Zur Homepage der Dissertation

# Eine Entwurfsmethodik für Data Warehouses

Dissertation
zur Erlangung des Grades des
Doktors der Ingenieurswissenschaften
am Fachbereich Informatik
der Carl von Ossietzky Universität Oldenburg
vorgelegt von:

**Dipl.-Inform. Olaf Herden**

Gutachter:

Prof. Dr. H.-J. Appelrath

Prof. Dr.-Ing. H. F. Schweppe

Tag der Disputation: 21. Dezember 2001

# Danksagung

Mein erster Dank im Rahmen dieser Dissertationsschrift, die während meiner Zeit als wissenschaftlicher Mitarbeiter am OFFIS (Oldenburger Forschungs– und Entwicklungsinstitut für Informatik–Werkzeuge und –Systeme) entstand, geht an Herrn Prof. Appelrath für die umsichtige Betreuung und Förderung dieser Arbeit. Er hat einerseits hilfreiche Leitlinien aufgezeigt, andererseits aber auch die nötigen akademischen Freiräume gewährt. Ebenso danke ich Herrn Prof. Schweppe für die Übernahme der Zweitbegutachtung.

Ein weiterer Dank geht an die im Umfeld Data Warehousing tätigen Kollegen des OFFIS, wobei ich insbesondere Holger Hinrichs und Arne Harren hervorheben möchte. Die Diskussionen und der Ideenaustausch mit beiden haben wesentlich zum Fortschritt dieser Arbeit beigetragen.

Für die Implementierungsarbeiten und Zusammenarbeit im ODAWA–Projekt bedanke ich mich bei Jürgen Meister und Jens Happe, für die Unterstützung bei der Evaluation bei den EKN–Mitarbeitern Kirsten Panienski, Joachim Kieschke und Martin Rohde.

Eine besonders harte Aufgabe im Rahmen einer solchen Arbeit ist das Korrekturlesen, wofür ich Heiko Tapken, Thorsten Teschke und Arne Harren für ihren akribischen Einsatz und so manch konstruktiven Hinweis danken möchte.

Weiterhin sage ich danke all den unbekannten Gutachtern von Workshops und Konferenzen wie auch den vielen, namentlich hier nicht alle zu nennenden Teilnehmern dieser Veranstaltungen, die durch ihre Reviews bzw. Diskussionsbeiträge den einen oder anderen Impuls gegeben haben. Insbesondere möchte ich hier die regelmäßigen Teilnehmer des GI–Arbeitskreises „Grundlagen des Data Warehousing“ nennen.

Nicht zuletzt möchte ich meinen Eltern für ihre Unterstützung und ihr Verständnis während meiner Studien– und Promotionszeit herzlich danken. Ihnen widme ich diese Arbeit.

Oldenburg, im Oktober 2001

Olaf Herden

# Zusammenfassung

Seit Jahren finden Data Warehouses (DWHs) als Kern entscheidungsunterstützender Informationssysteme ein starkes Interesse in Forschung und Praxis. Unter einem DWH wird dabei eine typischerweise separat von den operativen Systemen gehaltene Datenbank verstanden, deren Daten aus verschiedenen organisationsinternen Quellen integriert und häufig durch externe Daten angereichert werden. Anwendungsszenarien finden sich sowohl im betriebswirtschaftlichen Kontext wie auch in den Naturwissenschaften oder in medizinischen Registern.
Alle diese Anwendungen zeichnen sich dadurch aus, dass auf ihnen basierende Entscheidungen bzw. deren Konsequenzen sehr weitreichend und kostenintensiv sind, so dass die zugrundeliegende Aussage auf einer soliden Grundlage basieren muss. Hierzu ist es notwendig, dass das DWH einige Qualitätseigenschaften wie gute Wartbarkeit, Erweiterbarkeit und Skalierbarkeit aufweist.
Die heutige Praxis der Entwicklung von DWHs ist jedoch häufig noch durch Defizite geprägt: Der konzeptionellen Modellierung wird keine ausreichende Aufmerksamkeit geschenkt, physische Optimierungsmaßnahmen sind schlecht koordiniert, und die Behandlung von Metadaten wird vernachlässigt. Schließlich bieten existierende DWH–Werkzeuge keine Unterstützung heterogener Umgebungen. Daher scheitern einer Studie der Meta Group zufolge 20% der Projekte einer DWH–Einführung und 50% können als nur teilweise erfolgreich eingestuft werden.
Ziel dieser Arbeit ist die Konzeption einer Entwurfsmethodik, die sowohl auf Erfahrungen aus der Realisierung herkömmlicher Datenbanken (Grundlage des Ansatzes ist der allgemein akzeptierte Drei–Ebenen–Entwurf) zurückgreift, als auch die Besonderheiten von DWHs berücksichtigt. Für die konzeptionelle Ebene wurde dabei mit der MML (Multidimensional Modeling Language) eine Sprache entworfen, die sowohl multidimensionale als auch objektorientierte Aspekte aufweist. Als zugehörige graphische Notation wurde mit der $_m$UML eine multidimensionale Erweiterung der UML vorgenommen. Die Abbildung auf die logische Ebene wurde für das relationale Datenmodell realisiert, wobei eine abbildungsorientierte Vorgehensweise verfolgt wird, die neben Relationen und Attributen vor allem reichhaltige Metadaten erzeugt. Der Prozess des physischen Datenbankentwurfs schließlich läuft in drei Teilschritten ab: Nach einer eher technisch motivierten Erzeugung eines initialen Schemas wird dieses im zweiten Schritt durch z. B. Denormalisierungen den besonderen Bedürfnissen des Zielsystems angepasst. Für den letzten Teilschritt des physischen Datenbankentwurfs schließlich wird ein Framework vorgeschlagen, mit dem aufgrund von zusätzlichen Informationen über ein Schema (wie z. B. Extensionsgröße oder Verteilung von Attributwerten) und einer Menge definierter Aufgaben, die auf dem DWH auszuführen sind, ein koordiniertes Durchführen verschiedener Optimierungsmaßnahmen ermöglicht wird.

## Abstract

Recently, data warehouses (DWHs) as a decision support system's core caused a lively interest in research as well as in practice. Typically, a DWH is a database that is separated from the operational systems of an organisation. The data in a DWH are integrated from different inner–organisational data sources, probably enriched with external data.
Scenarios for applications of DWHs can be found in the economical domain as well as in natural sciences and medical registries. All these applications have in common that a decision's consequences are often cost–intensive. Therefore, statements representing the basis for such decisions have to be derived from a DWH that fulfills quality aspects like maintainability, extensibility and scaleability.
Today's development of DWHs however, is characterised by various problems: conceptual modelling is neglected, physical tuning actions are coordinated poorly and metadata handling is not payed sufficient attention. Moreover, existing tools do not support heterogeneous environments. A study carried out by Meta Group gives evidence of this statement — 20% of all DWH introduction projects fail and half of them is to be classified as only partially successful.
The goal of this thesis is the conception of a design method for DWHs. On the one hand this method is built on established techniques of operational database design (the approach is based on the widely accepted three level design), on the other hand specific aspects of DWHs are considered. For conceptual modelling the language MML (Multidimensional Modeling Language) is defined. MML features multidimensional as well as object oriented constructs. It is supplemented by the graphical notation $_m$UML, a multidimensional extension of the UML. The transformation to the logical layer is realised for the relational data model. Besides relations and attributes this transformation generates multifarious kinds of metadata. The physical database design step is divided into three substeps: after creating an initial schema a refinement is done which reflects the specific needs of the used OLAP server or database management system, respectively. As last substep of physical database design, a framework for schema tuning is proposed. The application of this framework supplements the schema with additional information (e. g. size of extension or derivation of attribute values) and specifies a set of tasks operating on the DWH. Thereby, the coordination of different tuning actions becomes possible.

# Inhaltsverzeichnis

# Teil I

# Einleitung und Grundlagen

# Kapitel 1

# Einleitung

## 1.1 Informationssysteme

Ein Informationssystem (IS) ist ein rechnerbasiertes System zum Sammeln, Manipulieren und Speichern von Daten, das die Arbeit in einer Organisation unterstützt. Historisch standen zunächst *datenverarbeitende Systeme* (auch als OLTP–Systeme (Online Transaction Processing) bezeichnet) im Mittelpunkt des Interesses, die das operative Geschäft häufig wiederkehrender Aktionen erleichtern sollten, z. B. Buchhaltungssysteme oder Platzreservierungssysteme bei Fluggesellschaften. Die Verarbeitung von Daten findet bei solchen Systemen größtenteils aufgrund von Benutzerinitiative statt, indem gezielt Daten angefragt werden oder eine datenändernde Transaktion durchgeführt wird. Manchmal geschieht die Nutzung auch automatisch, indem zu bestimmten fest definierten Zeitpunkten sog. Batch–Aufgaben angestoßen werden.
Waren datenverarbeitende Systeme in den Anfangsjahren ein Hilfsmittel für die tägliche Arbeit, so sind sie heute als essentiell wichtige Systeme einzustufen, von denen die Organisationen abhängig sind, um ihre tägliche Arbeit durchführen zu können. Daneben bestand aber schon seit den 60er Jahren der Wunsch auch Entscheidungsträgern verschiedener Funktionsbereiche und Hierarchieebenen einer Organisation als Grundlage zur Entscheidungsunterstützung die nötigen Informationen zur Verfügung zu stellen. Wünschenswert ist hierbei, dass die Bereitstellung der benötigten Informationen zeitnah, fehlerfrei, flexibel, ergonomisch, effizient und effektiv erfolgt. Dies kann nicht nur mittels der unmittelbar in der Datenbank abgelegten Daten erfolgen, sondern darüber hinaus müssen auch Informationen ermittelt werden, die aus diesen Daten durch Anwendung von z. B. statistischen Analysen gewonnen werden können. So sollte ein Buchhaltungssystem einen Überblick über Konten und Kostenstellen gewährleisten, und beim Platzreservierungssystem sollten Statistiken über die Auslastung bestimmter Flugstrecken möglich sein. Um solche Arten von Datenanalysen, die über die Unterstützung der operativen Aufgaben hinausgehen, zu unterstützen, entstanden in den letzten Jahrzehnten verschiedene Konzepte und Systeme, die auf den operativen Systemen aufsetzten und unter Bezeichnungen wie *Management Information System* (MIS), *Executive Information System* (EIS), *Führungsinformationssystem* (FIS), *Chefinformationssystem* (CIS), *Entscheidungsunterstützungssystem* (EUS) oder *Decision Support System* (DSS) bekannt wurden[1]. Dabei entstand eine Architekturform wie in Abbildung 1.1 zu sehen: Aufbauend auf den operativen Systemen wurden *Berichtswerkzeuge*, *integrierte Lösungen* und mit der zunehmenden Verbreitung von PC–Arbeitsplätzen in den 80er und 90er Jahren auch auf *Tabellenkalkulationen* basierende Lösungen angesiedelt.

[1] Einen Überblick über die historische Entwicklung geben [GGC97] und [Kem99].

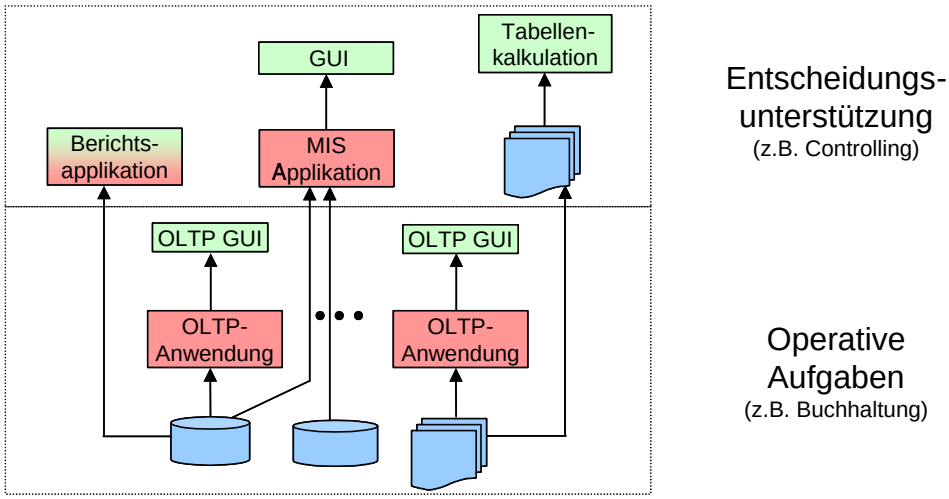



Abbildung 1.1: Architektur ohne Data Warehouse

Auf einer solchen Architektur basierende Systeme bringen jedoch eine Reihe von Problemen mit sich:

- Meistens wird eine *Beschränkung auf interne Daten* und dabei häufig auch nur auf ein System innerhalb einer Vielzahl von Systemen in einer Organisation vorgenommen; gerade für mittel– und langfristige Entscheidungen sind aber auch externe Daten, die z. B. von Meinungsforschungsinstituten erhoben werden oder von statistischen Ämtern stammen, von Belang.
- Oftmals werden in unterschiedlichen Teilorganisationen und deren Informationssystemen voneinander abweichende Begriffe verwendet, so dass Berichte aus unterschiedlichen Systemen aufgrund *uneinheitlicher Terminologie* nur schwer oder gar nicht vergleichbar sind.
- Daten in den operativen Systemen verweilen nur für einen beschränkten Zeitraum, so dass gerade zur Entscheidungsunterstützung wichtige Langzeitanalysen nicht möglich sind.
- Durch den direkten Zugriff auf die operativen Systeme ist diese Architekturform *wenig flexibel* gegenüber neuen Anforderungen.
- Durch die fehlende integrierte Sicht auf die Daten in unterschiedlichen Datenquellen sind Auswertungen nur *schwer vergleichbar*.
- Durch die direkte Kopplung an operative Datenquellen treten *technische Probleme* wie Performanzengpässe und *organisatorische Probleme* wie datenschutzrechtliche Aspekte auf.

Aufgrund der Existenz dieser technischen Nachteile können Architekturen wie die in Abbildung 1.1 skizzierte entscheidungsunterstützende Aufgaben in Organisationen nur in eingeschränktem Maße unterstützen. Andererseits wuchs in den letzten Jahren der Bedarf nach Datenanalysen als Entscheidungsgrundlage beträchtlich. Diese Tendenz begründet sich z. B. in der Marktsättigung im klassischen Handel oder in der Liberalisierung des Telekommunikations– und Energieversorgungsmarktes. Als Anforderungen an diese Systeme sind dabei zu nennen:

- Flexibilität, Interaktion und Effizienz beim Analysieren
- Vorliegen eines historisierten Datenbestandes
- Integration und Bereinigung des Datenbestandes

- Verwaltung sehr großer Datenbestände
- Entkopplung von operativen Systemen.

Um all diesen Anforderungen gerecht zu werden, hat sich in den 90er Jahren mit *Data Warehouse–Systemen* (DWS) eine Klasse von Architekturen herausgebildet, in deren Zentrum eine als *Data Warehouse* (DWH) bezeichnete Datenbank steht, die die Daten aus verschiedenen operativen Quellen einer Organisation integriert und eventuell um externe Daten anreichert. Neben dem klassischen Berichtswesen ist das vorrangige Anwendungsfeld eine als *OLAP* (Online Analytical Processing) bezeichnete Form der interaktiven und explorativen Analyse von im DWH vorliegenden Daten. Abbildung 1.2 skizziert die typische Architektur eines DWS[2].

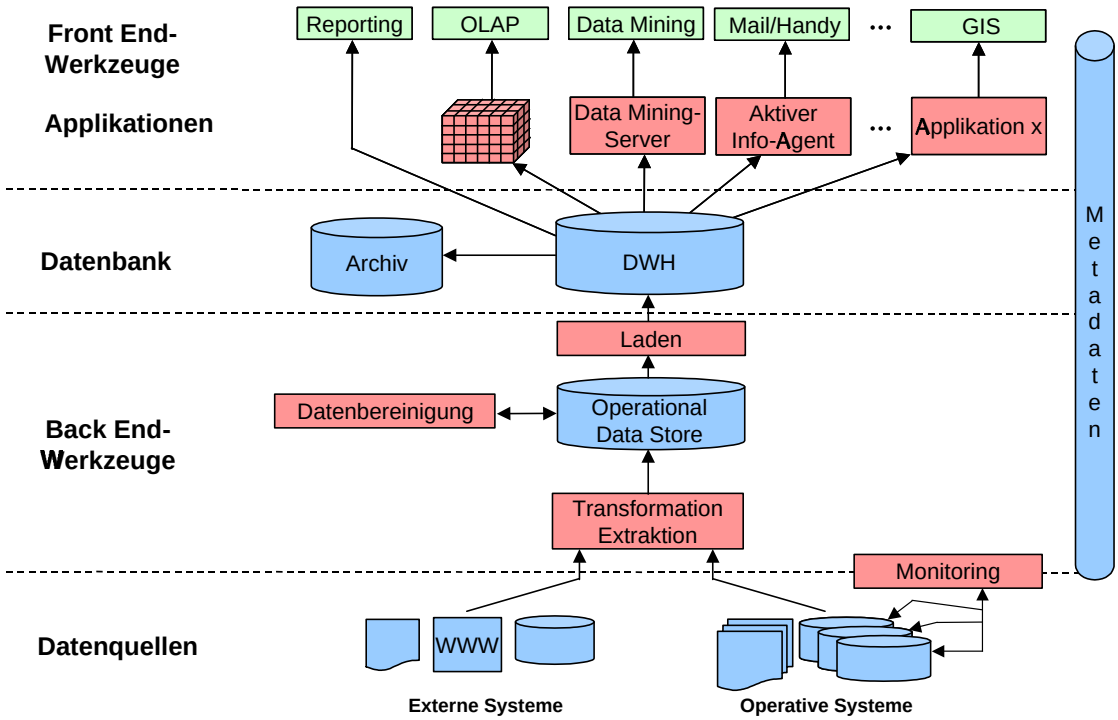



Abbildung 1.2: Architektur mit Data Warehouse

Die Anwendungsfelder für DWS sind u. a.:

- *Betriebswirtschaft*: In praktisch jedem Bereich eines Unternehmens finden sich auf den unterschiedlichen Ebenen Anwendungsfelder, vor allem im Marketing und Rechnungswesen/Controlling durch z. B. Einsatz klassischer Produkt–, Kunden– oder Segmentanalyse. Aktuell bekanntestes Projekt ist das DWH von Wal Mart [Wes00].
- *Wissenschaft und Technik*: Bei empirischen Untersuchungen in der Wissenschaft fallen oft große Datenmengen, z. B. in Form von Messwerten, an. Ein bekanntes Projekt in diesem Sektor ist *Earth Observing System* (EOS) [Mic91] aus dem Bereich der Klima– und Umweltforschung, in dem große Mengen an meteorologischen Daten von Bodenstationen und Satelliten gesammelt werden, wobei täglich ca. 1 TeraByte neue Daten hinzukommt. Mit Hilfe von statistischen Untersuchungsmethoden sollen schließlich Informationen extrahiert werden, die zur Gewinnung von neuen Erkenntnissen beitragen.

[2] Die einzelnen Komponenten dieser Architektur werden in Kapitel 2 detailliert vorgestellt.

- *Statistik*: Schon seit den 70er Jahren sind *Statistical and Scientific Databases* (SSDB) [Mic91, Ruf97] bekannt, die analog zu einem DWH die Integration, Verarbeitung und Analyse großer Rohdatenmengen zum Ziel haben. Ein Teilgebiet ist hier die *Epidemiologie*, die sich mit der Untersuchung der Verteilung und der Determinanten von Gesundheitszuständen oder –ereignissen in spezifischen Bevölkerungsgruppen und der Anwendung dieser Erkenntnisse zur Kontrolle von Gesundheitsproblemen beschäftigt[3].

## 1.2 Motivation

Alle oben erwähnten Szenarien für den Einsatz von DWH weisen die Gemeinsamkeit auf, dass die möglichen Folgen der Analysen wirtschaftlich (z. B. Umstellen der Produktpalette) oder „politisch“ (z. B. Bekanntgabe von Krankheitshäufungen an einem bestimmten Ort) große Konsequenzen nach sich ziehen können. Aus diesem Grunde ist neben einem „sauberen“ Datenbestand auch eine gut konzipierte Datenbank unerlässlich, die sinnvolle Analysen in nachvollziehbarer Art erst ermöglicht. *Gut konzipiert* zielt auf Eigenschaften wie Wartbarkeit, Erweiterbarkeit, Skalierbarkeit und Performanz ab.
Als weiterer Beleg für die Praxisrelevanz der Thematik ist das hohe Investitionsvolumen von Unternehmen in DWH zu nennen. So nahm in den USA der DWH–Markt von 2 Mrd. US–Dollar im Jahre 1995 auf 8 Mrd. US–Dollar in 1998 zu [KRRT98]. Ein Anhalten dieses Trends wird in verschiedenen Studien auch für die kommenden Jahre erwartet. In [Met00] wird ein Ansteigen des OLAP–Marktes in Deutschland von knapp 100 Mio. US–Dollar 1998 über gut 200 Mio. US–Dollar in 2000 auf rund 600 Mio. US–Dollar im Jahre 2002 vorhergesagt. [Ste00] prognostiziert ein Ansteigen der weltweiten Investition von knapp 60 Mrd. US–Dollar auf über 150 Mrd. US–Dollar in 2005.
In der Literatur herrscht im Wesentlichen Einigkeit darüber, dass existierende Entwurfsmethodiken nur eingeschränkt auf den Entwurf von DWH übertragbar sind, z. B. „Regardless of the underlying paradigm, traditional (individual) software systems are developed according to certain requirements. In contrast to that, data warehouses are usually built upon existing operational systems, with requirements being clear at most partially. This is due to the fact that operational systems are developed to support specific, standardized business transactions, while the data warehouse has to support ever–changing, individual decisions in a dynamic business setting.“ [Win99]
Dies zeigt, dass die heutige Praxis des DWH–Entwurfs aufgrund des Übernehmens existierender Entwurfsmethodiken häufig defizitär ist, u. a.:

- wird die konzeptionelle Modellierung „stiefmütterlich“ behandelt,
- werden physische Optimierungsmaßnahmen meistens isoliert betrachtet, anstatt in Kosten–/Nutzenanalysen systematisch koordiniert zu werden,
- wird die Koordination des Metadatenmanagements durch Verwendung proprietärer, lokaler Repositories nicht umfassend genug gehandhabt,
- weisen existierende Werkzeuge eine geringe Offenheit in Form von Schnittstellen auf,
- fokussieren existierende Ansätze auf genau ein Zielsystem.

Die folgenden Ergebnisse verschiedener Studien sollen diesen Zustand belegen: So scheitern laut einer Studie der Meta Group in der Praxis viele Projekte vollständig und 40% aller auf diesem Gebiet angesiedelten Projekte sind als (teilweise) gescheitert zu klassifizieren [Met99b].
In [Dat99] hat das Unternehmen DATA MART Consulting in Zusammenarbeit mit der TU Darmstadt

[3] In dieser Domäne ist auch das Beispiel der im Rahmen dieser Arbeit durchgeführten Evaluation angesiedelt.

eine Studie erarbeitet, in der Theorie und Realität bei bereits realisierten Data Mart– bzw. DWH–Projekten unter die Lupe genommen werden. Ausgangspunkt der Untersuchung ist eine Befragung von knapp 100 Unternehmen des deutschsprachigen Raums, die bereits Data Mart– und/oder DWH–Projekte durchgeführt haben. Zu den Fragestellungen zählten u. a. die betriebswirtschaftlichen Anforderungen, das Projektmanagement sowie die eingesetzten Technologien. Die Ergebnisse sind in verschiedene Matrizen eingeflossen, die nach Fachbereichen bis hin zur Geschäftsführung bzw. den Vorständen oder nach Branchen gegliedert sind. Anhand der Aufstellungen wurden die aktuellsten Informationen der jeweiligen Softwarehersteller eingeholt. In einem Testcenter wurden schließlich die erarbeiteten Anforderungen anhand von Beispiel–Anwendungen durchgespielt. Eine abschließende Befragung der Unternehmen glich die Ergebnisse der Tests mit den tatsächlich eingesetzten Technologien und Werkzeugen ab. Als wesentliche Ergebnisse konnten dabei festgehalten werden:

- 15% der DWH–Projekte wurden nie aktiv.
- In einigen Branchen wurde nur ein geringer Prozentsatz der (betriebswirtschaftlichen) Anforderungen erfüllt.

Als Ursachen für die Resultate der beiden Studien werden eine Reihe von Gründen angeführt, zu denen neben der Problematik der Datenbeschaffung und –integration auch das methodische Vorgehen genannt werden.
Ebenso stellt eine Studie des Bundesministeriums für Bildung und Forschung über die Softwareentwicklung (wozu insbesondere auch der Datenbankentwurf zu zählen ist) in Deutschland verschiedene Defizite fest [Bun01], u. a.:

- Nur die Hälfte aller Unternehmen, die in Deutschland Software entwickeln, setzen ein Vorgehensmodell ein.
- Durch fehlende systematische Wege zur Problemfindung können Entwicklungsentscheidungen häufig nicht nachvollzogen werden, womit eine langfristige Pflege der Anwendungen nur mit großem Aufwand möglich ist.
- Qualitätssicherung findet in der Regel erst in den späten Phasen der Softwareentwicklung statt.

Mit [DHP+99] stellt eine weitere empirische Studie Defizite bez. des Softwareentwicklungsprozesses fest:

- In Prozessdefinitionen beschriebene Vorgehensmodelle sind häufig zu allgemein und nicht adäquat für die angegebene Aufgabenstellung.
- In einem Viertel der untersuchten Fälle wurde sowohl auf eine formale wie semi–formale Beschreibungstechnik verzichtet, es werden lediglich textuelle Beschreibungen erstellt und direkt daraus der Quellcode abgeleitet.
- Nur in knapp der Hälfte der Fälle kamen CASE–Werkzeuge zum Einsatz.
- In sehr vielen Projekten wurde nur unzureichend oder gar nicht dokumentiert.

Eine auf den DWH–Kontext fokussierende Untersuchung ist [Gar98], in der als Gründe für das Scheitern von DWH–Projekten u. a. mangelnde Kommunikation mit den potenziellen Benutzern und das Nichtanwenden einer bewährten Methodik genannt werden. In [Kim96] werden E/R–basierte Datenmodelle aufgrund ihrer schlechten Kommunikationsbasis mit dem Endbenutzer und der fehlenden Navigierbarkeit durch DBMS als ungeeignet für den DWH–Entwurf eingestuft.
Diese Überlegungen und Studienergebnisse lassen den Bedarf nach einer Entwicklungsmethodik für DWH aufkommen.

## 1.3 Zielsetzung

Im Rahmen dieser Arbeit soll eine Entwurfsmethodik für DWH konzipiert werden, die einerseits bewährte Konzepte aus dem Entwurf herkömmlicher Datenbanken übernimmt, andererseits aber spezielle Aspekte von DWH berücksichtigt. Insbesondere sind dies:

- Das Vorgehen soll dem *Drei–Ebenen–Entwurf* folgen.
- Während der konzeptionellen Modellierung soll eine *multidimensionale Sicht* auf die Daten ermöglicht werden, weil diese in der Analysephase für ein gutes Problemverständnis und als Kommunikationsbasis mit potenziellen Anwendern benötigt wird.
- Die konzeptionelle Modellierung ist wichtigster Bestandteil des Prozesses, daher soll schon in dieser frühen Phase der *Qualitätssicherung* besondere Bedeutung zukommen.
- Nach Beendigung des konzeptionellen Entwurfs soll die Methodik das weitere Vorgehen weitgehend automatisch vorgeben, an ausgewählten Stellen soll aber *Interaktion* durch den Benutzer möglich sein.
- Die Methodik soll *zielsystemunabhängig* sein.
- Um Nachvollziehbarkeit des Entwicklungsprozesses sowie die für langfristige Wartung und Erweiterbarkeit notwendige *Dokumentation* zu gewährleisten, soll der gesamte Prozess Metadaten–gestützt erfolgen.
- Zum Nachweis der Umsetzbarkeit der Methodik soll die softwaretechnische Umsetzung in Form eines *Prototypen* erfolgen.
- Um die Anwendbarkeit der Methodik zu zeigen, soll mit Hilfe dieses Prototypen eine *Evaluation* an einem ausreichend großen Beispiel durchgeführt werden.

## 1.4 Aufbau der Arbeit

Der Aufbau der Arbeit gliedert sich in vier Teile. Teil I stellt neben dieser Einleitung grundlegende Begriffe und Konzepte aus den Bereichen Data Warehouse, Datenmodelle bzw. –modellierung mit dem Schwerpunkt multidimensionales Datenmodell, Entwurfsmethodiken für Datenbanken, physischer Datenbankentwurf und Metadaten vor. Teil II widmet sich der im Rahmen dieser Arbeit konzipierten Entwurfsmethodik. In Teil III werden die softwaretechnische Umsetzung des Konzeptes beschrieben und wesentliche Ergebnisse der Evaluation dargestellt. Eine Zusammenfassung und ein Ausblick folgen in Teil IV, bevor einige Anhänge und Verzeichnisse die Arbeit abschließen.

# Kapitel 2

# Data Warehouse–Systeme

In diesem Kapitel soll die allgemeine Architektur eines Data Warehouse–Systems (DWS) vorgestellt werden. Ein DWS enthält neben der Datenbank (dem DWH) „die Menge aller Komponenten (und deren Beziehungen untereinander), die für das Beschaffen, Aufbereiten, Speichern und Analysieren von Daten“ [BG01] notwendig sind. Abbildung 2.1 zeigt eine solche Architektur, deren einzelne Komponenten im Folgenden vorgestellt werden.

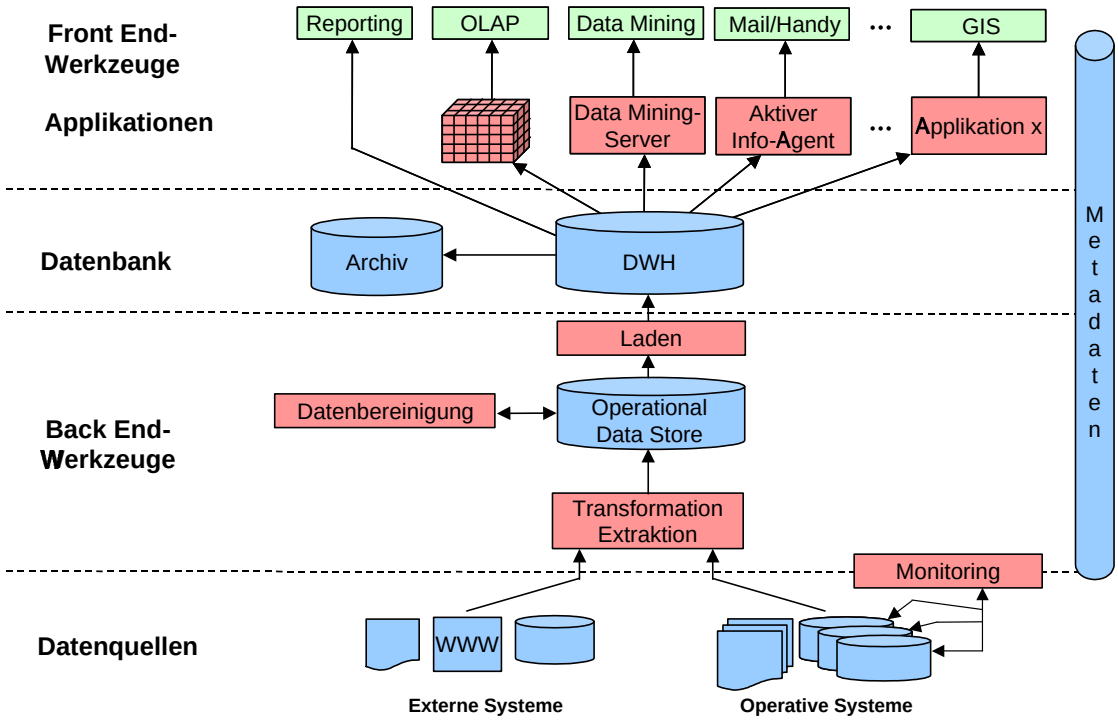



Abbildung 2.1: Referenzarchitektur DWS

## 2.1 Datenquellen

Datenquellen sind der Ausgangspunkt einer datenflussorientierten Betrachtung der Architektur. Die typischerweise stark heterogenen Datenquellen können internen Ursprungs (operative Systeme der eigenen Organisation) oder aber extern sein, wobei insbesondere das Internet als Quelle in Frage

kommt. Der Auswahl geeigneter Quellen zum Aufbau eines DWS kommt erhebliche Bedeutung zu. Bei dieser Auswahl spielen insbesondere die Faktoren Qualität der Daten, technische wie organisatorische Verfügbarkeit und der Preis für den Erwerb der Quelldaten eine Rolle.

## 2.2 Back End–Bereich

Der Back End–Bereich wird von einer Reihe von Werkzeugen gebildet, die zwischen den Datenquellen und dem DWH angesiedelt sind. Neben *Monitoren* (siehe Abschnitt 2.2.1) sind dies vor allem Werkzeuge zur *Extraktion* (siehe Abschnitt 2.2.2), *Transformation* (siehe Abschnitt 2.2.3) und zum *Laden* (siehe Abschnitt 2.2.4). Aufgrund der Anfangsbuchstaben der Werkzeugtypen wird der Back End–Bereich auch *ETL–Bereich* und der hier durchgeführte Prozess entsprechend *ETL–Prozess* genannt. Eine weitere wichtige Aufgabe des Back End–Bereichs ist die Sicherstellung der *Datenqualität*, die durch ein durchgängiges und umfassendes Datenqualitätsmanagementsystem gewährleistet werden sollte [Hin00, Hin01, HA01], welches eine hohe Güte der zu analysierenden Daten garantiert[1]. Als physischer Zwischenspeicher dient im Back End–Bereich eine als *Operational Data Store* (ODS) bezeichnete Datenbank.

### 2.2.1 Monitore

*Monitore* sind für die Entdeckung von Datenänderungen (neue, geänderte oder gelöschte Datensätze) in Quellsystemen zuständig. Um den ODS — und nachfolgend auch das DWH — aktuell zu halten, müssen Aktualisierungen in den Quellsystemen inkrementell in das DWS „propagiert“, d. h. diesem bekannt gegeben werden. Die konkrete Funktionsweise eines Monitors hängt unmittelbar von den Charakteristika des angeschlossenen Quellsystems[2] sowie von den Anforderungen der Analysekomponenten ab. Folgende Monitoring–Strategien können unterschieden werden [VGD99]:

- *Trigger–basiert:* Handelt es sich bei der Datenquelle um ein Datenbanksystem, welches aktive Mechanismen in Form von Triggern unterstützt, kann das Monitoring erfolgen, indem jede Datenmanipulation einen Trigger auslöst (z. B. Post Update), der das geänderte Tupel in eine Datei oder eine andere Datenstruktur schreibt.
- *Replikationsbasiert:* Moderne DBMS bieten Replikationsdienste an. Diese Dienste können so spezifiziert werden, dass sie geänderte Tupel in spezielle Tabellen schreiben.
- *Zeitstempelbasiert:* Jedem Datensatz ist ein Zeitstempel zugeordnet, der im Falle einer Änderung des Datensatzes auf den Zeitpunkt der Änderung gesetzt wird. Anhand der Zeitstempel kann später entschieden werden, welche Datensätze sich nach dem Zeitpunkt der letzten Extraktion geändert haben. Temporale DBMS [Sno95] bieten eine explizite Unterstützung der Zeitdimension an, sind aber bisher nicht über den Forschungsstatus hinausgekommen.
- *Log–basiert:* In diesem Fall nutzt man die Fähigkeit von DBMS aus, vorgenommene Transaktionen in einer Log–Datei zu protokollieren. Durch Analyse einer solchen Log–Datei kann ermittelt werden, welche Daten sich geändert haben.
- *Snapshot–basiert:* Bei dieser Variante wird der Datenbestand einer Quelle in periodischen Zeitabständen in eine Datei, den sog. *Snapshot*, geschrieben. Durch einen Vergleich von Snapshots (Delta–Berechnung) können Änderungen identifiziert werden.

[1] Da dieser Aspekt außerhalb des Fokus dieser Arbeit liegt, wird in diesem Kapitel nicht näher darauf eingegangen.
[2] Als Quellsysteme kommen nicht nur DBMS in Frage, sondern auch Flat Files, XML–Dokumente etc.

Von den vorgestellten Monitoring–Strategien erfordert die Snapshot–basierte Variante den größten (Implementierungs–)Aufwand, da sie keine vom Quellsystem bereitgestellten Dienste nutzt. Jedoch ist gerade bei Altsystemen, die solche Dienste i. Allg. nicht anbieten, Snapshot–Monitoring häufig die einzige anwendbare Technik zur Entdeckung von Änderungen im Datenbestand.

### 2.2.2 Extraktionskomponenten

Die *Extraktionskomponente* einer an ein DWS angebundenen Datenquelle ist für die Übertragung von Daten in den Transformationsbereich verantwortlich. Je nach verwendeter Monitoring–Strategie (siehe Abschnitt 2.2.1) gestaltet sich die Extraktion sehr unterschiedlich: Bei der Trigger–basierten Variante sind die geänderten Tupel aus den entsprechenden Dateien auszulesen, bei Verwendung der Replikationsdienste können sie per SQL–Anfrage aus den Replikationstabellen selektiert werden. Die zeitstempelbasierte Variante erfordert lediglich die Selektion von Tupeln anhand ihres Zeitstempels. Bei der Log– bzw. Snapshot–Variante hängt das Vorgehen von der gewählten Umsetzung der Log–Analyse bzw. des Snapshot–Vergleichs ab. Werden die als geändert identifizierten Tupel beispielsweise in eine Datei geschrieben, so ist diese Datei zu importieren.
Eine grundlegende Entscheidung besteht in der Festlegung, welche Datenquellen bzw. Ausschnitte daraus in ein DWS zu integrieren sind. Diese Selektion hängt stark von der inhaltlichen Relevanz der Datenquellen für die geplanten Auswertungen sowie von der Qualität der Quelldaten ab. Bei der Extraktion werden die selektierten bzw. durch das Monitoring als geändert identifizierten Daten aus den quellenspezifischen Datenstrukturen ausgelesen und in die Datenstrukturen des Transformationsbereichs überführt. Aufgrund der u. U. großen Datenvolumina kann eine Komprimierung der zu transferierenden Daten sinnvoll sein. Die Zeitpunkte, an denen eine Extraktion durchgeführt wird, sollten je nach Bedarf unterschiedlich gewählt werden. Es werden folgende prinzipielle Vorgehensweisen unterschieden [KRRT98]:

- *Periodisch:* Die Extraktion wird in periodischen Zeitabständen durchgeführt, wobei die Periodendauer von der Dynamik der Daten bzw. von den gestellten Anforderungen an die Aktualität der Daten abhängt. So sind z. B. Börsenkurse oder Wetterdaten i. Allg. (mehrmals) täglich zu aktualisieren, während Daten über technische Spezifikationen von Produkten typischerweise beständiger sind und daher mit einer längeren Periodendauer auskommen.
- *Anfragegesteuert:* In diesem Fall wird die Extraktion durch eine explizite Anfrage angestoßen. Wenn beispielsweise eine Produktgruppe um einen neuen Artikel erweitert wird, so kann die Extraktionskomponente angewiesen werden, die in den operativen Quellen zu diesem neuen Artikel gespeicherten Informationen in das DWS zu übertragen.
- *Ereignisgesteuert:* Häufig ist es sinnvoll, einen Extraktionsvorgang durch ein Zeit–, Datenbank– oder externes Ereignis auszulösen. Ein typisches Datenbankereignis wäre beispielsweise das Erreichen einer à priori festgelegten Anzahl von Änderungen. Ein externes Ereignis würde z. B. vorliegen, wenn ein Börsenindex eine bestimmte Marke über– oder unterschreitet. Strenggenommen sind auch periodische und anfragegesteuerte Extraktionen ereignisgesteuert, da sie einem speziellen Zeitereignis bzw. einem durch den Anwender ausgelösten Ereignis entsprechen.
- *Sofort:* Bei besonders hohen Anforderungen an die Aktualität von Daten, z. B. bei Börsenkursen, kann es erforderlich sein, Änderungen in den operativen Quellen unmittelbar in den ODS eines DWS zu propagieren. Die Daten im ODS sind damit praktisch immer genauso aktuell wie die Daten in den operativen Systemen.

Die technische Realisierung der Extraktion erfolgt typischerweise mit Hilfe von Gateways (Schnittstellen zwischen Netzwerken) und Standard–Datenbankschnittstellen (z. B. ODBC oder OLE DB [Woo99]).

### 2.2.3 Transformationskomponente

Ein DWS wird i. Allg. von mehreren heterogenen Datenquellen versorgt, die Daten mit disjunkten oder sich überschneidenden Inhalten in unterschiedlichen Repräsentationsformen liefern. Bevor die aus den Quellen extrahierten Daten in ein DWH geladen werden können, müssen sie in einen Zustand einheitlicher Repräsentation gebracht werden. Transformationen, die dem Zweck der Vereinheitlichung dienen, werden unter dem Begriff *Data Migration* zusammengefasst. Die transformierten Daten werden schließlich im ODS abgelegt. Dabei ist ggf. eine Historisierung inkrementeller Aktualisierungen vorzunehmen. Ein im DWS gespeicherter Datensatz, zu dem es eine Änderung in einer Datenquelle gegeben hat, darf daher nicht grundsätzlich mit den geänderten Werten überschrieben werden. Stattdessen ist der geänderte Datensatz ggf. mit einem Zeitstempel zu versehen und zusätzlich zu dem bereits „veralteten“ Datensatz im DWS abzulegen.
In diesem Schritt wird ausschließlich die Transformation von Daten behandelt, nicht hingegen die Transformation von Datenstrukturen bzw. Schemata. Schemaintegration ist natürlich eine entscheidende Voraussetzung für die Datenintegration. Zu diesem Thema sei auf die bereits umfassende Literatur (z. B. [Con97]) verwiesen.

### 2.2.4 Ladekomponente

Die *Ladekomponente* ist für die Übertragung von konsolidierten Daten aus dem ODS in die analyseorientierten Strukturen des DWH zuständig. Zur technischen Durchführung dient dabei oft das Ladewerkzeug des jeweils zugrunde liegenden DBMS, z. B. der *Oracle SQL*Loader* [DG98]. Es kann zwischen *Online*– und *Offline*–Ladevorgängen unterschieden werden [AM97a]: Bei Online-Ladevorgängen steht das DWH auch während des Ladens für Anfragen zur Verfügung, bei Offline-Ladevorgängen ist dies nicht der Fall. Üblicherweise findet nur das initiale Laden offline statt. Inkrementelle Updates sollten hingegen nur online durchgeführt werden, um den laufenden Betrieb des DWS nicht unterbrechen zu müssen. Das Zeitfenster für den Ladevorgang sollte allerdings so gewählt werden, dass die Beeinträchtigung der Benutzung minimiert wird, z. B. nachts oder an Wochenenden. Aufgrund der möglicherweise sehr großen Datenvolumina können spezielle Maßnahmen zur Effizienzsteigerung erforderlich werden, z. B. durch Parallelisierung der Ladevorgänge. Weiterhin ist analog zur Extraktionskomponente eine angemessene Ausnahme– und Fehlerbehandlung unerlässlich.

## 2.3 Datenbank

In dieser Architekturschicht sind mit dem DWH (siehe Abschnitt 2.3.1) und dem *Archiv* (siehe Abschnitt 2.3.3) die zwei wesentlichen datenspeichernden Komponenten angesiedelt. Als Variante bzw. denkbare Gestaltungsmöglichkeit des DWH sind sog. *Data Marts* populär, die in Abschnitt 2.3.2 kurz vorgestellt werden.

### 2.3.1 Data Warehouse

Zentrale Komponente eines DWS ist das DWH, eine durch folgende Charakteristika gekennzeichnete Datenbank:

- Die Datenbank ist physisch realisiert,
- sowohl Schema wie auch Daten sind integriert,
- das Schema ist analyse–orientiert,

- i. Allg. werden an den Daten keine Modifikationen vorgenommen,
- in der Regel liegen die Daten historisiert vor.

Die wesentlichen Unterschiede zwischen herkömmlichen operativen Datenbanken und DWHs fasst Tabelle 2.1 zusammen [BS97, BG01].

| **Kriterium** | **Operative Datenbank** | **DWH** |
|---|---|---|
| **Zweck und Anwendercharakteristika** | | |
| Verwendung | Transaktional, Unterstützung und Abwicklung der Geschäftsprozesse | Analytisch, Information für Entscheidungsträger |
| Anwendertyp | Sachbearbeiter | Analysten, Manager |
| Anwenderzahl | Sehr viele | Wenige |
| **Zugriffscharakteristika** | | |
| Zugriffsart von Applikation | Lesen, Schreiben, Modifizieren, Löschen | Lesen, periodisches Hinzufügen |
| Transaktionsdauer und –typ | Kurze Lese– und Schreibtransaktionen | Lange Lesetransaktionen |
| Abfragestruktur | Einfach strukturiert | Komplex, rechenintensiv |
| Abfragetyp | Vorhersagbar | Häufig ad–hoc–Anfragen |
| Zugriffsart | Einzelne Datensätze | Bereichsanfragen |
| Volumen der Anfrageergebnisse | Wenige Datensätze | Viele Datensätze |
| Erwartete Antwortzeiten | (Milli–)Sekunden | Sekunden bis Minuten |
| Update | Laufend, konkurrierend | Ergänzend |
| **Daten– und Schemacharakteristika** | | |
| Datenquellen | (Meistens) eine | (Fast immer) mehrere |
| Eigenschaften der Daten | Nicht abgeleitet, zeitaktuell, autonom, dynamisch | Abgeleitet, konsolidiert, historisiert, integriert, stabil |
| Granularität der Daten | Detaildaten | Detaildaten und Aggregate |
| Aktualität der Daten | Online, Realtime | Unterschiedlich, aufgabenabhängig |
| Zeithorizont der Daten | Aktuelle Daten | Historisierte Daten |
| Datenschema | Normalisiert | Zweckgebunden analyse–orientiert |
| Datenvolumen | Megabyte bis Gigabyte | Gigabyte bis Terabyte |

Tabelle 2.1: Vergleich operative Datenbanken und DWH

Diese Übersicht unterstellt die idealtypische Vorstellung einer weitgehenden „Orthogonalität“ von OLTP–Datenbanken und DWHs aus der gängigen Literatur. In einigen Quellen (z. B. [ZS99]) wird darauf hingewiesen, dass „die Welt in Wirklichkeit nicht so schwarzweiß ist“. So können in bestimmten Anwendungsszenarien auch in den OLTP–Systemen historisierte Daten existieren (z. B. Status bei Bestellungen) oder bestimmte Anwendungen (z. B. Marketing–Kampagnen) können ein Schreiben auf das DWH verlangen. Ebenso wird man in der Praxis in den operativen Systemen nicht durchgängig auf normalisierte Schemata treffen.

### 2.3.2 Data Marts

Die Realisierung eines zentralen DWH ist in einigen Fällen konzeptionell oder technisch schwer durchsetzbar. Beispielsweise kann eine solche Lösung zu teuer sein oder das Projekt seiner Entstehung zu komplex oder ressourcenintensiv. Technische Probleme kann eine zentrale Lösung hinsichtlich der Skalierbarkeit bei steigender Benutzerzahl und/oder anwachsenden Datenbeständen aufwerfen. Aus diesem Grunde haben sich *Data Marts* („kleine DWHs“) als Lösung etabliert, was aus Datenbanksicht einer Verteilung des DWH–Datenbestandes entspricht. Prinzipiell lassen sich die beiden in Abbildung 2.2 dargestellten Ausprägungen unterscheiden. *Abhängige* Data Marts sind Extrakte aus einem zentralen, integrierten DWH, während *unabhängige* Data Marts als isolierte Sichten auf die Quellsysteme ohne Verwendung einer „großen, gemeinsamen Datenbasis“ realisiert werden.

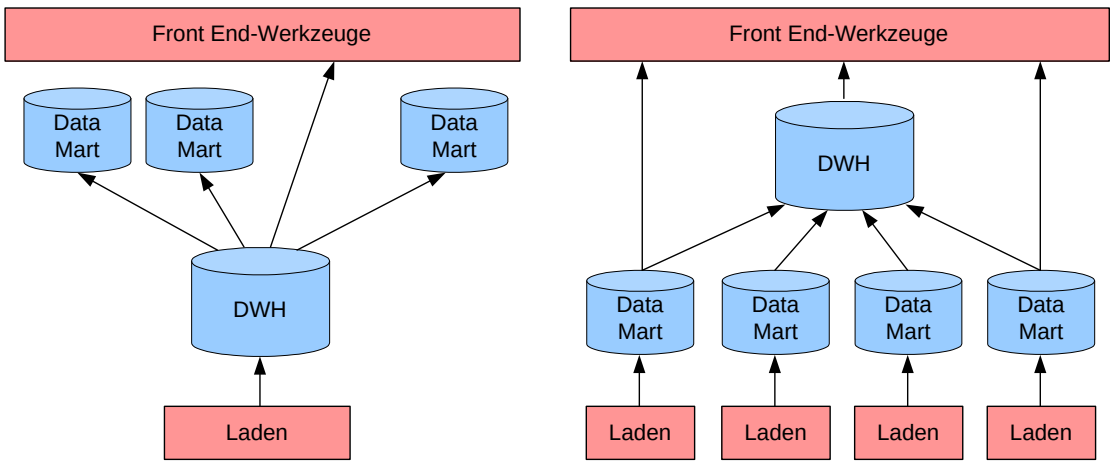



Abbildung 2.2: Abhängige und unabhängige Data Marts

Bei der Konzeption von Data Marts ist die wichtigste Frage, welche Daten der Extrakt enthalten soll. Prinzipiell gibt es drei verschiedene Arten, die allerdings häufig in Kombination angewendet werden:

- *Struktureller Extrakt*: Nur ein Teil des DWH–Schemas wird in den Data Mart repliziert und somit wird nur eine bestimmte Gruppe von Analysen ermöglicht.
- *Inhaltlicher Extrakt*: Es wird zwar das gesamte Schema, aber nur ein Teil der Extension im Data Mart vorgehalten, z. B. nur die Daten einer bestimmten Periode oder einer bestimmten Organisationseinheit.
- *Aggregierter Extrakt*: In den Data Mart werden die Daten mit einem geringeren Detaillierungsgrad, d. h. auf einer höheren Abstraktionsebene, gespeichert, so dass sich das Datenvolumen verringert.

### 2.3.3 Archiv–Datenbank

In einem DWS muss für eine definierte „Entsorgung“ der Daten aus dem DWH gesorgt werden. Gründe hierfür können mangelndes Interesse an veralteten Daten oder aber Platz– und Performanzprobleme sein. Bei dieser Entsorgung stellt sich die Frage nach endgültigem Löschen oder der Übernahme in eine *Archiv–Datenbank*. In dieser werden die Daten an einem separaten Ort langfristig gehalten,

um sie im Bedarfsfall wieder im DWH zur Verfügung stellen zu können. Häufig wird der Archivierungsvorgang auch mit einem erneuten Schreibvorgang in das DWH kombiniert: Die detaillierten Daten werden in die Archiv–Datenbank übertragen, eine verdichtete Version (aggregierter Extrakt) dieser Daten wird im DWH abgelegt.

## 2.4 Front End–Werkzeuge

*Front End–Werkzeuge* sind auf dem DWH basierende Applikationen einschließlich ihrer Benutzerschnittstellen. Je nach Funktionalitätsumfang lassen sich verschiedene Klassen von Front End–Werkzeugen identifizieren.

#### Berichts– und Abfragewerkzeuge

*Berichtswerkzeuge* erzeugen mittels vordefinierter, eventuell parametrisierter Abfragen Auswertungen der Daten, reichern diese eventuell um einfache arithmetische Operationen an und repräsentieren sie in Form von Berichten. Diese können tabellarisch oder in Form von Diagrammen dargestellt sein. Berichtswerkzeuge können entweder für die Generierung periodisch wiederkehrender Berichte im Batch–Betrieb eingesetzt werden oder als Endbenutzeranwendungen für unregelmäßige Auswertungen zur Verfügung stehen.
Ebenfalls in diese Kategorie eingeordnet werden können *Abfragewerkzeuge*, bei denen zwischen DB und Benutzer eine Zwischenschicht verwendet wird, die es durch „Point and Click“–Bedienung ermöglicht, Anfragen zu formulieren und so dem Endbenutzer das Formulieren komplexer SQL–Anfragen abnimmt.

#### OLAP–Werkzeuge

*OLAP–Werkzeuge* (On–Line Analytical Processing) bieten die Möglichkeit der interaktiven Datenanalyse. Die Sicht auf die Daten erfolgt meistens multidimensional (siehe Abschnitt 3.1). Dem Anwender wird die Möglichkeit geboten, in Abhängigkeit von der aktuellen Fragestellung eine Zusammenstellung der Daten hinsichtlich Verdichtungsgrad und Präsentation vorzunehmen. Damit bieten sie gegenüber Berichtswerkzeugen die Vorteile einer interaktiven, auf die individuellen Bedürfnisse zugeschnittenen Datenanalyse.

#### Data Mining–Werkzeuge

Im Gegensatz zu OLAP ist das *Data Mining* ein induktiver Prozess. Es werden keine Vermutungen menschlicher Benutzer durch interaktive Analysen erhärtet, sondern z. B. anhand von Algorithmen des maschinellen Lernens und statistischer Verfahren versucht, bisher unbekannte Zusammenhänge und Trends im Datenbestand zu entdecken. Obwohl Data Mining auch ohne DWH möglich ist, bietet das Aufsetzen auf einem DWH erhebliche Vorteile. So können viele Zusammenhänge erst durch den Integrationsaspekt verschiedener Datenbestände oder den Versionierungsaspekt eines Datenbestandes entdeckt werden. Auf der anderen Seite sei aber auch auf die Nachteile des Data Mining in einer DWH–Umgebung hingewiesen, z. B. muss die für OLAP optimierte Struktur des Schemas nicht für Data Mining geeignet oder gar optimal sein. Darüber hinaus können durch die Bereinigung und Integration von Daten manche Analysen nicht mehr sinnvoll durchgeführt werden. Eine Gegenüberstellung der Vor– und Nachteile findet sich in [Gün00].

Tabelle 2.2 fasst die drei herkömmlichen Front End–Werkzeuge nochmals zusammen [Kem99].

| **Kriterium** | **Berichte** | **OLAP** | **Data Mining** |
|---|---|---|---|
| Systemausrichtung | Berichtsorientiert | Berichtsorientiert | Algorithmisch |
| Nutzungsinitiative | Systeminduziert | Benutzerinduziert | Benutzerinduziert |
| Nutzungsfrequenz | (A–)periodisch | A–periodisch | A–periodisch |
| Erforderliche DV–Kompetenz der Benutzer | Wenig | Mittel | (Sehr) groß |

Tabelle 2.2: Vergleich herkömmlicher Front End–Werkzeugklassen

**Sonstige Front End–Werkzeuge**

Neben den drei „klassischen“ auf einem DWH aufsetzenden Front End–Komponenten sind beliebige Applikationen denkbar, die sich mit Daten aus dem DWH versorgen. Exemplarisch ist in der Referenzarchitektur in Abbildung 2.1 auf Seite 9 ein mobiler Informationsagent genannt, der das DWH als Datenbasis nutzt und durch zusätzliches Wissen kritische Fälle erkennt, in denen er Benutzer benachrichtigt. Unter Berücksichtigung der Tatsache, dass (fast) alle entscheidungsrelevanten Daten einen Raumbezug haben, ist auch die in der Abbildung dargestellte Kombination mit einem GIS (Geographisches Informationssystem) eine interessante Möglichkeit, raumbezogene Auswertungen attraktiv aufbereitet zu präsentieren.

## 2.5 Sonstige Werkzeuge

Ergänzt werden die bisher vorgestellten Komponenten in einem DWS typischerweise um einen *DWS–Manager* (siehe Abschnitt 2.5.1) und ein *Metadaten–Repository* (siehe Abschnitt 2.5.2). Fasst man den Begriff der Werkzeuge in einem DWS etwas weiter, so gehören auch Entwurfswerkzeuge für das DWH dazu. Weil diese im Kontext der Arbeit eine wichtige Rolle spielen, werden sie separat in Abschnitt 5.2 behandelt.

### 2.5.1 DWS–Manager

Der *DWS–Manager* ist ein Administrationswerkzeug, das für die Steuerung und Überwachung der einzelnen, im DWS stattfindenden Prozesse zuständig ist. Dazu koordiniert er insbesondere das Zusammenspiel der beteiligten Komponenten, inklusive einer angemessenen Ausnahme– und Fehlerbehandlung. Da Daten üblicherweise im Batch–Betrieb integriert werden, wäre es unpraktikabel, den gesamten Integrationsprozess bei Vorliegen einer Ausnahme oder eines Fehlers anzuhalten und erst nach Behebung des Problems fortzusetzen – zumindest, solange das Problem lokal begrenzt ist, z. B. im Falle der Integritätsverletzung eines Datensatzes. Stattdessen wird eine Protokollierung des Problemfalls vorgenommen und der Integrationsprozess fortgesetzt. Datensätze, bei denen Probleme aufgetreten sind, können dann später manuell nachbearbeitet werden. Nur für den Fall, dass die Anzahl der Ausnahmen bzw. Fehler ein zuvor definiertes Limit übersteigt oder der Schweregrad des Problems eine Fortsetzung der Integration nicht zulässt (z. B. Hardware–Fehler oder fehlerhaftes Datenformat), wird der Prozess abgebrochen. Dabei muss der DWS–Manager ein geregeltes Anhalten und Wiederaufsetzen von Prozessen sicherstellen. Weitere Aufgaben des DWS–Managers sind die Unterstützung des DWS–Administrators bei der Prozessplanung und die Kommunikation mit dem DWS–Administrator während der Prozessausführung.

### 2.5.2 Metadaten–Repository

Eine Grundvoraussetzung für den effektiven Umgang mit Daten ist das Vorhandensein von Wissen über deren Syntax und Semantik. Sind diese *Metadaten* formal repräsentiert und zentral verfügbar, können Datenverarbeitungsprozesse einerseits automatisiert, andererseits sehr flexibel gestaltet werden. In der DWS–Architektur ist das sog. *Metadaten–Repository* für die zentrale Verwaltung und Bereitstellung von Metadaten zuständig. Üblicherweise werden *administrative, domänenspezifische* und *operative* Metadaten unterschieden [CD97].
Zu den administrativen Metadaten zählen u. a. :

- *Schemainformationen,* d. h. Metadaten im klassischen Sinne als „Daten über Daten".
- Informationen über *Quell– und Zielsysteme,* z. B. technische Charakteristika über den Zugriff wie Rechner– oder DB–Namen.
- *Datenabhängigkeiten* in Form von Transformationsregeln zwischen den Quellsystemen und dem DWH.
- *Datenabhängigkeiten* in Form von Anfragen o. ä. zwischen dem DWH und den Front End–Werkzeugen.

Operative Metadaten sind u. a. :

- *Systemstatistiken* für die Ressourcenplanung und Optimierung, d. h. Anfragemuster oder nutzer– bzw. gruppenspezifisches Nutzungsverhalten.
- Informationen über *Scheduling, Logging* und *Jobausführung* des DWH.
- Regeln und Funktionen für das *Nachladen* und die *Archivierung.*

Den domänenspezifischen Metadaten werden u. a. zugerechnet:

- *Informationsmodelle* und *konzeptionelle Datenschemata,* die der implementierungsunabhängigen Dokumentation dienen.
- Organisations– bzw. branchenspezifische *Vokabulare*, *Terminologien* und *Taxonomien.*
- *Abbildungen* zwischen diesen drei Begriffswerken und den korrespondierenden Elementen im DWH.
- Informationen über *Organisationsstrukturen* und *Geschäftsprozesse.*
- *Konzeptionelle Beschreibungen* von Berichten, Anfragen, Kennzahlen etc.
- Angaben über die *Datenqualität.*

Das Metadaten–Repository kommuniziert mit den anderen DWS–Komponenten, die entweder Metadaten anfordern (z. B. Schemabeschreibungen) oder aber ihrerseits erzeugte Metadaten im Repository ablegen (z. B. Zugriffsstatistiken)[3].

[3]Typischerweise existieren in der Praxis neben einem zentralen Repository bei den einzelnen Werkzeugen lokale Datenhaltungskomponenten, in denen Metainformationen abgelegt werden. Siehe hierzu Abschnitt 4.4.2

## 2.6 Zusammenfassung

In diesem Kapitel wurde eine grundlegende DWS–Architektur vorgestellt und die einzelnen Komponenten, ihre Aufgaben sowie ihr Zusammenspiel beschrieben.
In der Praxis gibt es mittlerweile eine Vielzahl von Herstellern, die einen Teil oder auch das gesamte Spektrum eines DWS abzudecken versuchen. Diese Anbieter kommen sowohl aus dem Datenbanksektor (z. B. *Oracle*) wie auch aus dem Bereich betrieblicher Standardsoftware (z. B. *SAP*). Daneben existieren auch diverse, meist kleinere Nischenanbieter. Auf eine Nennung konkreter Anbieter und Werkzeuge wird jedoch auf Grund der schnellen Alterung solcher Informationen an dieser Stelle verzichtet. Stattdessen sei auf die Marktübersichten und Studien [Ovu98, MBS00, MBS01][4] hingewiesen.

[4] Die Auswahl fiel auf diese Studien, weil sie regelmäßig aktualisiert werden, z. B. [Ovu98] vierteljährlich.

# Kapitel 3

# Multidimensionale Datenmodelle

Gegenstand dieses Kapitels sind multidimensionale Datenmodelle. Dabei werden zunächst in Abschnitt 3.1 die Grundbegriffe eingeführt. Anschließend werden in Abschnitt 3.2 aus der Literatur herausgearbeitete Anforderungen an konzeptionelle multidimensionale Datenmodelle gestellt, bevor in Abschnitt 3.3 konkrete Modelle präsentiert werden. Ein Vergleich dieser unterschiedlichen Modelle erfolgt in Abschnitt 3.4, bevor eine Zusammenfassung in Abschnitt 3.5 das Kapitel abschließt.

## 3.1 Grundbegriffe

Zur Einführung der Grundbegriffe multidimensionaler Datenmodelle soll das folgende Szenario dienen: In einem Unternehmen sollen die Verkaufszahlen von Produkten pro Tag und Filiale analysiert werden; relevante Zeiteinheiten neben dem Tag sind Woche, Monat, Quartal und Jahr; die Produkte sollen einerseits zu Produktgruppen, andererseits zu Marken und Herstellern zusammengefasst werden können; Filialen können immer einer Stadt zugeordnet werden, diese einer Region und diese wiederum einem Land[1].

### 3.1.1 Statische Aspekte

Hauptcharakteristikum multidimensionaler Datenmodelle ist die Klassifikation von Daten in *quantifizierende* und *qualifizierende* Daten.
*Fakten* sind dabei Datenobjekte, die sowohl *quantifzierende* wie auch *qualifizierende* Eigenschaften besitzen. Die quantifizierenden Eigenschaften beinhalten für die Organisation relevante Daten, die während einer Datenanalyse weitergehend untersucht werden können. Qualifizierende Eigenschaften dienen der näheren Beschreibung der quantifizierenden Eigenschaften, wodurch diese eine Bedeutung erhalten. Ein *Fakt* setzt sich aus einem oder mehreren *Faktattributen* (synonym *Kennzahlen* oder *Maßzahlen*) zusammen, die zumeist numerisch sind und den quantifizierenden Aspekt bestimmen. Den qualifizierenden Aspekt von Fakten beschreiben die *Dimensionen.* Dimensionen beantworten typischerweise Fragen wie „*Wo*, *Wann*, *Warum*, ... ist der Fakt aufgetreten?“. Im Beispiel ist die Verkaufszahl ein Fakt, die konkrete Anzahl verkaufter Produkte ist eine Kennzahl, und das Produkt (*Was* wurde verkauft?), die Filiale (*Wo* wurde es verkauft?) und der Tag (*Wann* wurde es verkauft?) stellen Dimensionen dar. Die Anzahl der Dimensionen eines Fakts wird als seine *Dimensionalität* bezeichnet. Die um ein Fakt angeordneten Dimensionen spannen einen (multidimensionalen) Datenraum auf, der als *Datenwürfel* (oder kurz *Würfel*) bezeichnet wird. Bis zu dreidimensionalen

[1] Dieses Beispiel ist in der Literatur mittlerweile zu einem „Standardbeispiel“ geworden. In Teil II dieser Arbeit findet es in einer erweiterten Form Anwendung als durchgängiges Beispiel.

Fällen wie im Beispiel lässt sich der Würfel graphisch gut veranschaulichen (siehe Abbildung 3.1).

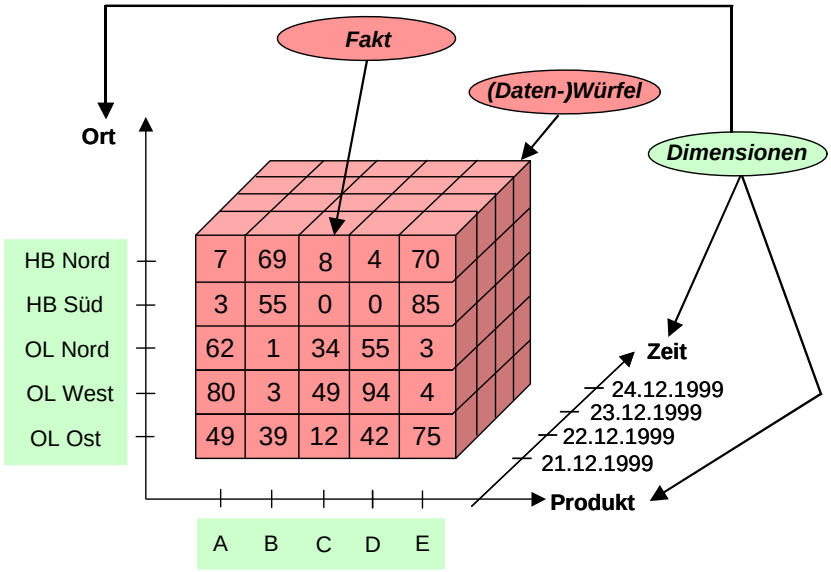
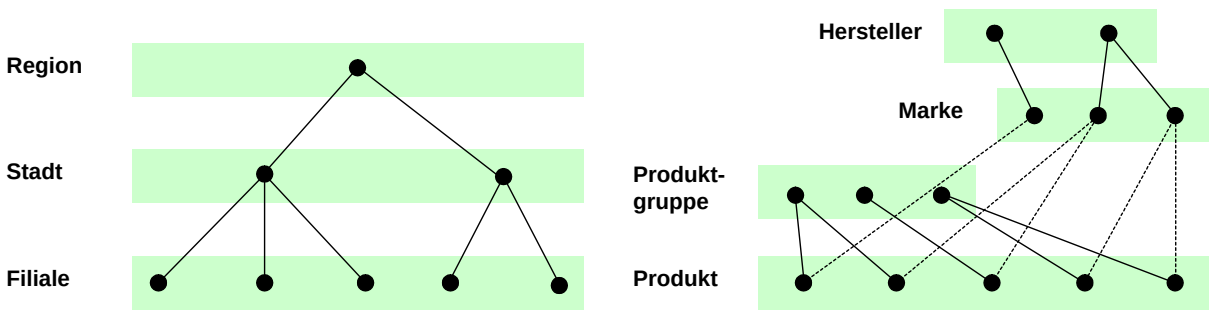



Abbildung 3.1: (Daten–)Würfel

Weil multidimensionale Datenschemata einen analyseorientierten Charakter besitzen, werden die Daten auf den *Dimensionen* auswertungsorientiert zusammengefasst. Eine solche Zusammenfassung wird als *Hierarchieebene* (synonym auch *Aggregations–* oder *Verdichtungsebene*) bezeichnet. Eine Menge aufeinander aufbauender Hierarchieebenen heißt *Dimensionshierarchie* oder kurz *Hierarchie* oder auch *Verdichtungspfad*. Das Zusammenfassen von Daten entlang einer Hierarchie wird als *Verdichtung* (synonym auch *Gruppierung* oder *Aggregation*) bezeichnet. Dieses Zusammenfassen der Daten erfolgt mittels einer Berechnungsvorschrift, die entsprechend als *Verdichtungs–*, *Gruppierungs–* oder *Aggregationsfunktion* bezeichnet wird.
Innerhalb einer Dimension kann es Fälle geben, in denen auf eine Hierarchieebene alternativ mehrere andere folgen können, indem aufgrund verschiedener Merkmale verdichtet wird. In diesem Falle spricht man von *multiplen Hierarchien* oder *Mehrfachhierarchien*. Werden verzweigende Pfade innerhalb der Hierarchie wieder zusammengeführt, so spricht man auch von *alternativen Verdichtungspfaden.* Abbildung 3.2 zeigt für das Beispiel aus Abbildung 3.1 eine einfache Hierarchie auf der Ortsdimension und eine Mehrfachhierarchie auf der Produktdimension.




(a) Einfache Hierarchie (b) Multiple Hierarchie

Abbildung 3.2: Einfache und Mehrfachhierarchien

Die Stufe des Verdichtungsgrades von Daten innerhalb einer Hierarchie bezeichnet man als *Granularität*. Hierbei besitzen die *Detaildaten* den niedrigsten Verdichtungsgrad bzw. die feinste Granularität, zusammengefasste Daten haben entsprechend einen höheren Verdichtungsgrad und damit eine gröbere Granularität.
Es ist möglich, dass bei der Zuordnung von Elementen einer Hierarchieebene zur nächsthöheren (oder nächstniedrigeren) Ebene nicht immer zugehörige Elemente existieren. In diesem Fall ergibt sich auf Instanzebene ein unbalancierter Baum, wie er Abbildung 3.3 zu sehen ist. Man spricht in diesem Falle von einer *unbalancierten Hierarchie*.

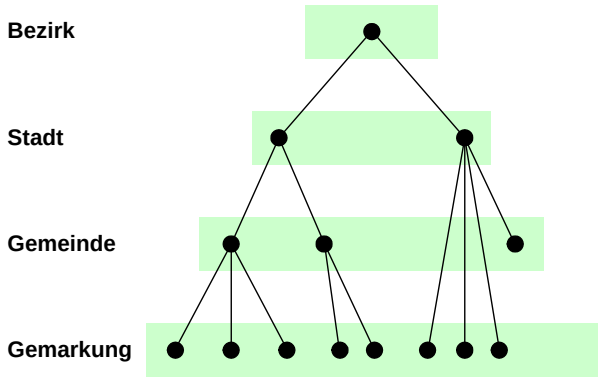
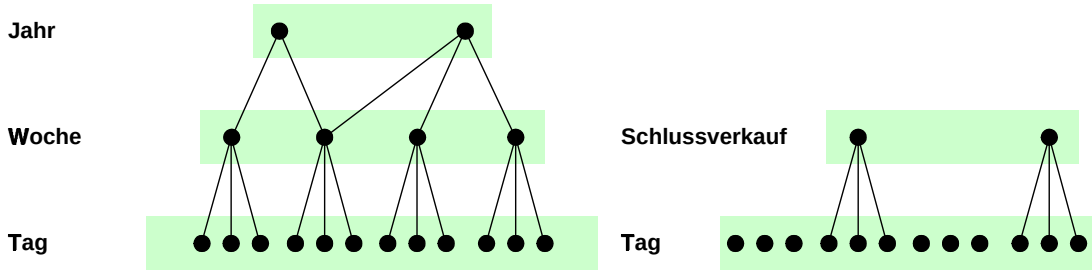



Abbildung 3.3: Unbalancierte Hierarchie

In [Glu96, Wie00] wird die Problematik der *anteiligen Verrechnung* vorgestellt, die durch Zuordnung eines Elementes einer Hierarchieebene zu mehreren Elementen der nächst höheren Ebene mittels einer Berechnungsvorschrift entsteht (siehe Abbildung 3.4(a)). Bei einer Verdichtung der Daten muss berücksichtigt werden, dass ein Wert nicht mehrfach zu 100% in die Berechnung einbezogen werden darf. Die herkömmliche Definition einer Dimensionshierarchie mit — auf Instanzebene — eindeutig identifizierbaren Elternknoten ist somit nicht verwendbar. Ebenso können Fälle auftreten, in denen nicht alle Instanzen einer Hierarchieebene an der Verdichtung teilnehmen (siehe Abbildung 3.4(b)). In diesem Falle spricht man von einer *nicht–vollständigen Verdichtung*. Beim Navigieren entlang einer solchen Verdichtung wird anschaulich die Datenbasis um die nicht an der Verdichtung teilnehmenden Instanzen „ausgedünnt“. Hierbei ist insbesondere zu beachten, dass ein Hierarchiepfad, der eine nicht–vollständige Verdichtung beinhaltet, i. Allg. nicht wieder mit anderen Hierarchieebenen zusammengeführt werden darf, weil dann aufgrund der zuvor verlorenen Daten falsche Werte zustande kommen.




(a) Anteilige Verrechnung (b) Nicht-vollständige Verdichtung

Abbildung 3.4: Anteilige Verrechnung und nicht–vollständige Verdichtung

Beim Navigieren entlang der Verdichtungspfade werden die Daten entsprechend einer Verdichtungsoperation verrechnet. Hierbei ist nicht für jede Kennzahl jede Operation anwendbar. So erhält man beispielsweise in einer metereologischen Datenbank bei Addition der Kennzahl „Temperatur“ bez. der Dimension „Ort“ falsche Werte. Einen Überblick über diese Problematik findet sich in [LS97, Wie00]. Die Eigenschaft einer Kennzahl, bez. einer Dimension bestimmte Verdichtungsoperatoren zu besitzen, wird als *Aggregierbarkeit* oder *Additivität* bezeichnet[2].

### 3.1.2 Dynamische Aspekte

Unter den dynamischen Aspekten multidimensionaler Datenmodelle werden Operationen auf den statischen Strukturen verstanden. Häufigste Operation ist das in Abbildung 3.5 dargestellte Wechseln zwischen den Hierarchieebenen, das als *Drilling* bezeichnet wird. Das Wechseln auf eine gröbere Hierarchieebene heißt *Roll–Up*, die inverse Operation — die Verfeinerung der Hierarchieebene — wird als *Drill–Down* bezeichnet.

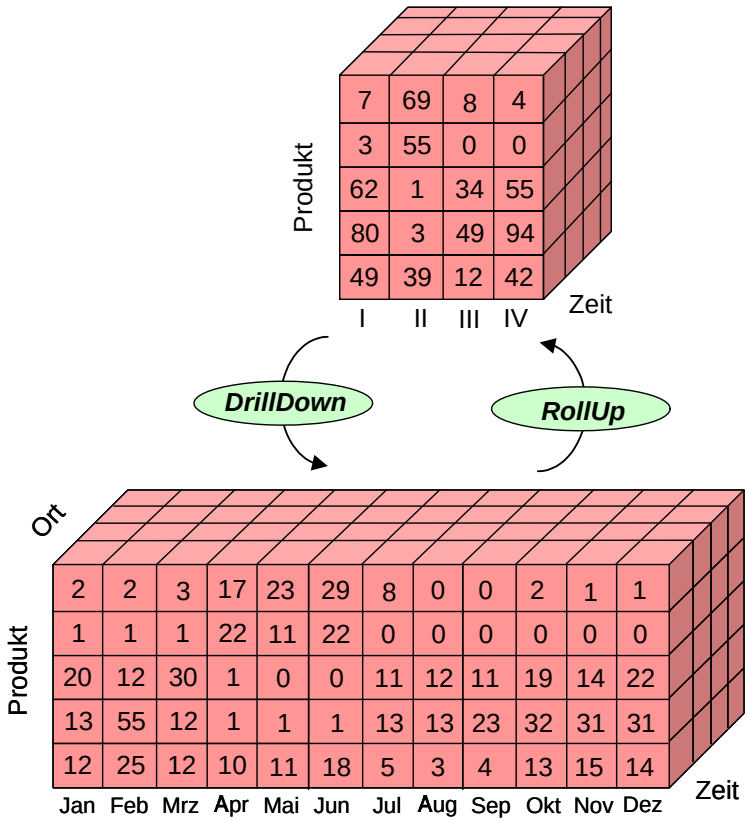



Abbildung 3.5: *Roll–Up*– und *Drill–Down*–Operator

Abbildung 3.6 zeigt das *Pivotisieren* bzw. *Rotieren*, bei dem der Datenwürfel in eine für die aktuelle Datenanalyse günstige Position bewegt wird.

[2] In Arbeiten, die dem Kontext statistischer Datenbank entstammen, wie z. B. [LS97, Wie00], wird von *Summierbarkeit* gesprochen, weil in statistischen Datenbanken Summenbildung häufig die einzige Operation ist.

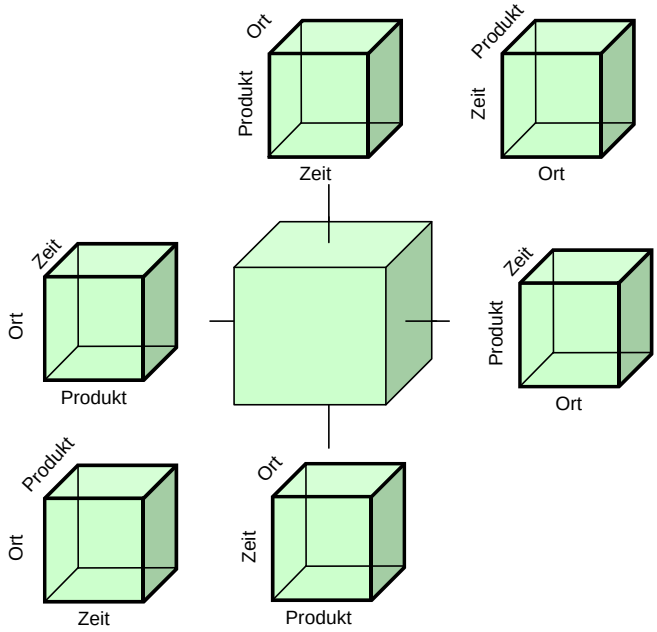



Abbildung 3.6: *Pivoting*– bzw. *Rotation*–Operator

Schließlich ist auch das benutzergesteuerte Erforschen des Datenwürfels eine wichtige multidimensionale Operation. Durch Restriktion der Dimensionselemente werden hierbei die sichtbaren Daten auf einen Teilwürfel oder eine Scheibe eingeschränkt. Diese als *Slice and Dice* bezeichnete Operation ist in Abbildung 3.7 veranschaulicht.

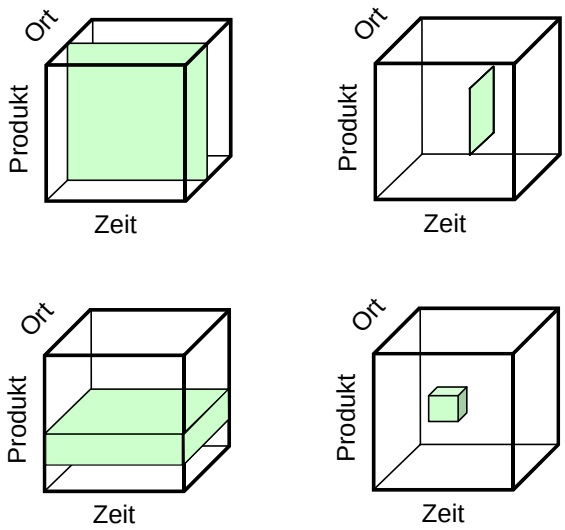



Abbildung 3.7: *Slice and Dice*–Operator

## 3.2 Datenmodellanforderungen

In diesem Abschnitt werden die speziellen Anforderungen an die konzeptionelle Modellierungsphase von DWHs genannt. Als Grundlage dienen dabei sowohl theoretische Überlegungen [Inm96, AM97b, Tho97, SBHD98a] als auch praktische Experimente [BCJJ98, PJ99, Wes00]. Generell muss es für die adäquate Darstellung der multidimensionalen Sachverhalte möglich sein, sowohl *komplex strukturierte* Dimensionen als auch *komplex strukturierte* Fakten modellieren zu können. Zur Definition „komplex strukturierter Fakten“ gehören dabei folgende Anforderungen:

- Ein (Daten–)Würfel bzw. ein Schema sollte *beliebig viele Fakten* enthalten dürfen, um den gesamten Umfang des DWH modellieren zu können.
- Ein (Daten–)Würfel bzw. ein Fakt sollte *beliebig viele Kennzahlen* enthalten dürfen, denn dies ist die natürlichste Möglichkeit, in der Realwelt gleichzeitig auftretende Kennzahlen darzustellen.
- *Beziehungen zwischen Fakten,* wie z. B. Spezialisierungen oder Aggregationen, sollten explizit dargestellt werden können, weil diese Abhängigkeiten das Verständnis der Daten während der konzeptionellen Modellierung erheblich vertiefen.
- Ein Fakt sollte eine *innere Struktur* besitzen, denn Fakten bestehen nur in Ausnahmefällen aus einer einzigen numerischen Kennzahl. Ein Fakt kann einerseits sowohl mehrere Attribute besitzen, die beispielsweise in Form einer Verbundstruktur angeordnet sind, und zum anderen können auch textuelle „Kennzahlen“ oder Eigenschaften existieren, deren Domäne ein Aufzählungstyp ist.
- Die Markierung *abgeleiteter Attribute* und die Angabe der Berechnungsvorschrift sollten im Datenschema möglich sein. Gerade in OLAP–Szenarien existieren häufig eine Reihe abgeleiteter Attribute [SBHD98a], die in dieser Form nicht im Datenbestand enthalten sind, sondern sich durch Anwendung einer Funktion aus anderen Kennzahlen berechnen lassen.
- Die *Additivität von Kennzahlen* sollte im Schema explizit angegeben werden können, denn die falsche Anwendung von Verdichtungsoperatoren kann zu falschen Auswertungen mit möglicherweise sogar schwerwiegenden Folgen führen.
- Auch Kennzahlen sollten eine *innere Struktur* besitzen können, um so Zusammengehörigkeiten bzw. Abhängigkeiten dieser Werte natürlich abbilden zu können.

Die Anforderung der Modellierung „komplex strukturierter Dimensionen“ lässt sich wie folgt präzisieren:

- Die Modellierung von *Hierarchieebenen* sollte möglich sein, um die Daten auf einem für die spätere Analyse adäquaten Verdichtungsgrad darzustellen.
- Zwischen Hierarchieebenen sollte die Darstellung von *Verdichtungspfaden* möglich sein, um Verdichtungspfade für potenzielle Auswertungen definieren zu können.
- Die Struktur sollte ein *gerichteter azyklischer Graph* (DAG[3]) sein, denn eine reine Baumstruktur schränkt die Modellierungsmöglichkeiten erheblich ein.
- Darüber hinaus können zwischen Hierarchieebenen weitere Beziehungen wie *Generalisierungen* und *Assoziationen* bestehen, die ebenfalls explizit formuliert werden sollten.

[3] Directed acyclic graph.

- In der Hierarchiestruktur sollten *Mehrfachhierarchien* möglich sein.
- In der Hierarchiestruktur sollten *alternative Verdichtungspfade* möglich sein.
- In der Hierarchiestruktur sollten *unbalancierte Hierarchien* möglich sein.
- Bei der Verdichtung sollte die Darstellung *anteiliger Verrechnungen* möglich sein.
- Die Modellierungssprache sollte ein Konzept zur Handhabung *nicht–vollständiger Verdichtungen* anbieten.
- Für jede Hierarchieebene sollten darüber hinaus *zusätzliche Attribute* definierbar sein, die nicht unmittelbar der Hierarchiebildung dienen, weil die Dimensionsstrukturen allein keine näheren Informationen über die qualifizierenden Daten enthalten. Diese Dimensionsattribute können später während der Analysephase von den Anwendern für eine dynamisch entstehende Strukturierung der Dimensionen verwendet werden.
- Für jede Hierarchieebene sollten *Schlüsselattribute* definiert werden können.
- Innerhalb einer Hierarchieebene können Datenobjekte mit unterschiedlichen Dimensionsattributen existieren, was insbesondere bei Hierarchiebildungen Probleme nach sich ziehen kann [LRT96], so dass die Modellierung *unterschiedlicher Datenobjekttypen* unterstützten sollte.
- Insbesondere sollten auch *optionale Dimensionsattribute* definiert werden können.

Das klassische ERM (*Entity Relationship–Modell*) [Che76] ist für die konzeptionelle Modellierung von OLTP–Datenbanken weit verbreitet. Seine Vorteile liegen in der geringen Anzahl von Konstrukten und der damit verbundenen einfachen Anwendbarkeit. Die Hauptkonstrukte sind *Entitätstypen*, die Objekte der Diskurswelt beschreiben, *Beziehungstypen*, die Abhängigkeiten zwischen diesen Objekten festhalten und *Attribute*, die die Eigenschaften der Entitäten und Beziehungen darstellen. Es ist jedoch offensichtlich, dass das klassische ERM den oben genannten Anforderungen nicht gewachsen ist, insbesondere aufgrund der Existenz nur eines einzigen Beziehungstyps. Zwar sind durch Ergänzungen wie z. B. Spezialisierungen oder Kompositionen diverse *erweiterte E/R–Modelle* entstanden (für einen Überblick siehe [Teo90, Tha00]), welche auch die Anforderungen komplexer Objekte erfüllen, aber die Darstellung von Konzepten wie Dimensionen und Hierarchien nur unzureichend zulassen.
Aus den Anforderungen in der Literatur lassen sich noch zwei Punkte nennen, die im Rahmen dieser Arbeit nicht als Anforderung für konzeptionelle Datenmodelle gelten sollen:

- *Viele–zu–Viele*–Beziehungen zwischen Fakten und einer Dimension [PJ99, TKS01]:
  Dieses ist ein nur scheinbar natürliches Konstrukt, in Wirklichkeit verbirgt sich hinter der Beziehung ein weiteres Fakt, welches auch explizit modelliert werden sollte.
- Unsicherheit [PJ99]:
  Aussagen über Unsicherheit oder allgemein Güte der Datenqualität sind wichtige Aspekte, allerdings sollten sie als Metadaten im Repository abgelegt werden und nicht Bestandteil der konzeptionellen Modellierung sein.

## 3.3 Konzeptionelle multidimensionale Datenmodelle

### 3.3.1 Multidimensional E/R–Modell

In [SBHD98b] wird eine auf dem E/R–Modell basierende Erweiterung zur Repräsentation multidimensionaler Aspekte vorgestellt. Dieses als *Multidimensional E/R Model* (MERM) bezeichnete Modell nimmt dabei keine tiefgreifenden Veränderungen am E/R–Modell vor, sondern ergänzt es um drei neue Notationselemente, die jeweils Spezialisierungen existierender Entitäts– bzw. Beziehungstypen sind.
Grundgedanke dieser Vorgehensweise ist die Übernahme eines etablierten Modells inklusive dessen formaler Fundierung. Durch diesen Ansatz wird u. a. die einfachere Erweiterung bestehender Forschungsergebnisse auf den multidimensionalen Fall ermöglicht, da nur die benötigten Veränderungen quasi „inkrementell“ diskutiert werden müssen. Eine besondere Bedeutung kommt hierbei dem Aspekt der automatischen Schemaerzeugung zu. Die neuen MERM–Sprachelemente bieten neben der Unterscheidung qualifizierender und quantifizierender Daten die Modellierung von Dimensionshierarchien. Vorrangig wurde hierbei auf eine minimale Erweiterung für die Darstellbarkeit multidimensionaler Daten geachtet, so dass die Ergänzungen einfach zu erlernen sind und die Konstrukte auch in mächtigere E/R–Modelle integriert werden können. Die drei multidimensionalen Spezialisierungen der ERM–Modellierungskonstrukte sind in Abbildung 3.8 mit ihren graphischen Symbolen zu sehen. Für die Strukturierung von Dimensionen wird ein direkter Entitätsuntertyp im Sinne einer *Dimensionsebene* eingeführt, der im Gegensatz zum ursprünglichen Entitätstyp die beiden neuen *Fakt–* und *Rolls-Up*–Beziehungen verwenden darf. Da jede Hierarchieebene durch eine separate Entitätsmenge repräsentiert wird, lassen sich auf einfache Weise ebenenabhängige Dimensionsattribute in ein Datenschema aufnehmen. Unklarheiten während der Modellierung kann die graphische Darstellung von Dimensionsebenen verursachen, da das Symbol einer normalen Entität verwendet wird. Die Einsetzbarkeit der verschiedenen Beziehungstypen ist dadurch nicht sofort ersichtlich.

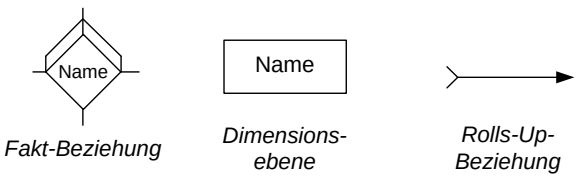



Abbildung 3.8: Graphische MERM–Notationselemente

Die hierarchische Struktur einer Dimension wird mittels einer gerichteten, binären *Rolls-Up–Beziehung* zwischen zwei Dimensionsebenen modelliert. Dieser Ansatz ermöglicht die Modellierung von alternativen Pfaden, Mehrfachhierarchien sowie die gemeinsame Nutzung derselben Dimensionsebenen in verschiedenen Hierarchien. Die individuelle Gestaltung von Dimensionshierarchien ist nur durch eine geforderte Zyklenfreiheit eingeschränkt, durch die unendliche Roll-Up–Pfade vermieden werden. Die *Fakt–Beziehung* stellt den zentralen Punkt eines Datenschemas dar und dient der Aufnahme mehrerer Kennzahlen in Form von Attributen. Durch die Anbindung verschiedener Dimensionsebenen wird der gewünschte mehrdimensionale Zusammenhang hergestellt. Ein MERM–Schema ist nicht auf eine Fakt–Beziehung beschränkt, so dass auch komplexere Abhängigkeiten zwischen Fakt–Beziehungen modelliert werden können. Abbildung 3.9 zeigt das Beispiel mit den Verkaufszahlen in MERM–Notation.

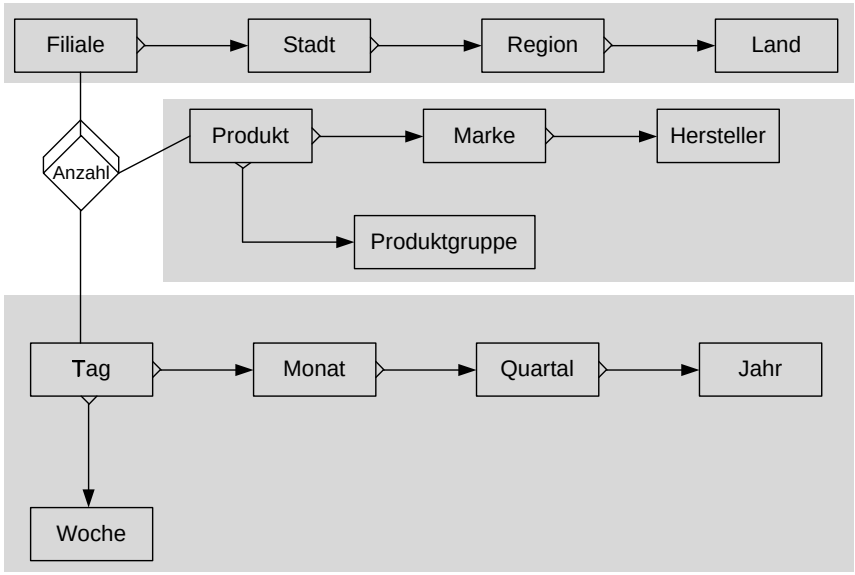



Abbildung 3.9: MERM–Beispielschema

Bereits in der hier vorgestellten Form ermöglicht das MERM eine weitreichende Modellierung des betrachteten Weltausschnitts, wobei Art und Aufbau der einzelnen Kennzahlen bzw. Dimensionsattribute durch die Ausdrucksmächtigkeit des zugrunde liegenden E/R–Modells bestimmt sind. Steigen die Anforderungen an die Struktur der Daten, so kann das MERM um entsprechende E/R–Konstrukte erweitert werden, z. B. würde die Generalisierungsbeziehung die Unterscheidung verschiedener Objekttypen erlauben. Als Nachteil kann sich herausstellen, dass das MERM keine graphische Repräsentation abgeleiteter Attribute ermöglicht. Analog zum ERM stellt ein MERM–Schema nur statische Aspekte der Daten dar, eine Unterscheidung der beiden Attributtypen ist somit nicht möglich. Informationen über abgeleitete Attribute lassen sich jedoch zusammen mit den benötigten Berechnungsvorschriften separat notieren oder mit einem orthogonalen, funktionalen Modell erfassen.

### 3.3.2 starER–Modell

Wie beim multidimensionalen E/R–Modell wird im starER–Modell [TBC99] das konventionelle ERM erweitert, indem spezielle Beziehungstypen eingeführt werden, um die Modellierung von Hierarchien zu ermöglichen. Die Konstrukte des starER–Modells sind in Abbildung 3.10 dargestellt:

- Eine *Faktenmenge* repräsentiert eine Menge von Fakten der Realwelt mit gleichen Eigenschaften im Sinne des multidimensionalen Datenmodells, ihre graphische Darstellung ist ein Kreis.

- Eine *Entitätsmenge* ist ein Entitätstyp im Sinne der klassischen E/R–Modellierung, d. h. sie repräsentiert eine Menge von Realwelt–Objekten mit gleichen Eigenschaften. Die graphische Darstellung ist ein Rechteck.

- Eine *Beziehungsmenge* repräsentiert eine Menge von Assoziationen zwischen zwei *Entitätsmengen* oder einer *Entitätsmenge* und einer *Faktenmenge*. Zulässige Kardinalitäten sind *Viele–zu–Viele* (N:M), *Viele–zu–1* (M:1) und *1–zu–Viele* (1:M), graphische Darstellung ist eine Raute. Um die Aggregierbarkeit anzuzeigen, können Attribute mit *S* (*stock* = Bestandsgröße), *F* (*flow* = Bewegungsgröße) oder *V* (*value per unit* = Wert pro Stück) gekennzeichnet werden.

- Zwischen *Faktenmengen* gibt es *Beziehungsmengen* der speziellen Typen *Spezialisierung* bzw. *Generalisierung*, *Aggregation* und *Membership*, die Notationen sind in Abbildung 3.10 zu sehen.
- *Attribute* beschreiben die statischen Eigenschaften von *Fakten–*, *Entitäts–* und *Beziehungs–* Instanzen und werden als Ellipse dargestellt, die mit ihrem Bezugsobjekt verbunden ist.

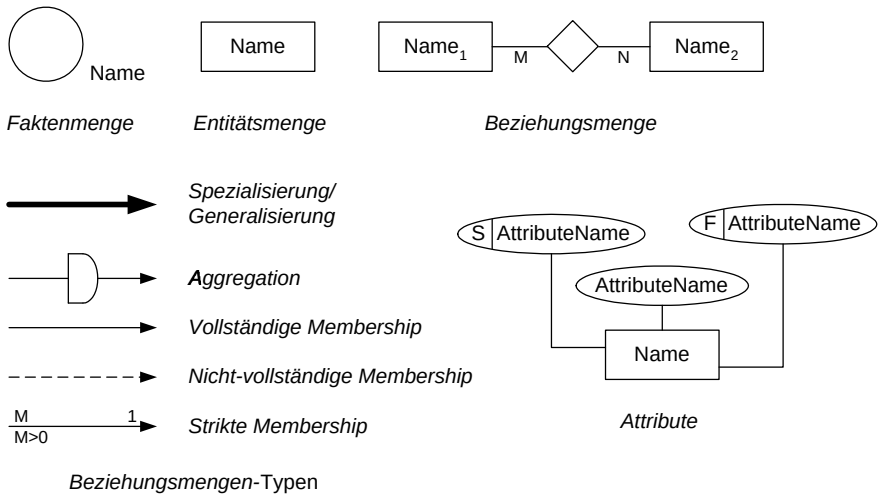



Abbildung 3.10: Konstrukte des starER–Modells

Das Beispiel „Verkäufe“ ist in Abbildung 3.11 dargestellt.

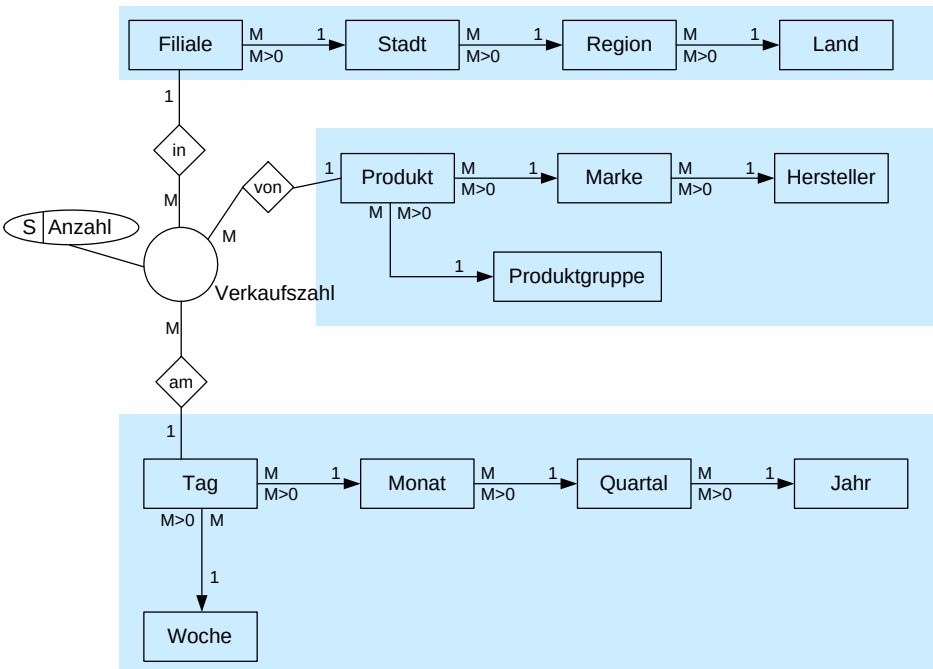



Abbildung 3.11: starER–Beispielschema

Zusammenfassend lässt sich zum starER–Modell sagen, dass mit dem Erweitern des E/R–Modells auf einer bewährten, formal fundierten Basis aufgesetzt wird. Gegenüber dem MERM bietet es mit Generalisierung und Aggregation erweiterte Beziehungskontrukte an. Ebenso positiv ist die Klassifikation der Kennzahlen und ihre darauf aufbauende Additivität zu erwähnen. Negativ sind die unterschiedlichen Beziehungstypen zwischen Entitätsmengen zu bewerten. Das in Abbildung 3.12 dargestellte Schemafragment [TBC99] ist in dieser Form nicht sinnvoll. So bleibt die Frage, warum die „Halbjahre“ als Aggregation und nicht als normale Verdichtungspfade modelliert worden sind. Ebenso sind die *Viele–zu–Viele*–Beziehungen zwischen einer Faktmenge und einer Entitätsmenge fragwürdig, denn hinter dieser Beziehung verbirgt sich implizit ein weiteres Fakt und sollte daher auch als solches modelliert werden.

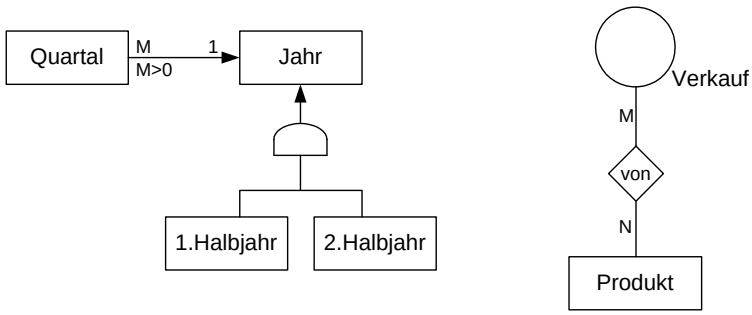



Abbildung 3.12: starER–Beispielschema: Kritische Punkte

### 3.3.3 ADAPT (Application Design for Analytical Processing Technologies)

MitADAPT einer Vielzahl speziell auf die Modellierung von Datenstrukturen für analytische Anwendungen ausgerichteter Notationselemente ist die in [Bul96] vorgeschlagene Notation ADAPT (*Application Design for Analytical Processing Technologies*) ausgestattet. Ziel dieser Modellierungstechnik ist die Darstellung multidimensionaler Datenstrukturen in Verbindung mit den darauf anwendbaren Berechnungsvorschriften, die z. B. durch einen Datenbank–Server bereitgestellt werden. Neben Daten und Funktionen besitzt ADAPT einige Konstrukte für die Verknüpfung mit Datenquellen und Front End–Werkzeugen. Auf diese Weise kann in einem Schema bereits festgelegt werden, ob der Datenzugriff interaktiv oder beispielsweise mit einem Tabellenkalkulationsprogramm erfolgen soll. Die Einbeziehung der Datenquellen soll eine Vorab–Optimierung des Datenschemas ermöglichen, indem die Datengranularität vorgegeben wird. Ein spezieller *SQL Drill–Thru*–Operator zeigt dabei an, dass feinere Daten nicht gespeichert werden, sondern bei Bedarf aus der Quelldatenbank abgefragt werden müssen. Beide Aspekte nehmen allerdings logische bzw. physische Entwurfsentscheidungen vorweg.
Grundlegende Elemente der ADAPT–Notation sind *Variablen* bzw. *Würfel*, *Formeln* und *Dimensionen* (siehe Abbildung 3.13) [JT98]. Abgesehen von dem zusätzlich eingeführten *Datenquellen*–Symbol, mit dem sich die Herkunft der Daten beschreiben lässt, entsprechen die drei Konstrukte den entsprechenden Begriffen der multidimensionalen Datensicht. Der *Datenwürfel* ist das Kernstück eines ADAPT–Schemas und stellt normalerweise eine einzige betriebswirtschaftliche Variable[4] dar. Im unteren Teil des Datenwürfel–Symbols werden alle relevanten Dimensionen eingetragen[5].

[4] Der Begriff Variable ist in diesem Kontext gleichbedeutend mit dem Begriff Kennzahl.

[5] Eine Aufnahme der Dimensionsnamen in den Datenwürfel erscheint auf den ersten Blick überflüssig, da später die benötigten Dimensionen graphisch mit dem Datenwürfel verbunden werden. In [Bul96] wurde aber ursprünglich eine Trennung der Ansichten von Würfeln und Dimensionen vorgeschlagen, die jedoch in der Literatur zunehmend durch eine kombinierte Darstellung ersetzt wird [JT98].

Existieren mehrere Variablen, die dieselbe Menge von Dimensionen besitzen, so können sie durch einen gemeinsamen Datenwürfel dargestellt werden. Die einzelnen Variablen werden dabei in eine spezielle *Kennzahlendimension* aufgenommen. Abgeleitete Attribute werden als normale Variablen modelliert und zusätzlich mit einem Formel–Symbol versehen, das den Funktionsnamen und die konkrete Berechnungsvorschrift enthält.

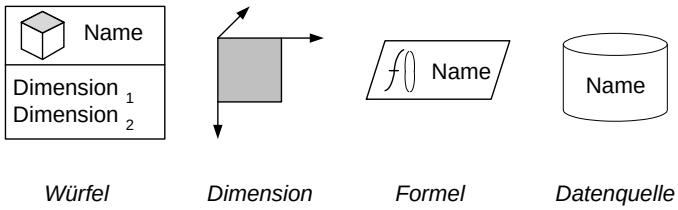



Abbildung 3.13: ADAPT–Kernelemente

Für eine zweckmäßige Modellierung qualifizierender Daten besitzt ADAPT jeweils sechs verschiedene Dimensions– und Dimensionselementtypen. Die zugehörigen Symbole sind in den Abbildungen 3.14 und 3.15 dargestellt. Eine *aggregierende Dimension* besitzt mindestens eine Hierarchie, die den gewünschten Verdichtungspfad repräsentiert, indem sie sich in mehrere Hierarchieebenen gliedert. Die einzelnen Ebenen werden dabei direkt dem Hierarchiesymbol untergeordnet, so dass in einer Dimension mehrere alternative Hierarchien existieren können und Dimensionselemente nicht zwingend darin enthalten sein müssen. *Partitionierende Dimensionen*, die auch als *Versions-* oder *Szenariodimensionen* bezeichnet werden, stellen verschiedene Varianten der Variablen dar, wie z. B. Plan– oder Ist–Werte. Besitzen Elemente eine natürliche Ordnungsbeziehung, so kann dies durch eine *sequentielle Dimension* ausgedrückt werden. Ein Beispiel für diesen Dimensionstyp ist die Zeitdimension. Jedes Dimensionselement einer aggregierenden oder sequentiellen Dimension kann weitere, im Analyseprozess nutzbare Attribute besitzen. Diese Zusatzattribute werden in einer eigenen *Eigenschaftsdimension* abgelegt. Neben der bereits erwähnten *Kennzahlendimension* existiert der *Tupeldimensionstyp*, durch den aus der Kombination von Elementen zweier Dimensionen eine neue Dimension gebildet werden kann.

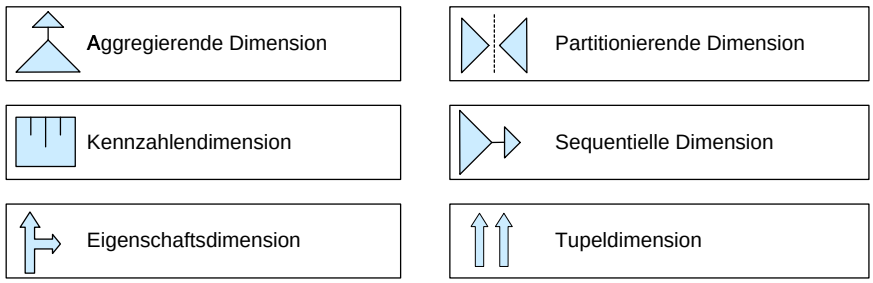



Abbildung 3.14: ADAPT–Dimensionstypen

Hinsichtlich der Verdichtung von Daten ist die *Hierarchie* der wichtigste Dimensionselementtyp. Jede Stufe innerhalb der Hierarchie wird durch eine eigene *Hierarchieebene* dargestellt. Die oberste Ebene, die direkt mit dem Hierarchie–Symbol verbunden ist, stellt dabei die höchste Aggregationsstufe dar. Die unterste (über einem SQL Drill–Thru–Operator angeordnete) Ebene entspricht der feinsten Granularität, mit der die Daten in der dargestellten Datenbank abgelegt sind. Gegenüber Hierarchieebenen besitzen Kennzahlendimensionen und partitionierende Dimensionen *Dimensionswerte*. Jedes Dimensionselement kann mit *Dimensionsattributen* um weitere Beschreibungen ergänzt

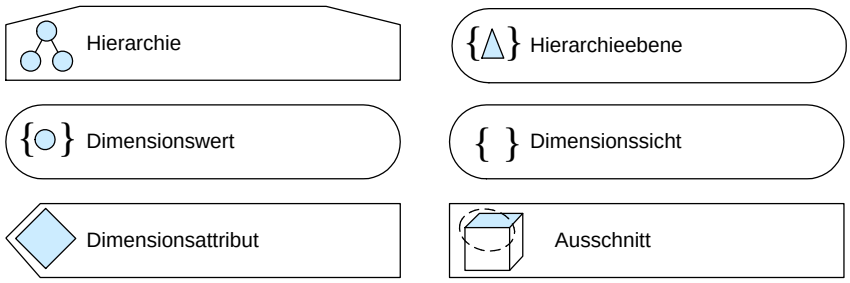



Abbildung 3.15: ADAPT–Dimensionselementtypen

werden. Andere Sichtweisen auf Dimensionswerte bzw. Hierarchiestufen oder die Betrachtung einer Teilmenge des Wertebereiches einer Dimension ermöglichen *Dimensionssichten* und *Ausschnitte.*
Weitere Notationselemente sind Beziehungstypen, die Abhängigkeiten zwischen Dimensionen beschreiben. Darin enthalten ist einerseits der aus dem E/R–Modell bekannte Beziehungstyp (hier *Dimensionsbeziehung* genannt) und zusätzlich drei Symbole für die Modellierung von *Unter–* und *Teilmengenbeziehungen*. Der *Filter* als weiteres Symbol dient der Definition von Auswahlkriterien für Dimensionssichten.
Wie bereits im vorangegangenen Abschnitt soll auch hier das Beispielschema mittels der betrachteten Notation vorgestellt werden: Abbildung 3.16 zeigt den Datenwürfel für die Verkaufszahlen.

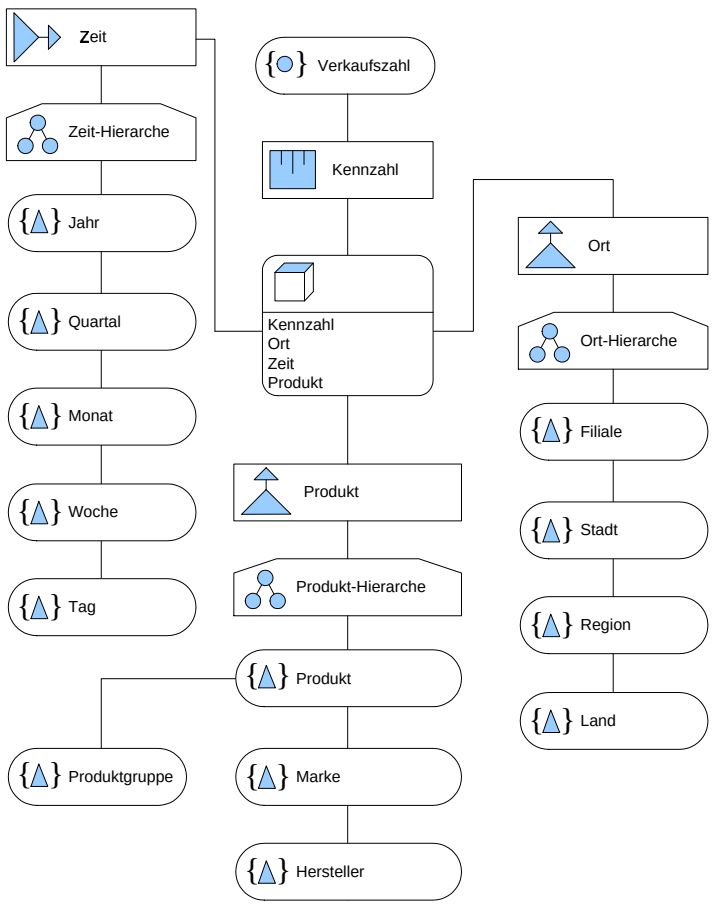



Abbildung 3.16: ADAPT–Beispieldiagramm

Durch die mögliche Einbeziehung von Implementierungsdetails, wie beispielsweise die Optimierung des Schemas durch den SQL Drill–Thru–Operator, in die Datenmodellierung erfolgt eine für die konzeptionelle Entwurfsebene unerwünschte Vermischung mit logischen bzw. physischen Gesichtspunkten. Problematisch für den praktischen Einsatz ist zudem die verwirrend große Anzahl der Beschreibungsobjekte, die einerseits unzureichend spezifiziert [GG98] und andererseits nicht eindeutig in ihrer Verwendung, d. h. semantisch nicht „sauber“ fundiert ist. Ein ADAPT–Schema unterliegt somit immer einer Interpretation. In [JT98] wird darauf hingewiesen, dass gerade die Einordnung einer Dimension in einen der sechs Dimensionstypen nicht immer klar ist: Besitzen aggregierende Dimensionen neben der eigentlichen Hierarchie weitere Elemente, so kann sie auch einer Partitionierung entsprechen. Ebenso impliziert eine sequentielle Zeitdimension mit verschiedenen Granularitätsstufen eine Aggregationshierarchie und könnte daher auch als aggregierende Dimension verstanden werden.

### 3.3.4 DFM (Dimensional Fact Model)

Als weitere graphische Notation zur Modellierung von Datenstrukturen für das DWH wird in [GMR98a, GMR98b] das *Dimensional Fact Model* (DFM) vorgestellt. Ein konzeptionelles Datenschema unterteilt sich bei diesem Ansatz in mehrere themenorientierte *Fakt–Schemata*, deren Basiselemente *Fakten*, *Attribute*, *Dimensionen* und *Hierarchien* sind. Im Mittelpunkt eines Fakt–Schemas steht ein bestimmter zur Datenanalyse relevanter Bereich, der im DFM als *Fakt* bezeichnet wird und zur Aufnahme von Daten eine beliebige Anzahl verschiedener *Fakt–Attribute* besitzen kann. Das Modell beschränkt sich bei der Art von Fakt–Attributen auf numerische Typen bzw. Typen mit kontinuierlichem Wertebereich, so dass Fakt–Attribute selbst keine komplexere innere Struktur aufweisen können. Die graphische Darstellung von Fakten erfolgt durch ein zweigeteiltes Rechteck, welches den jeweiligen Fakt–Namen und eine Auflistung der Namen der einzelnen Fakt–Attribute enthält.
Von diesem zentralen Fakt zweigen die eine Baumstruktur bildenden Dimensionen ab. Die Knoten des Baumes sind die Hierarchieebenen. Sie werden durch einen Kreis dargestellt und beschreiben ein Attribut, mit dem ein Element auf der entsprechenden Hierarchieebene charakterisiert werden kann. Außer der Angabe des Attributnamens erfolgt keine nähere Aufgliederung der einzelnen identifizierenden Eigenschaften, so dass es nicht möglich ist, durch Kombinationen von Attributen einen konzeptionellen Schlüssel festzulegen. Besteht eine modellierte Beziehung zwischen zwei Attributen nicht bei jeder Kombination von Attributausprägungen, so kann dies für spätere Entwurfsphasen im Fakt–Schema vermerkt werden. Diese *optionale Beziehung* wird durch einen Querstrich auf der zugehörigen Kante dargestellt. Neben kreisförmigen Knoten existiert ein zweiter Typ zur Darstellung *nicht-dimensionaler* Attribute[6]. Diese als Strich darzustellenden Attribute können nur als Blätter im Dimensionsbaum auftreten und beschreiben Zusatzinformationen, die nicht für eine Hierarchiebildung geeignet sind. Abbildung 3.17 fasst die Notationselemente zusammen.


Abbildung 3.17: DFM–Notationselemente

[6] In anderen konzeptionellen Modellen werden diese *nicht–dimensionalen Attribute* als *Dimensionsattribute* bezeichnet.

Das Beispiel der Verkaufszahlen ist in Abbildung 3.18 dargestellt.

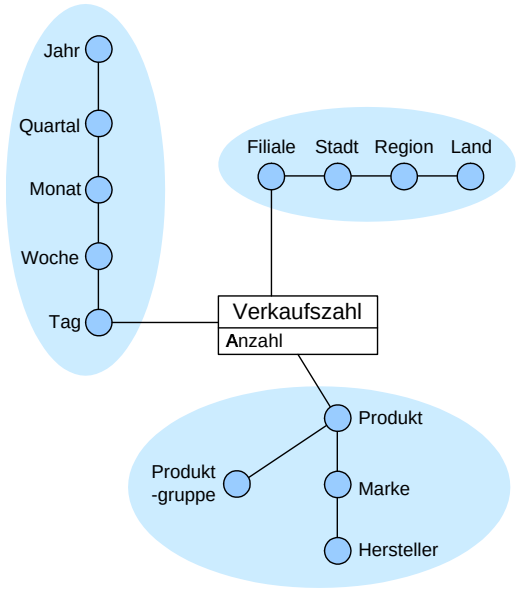



Abbildung 3.18: DFM–Beispielschema

Zur Modellierung der Additivität erlaubt das DFM pro Kombination aus Fakt–Attribut und verbundener Dimension die Angabe der zulässigen Verdichtungsoperatoren. Neben einer vollständigen Additivität werden noch die zwei Klassen *nicht–* und *halb–additiver* Kennzahlen unterschieden, wobei halb–additive Fakt–Attribute gegenüber nicht–additiven mindestens in einer Dimension summierbar sind. Obwohl eine Kennzahl nicht summierbar ist, schließt dies nicht aus, dass Aggregationen durch andere Operatoren erfolgen können. Bildet ein Attribut bezüglich der Additivität eine Ausnahme, wird dies durch eine gestrichelte Linie zwischen Fakt–Attribut und der Dimension im Fakt–Schema gekennzeichnet. Abbildung 3.19 skizziert bez. des Temperatur–Attributes die Tatsache, dass die Aufsummierung mehrerer Datensätze nicht sinnvoll ist und stattdessen der *avg*–Operator zur Berechnung der Durchschnittstemperatur Anwendung findet.

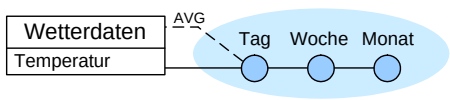



Abbildung 3.19: DFM: Darstellung der Additivität

Abschließend lässt sich anmerken, dass die strenge Baumstruktur der Fakt–Schemata bei der Dimensionsmodellierung lediglich die Darstellung von Mehrfachhierarchien erlaubt, was die Möglichkeiten erheblich einschränkt. So sind beispielsweise die in Abbildung 3.20 skizzierten alternativen Hierarchiepfade und gemeinsam genutzten Hierarchieebenen, die komplexe Zusammenhänge innerhalb der qualifizierenden Daten darstellen, durch die Restriktion auf einen Baum nicht möglich. Als positiver Aspekt des DFM ist die Möglichkeit zur differenzierten Darstellung der zulässigen Verdichtungsoperatoren zu nennen.

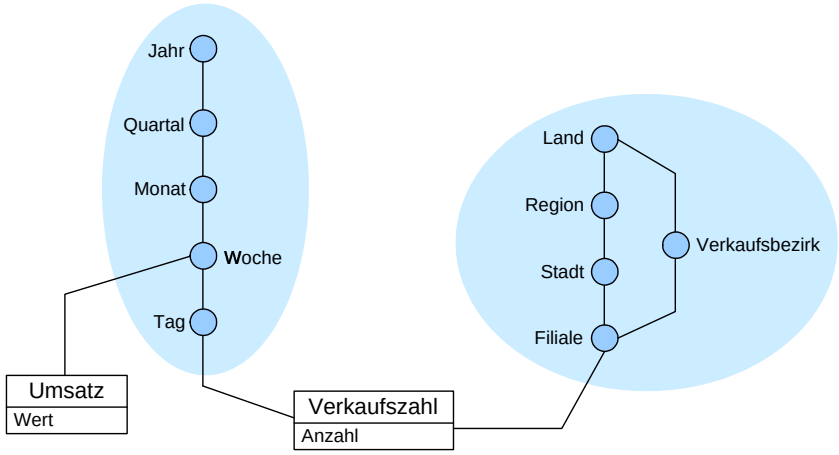



Abbildung 3.20: DFM: Nicht mögliche Darstellungen

### 3.3.5 MD–Modell: Multidimensional Data Model

Das MD–Modell (Multidimensional Data Model) [CT98a] basiert auf den beiden Konstrukten *Dimension* und *F–Tabelle*. Dimensionen werden dabei im Sinne der multidimensionalen Terminologie aus Abschnitt 3.1 verstanden. Jede Dimension besteht entsprechend aus einer Menge von *Ebenen*, die als Datendomänen unterschiedlicher Granularität aufgefasst werden. Innerhalb einer Dimension sind die Instanzen unterschiedlicher Ebenen durch eine Familie von *RollUp*–Funktionen miteinander verbunden. F–Tabellen schließlich werden als Funktionen einer speziellen Kombination von Ebenen auf einer Kennzahl definiert. Die graphische Notation ist in Abbildung 3.21 dargestellt.

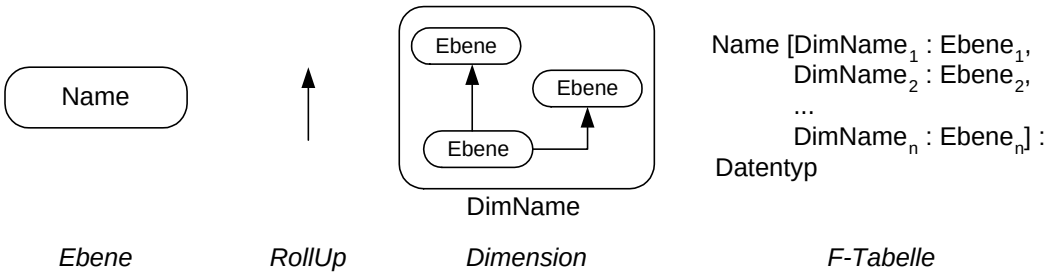



Abbildung 3.21: MD–Notationselemente

Abbildung 3.22 zeigt anhand des Verkaufszahlen–Beispiels ein komplettes MD–Schema.

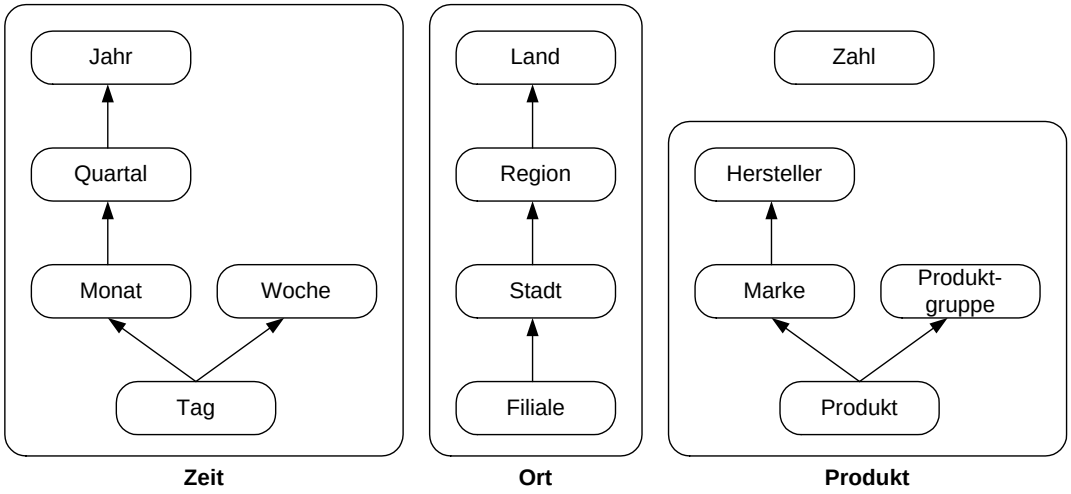



Abbildung 3.22: MD–Beispielschema

Als Bewertung kann festgehalten werden, dass das Schema aufgrund der geringen Anzahl der Modellierungskonstrukte relativ übersichtlich bleibt. Das Definieren sog. *abgeleiteter F–Tabellen*, die in mindestens einer Dimension eine höhere Ebene als die feingranularste benutzen (z. B. ist „Geplanter Absatz [Zeit: Monat, Ort: Filiale, Produkt: Produkt] : Zahl“ möglich), erlaubt die gemeinsame Verwendung von Dimensionen für verschiedene F–Tabellen. Andererseits könnte diese Möglichkeit auch zu einer Vorwegnahme physischer Entwurfsentscheidungen in Form von Materialisierungen führen. Das fehlende graphische Symbol für F–Tabellen macht die Fakt–Dimensions–Zusammenhänge optisch nicht klar, andererseits kann ein Schema relativ kompakt dargestellt werden.

### 3.3.6 MAC – Multidimensional Aggregation Cube

Das in [TKS01] vorgeschlagene Modell MAC (Multidimensional Aggregation Cube) beschreibt Daten wie folgt: *Dimensionsebenen* stellen mögliche Verdichtungen der Diskurswelt dar, verschiedene Dimensionsebenen können durch *Drillingbeziehungen* miteinander verknüpft werden. Eine Menge von Drillingbeziehungen bildet einen *Dimensionspfad*, sofern einige strukturelle Bedingungen erfüllt sind. Ein oder mehrere Dimensionspfade, die gemeinsame Dimensionsebenen haben, bilden eine *Dimension*. Schließlich sind *Multidimensional Aggregation Cubes* (MAC) als Beziehung zwischen den Domänen einer oder mehrerer Dimensionen definiert. Ein MAC kann eine oder mehrere Kennzahlen haben, von denen jede ein atomares Attribut der durch den MAC definierten Beziehung ist. Die Instanz eines MAC wird als *(Würfel–)Zelle* (*(MAC) cell*) bezeichnet.
Die graphischen Notationsprimitive sind in Abbildung 3.23 dargestellt.

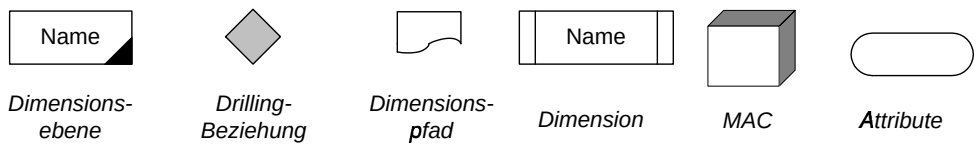



Abbildung 3.23: Konstrukte des MAC–Modells

Als Besonderheit verfügt das MAC–Modell über die Möglichkeit der Definition von *Analysepfaden* (*analysis paths*), die das Aufspüren der Dimensionshierarchien während der konzeptionellen Modellierung erleichtern soll. Dem Problem des Zusammenführens nicht–vollständiger Verdichtungen (siehe Abschnitt 3.1.1 auf Seite 21) wird im MAC–Modell mit Hilfe einer *ALL*–Ebene begegnet, die wieder alle Elemente der Dimension zusammenfasst. Die Modellierung des Beispiels erfolgt in Abbildung 3.24.

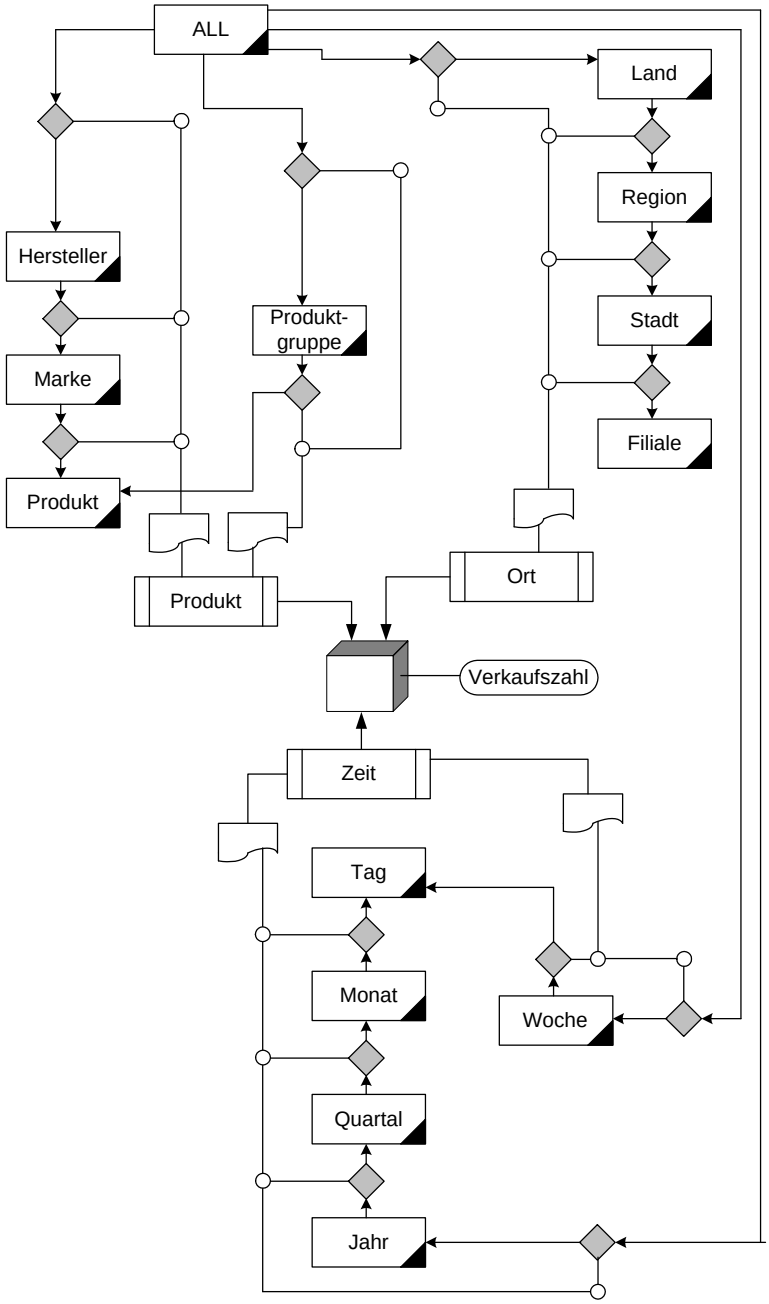



Abbildung 3.24: MAC–Beispielschema

Das MAC–Modell stellt die wesentlichen multidimensionalen Modellierungskonstrukte zur Verfügung und bietet als einziges Modell Analysepfade an. Diese erweitern jedoch nicht die Mächtigkeit des Modells, sondern sind eher als Instrument zur Kommunikation mit potenziellen Anwendern nützlich. Teilweise besitzt das Modell überflüssige Konstrukte, wie z. B. *Dimensionspfad*, die das Lesen der Schemata eher erschweren. Von zweifelhaftem Wert ist auch das *ALL*–Konstrukt, denn durch Weglassen einer einschränkenden Bedingung auf einer Dimension erhält man immer alle Werte dieser Dimension.

### 3.3.7 OLAP–orientierte Ansätze

Neben den im letzten Abschnitt vorgestellten Modellen gibt es in der Literatur eine Reihe multidimensionaler Datenmodelle, deren Fokus nicht primär auf statische Strukturen zielt, sondern vielmehr auf die Datenmanipulation ausgerichtet ist. Insbesondere sind hier die Modelle von Li und Wang [LW96], von Gyssens und Lakshmanan [GL97] und das Modell MADEIRA[7] [Wie00] zu nennen. Da diese Modelle im Rahmen dieser Arbeit aber eine untergeordnete Rolle spielen, soll auf eine vertiefte Betrachtung verzichtet werden, ein vergleichender Überblick findet sich in [Wie00].

## 3.4 Vergleich der Datenmodelle

Basierend auf den Datenmodellanforderungen aus Abschnitt 3.2 werden die in Abschnitt 3.3 vorgestellten Modelle

- MERM (*Multidimensional E/R–Modell*),
- *starER*–Modell,
- ADAPT (*Application Design for Analytical Processing Technologies*),
- DFM (*Dimensional Fact Model*),
- MD–Modell (*Multidimensional Data Model*) und
- MAC–Modell (*Multidimensional Aggregation Cube*)

in den Tabellen 3.1 und 3.2 bez. der Kriterien

- formale *Modellgrundlage*,
- möglicherweise existierender *Werkzeugunterstützung*,
- *innere Struktur* von Fakten,
- *Beziehungen* zwischen Fakten,
- Eigenschaften von *Kennzahlen*,
- dimensionale Eigenschaften auf *Schemaebene* und
- dimensionale Eigenschaften auf *Instanzebene*

vergleichend gegenübergestellt.
Neben den bei den einzelnen Modellen in den Abschnitten 3.3.1 bis 3.3.6 genannten Kritikpunkten lassen sich folgende modellübergreifende Aspekte nennen:

- Kein Modell ermöglicht es, Beziehungen zwischen Fakten auszudrücken.
- Die anteilige Verrechnung wird in den meisten Modellen vernachlässigt.
- Kein Modell basiert auf einer objektorientierten Grundlage.

[7] *M*odelling *A*nalyses of *D*ata in *E*pidemiological *I*nte*RA*ctive studies.

| | | | Konzeptionelle Datenmodelle | | |
|---|---|---|---|---|---|
| | | | Multidimensional E/R Model[8](MERM) | starER–Modell | Application Design for Analytical Processing Technologies (ADAPT) |
| Modellgrundlage | | | E/R–Modell | E/R–Modell | Keine, proprietärer Ansatz |
| Werkzeugunterstützung | | | Editor GraMMi | Nein | Notationselemente existieren als Template für das Zeichenwerkzeug Visio |
| **Fakten** | Anzahl der Datenwürfel/Fakten pro Schema | | Unbegrenzt | Unbegrenzt | Unbegrenzt |
| | Anzahl der Kennzahlen pro Datenwürfel/Fakt | | Unbegrenzt | Unbegrenzt | 0 bis 1; mit Kennzahlendimension: unbegrenzt |
| | Beziehungen zwischen Fakten | | Sind nicht möglich | Sind nicht möglich | Sind nicht möglich |
| | Struktur eines Fakts | | Menge von Kennzahlattributen | Menge von Kennzahlattributen | Menge von Kennzahlattributen |
| | **Kennzahlen** | Additivität | Wird nicht berücksichtigt | Pro Attribut kann eine Aggregierbarkeitseigenschaft angegeben werden | Wird nicht berücksichtigt |
| | | Abgeleitete Attribute | Keine schema–inhärente Unterscheidung zu normalen Attributen möglich | Keine schema–inhärente Unterscheidung zu normalen Attributen möglich | Darstellbar in Kombination mit Berechnungsvorschrift |
| | | Innere Struktur | Keine nähere Beschreibung der Kennzahlen | Keine nähere Beschreibung der Kennzahlen | Keine nähere Beschreibung der Kennzahlen |
| **Dimensionen** | **Schemaebene** | Hierarchieebenen | Können im Schema dargestellt werden | Können im Schema dargestellt werden | Können im Schema dargestellt werden |
| | | Hierarchiepfade | Können im Schema dargestellt werden | Können im Schema dargestellt werden | Können im Schema dargestellt werden |
| | | Hierarchiestruktur | DAG | DAG | Keine Angabe |
| | | Sonstige Beziehungen zwischen Hierarchieebenen | Wird nicht unterstützt | Assoziationen, Generalisierungen und Aggregationen sind möglich | Wird nicht unterstützt |
| | | Mehrfachhierarchien | Werden durch DAG unterstützt | Werden durch DAG unterstützt | Sind darstellbar |
| | | Alternative Verdichtungspfade | Werden durch DAG unterstützt | Werden durch DAG unterstützt | Sind darstellbar |
| | | Anteilige Verrechnung | Wird nicht unterstützt | Kardinalitätsangabe möglich, aber keine Berechnungsvorschrift | Wird nicht unterstützt |
| | | Schlüsselattribute | Jeder Hierarchieebene können identifizierende Attribute zugeordnet werden | Keine Angabe | Hierarchieebenen werden als abstrakte Objekte behandelt |
| | | Dimensionsattribute | Werden pro Hierarchieebene angegeben | Werden pro Hierarchieebene angegeben | Werden pro Hierarchieebene angegeben |
| | **Instanzebene** | Unbalancierte Hierarchien | Können im Schema nicht modelliert werden[9] | Können mit Hilfe nicht–kompletter Beziehungen modelliert werden | Können im Schema nicht modelliert werden |
| | | Unterschiedliche Objekttypen | Sind nicht vorgesehen | Sind mittels Generalisierung darstellbar | Sind nicht vorgesehen |
| | | Optionale Dimensionsattribute | Sind nicht vorgesehen | Sind nicht vorgesehen | Sind nicht vorgesehen |

Tabelle 3.1: Eigenschaften der konzeptionellen Datenmodelle (I)

[8] Die Betrachtung des MERM erfolgt auf der Basis des einfachen E/R-Modells.

[9] Neben numerischen Datentypen sind im Dimensional Fact Model auch Kennzahlen mit einem anderen kontinuierlichen Wertebereich zugelassen.

| | | | Konzeptionelle Datenmodelle | | |
|---|---|---|---|---|---|
| | | | Dimensional Fact Model (DFM) | MD–Modell (Multidimensional Data Model) | MAC–Modell (Multidimensional Aggregation Cube) |
| Modellgrundlage | | | Keine, proprietärer Ansatz | Keine, proprietärer Ansatz | Keine, proprietärer Ansatz |
| Werkzeugunterstützung | | | Keine Angabe | Nein | Nein |
| **Fakten** | Anzahl der Datenwürfel/Fakten pro Schema | | 1; Fakt ist Wurzel des baumförmigen Schemas | Unbegrenzt | Unbegrenzt |
| | Anzahl der Kennzahlen pro Datenwürfel/Fakt | | Unbegrenzt | 1 | Unbegrenzt |
| | Beziehungen zwischen Fakten | | Sind nicht möglich | Sind nicht möglich | Sind nicht möglich |
| | Struktur eines Fakts | | Menge von Kennzahlattributen | Ein Attribut | Menge von Kennzahlattributen |
| | **Kennzahlen** | Additivität | Pro Kennzahl und Dimension ist die Angabe alternativer Funktionen möglich | Wird nicht berücksichtigt | Wird nicht berücksichtigt |
| | | Abgeleitete Attribute | Werden nicht berücksichtigt | Werden nicht berücksichtigt | Werden nicht berücksichtigt |
| | | Innere Struktur | Nur numerische Kennzahlen erlaubt[10] | Attribut beliebigen Datentyps | Menge von Attributen |
| **Dimensionen** | **Schemaebene** | Hierarchieebenen | Können im Schema dargestellt werden | Können im Schema dargestellt werden | Können im Schema dargestellt werden |
| | | Hierarchiepfade | Können im Schema dargestellt werden | Können im Schema dargestellt werden | Können im Schema dargestellt werden |
| | | Hierarchiestruktur | Baumstruktur | DAG | DAG |
| | | Sonstige Beziehungen zwischen Hierarchieebenen | Wird nicht unterstützt | Wird nicht unterstützt | Wird nicht unterstützt |
| | | Mehrfachhierarchien | Werden durch Baumstruktur unterstützt | Werden durch DAG unterstützt | Werden durch DAG unterstützt |
| | | Alternative Verdichtungspfade | Sind aufgrund der Baumstruktur nicht erlaubt | Werden durch DAG unterstützt | Werden durch DAG unterstützt |
| | | Anteilige Verrechnung | Wird nicht unterstützt | Wird nicht unterstützt | Wird nicht unterstützt |
| | | Schlüsselattribute | Hierarchieebenen werden als abstrakte Objekte behandelt | Hierarchieebenen werden als abstrakte Objekte behandelt | Möglich |
| | | Dimensionsattribute | Werden pro Hierarchieebene angegeben[11] | Werden pro Hierarchieebene angegeben | Werden pro Hierarchieebene angegeben |
| | **Instanzebene** | Unbalancierte Hierarchien | Können im Schema nicht modelliert werden | Können im Schema modelliert werden | Können im Schema modelliert werden |
| | | Unterschiedliche Objekttypen | Sind nicht vorgesehen | Sind nicht vorgesehen | Sind nicht vorgesehen |
| | | Optionale Dimensionsattribute | Werden im Schema markiert; es erfolgt jedoch keine Angabe, wann ein Attribut optional ist | Sind nicht vorgesehen | Sind nicht vorgesehen |

Tabelle 3.2: Eigenschaften der konzeptionellen Datenmodelle (II)

[10] Dimensionsattribute werden im Dimensional Fact Model als *non-dimensional attributes* bezeichnet.

[11] Durch Einbeziehung des Generalisierungskonstrukts in den ERM-Kern des MERMS ist eine unbalancierte Hierarchie durch Entitätsuntertypen modellierbar.

## 3.5 Zusammenfassung

In diesem Kapitel wurden multidimensionale Datenmodelle behandelt. Dabei wurden zunächst in Abschnitt 3.1 statische und dynamische Aspekte der multidimensionalen Sichtweise auf Daten eingeführt. Bei den statischen Aspekten wurden neben grundlegenden Charakteristika, wie z. B. der Klassifikation der Daten in quantifizierende und qualifizierende, insbesondere vielfältige Möglichkeiten der Hierarchiebildung in Dimensionen betrachtet. Die dynamischen Aspekte sind im Wesentlichen die von OLAP–Werkzeugen (siehe auch Abschnitt 2.4) zur Verfügung gestellten Operationen.
Aufbauend auf diesen Grundbegriffen wurden in Abschnitt 3.2 unter Berücksichtigung von Publikationen sowohl mit reinem Forschungscharakter als auch mit praktischem Projekthintergrund eine Reihe von Anforderungen an Datenmodelle für die konzeptionelle multidimensionale Modellierung herausgearbeitet. Abschnitt 3.3 stellte sechs existierende Datenmodelle vor, nannte dabei Stärken und Schwächen der einzelnen Ansätze. Abschnitt 3.4 schließlich verglich die vorgestellten Modelle mit den Anforderungen und hielt einige modellübergreifende Kritikpunkte fest.

# Kapitel 4

# Realisierung von Data Warehouses

In diesem Kapitel werden grundlegende Fragen der Realisierung von DWH behandelt. Dabei werden in Abschnitt 4.1 zunächst verschiedene physische Speicherungsformen vorgestellt und ihre Vor– und Nachteile diskutiert. Die Abschnitte 4.2 und 4.3 widmen sich der „relationalen Welt“, indem zunächst verschiedene relationale Schematypen eingeführt und anschließend physische Optimierungsmöglichkeiten dieser Schemata diskutiert werden. Abschnitt 4.4 geht auf die in einem DWS wichtigen Metadaten ein, bevor Abschnitt 4.5 mit einer Zusammenfassung schließt.

## 4.1 Umsetzungsmöglichkeiten des multidimensionalen Datenmodells

Obwohl sowohl Entwickler und Modellierer während der Entwurfsphase als auch OLAP–Werkzeuge während der Datenanalysephase eine multidimensionale Sichtweise (siehe Abschnitt 3.1) auf die in einem DWH verwalteten Daten haben, existieren verschiedene Möglichkeiten der physischen Realisierung. Aufgrund ihrer weiten Verbreitung und Etablierung in Organisationen sowie ihres in der Zwischenzeit erlangten technischen Reifegrades kommen häufig relationale DBMS auch für DWH zum Einsatz. Man spricht in diesem Falle von *ROLAP*–Systemen (Relationales OLAP). Findet hingegen eine direkte Speicherung in multidimensionalen Strukturen statt, so spricht man von *MOLAP*–Systemen (Multidimensionales OLAP). Diese beiden Speicherungsformen weisen verschiedene Vor– und Nachteile auf [AAD+96, CD97, DNR+97, Mar98, Ken99]:

- Relationale Systeme sind in Organisationen weit *verbreitet* und *etabliert*, sie haben sich auch in der Verwaltung großer Datenmengen bewährt. Ebenso existieren sowohl intern (z. B. Sicherheitskonzepte) wie auch extern (z. B. Administrationswerkzeuge von Drittherstellern) eine Reihe von hilfreichen Werkzeugen.
- Ebenso haben relationale Systeme in der Zwischenzeit einen hohen technischen *Reifegrad* erlangt, und mit SQL liegt eine (weitestgehend) *standardisierte Zugriffssprache* zur Verfügung. MOLAP–Systeme als vergleichsweise junge Produkte hingegen besitzen in puncto Performanz–Optimierungen, Lastverteilung etc. weniger Konfigurationsmöglichkeiten, ebenso existieren keine allgemein anerkannten Standards bez. Datenformaten, Abfragesprachen oder Programmierschnittstellen.
- ROLAP–Systeme speichern die Daten in Relationen, die eine Untermenge des Kreuzproduktes aller Wertebereiche (d. h. im multidimensionalen Fall Instanzen der feingranularsten Ebenen jeder Dimension) sind. Somit werden nur vorkommende Wertekombinationen gespeichert. MOLAP–Systeme hingegen bilden einen Datenwürfel direkt auf den physischen Speicher ab, indem dieses mehrdimensionale Konstrukt zu einer eindimensionalen Liste linearisiert wird.

Dadurch wird für jeden möglichen Wert (sprich jede Würfelzelle) Speicherplatz reserviert. Dies führt bei dünn besetzten Würfeln zu Speicherplatzverschwendung. Praktische Analysen [Mic95] haben ergeben, dass insbesondere in betriebswirtschaftlichen Applikationen häufig nur ein Besetzungsgrad von 20% zu erwarten ist, was den ROLAP–Systemen Vorteile verspricht. MOLAP–Systeme versuchen, diesen Nachteil durch verschiedene Formen der Komprimierung zu kompensieren [ZDN97, MK98].

- Weiterhin werden in einem MOLAP–System alle möglichen Verdichtungen über die Dimensionen im Voraus berechnet, so dass sich eine lange Zeit für das sog. *Aufbereiten* oder *Vorberechnen* der Daten ergibt. Diese Zeit entfällt bei ROLAP–Systemen vollständig. Beim *Nachladen von Daten* in das DWH sind in einem ROLAP–System lediglich neue Tupel hinzuzufügen, während ein MOLAP–System den kompletten Würfel neu berechnen muss. Analoges gilt nach dem Löschen von Daten im DWH bei der Archivierung.

- *Anfragen* an die Speicherstruktur sind bei einer MOLAP–Realisierung *einfach formulierbar*, während bei ROLAP–Systemen vom Server immer erst eine Übersetzung in relationale Strukuren vorgenommen werden muss, was häufig zu komplexen SQL–Anweisungen führt.

- Weiterhin sind die *Anfragen* in MOLAP–Realisierungen aufgrund der direkten Speicherung i. d. R. sehr effizient ausführbar, während in einer ROLAP–Umgebung das DBMS zur Laufzeit eine hohe Verarbeitungsleistung erbringen muss, so dass Anfragen nicht immer performant beantwortet werden können.

Unter Abwägung der verschiedenen Aspekte hat sich in den letzten Jahren zunehmend eine als *HOLAP* (Hybrides OLAP) bezeichnete Mischform etabliert, die die Detaildaten in relationaler Form speichert und gewisse Verdichtungen zusätzlich multidimensional vorhält. Abbildung 4.1 zeigt nochmals die drei Realisierungsformen: Der Benutzer hat in seinem Front End–Werkzeug auf jeden Fall eine multidimensionale Sicht auf die Daten. Im Falle einer MOLAP– oder ROLAP–Realisierung wird die Anfrage vom Server in einen entsprechenden DB–Zugriff umgewandelt. Bei der HOLAP–Lösung greift der Server auf die multidimensionalen Aggregate oder, falls diese die Anfrage nicht beantworten können, auf die relationale DB zurück. Dabei ist für den Entwicklungsprozess der DB festzuhalten, dass i. d. R. nur die relationale DB gestaltet werden kann und die Verwaltung der multidimensionalen Aggregate vollständig vom HOLAP–Server übernommen wird, was in der Abbildung durch die etwas abgesetzte Darstellung des rechten Würfels angedeutet ist.

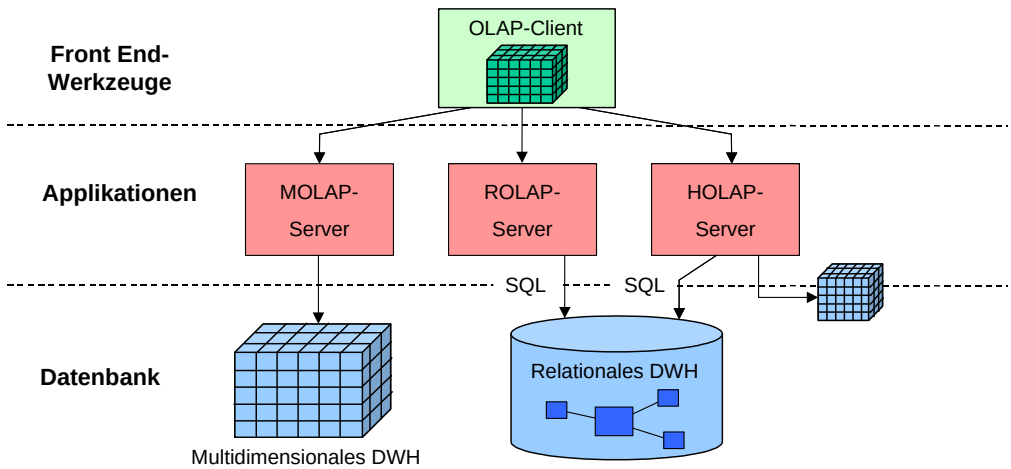



Abbildung 4.1: Realisierungsmöglichkeiten MOLAP, ROLAP und HOLAP

Der Vollständigkeit halber seien an dieser Stelle auch die beiden in einigen Veröffentlichungen vorzufindenden Begriffe *DOLAP* (Desktop OLAP) bzw. *COLAP* (Client OLAP) erwähnt. Dahinter verbirgt sich keine weitere grundlegende Speichertechnik, sondern lediglich eine Architekturform, in der kein separater OLAP–Server existiert. Die komplette Verarbeitung der multidimensionalen Daten erfolgt auf dem Client. Die eigentliche OLAP–Datenbank wird lokal auf der Festplatte der Arbeitsstation abgelegt. Diese Technik kann insbesondere ergänzend eingesetzt werden, indem z. B. Außendienstmitarbeiter einen für sie relevanten Teil der Daten auf ihrem mobilen Endgerät zur Verfügung gestellt bekommen.

## 4.2 Relationale Realisierungen

Dieser Abschnitt beschäftigt sich mit der Darstellung unterschiedlicher multidimensionaler Konstrukte innerhalb des relationalen Datenmodells, das gegenüber dem multidimensionalen Modell eine erheblich „geringere Semantik“ besitzt. Ziel ist es hierbei, multidimensionale Strukturen im relationalen Modell so abzubilden, dass

- domänenspezifische Aspekte, wie z. B. festgelegte Hierarchiepfade, möglichst erhalten bleiben,
- die Übersetzung multidimensionaler Anfragen möglichst effizient geschehen kann,
- die Abarbeitung der übersetzten Anfragen durch das RDBMS möglichst effizient erfolgen kann,
- die Aktualisierung der Tabellen beim Laden und Archivieren möglichst effizient erledigt werden kann.

In Abschnitt 4.2.1 wird zunächst eine Notation für relationale Schemata eingeführt, bevor in den Abschnitten 4.2.2 und 4.2.3 mit dem sog. *Schneeflockenschema* und dem *Sternschema* die in Literatur und Praxis bekanntesten Vertreter relationaler Realisierungen vorgestellt werden. Abschnitt 4.2.4 beschreibt weitere relationale Realisierungen, die im Wesentlichen Varianten und Abwandlungen der beiden Grundformen sind.

### 4.2.1 Notation

Für die Abbildungen dieses Kapitels soll die in der DB–Literatur (siehe z. B. [HS00, CBS98]) übliche Notation gewählt werden (siehe Abbildung 4.2): Eine Tabelle wird als zweigeteiltes Rechteck dargestellt, im oberen Bereich steht der Name, im unteren durch einen waagerechten Strich abgegrenzten Bereich sind die Attribute, per Doppelpunkt von ihrem Datentyp getrennt, aufgelistet. Spielt der Datentyp im aktuellen Kontext keine wesentliche Rolle, so kann er in der Notation entfallen. Primärschlüsselattribute werden **fett** geschrieben, die Primär–Fremdschlüsselbeziehungen durch Kanten dargestellt. Die Anzahlen an der Beziehung beteiligter Tupel werden an die Enden der Kante geschrieben. Zur Darstellung des multidimensionalen Sachverhaltes werden Fakttabellen dunkel und dimensionale Tabellen hell hinterlegt.

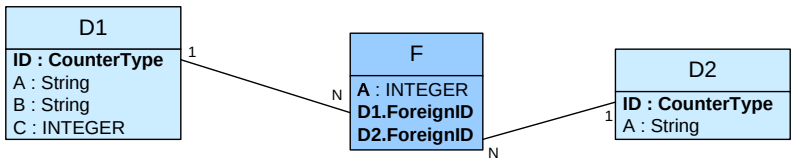



Abbildung 4.2: Darstellung von Tabellen, Attributen und Beziehungen

Diese Notation wird später in Kapitel 9 wieder aufgegriffen. Dort ist die Darstellung weiterer Informationen, z. B. über den Verbindungstyp oder das Verhalten innerhalb einer referentiellen Beziehung, relevant. Diese wird innerhalb eines farbig nicht hinterlegten Rechteckes mit abgerundeten Ecken dargestellt, das an der Beziehungskante platziert ist. Ebenso werden Constraints in farbig nicht hinterlegten Rechtecken mit abgerundeten Ecken dargestellt. Auch diese sind nahe ihrem Bezugsobjekt positioniert. Beispiele dieser Notation sind in Abbildung 4.3 zu sehen: Die Beziehung zwischen den Tabellen „F“ und „D2“ wird als „DIMENSION“ gekennzeichnet und der Wertebereich des Attributs „C“ der Tabelle „D1“ wird auf Werte größer oder gleich 5 eingeschränkt. Um die Übersichtlichkeit zu wahren, soll das Darstellen dieser Zusatzinformationen *sparsam* verwendet werden.

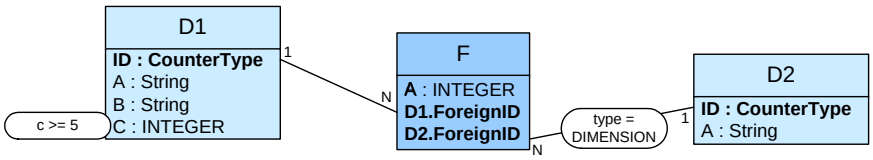



Abbildung 4.3: Darstellung von Primärschlüsselrollen und Constraints

### 4.2.2 Schneeflockenschema

Das *Schneeflockenschema* ist eine direkte Möglichkeit, Dimensionshierarchien darzustellen, indem für jede Hierarchieebene eine eigene Tabelle angelegt wird.

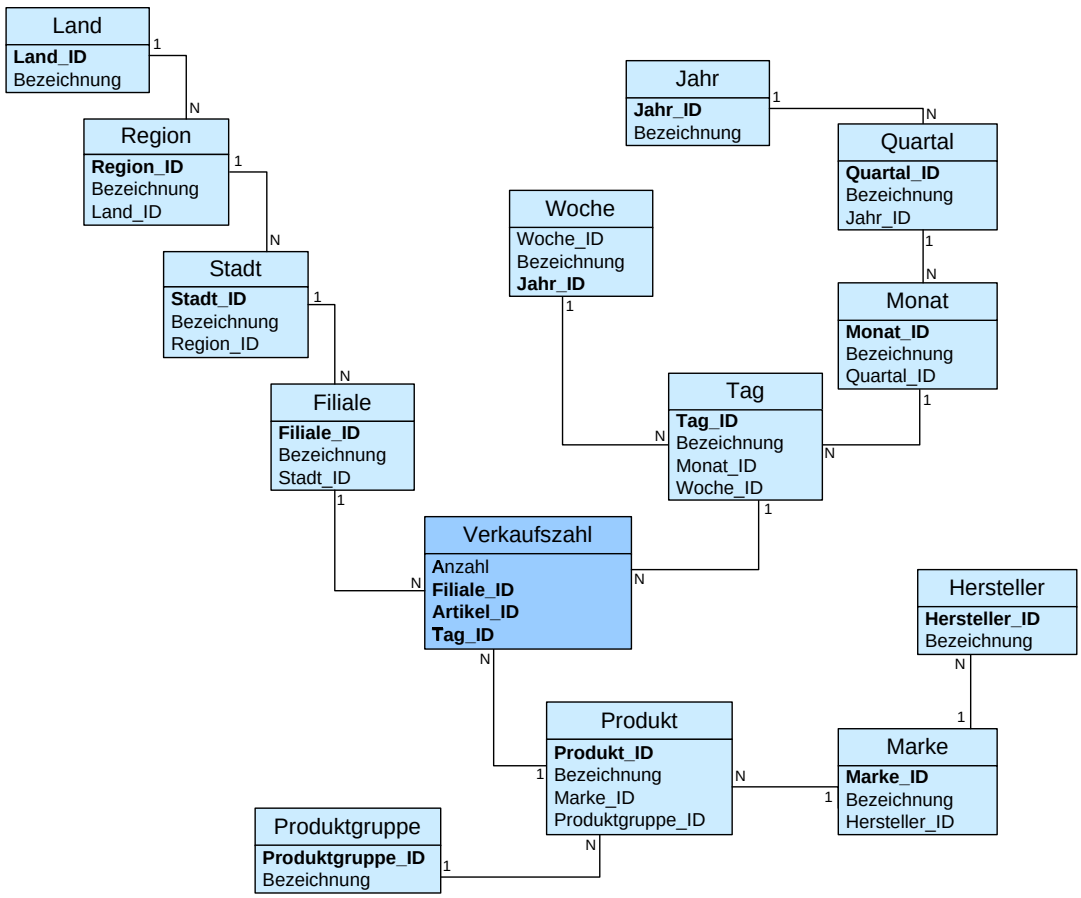



Abbildung 4.4: Schneeflockenschema

Jede Tabelle besitzt neben einem Surrogat alle Attribute der Hierarchieebene sowie zusätzlich Fremdschlüsseleinträge der direkt darüberliegenden Hierarchieebenen. Die Kennzahlen werden in einer sog. Fakttabelle vorgehalten, die neben einer Spalte pro Kennzahl Fremdschlüsseleinträge der Hierarchieebenen der feinsten Granularität jeder Dimension besitzt. Die Menge der Fremdschlüsseleinträge in einer Faktentabelle beschreibt genau eine Zelle des multidimensionalen Datenraums und bildet daher in ihrer Gesamtheit den Primärschlüssel der Faktentabelle. Abbildung 4.4 zeigt diese Form der relationalen Realisierung für das Beispiel „Verkaufszahl“, das schon in Kapitel 3 zur Vorstellung der multidimensionalen Datenmodelle genutzt wurde.
Das Schneeflockenschema ist *normalisiert* bez. der durch die Hierarchiestrukturen induzierten funktionalen Abhängigkeiten, womit z. B. Änderungsanomalien vermieden werden. Andererseits besitzt das Schema Nachteile hinsichtlich der Anfrageverarbeitung, denn gerade bei Hierarchien mit vielen Ebenen fallen viele Verbundoperationen an, die in relationalen DBMS typischerweise sehr teuer sind. Bei einer Auswertung nach „Land“, „Jahr“ und „Hersteller“ müsste beispielsweise ein Verbund über 12 Tabellen gebildet werden.

### 4.2.3 Sternschema

Das *Sternschema* [KRRT98] ist eine relationale Realisierung, die bei Anfragen das Bilden teurer Verbundoperationen vermeidet, indem die zu einer Dimension gehörenden Tabellen zu einer einzigen Tabelle denormalisiert werden. So ist bei n Dimensionen für eine beliebige Anfrage, unabhängig von der Anzahl der Hierarchieebenen, ein Verbund über n+1 Tabellen zu realisieren. Abbildung 4.5 zeigt das bekannte Beispiel, die sternförmige Platzierung der dimensionalen Tabellen um die zentrale Faktentabelle gibt dem Schematyp seinen Namen.

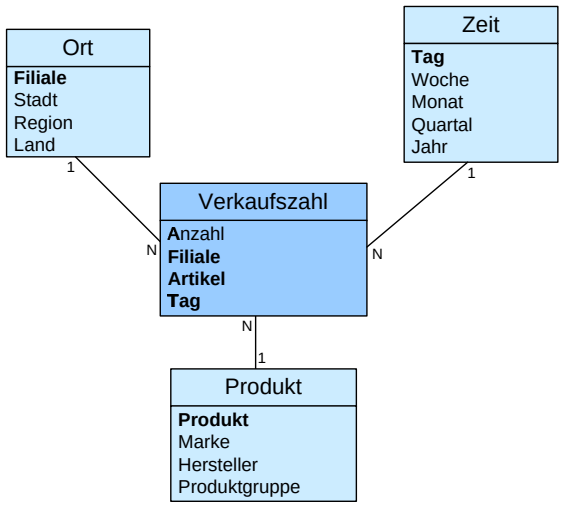



Abbildung 4.5: Sternschema

Der Preis für die bessere Anfrageperformanz durch Senkung der an Verbunden beteiligten Tabellen sind Redundanzen in den dimensionalen Tabellen, die bei der Denormalisierung entstehen. Beispielsweise steht eine Stadt als Objekt der zweituntersten Hierarchieebene so oft in der Tabelle, wie es zu ihr gehörige Tupel der untersten Ebene (d. h. Filialen) gibt. Dadurch sind einerseits Änderungsanomalien „vorprogrammiert“, andererseits kostet die Redundanz Speicherplatz. Dennoch existieren eine Reihe für DWH typische Charakteristika, die für ein Sternschema gegenüber einem Schneeflockenschema sprechen:

- Einschränkungen von Anfragen werden häufig auf höherer Granularitätsstufe vorgenommen; hierbei führen die eingesparten Verbundoperatoren zu einer schnelleren Anfrageverarbeitung.

- Das Datenvolumen der Dimensionen ist im Verhältnis zur Größe der Faktentabelle relativ klein. Daher fallen die durch die Redundanzen verursachten Platzverluste (im Verhältnis zur Gesamtextension der DB) nicht besonders stark ins Gewicht.

- Änderungen der Daten in den Dimensionen (z. B. Umstellung der Produktpalette oder Eröffnung einer neuen Filiale) treten erheblich seltener auf als das Hinzufügen neuer Faktdaten (neue Verkäufe). Außerdem werden diese Änderungen im Zuge des Ladeprozesses unter kontrollierten Bedingungen vorgenommen und entstehen nicht durch Benutzerinteraktion, wodurch die Gefahr von Änderungsanomalien stark eingeschränkt wird.

### 4.2.4 Sonstige Schemaformen

Neben den beiden in den vorangegangenen Abschnitten vorgestellten Schematypen existieren in Literatur und Praxis diverse weitere Vorschläge, die im Folgenden überblicksartig genannt werden sollen:

- Bei einer *Mischform aus Stern– und Schneeflockenschema* [BG01] wird durch Abwägen der Argumente für und wider die beiden Typen für jede Dimension entschieden, ob sie denormalisiert wird oder nicht.

- Mehrere, möglicherweise über gemeinsame dimensionale Tabellen verbundene, Fakttabellen enthaltende Schemata werden manchmal auch als *Multi–Fakttabellen–Schema* oder *Galaxieschema* [KRRT98] bezeichnet.

- Werden in einem Schema neben den Basisdaten auch verdichtete Daten vorgehalten, so ergeben sich mehrere aufeinander aufbauende Stern– oder Schneeflockenschemata. In [KRRT98] wird dieser Schematyp als *Fact Constellation–Schema* bezeichnet.

- Als Variante des Sternschemas ist neben der Anordnung der Hierarchieebenen als Spalten (also horizontal) auch eine *vertikale (oder rekursive) Darstellung* denkbar. Hierbei besitzt jedes Tupel den Eintrag selbst und eine Selbstreferenz auf die Tabelle, durch die die Hierarchiestruktur abgebildet wird.

- Noch einen Schritt weiter geht das in [BD99] vorgeschlagene sog. *Kollabierte Sternschema*. Bei Existenz nur einer dimensionalen Tabelle für alle Dimensionen eines Fakts werden in zusätzlichen Tabellen die (Meta–)Informationen über Dimensionszugehörigkeit und Hierarchiebildungen festgehalten.

- Im System *SAP Business Warehouse* [SAP97, SSL01] kommt eine Variante des Sternschemas zum Einsatz, bei der zu den dimensionalen Tabellen zusätzlich Stammdatentabellen existieren. In diesen Stammdatentabellen werden beschreibende Attribute von Dimensionsobjekten vorgehalten mit dem Ziel einerseits eine enge Kopplung zur Datenquelle zu haben und andererseits können verschiedene dimensionale Tabellen die Informationen in Stammdatentabellen gemeinsam nutzen.

- Die Werkzeuge der Firma *Microstrategy* [Mic99] setzen eine Variante des Schneeflockenschemas ein, bei dem die Fremdschlüssel der höheren Hierarchieebenen nicht nur in der unmittelbar darunterliegenden, sondern auch in noch tiefer liegenden Hierarchieebenen eingetragen werden, wodurch bei Anfragen Verbundoperationen eingespart werden.

## 4.3 Relationale Optimierungsmöglichkeiten

In einem DWH spielen Optimierungstechniken i. d. R. eine größere Rolle als in OLTP–DB. Diese Aussage begründet sich in dem großen Datenvolumen, vor allem der Fakttabelle, in den durch analytische Auswertungen zu erwartenden umfangreichen, multidimensionalen Auswahlbedingungen und Aggregationen sowie die häufig vorhandene Notwendigkeit von Gruppierungen und Sortierungen für Berichte. Daneben fallen beim Nachladen des DWH große Datenmengen an, die eine Vielzahl von Änderungen bestehender Strukturen, wie z. B. Indexen oder Materialisierungen nach sich zieht. Aus diesem Grunde sind in den letzten Jahren eine Vielzahl von Untersuchungen und Veröffentlichungen über die Verwendung existierender Optimierungstechniken und Erweiterungen dieser Techniken unter Berücksichtigung von DWH–Spezifika entstanden. Die folgenden Abschnitte erheben keinen Anspruch auf Vollständigkeit, sondern dienen vielmehr der Vermittlung eines groben Überblickes. Bei Bedarf geben die referenzierten Literaturquellen weiterführende Informationen.

### 4.3.1 Indexierung

Indexstrukturen sind redundante Strukturen zur Optimierung von selektiven Lesezugriffen, was durch eine Reduzierung der für die Anfrage zu lesenden Datenseiten erreicht wird. Als Standard–Indexstruktur in OLTP–DB haben sich *B–Bäume* [BM72] und *B*–Bäume* [Wed74] herauskristallisiert. Diese ermöglichen den effizienten Zugriff bei sog. Punktanfragen, d. h. gezielter Zugriff auf einen Datensatz oder eine relativ kleine Datenmenge. In DWH jedoch werden typischerweise viele Datensätze umfassende sog. Bereichsanfragen gestellt, so dass diese herkömmlichen Indexierungstechniken nur beschränkt einzusetzen sind. Auf der anderen Seite können im Gegensatz zu OLTP–DBen in DWH komplexere Indexstrukturen eingesetzt werden, weil die Problematik der Indexpflege bei massiv konkurrierenden Datenänderungen entfällt.
So sind eine Reihe neuer Indextypen konzipiert worden, z. B. mehrdimensionale Indexstrukturen. Ein bekannter Vertreter ist der *R–Baum*[1] [Gut84], der das Indexieren von Bereichen in Form von Rechtecken des multidimensionalen Datenraumes ermöglicht. Auf dem R–Baum basierend sind eine Reihe von Varianten entstanden, z. B. der*Gepackte R–Baum* [RL85], der den freien Speicherplatz in der Indexstruktur zu minimieren versucht, der *R+–Baum* [SRF87], der durch Vermeidung überlappender Bereiche die Suche im Baum beschleunigt, und der sich durch geänderte Einfüge– und Split–Operationen auszeichnende *R*–Baum* [BKSS90]. Diese letzte Variante wurde in [BKK96] zum *X–Baum* erweitert, der durch variable Knotengrößen Überlappungen im Inneren des Baumes weitgehend vermeidet. Speziell im DWH–Kontext entstand der *UB–Baum*[2] [RMF+00], der die Indexierung mittels eindimensionaler Einbettung multidimensionaler Punktobjekte realisiert.
Als weitere Indexform besitzen im DWH–Bereich *Bitmap–Indizes* [CI98] große Bedeutung. Bitmap–Indizes sind eindimensionale Indexstrukturen, die sich insbesondere für Attribute mit geringer Kardinalität eignen. Für jede mögliche Ausprägung des Attributes wird eine Bitliste angelegt, auf der effizient boolesche Verknüpfungen zur Auswertung von mehrdimensionalen Suchausdrücken realisiert werden können. Neben dieser Form des einfachen Bitmap–Index existieren eine Reihe von sog. kodierten Bitmap–Indizes [WB98], in denen das Setzen eines Bits in der Liste statt des konkreten Wertes die Zugehörigkeit zu einem Intervall oder Bereich angibt.
Interessant sind vergleichende Ansätze mit Untersuchungen, welche Indexierungstechnik unter bestimmten Bedingungen bessere Resultate erwarten lässt [JL99].

[1] R–Baum: Rectangle tree.
[2] UB–Baum: Universal B–Baum.

### 4.3.2 Partitionierung

Partitionierung hat ihren Ursprung im Bereich verteilter und paralleler DBS [Rah94, OV99], wobei die Aufteilung einer Relation auf einzelne Rechnerknoten mit dem Ziel der Lastverteilung im Vordergrund steht. Dabei werden die zwei Phasen der *Fragmentierung* (Bestimmung der Verteilungseinheiten) und der *Allokation* (Zuordnung der Fragmente zu physischen Einheiten wie Plattenspeichern oder Rechnerknoten) unterschieden. Aber auch in nicht–verteilten DBen können durch Partitionierungen Performanzsteigerungen erreicht werden, indem eine Tabelle mit umfangreicher Extension auf mehrere kleinere, dann als *Partitionen* bezeichnete Tabellen aufgeteilt wird. Aufgund ihrer Extensionsgröße bietet sich insbesondere die Faktentabelle zur Partitionierung an. Im Wesentlichen wird zwischen den in Abbildung 4.6 dargestellten Varianten *horizontaler* und *vertikaler* Partitionierung unterschieden.

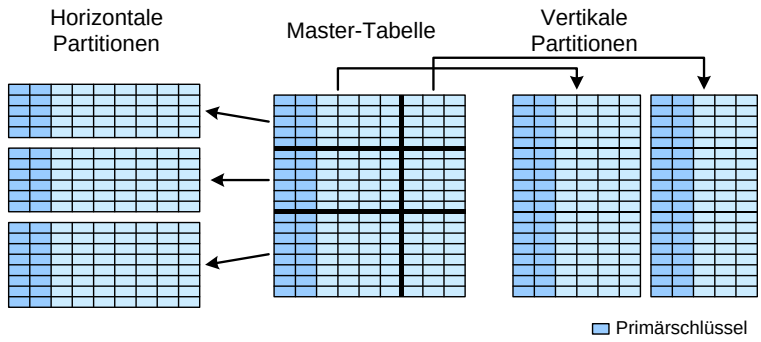



Abbildung 4.6: Horizontale und vertikale Partitionierung

Horizontale Partitionierung kann *zufällig* z. B. nach dem *Round Robin–Verfahren* oder *wertebasiert* erfolgen [KN99, Dor99]. Während die wertebasierte Partitionierung bei DB–Operationen bestimmte Partitionen ausschließt, kann zufällige Partitionierung zur Erhöhung des Parallelitätsgrades von Operationen genutzt werden. Bei der wertebasierten Variante kann wiederum zwischen *Bereichs*– und *Hashpartitionierung* unterschieden werden [BG01].Während bei der Hashpartitionierung eine Funktion über die Fragmentierung der Tupel entscheidet, geschieht die Bereichspartitionierung aufgrund semantischer Kriterien. Im Kontext von DWH bieten sich häufig Ort und Zeit als Kriterien zur Fragmentierung an. Werden beispielsweise in einem DWH die Zahlen der letzten vier Jahre gespeichert, könnte die gesamte Faktentabelle durch Bereichsfragmentierung in vier Partitionen aufgesplittet werden.
Vertikale Partitionierung bietet sich vor allem für besonders „breite“ Tabellen, d. h. solche mit vielen Attributen, an. Im DWH kann dies für einige Dimensionstabellen zutreffen. Weil für das Wiederzusammensetzen der Partitionen jedoch eine relativ teure 1:1–Verbundanfrage nötig ist, besitzt die vertikale Partitionierung im DWH–Umfeld nur eine untergeordnete Bedeutung.
[MWM99, GMR00] sind als zentrale Arbeiten zur Partitionierung in DWH zu nennen.

### 4.3.3 Materialisierte Sichten

Unter *materialisierten Sichten* wird die explizite Speicherung von Anfrageergebnissen mit dem Ziel der Beschleunigung erneuter Anfragen verstanden. Aufgrund ähnlicher, häufig vorkommender Anfragen, Lokalität bei Auswertungen und einem relativ stabilen Datenbestand sind materialisierte Sichten eine sehr effektive Optimierungsoption in DWH [Rou98, GMRR01]. Wesentliche Realisierungsaspekte sind die *Auswahl* zu materialisierender Sichten und die *Aktualisierung* bei Datenänderungen

[BLT86, TB88, GL95]. Die Auswahl der zu materialisierenden Sichten kann *statisch* oder *dynamisch* erfolgen. Bei dynamischer Auswahl wird ein Caching von Anfrageergebnissen vorgenommen, wobei die Lokalität von Ad–Hoc–Anfragen genutzt wird. Besonders vorteilhaft ist dies bei interaktiven, aufeinander aufbauenden Anfragen, z. B. Roll Up–Operationen. Dieses Verfahren kann jedoch nur im Kern des DBMS realisiert werden, so dass es für den Entwurfsprozess (und damit im Rahmen dieser Arbeit) keine Bedeutung besitzt. Hier spielt vielmehr die statische Auswahl von Sichten eine Rolle, die durch den Entwickler (oder auch automatisiert durch ein Werkzeug) geschieht. Die Auswahl erfolgt aufgrund von Erfahrungen aus der Vergangenheit, meistens in Form von Anfragemustern, und bis zur nächsten Aktualisierung des DWH werden keine Veränderungen der materialisierten Sichten vorgenommen. In einem DWH bieten sich als materialisierte Sichten Kombinationen von Hierarchieebenen an. Dabei ist es in der Praxis aufgrund des benötigten Speicherplatzes bzw. des Aktualisierungsaufwandes i. Allg. nicht möglich, alle Kombinationen zu realisieren, denn die Anzahl möglicher Kombinationen wächst exponentiell mit der Anzahl der Dimensionen.
Ziel ist es daher, eine „möglichst optimale“ Teilmenge zu ermitteln, die einerseits die Anfrageperformanz verbessert, andererseits aber vertretbare Kosten bez. benötigtem Speicherplatz und Aktualisierungsaufwand aufweist.
Zum Thema Materialisierung im DWH sind in den letzten Jahren etliche Arbeiten entstanden. In [HRU96, Gup97] werden Algorithmen zur Auswahl zu materialisierender Sichten vorgestellt, die die Antwortzeiten unter der Nebenbedingung des verfügbaren Speicherplatzes minimieren. Diese Aufgabenstellung wird als *Sichtenauswahlproblem* oder *DWH–Konfigurationsproblem* [TS97] bezeichnet. Dieses Optimierungsproblem ist i. Allg. NP–vollständig, die genannten Arbeiten bieten daher als Lösung auf einer Greedy–Strategie basierende Algorithmen an, die polynomiale Laufzeit aufweisen und um einen konstanten Faktor von der optimalen Lösung abweichen. Erweitert werden diese Ansätze in [GHRU97, LQA97] auf die Auswahl von Sichten und Indizes. All diese Arbeiten vernachlässigen jedoch die Wartungskosten der materialisierten Sichten. Diesen Aspekt berücksichtigen u. a. die Arbeiten [GM99, CLF99], die die Auswahl zu materialisierender Sichten unter der Nebenbedingung einer vorgegebenen Zeit für die Wartung betrachten. Dieses kann in der Praxis Bedeutung haben, wenn für das Nachladen der Daten in ein DWH nur eine beschränkte Zeit zur Verfügung steht. In [URT99] wird der Ansatz aus [HRU96] wieder aufgegriffen und zusätzlich die Häufigkeit von Anfragen sowie die Kosten der Wartung der materialisierten Sichten berücksichtigt.
In [LH99] wird statt einer Greedy–Strategie ein genetischer Algorithmus zur Lösung des Optimierungsproblems verwendet. Hierbei wurden Resultate erzielt, die bei linearem Laufzeitverhalten um etwa 10% von der optimalen Lösung abwichen.
Daneben existieren noch eine Reihe weiterer Publikationen zum Thema, die das Problem der Auswahl zu materialisierender Sichten ohne Nebenbedingungen und ohne Aussagen zur Güte der Lösung betrachten [RSS96, YKL97, BPT97, TS97].
Meistens wird davon ausgegangen, dass während des Nachladens des DWH (und damit während der Pflege der materialisierten Sichten) kein lesender Zugriff für die auf dem DWH operierenden Applikationen zugelassen ist. In [QW97] wird ein Ansatz vorgestellt, der auch bei laufenden Aktualisierungen konsistente Lesezugriffe garantiert.
Neben der Auswahl der zu materialisierenden Sichten spielt auch ihre Wartung eine wichtige Rolle. Aufgrund der Datenvolumina ist ein vollständiges Neuberechnen der materialisierten Sichten i. Allg. nicht möglich, so dass *inkrementelle Aktualisierungen* vorgenommen werden müssen. Mit effizienten Algorithmen hierfür beschäftigen sich u. a. [GMS93, LMSS95, ZGHW95, MQM97, LS99, OAE00, Huy00, LY01]. Als weitere Problemstellung kann in einem DWH das Entdecken redundanter materialisierter Sichten bei Evolutionen im DWH auftreten, womit sich [The01] beschäftigt.

Abbildung 4.7 skizziert noch einmal die in diesem Abschnitt erwähnten physischen Optimierungsoptionen, wobei die im DWH–Kontext relevanten hinterlegt sind.

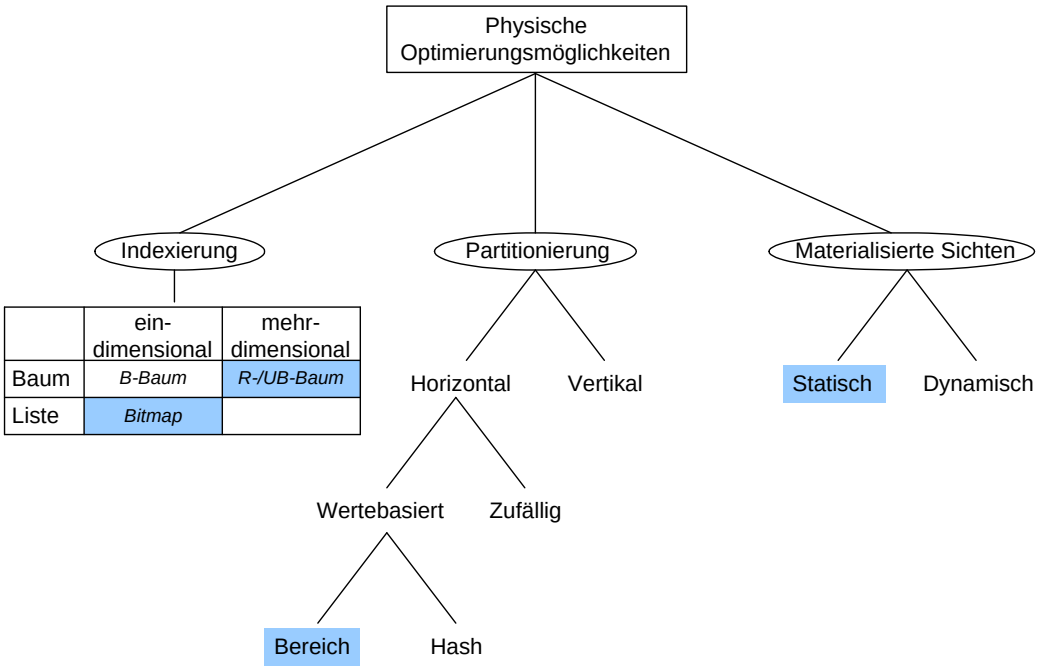



Abbildung 4.7: Physische Optimierungsmöglichkeiten

## 4.4 Metadaten

Wie in Abschnitt 2.5.2 motiviert wurde, spielt die Verwaltung von Metadaten in einem DWS eine wichtige Rolle. Durch das Zusammenspiel der vielen einzelnen Komponenten in einem DWS (siehe Kapitel 2) ergibt sich für die Verwaltung der in Abschnitt 2.5.2 aufgelisteten Metadatentypen eine Reihe von Anforderungen [Rah01]:

- Vorliegen eines *mächtigen Metadatenmodells* zur Repräsentation aller relevanten Typen von Metadaten,
- *Konsistente Bereitstellung* aller relevanten Metadaten auf aktuellem Stand,
- *Zugriffsmöglichkeiten* auf alle Metadaten über mächtige Schnittstellen,
- Vorhandensein einer *Versions– und Konfigurationsverwaltung*,
- *Unterstützung für technische und fachliche Aufgaben* der Benutzer,
- Möglichkeit der *aktiven Unterstützung* von DWH–Prozessen, z. B. automatisches Generieren von Transformationsskripten.

Im Folgenden sollen kurz existierende Standardisierungsbemühungen (Abschnitt 4.4.1) und unterschiedliche physische Realisierungen (Abschnitt 4.4.2) beschrieben werden.

### 4.4.1 Standards

Neben einer Vielzahl in der Praxis existierender proprietärer Metadatenmodelle, die von (Werkzeug–)Herstellern oder auch aus Forschungsprojekten [JJQV99, JLVV00] stammen, gibt es zwei nennenswerte Standardisierungsbemühungen: Zum einen ist dies das *OIM* (*Open Information Model*) der *MDC* (*Meta Data Coalition*), auf der anderen Seite das *CWM* (*Common Warehouse Metamodel*) der *OMG* (*Object Management Group*). Beide Modelle sind UML–basiert, ihre Strukturierung ist in Abbildung 4.8 gegenübergestellt. Das OIM ist in themenspezifischen Submodelle unterteilt. Die Menge aller Submodelle umfasst alle relevanten Aspekte des Informations–, DWH– und Wissensmanagement. Im CWM wird die zentrale Komponente das *CWM Foundation Model* von der UML abgeleitet ist. Um diesen Kern herum siedeln sich die 13 im rechten Teil von Abbildung 4.8 dargestellten Modelle an.

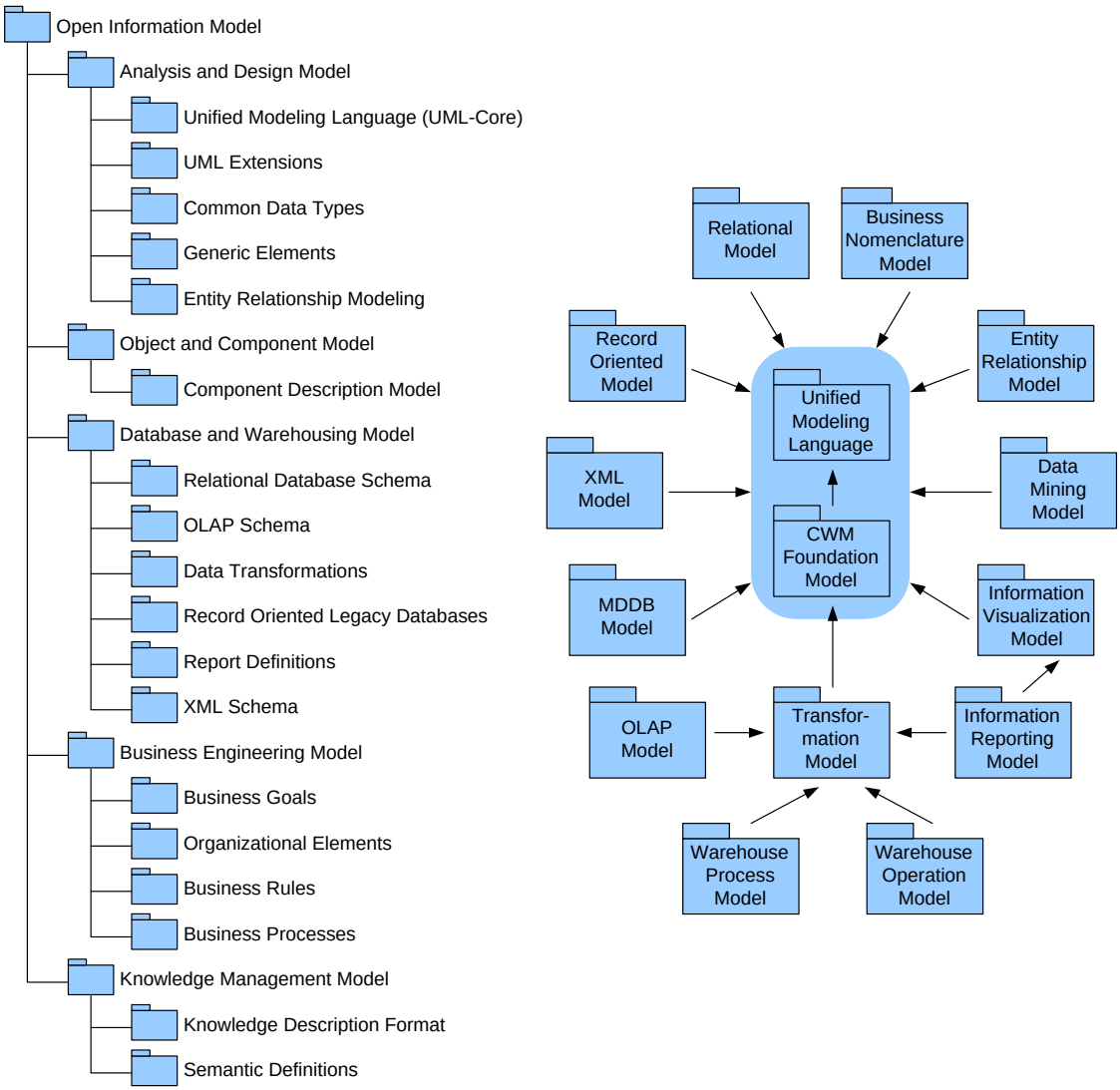



Abbildung 4.8: Strukturierung von OIM und CWM

Im folgenden werden die beiden Ansätze kurz skizziert, eine ausführlicher Vergleich kann in [VVS00] nachgelesen werden.
Das OIM [Met99a, BBC+99] wurde von der MDC, einer 1995 gegründeten Non Profit–Organisation von Herstellern, entwickelt. Ziel ist die Bereitstellung einer hersteller– und plattformunabhängigen

Spezifikation von Metadaten, damit verschiedene Werkzeuge über ein gemeinsam genutztes Informationsmodell interoperieren können. Das OIM unterstützt alle Phasen der Informationssysteme–Entwicklung von der Analyse bis zum Betrieb. Das OIM nutzt die UML (*Unified Modeling Language*) in dreifacher Weise:

- als Modellierungssprache für die Beschreibung der OIM–Modelle,
- als Hauptbestandteil des Submodells *Analysis and Design Model* und
- als Kernmodell des OIM, von dem die anderen Modelle objektorientierte Konzepte erben.

Diese letztgenannte Eigenschaft ermöglicht es, zur Handhabung werkzeugspezifischer Aspekte von den bestehenden (Sub–)Modellen Spezialisierungen zu erzeugen. Dies zeigt einerseits eine gute Offenheit des Standards im Sinne von Erweiterbarkeit, andererseits ergibt sich die Problematik, dass zwei Spezialisierungen eines gemeinsamen (Sub–)Modells auch wieder zueinander inkompatibel sind. Wenn nun verschiedene Hersteller Besonderheiten ihrer Werkzeuge durch neue Submodelle abbilden, entstehen somit eine Reihe von inkompatiblen Modellen unter einem gemeinsamen Dach.
Das CWM der OMG [Obj01] zielt auf eine Unterstützung des Metadatenaustausches wie er in DWSen auftritt. Dementsprechend enthält das CWM im Gegensatz zum OIM vor allem weiterreichende Modelle mit DWH–spezifischem Inhalt, wie z. B. Informationen zum Betrieb des DWH oder zum Data Mining.
Zur Entwicklung ist anzumerken, dass die Tätigkeiten der MDC im Herbst 2000 eingestellt worden, bei der Weiterentwciklung des CWM sollen jedoch Ideen des MDC einfließen.

### 4.4.2 Datenhaltung

Die konkrete Organisation der Metadatenhaltung in einem DWS kann in drei Formen geschehen:

- Alle Metadaten werden in einem *zentralen Repository* gehalten. Dieser Ansatz hat den Vorteil, dass keine Replikation von Metadaten notwendig ist, andererseits existiert aber eine hohe Abhängigkeit vom zentralen Repository. Darüber hinaus besitzen die einzelnen Komponenten nur eine eingeschränkte Autonomie.
- Bei einer vollständig *verteilten Metadatenverwaltung* sorgt jede einzelne Komponente des DWS für die Verwaltung ihrer Metadaten. Den Vorteilen maximaler Unabhängigkeit und schnellen Zugriffs auf die lokalen Ressourcen stehen als Nachteile neben den zahlreichen Verbindungen zwischen den Komponenten eines DWS zum Metadatenaustausch auch ein hoher Grad an Replikation und vor allem das Problem der Synchronisation der verschiedenen Metadatenbestände gegenüber.
- Als Kombination dieser beide Wege ergibt sich ein als *Shared Repository* bezeichneter Ansatz, der sowohl lokale Speicherung bei den einzelnen Komponenten vorsieht, wie auch ein von allen Komponenten gemeinsam genutztes, zentrales Repository.
  Die Vorteile dieser Lösung sind eine einheitliche Repräsentation gemeinsamer Metadaten, Wahrung lokaler Autonomie, Reduzierung des Metadatenaustauschs und eine kontrollierte Redundanz. Demgegenüber muss ein Konzept für die Kommunikation zwischen lokalen und globalem Repository etabliert werden. Ansätze hierfür werden in [DR00] vorgestellt.

## 4.5 Zusammenfassung

In diesem Kapitel wurden zunächst die möglichen physischen Realisierungen von DWH vorgestellt: Basierend auf der zugrunde liegenden Technik wird dabei zwischen ROLAP–, MOLAP– und HOLAP–Systemen unterschieden.
Anschließend wurden in Abschnitt 4.2 verschiedene relationale Realisierungen vorgestellt, wobei sich Schneeflocken– und Sternschemata als die beiden grundlegenden Schematypen herausgestellt haben, auf denen aufbauend eine Vielzahl von Schemavarianten existiert. Abschnitt 4.3 diente der kurzen Vorstellung unterschiedlicher Optimierungsmöglichkeiten relationaler DWH–Implementierungen. Dieser Abschnitt ist bewusst kurz gehalten und mit vielen Referenzen versehen, unter denen Details nachgelesen werden können. Als Quintessenz kann festgehalten werden, dass eine Vielzahl von Arbeiten zu Optimierungsmöglichkeiten und –verfahren existieren, die meisten jedoch eine Möglichkeit oder ein Verfahren nur isoliert (evtl. noch auf einen bestimmten Kontext eingeschränkt, aber nicht hinreichend breit und abstrahierend) betrachten.
Abschnitt 4.4 befasste sich mit Metadaten, deren große Bedeutung im Kontext von DWH bereits in Abschnitt 2.5.2 motiviert worden war, wobei Standards und physische Realisierung der Metadatenverwaltung in einem DWS behandelt wurden.

# Kapitel 5

# Entwurf von Informationssystemen

Im Mittelpunkt dieses Kapitels steht der Entwurf von Datenbanken. Dabei werden zunächst in Abschnitt 5.1 wichtige Begriffe aus dem Umfeld des Entwurfs operativer DBen eingeführt. Abschnitt 5.2 stellt verwandte Arbeiten zum DB–Entwurf mit Fokus auf DWHs vor. In Abschnitt 5.3 werden einige weitere Aspekte aus den Bereichen Software Engineering und DB–Entwurf behandelt, die auf die in Teil II dieser Arbeit konzipierte Entwurfsmethodik Einfluss haben.

## 5.1 Entwurf operativer Datenbanken

Weil die zentrale Datenhaltung für mehrere Anwendungssysteme über einen i. Allg. mehrjährigen Zeitraum ein kritischer Aspekt des Informationsmanagements ist, kommt dem Entwurf einer DB besondere Bedeutung zu [HS00]. Daher ist ein als *Entwurfsmethodik* bezeichneter strukturierter Ansatz notwendig, der unter Verwendung wohldefinierter Vorgehensweisen, Techniken, Werkzeuge und Dokumentationen den *Entwurfsprozess* der DB unterstützt bzw. erleichtert [CBS98]. Im Kontext von Entwurfsprozessen werden im Folgenden einige Begriffe eingeführt, die in Abbildung 5.1 zusammenfasst sind.

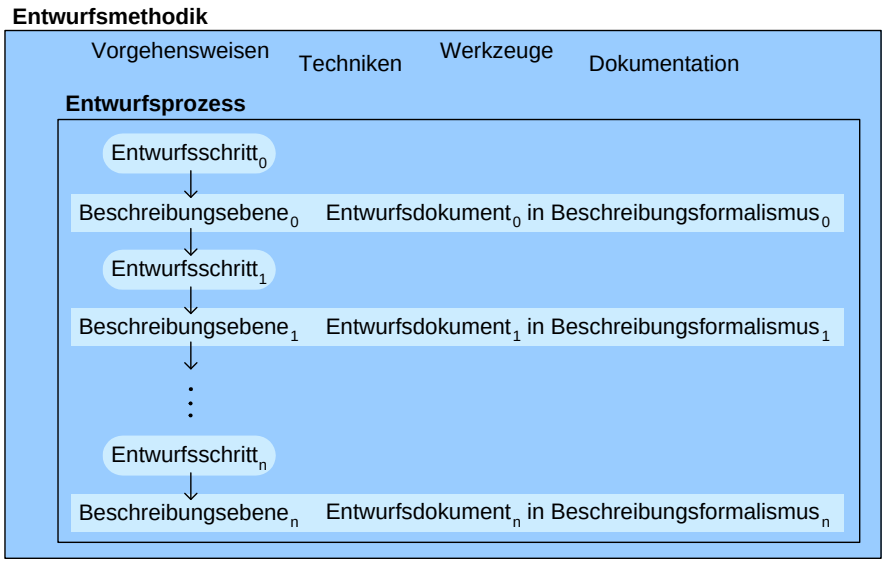



Abbildung 5.1: Begriffsbildung Datenbankentwurf

In Analogie zum Phasenmodell [PS94, Bal98, Som00] als klassischem Entwurfsvorgehen im Software Engineering wird der Entwurfsprozess als Abfolge von *Entwurfsdokumenten* (auch als *Modellierungen* bezeichnet) beschrieben. Jedes Entwurfsdokument wird mit den Mitteln eines *Beschreibungsformalismus* verfasst und gehört zu einer *Beschreibungsebene* (auch als *Abstraktionsebene* bezeichnet). Beginnend mit einer abstrakten, anwendungsnahen Beschreibungsebene und führt der Entwurfsprozess bis zur konkreten Realisierung der DB. Zwischen zwei Beschreibungsebenen erfolgt ein *Entwurfsschritt*, der ein Entwurfsdokument auf ein anderes abbildet, wobei der Beschreibungsformalismus beibehalten wird oder — was die Regel ist — wechseln kann. Einzelne Entwurfsschritte können manuell durchgeführt werden oder (in verschiedenem Grad) automatisiert sein. An jeden Entwurfsschritt sind zwei Bedingungen zu richten: *Vollständigkeit* (in [HS00] als *Informationserhalt* bezeichnet), die fordert, dass in der neuen Darstellungsform alle Informationen gespeichert werden können, die in der ursprünglichen Beschreibungsform möglich waren, und *Korrektheit* (in [HS00] als *Konsistenzerhaltung* bezeichnet), die besagt, dass Einschränkungen und Regeln der Ausgangsbeschreibung auch in der neuen Beschreibungsform respektiert werden.
Beim DB–Entwurf hat sich dabei im Laufe der Jahre ein *Drei–Ebenen–Entwurf* etabliert, der die drei Beschreibungsebenen *konzeptionell*, *logisch* und *physisch* umfasst. Begonnen wird der Drei–Ebenen–Entwurf mit dem konzeptionellen Entwurfsschritt, in dem ein Schema aller (bez. des Projektziels) benötigten Informationen entworfen wird, wobei möglichst vollständig von konkreten Software– bzw. DBMS–Systemen und auch allen weiteren physischen Randbedingungen abstrahiert wird. Anschließend folgt der *logische Entwurfsschritt*, der ein Schema der benötigten Informationen unter Verwendung eines konkreten Datenmodells (z. B. relational oder objektorientiert) gestaltet. Es herrscht jedoch wie in der konzeptionellen Ebene größtmögliche Unabhängigkeit vom verwendeten DBMS oder anderen physischen Randbedingungen. Schließlich folgt der *physische Entwurfsschritt*, in dem eine Beschreibung der Implementierung der Datenbank auf Sekundärspeicher erfolgt. Diese Beschreibung umfasst sowohl Speicherstrukturen als auch Zugriffsmethoden für einen effizienten Zugriff auf die Datenbank. Typische Beschreibungsformalismen sind auf der konzeptionellen Entwurfsebene das E/R–Modell, auf der logischen Ebene das Relationenmodell und auf der physischen Ebene die Datendefinitionssprache des eingesetzten DBMS.
Jede Entwurfsebene korrespondiert in gewisser Weise mit einem Entwurfsschritt des Software Engineering: Der konzeptionelle Entwurf entspricht der Analyse, der logische dem Entwurf und der physische der Implementierung.

## 5.2 Arbeiten zum Entwurf von DWHs

Aufgrund der unterschiedlichen Einsatzgebiete operativer DBen und DWHs ergeben sich Unterschiede zwischen den beiden Datenbanktypen (siehe auch Tabelle 2.1 auf Seite 13), die ein einfaches Übertragen der Entwurfsmethodiken operativer DBen nicht zulassen. Aus diesem Grunde sind eine Reihe von Arbeiten entstanden, die den DB–Entwurf mit dem Fokus auf DWHs betrachten. Diese werden in diesem Abschnitt kurz skizziert und bewertet[1].

### Babelfish

*Babelfish* ist ein Teilprojekt des DWH–Projektes *System 42* vom FORWISS[2] München, dessen Ziel die Entwicklung einer Methodik zum modellgestützten Entwurf und Betrieb von Repository–basierten DWHs ist. Dabei sollen vorhandene Modellierungs– und Entwurfsmethodiken auf ihre

[1] Die ersten Ansätze auf diesem Gebiet stammen von Kimball [Kim96] und Rautenstrauch [Rau97]. Während die erste Arbeit sehr stark auf die relationale Zielwelt ausgerichtet und sehr viele Aspekte des physischen Entwurfs behandelt, ist die zweite speziell auf die Realisierung mit einer Oracle–DB ausgelegt. Diese beiden Arbeiten besitzen somit eher „historische“ Bedeutung und werden daher nicht detailliert behandelt.

[2] Bayerisches Forschungszentrum für wissensbasierte Systeme.

Eignung im DWH–Umfeld untersucht und an die speziellen Erfordernisse dieser Umgebung angepasst werden. Resultat des Projektes ist die in Abbildung 5.2 skizzierte Architektur.

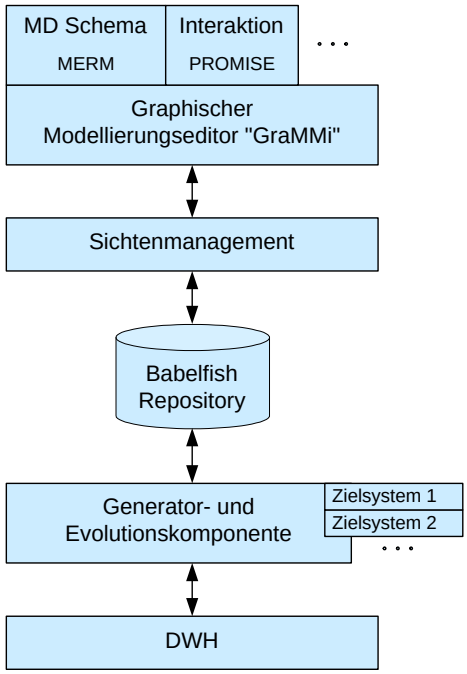



Abbildung 5.2: Architektur Babelfish

Als Beschreibungsformalismus der konzeptionellen Ebene kommt das in Abschnitt 3.3.1 vorgestellte MERM zum Einsatz. Neben diesen statischen Aspekten wurde versucht, in der frühen Entwicklungsphase der konzeptionellen Modellierung Benutzerinteraktionen einfließen zu lassen. Hierzu wurde die Notation PROMISE (Predicting User Query Behaviour in Multidimensional Information System Environments) eingeführt [Sap99, Sap00a, Sap00b]. MERM–Schemata wie auch PROMISE–Diagramme werden im graphischen Editor GraMMi (Graphical Meta–Data–Driven Modeling Tool) [SBH00] verarbeitet und über eine Zwischenschicht zum Sichtenmanagement im Repository gespeichert. Aufbauend auf dem Repository erfolgt durch eine Generator– und Evolutionskomponente eine zielsystemspezifische Transformation [HSB00a, HSB00b].
Als positive Aspekte des Babelfish–Ansatzes lässt sich einerseits die Berücksichtigung dynamischer Aspekte, andererseits die prinzipielle Erweiterbarkeit (sowohl weiterer Notationen als auch weiterer Zielsysteme) festhalten. Als Probleme bzw. offene Fragen bleiben die fehlende saubere Trennung zwischen logischem und physischem Entwurf sowie die unklare Integration statischer und dynamischer Aspekte bei der konzeptionellen Modellierung, ebenso unklar ist die Einbindung weiterer Notationen.

### MetaMIS

Das Projekt *MetaMIS* der Universität Münster [BH98, Hol99, HK99, Hol00] hatte das primäre Ziel, eine werkzeuggestützte Methodik zum konzeptionellen DWH–Entwurf zu erstellen und mit dem *MetaMIS Toolset* zu implementieren. Als zweites wurde das Werkzeug mit einem DWH–Werkzeug gekoppelt, um den Lebenszyklus eines DWH in den einzelnen Phasen von der Entwicklung (vom konzeptionellen Entwurf bis zur Implementierung) bis zum Betrieb (Aktualisierungen und Evolutionen) durchgängig zu unterstützen. Die Gesamtarchitektur des Werkzeugs ist in Abbildung 5.3 dargestellt.

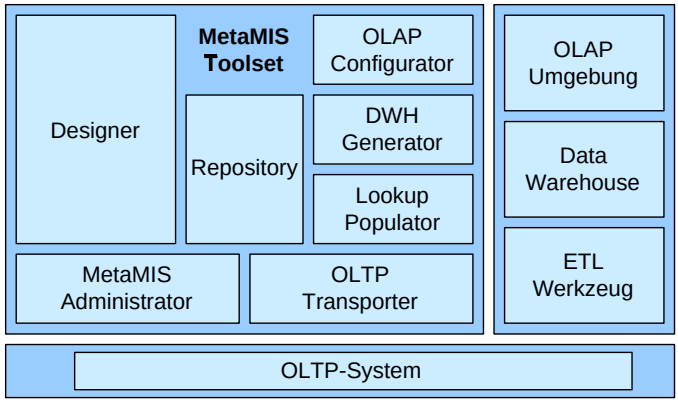



Abbildung 5.3: Architektur MetaMIS

Positiv zu erwähnen ist die Idee der Durchgängigkeit, inklusive des Populationswerkzeugs *Lookup Populator*, das ein Prototyping ermöglicht. Zu kritisieren ist die Tatsache, dass das im Repository abgelegte Metamodell sowohl Kennzahlen und Dimensionen als auch physische Aspekte wie Schlüsselangaben und applikationsspezifische Elemente wie Berichte enthält, womit diverse Entwurfsebenen gemischt werden. Weiterhin fokussiert das Metamodell lediglich auf den betriebswirtschaftlichen Kontext und ist somit nicht domänenunabhängig.

### WanD: Warehouse Integrated Designer

In [GMR98a, GR98] wird der in Abbildung 5.4 skizzierte Entwurfsprozess für DWHs vorgeschlagen, der auf den existierenden operativen Datenquellen aufsetzt. Begonnen wird hierbei mit einer *Analysephase*, in der aus existierenden Dokumentationen der operativen Systeme DB–Schemata einheitlich in E/R–Notation erstellt werden.

| **Schritt** | **Eingabe** | **Ausgabe** |
|---|---|---|
| Analyse der operativen DBen | Vorliegende Dokumentation | DB-Schema |
| Anforderungsanalyse | DB-Schema | Fakten, vorläufiger Workload |
| Konzeptioneller Entwurf | DB-Schema, Fakten, vorläufiger Workload | Konzeptionelles Schema |
| Workload-Verfeinerung, Schema validieren | Konzeptionelles Schema, vorläufiger Workload | Validiertes konzeptionelles Schema, Workload |
| Logischer Entwurf | Konzeptionelles Schema, Datenvolumen, Workload | Logisches Schema |
| Physischer Entwurf | Logisches Schema, Ziel-DBMS, Workload | Physisches Schema |

von WanD unterstützt

Abbildung 5.4: WanD: Warehouse Integrated Designer

Daran schließt sich eine *Anforderungsanalyse* an, in der die Erwartungen der potenziellen Benutzer an das zu entwerfende DWH festgestellt werden. Ist dieses abgeschlossen, wird das *konzeptionelle Schema* mittels einer semiautomatischen Vorgehensweise aus den operativen Quellen abgeleitet. Die Notation für das konzeptionelle Schema ist das in Abschnitt 3.3.4 vorgestellte Dimensional Fact Model. Daran schließt sich eine Phase der *Workload–Bestimmung* und der *Schemavalidierung* an. Ein

Workload besteht aus einer Menge von Anfragen, die Schemavalidierung prüft, ob das erzeugte konzeptionelle Schema diese Anfragen erfüllen kann. Ebenso werden in dieser Phase Annahmen über das Datenvolumen getroffen. Aufbauend auf diesen Informationen findet der *logische Entwurf* statt, der in diesem Ansatz neben der Bestimmung von Tabellen auch Materialisierungen und Partitionierungen umfasst. Die abschließende Phase des *physischen Entwurfs* nimmt in Abhängigkeit vom konkreten Zielsystem im Wesentlichen die Auswahl von Indizes vor. Diese Arbeiten sind in das prototypische CASE–Werkzeug WanD (Warehouse Integrated Designer) eingeflossen, die graue Hinterlegung in Abbildung 5.4 skizziert die im Werkzeug implementierten Phasen.
Als positiv ist das Aufbauen auf dem Drei–Ebenen–Ansatz zu erwähnen, zu kritisieren ist jedoch die Zuordnung der Aufgaben zu den Phasen logischer und physischer Entwurf. Das Bestimmen von Materialisierungen und Partitionierungen ist vom eingesetzten System abhängig und sollte daher Gegenstand des physischen Entwurfs sein. Wiederum positiv zu erwähnen ist die faktenbasierte Vorgehensweise bei der Ableitung des konzeptionellen Schemas, nachteilig hingegen ist die Anforderung des Vorliegens eines E/R–Schemas als Ausgangsbasis, denn gerade die Struktur externer, für das DWH interessanter Daten ist selten in einer E/R–Notation modelliert. Darüber hinaus bleibt unklar, was geschieht, wenn bei der Schemavalidierung eine unzureichende Bereitstellung von Daten aus den Quellen festgestellt wird.

### Konstruktion initialer Schemata nach Sinz et al.

In [BE99b, BE99a] wird ein Verfahren zur Gewinnnung initialer DWH–Schemata aus operativen DB–Schemata vorgeschlagen. Ausgangspunkt ist dabei die E/R–Variante SERM[3] [Sin88]. Eine Besonderheit des SERM ist die Forderung, dass bei der Visualisierung unabhängige Entitätstypen auf der linken Seite im Diagramm stehen, die abhängigen dann auf der rechten Seite folgen. Das Vorgehen beinhaltet im Wesentlichen die Schritte Identifikation von Kennzahlen, Dimensionen und Hierarchien sowie von Integritätsbedingungen zwischen den Dimensionshierarchien.

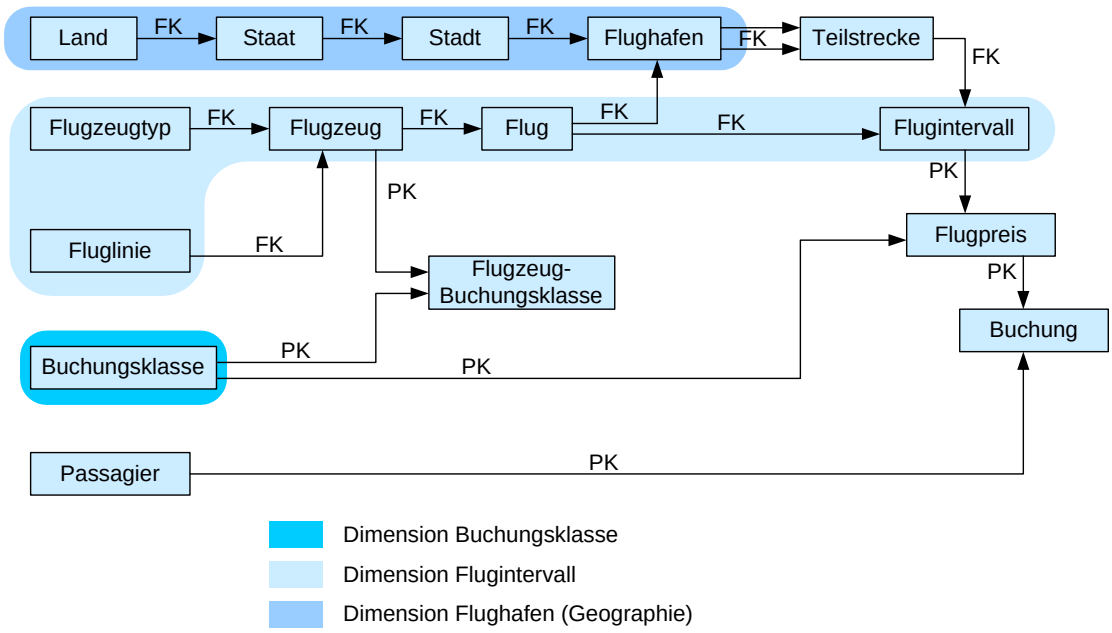



Abbildung 5.5: Ableitung initialer DWH–Schemata aus operativen Schemata

[3] SERM: Structured Entity Relationship Model.

Während der erste Schritt weitgehend manuell abläuft, versucht man im zweiten Schritt von der Visualisierungsvorschrift des SERM zu profitieren. Von links nach rechts ergeben sich im Diagramm Pfade von Entitätstypen mit *1–zu–Viele*–Beziehungen, die als potenzielle Kandidaten für Hierarchiepfade in Frage kommen, wie in Abbildung 5.5 zu erkennen ist. Auch der dritte Schritt profitiert von dieser Darstellung, denn sind die Pfade erst einmal entdeckt, so kann man auch Beziehungen zwischen diesen leicht erkennen.
Diese scheinbar sehr intuitive Vorgehensweise bringt aber auch Probleme mit sich. Weil die Beziehung das einzige Konstrukt im E/R–Modell (und auch im SERM) ist, werden hierdurch auch andere Sachverhalte als hierarchische, inhaltliche Abhängigkeiten dargestellt. Weiterhin ist das vorliegende Schema für eine OLTP–DB optimiert und kann für das DWH zu grob oder zu fein bez. der Hierarchieebenen sein. Als letzten Kritikpunkt kann man anmerken, dass durch eine solche Vorgehensweise das resultierende Schema sehr stark von den in den operativen DBen vorliegenden Schemata abhängig ist und somit möglicherweise nicht die Anforderungen potenzieller DWH–Benutzer erfüllt.

### Konstruktion initialer Schemata nach Peralta et al.

In [PMR99] wird der in Abbildung 5.6 skizzierte Ansatz vorgestellt. Als Eingabe dienen ein konzeptionelles Schema, die Quelldaten und Abhängigkeiten zwischen diesen. In einem ersten Schritt legt der Designer einige Entwurfsziele für das logische Schema fest. Ein solches Entwurfsziel kann beispielsweise das Einhalten von Normalisierungen oder umgekehrt das bewusste Denormalisieren sein. Aus diesen Entwurfszielen wird ein sog. Entwurfsplan generiert, der den Designer bei Anwendung einer Reihe von sog. Transformationsprimitiven hilft. Insgesamt gibt es 14 Transformationsprimitive, die Manipulationen am Schema vornehmen. Das Resultat ihrer Anwendung ist ein logisches DWH–Schema.

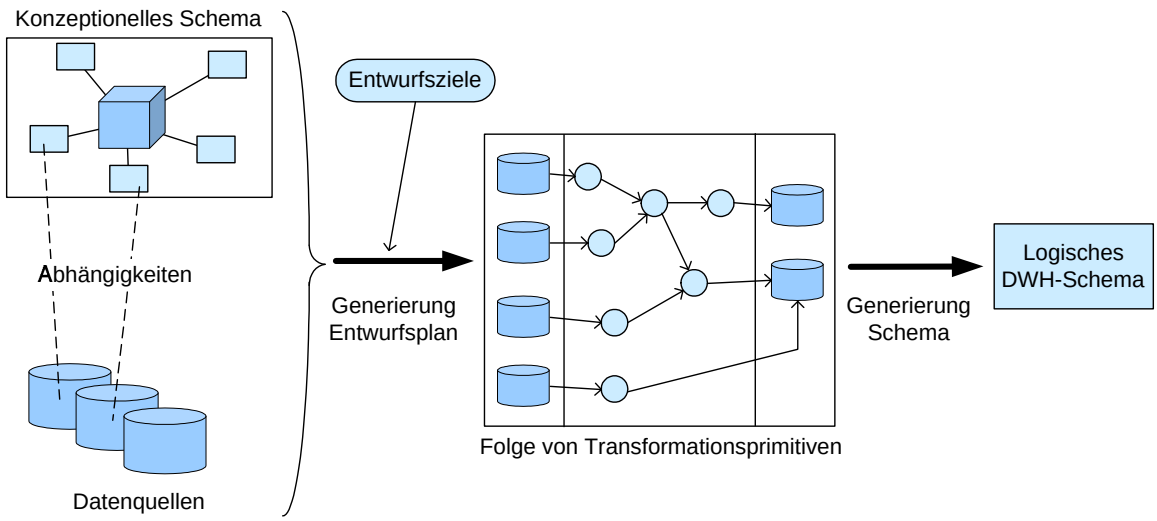



Abbildung 5.6: DWH–Entwurf mit Transformationsprimitiven

Positiv ist an diesem Ansatz das Bestimmen von Entwurfszielen durch den Entwickler zu nennen, denn durch die möglichen Vorgaben lässt sich ein auf die Datenanalyse bzw. das Zielsystem ausgerichtetes Schema erzeugen. Nachteilig ist zum einen die enge Verknüpfung zwischen Datenquellen und konzeptionellem Schema, die das Verständnis der konzeptionellen Modellierung unabhängig von jeglichen physischen Rahmenbedingungen durchbricht, und zum anderen die Beschränkung der Anwendung der Transformationsprimitive auf relationale Schemata, obwohl in DWS typischerwei-

se viele heterogene Quellsysteme existieren. Weiterhin bleibt die Schemaintegration bei nicht völlig disjunkten Beständen in den verschiedenen Datenquellen unklar.

### Vorgehen nach Cabibbo und Torlone

In [CT98b] wird das in Abbildung 5.7 skizzierte Vorgehen vorgeschlagen. Während einer *Analysephase eingehender Daten* werden die Anforderungen an die Datenanalyse aufgenommen, interne wie externe Quellen analysiert und für jede Quelle ein konzeptionelles Schema erstellt. Während der *Integration der Datenquellen* werden diese einzelnen konzeptionellen Schemata integriert, um ein einheitliches Schema aller Quellen zu haben. Daran schließt sich der *Entwurf des DWH* an, der zunächst eine Überführung des integrierten E/R–Schemas in ein konzeptionelles Schema vorsieht, wobei als Notation das in Abschnitt 3.3.5 vorgestellte MD–Modell zum Einsatz kommt. Daran schließt sich der logische Entwurf an, der ein Übertragen in z. B. relationale Strukturen vorsieht. Als letzte Phase erfolgt eine von den Autoren als *Entwurf der multidimensionalen DB* bezeichnete Tätigkeit, die sich ihrerseits in logischen Entwurf und Implementierung unterteilt. Hiermit ist die Gestaltung von Data Marts im Sinne von aggregierten Extrakten (siehe Abschnitt 2.3.2) gemeint.

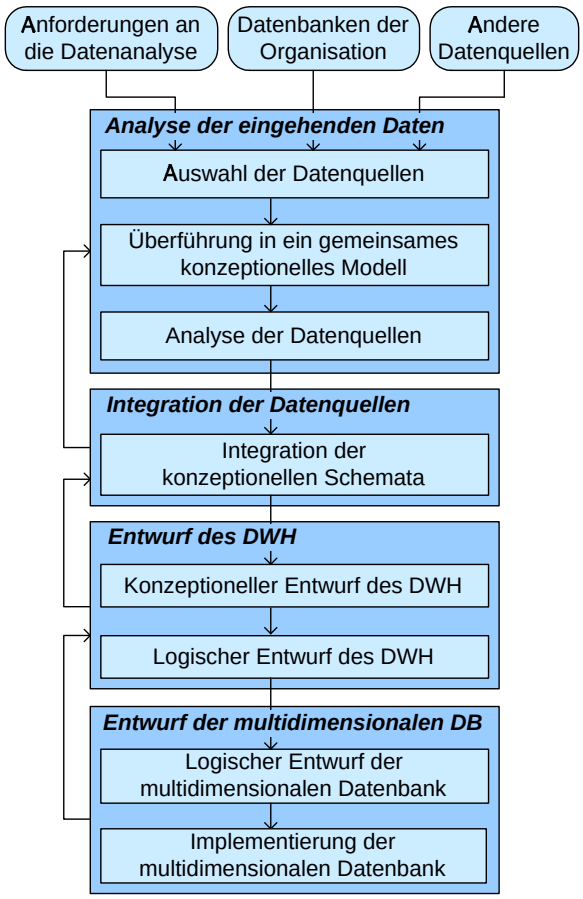



Abbildung 5.7: Vorgehen nach Cabibbo und Torlone

Die Methodik stellt einen umfassenden Rahmen mit sauberer Trennung der einzelnen Phasen dar. Das konstruierende Vorgehen des konzeptionellen DWH–Schemas bringt jedoch die bereits in den letzten beiden Abschnitten genannten Probleme mit sich, nämlich Verzicht auf eine völlig unabhängige konzeptionelle Modellierung durch das frühe Einbeziehen der Datenquellen und unklares Vorgehen

bei überlappender Datenbeständen in den Quellen. Ebenso bleibt die Beschreibung einzelner Schritte, z. B. logischer Entwurf und Auswahl zu realisierender Data Marts, unklar.

### Schemaentwurf nach Lechtenbörger

In [Lec01] wird ein auf dem Drei–Ebenen–Entwurf basierender Ansatz zum DWH–Schemaentwurf vorgeschlagen. Hierbei werden *Normalformen* und *Änderungsunabhängigkeit* als wesentliche Kriterien für die Qualität konzeptioneller und logischer Schemata genannt. Neben der Übertragung von Normalformen wie Vollständigkeit und Redundanzfreiheit sind auch zwei neue Normalformen für den DWH–Kontext definiert worden, die die Aspekte der Additivität, d. h. korrekter Resultate bei Verdichtungen, und Vermeidung von NULL–Werten betreffen. Änderungsunabhängigkeit eines DWH soll die effiziente Wartung ermöglichen, insbesondere in Umgebungen mit entkoppelten Datenquellen. Aufbauend auf diesen beiden Kriterien wird eine Entwurfsmethodik vorgeschlagen, die von der Anforderungsanalyse bis zum logischen Entwurf einer relationalen Implementierung reicht. Neben dem „klassischen“ Vorgehen in der Analysephase, das die Sichtung der Daten und Benutzeranforderungen umfasst, werden in dieser Phase auch die Schemata von operativen DBen untersucht, die als potenzielle Datenquellen in Frage kommen. In der Phase der konzeptionellen Modellierung werden die Ergebnisse der Anforderungsanalyse zu multidimensionalen Faktschemata angeordnet, wobei das in [HLV00] beschriebene multidimensionale Modell verwendet wird. Darüber hinaus wird ein Algorithmus vorgeschlagen, der die Resultate der Anforderungsanalyse automatisch in ein multidimensionales Schema überführt. Im Zuge des logischen Entwurfs wird eine Schematransformation vorgenommen, die eine relationale Implementierung des konzeptionellen Schemas in Form von änderungsunabhängigen materialisierten Sichten vornimmt. Weiterhin werden im resultierenden Schema NULL–Werte vermieden.
Positiv zu erwähnen ist das Aufbauen auf dem bewährten Drei–Ebenen–Ansatz, ebenso interessant ist die kombinierte Betrachtungsweise von neuen Anforderungen und existierenden Schemata während der Anforderungsanalyse. Durch die Änderungsunabhängigkeit wird das DWH als Menge materialisierter Sichten der Datenquellen aufgefasst, wodurch der im Back End–Bereich zu etablierende Integrationsaspekt verlorengeht. Das resultierende Schema erfüllt zwar die aufgestellten Kriterien, in der Praxis verlangt jedoch das eingesetzte DBMS bzw. der OLAP–Server zur effizienten Anfrageverarbeitung bestimmte Schematypen.

### Kommerzielle Werkzeuge

Aufgrund des sich stetig wandelnden Marktes an Werkzeugen und des Markteintritts ständig neuer Hersteller kann an dieser Stelle keine komplette Marktübersicht gegeben werden. Stattdessen werden die Werkzeuge in drei Klassen unterschieden und jeweils ein wichtiger Vertreter genannt[4].
Weit verbreitet ist der *Einsatz herkömmlicher Werkzeuge*, d. h. solche, die in einer Organisation etabliert sind und eine E/R–Notation anbieten. Vorteil dieser Werkzeuge sind Etablierung, Akzeptanz und meistens schon erreichte Produktreife. Ein Beispiel für ein solches Werkzeug ist *DeZign* der Firma Datanamic [Dat01]. Das Problem dieser Klasse von Werkzeugen ist die Verwendung der E/R–Notation, das Schema muss schon in einer frühen Entwurfsphase auf das eingesetzte Zielsystem ausgelegt sein (auch wenn das Werkzeug selber mehrere unterschiedliche DBMS unterstützt). Somit fallen konzeptioneller und logischer Entwurf praktisch zusammen. Ebenso sind DWH–spezifische Gestaltungen und Optimierungen wie z. B. Materialisierungen nicht möglich.
Zur Behebung dieser Mankos gibt es einige *herkömmliche Werkzeuge mit Zusätzen*, die ihre ursprüngliche E/R–Notation erweitern. Dieser Kategorie kann das Werkzeug *ERWin* [Com01] zugeordnet werden, das in den letzten Jahren in der Modellierung konventioneller DBen in der Praxis große Bedeutung erlangt hat. Es ermöglicht die Erstellung von E/R–Schemata und besitzt die DWH–spezifische

[4] Alle hier vorgenommenen Aussagen beziehen sich auf Herstellerangaben vom Sommer 2001.

Funktionalität, Stern– und Schneeflockenschemata zu modellieren. Hierauf aufbauend lassen sich unter Analysegesichtspunkten Performanz–optimierte Schemata generieren. Einen weiteren Vorteil bietet *ERWin* durch seine Einbettung in die sog. *ModelMart*–Umgebung [Com01], die eine professionelle Verwaltung der erstellten Schemata ermöglicht. Weiterhin bietet *ERWin* Möglichkeiten, Informationen über die Datenquellen, den Back End–Prozess und den Ladevorgang zu verwalten. Dies klingt einerseits nach einer integrierten Lösung und damit positiv, andererseits wird hierdurch aber der Schemaentwurf massiv durch die operativen Quellen beeinflusst. Als weiterer Problempunkt bleibt zu vermerken, dass die Ergänzungen lediglich relationale Realisierungen unterstützen und somit nur für solche Zielsysteme in Frage kommen.
Schließlich sind als weitere Klasse kommerzieller Werkzeuge *DWH–Anbieter–Werkzeuge* wie z. B. der *Oracle Warehouse Builder* [Ora01] zu nennen. Diese bilden ähnlich wie die Werkzeuge mit Ergänzungen einige Erweiterungen für den Entwurf von DWH, wie z. B. die Modellierung von Sternschemata. Die enge Anbindung an die DB führt zu sehr gut optimierten Schemata, weil der Hersteller eine größtmögliche Ausnutzung der physischen Optimierungsmöglichkeiten vornehmen kann. Auf der anderen Seite ist man natürlich nicht nur wie bei den beiden erstgenannten Werkzeugklassen vom logischen Modell abhängig, sondern auch vom konkreten physischen des Herstellers.
Als Fazit kann über existierende Werkzeuge festgehalten werden, dass diese i. Allg. eine E/R–basierte Notation verwenden und somit keine unabhängige, konzeptionelle Modellierung unterstützen. Bei Werkzeugen von DWH–Anbietern wie z. B. Oracle fehlt außerdem die Unabhängigkeit vom Zielsystem. Andererseits besitzen die Werkzeuge Stärken in der Generierung optimierter Schemata.

## 5.3 Weitere Aspekte des Software und Database Engineering

Dieser Abschnitt führt aus dem Bereich Software Engineering, speziell der Entwicklung von Datenbanken, einige Begriffe ein, auf die in Teil II zurückgegriffen wird.

### 5.3.1 Qualitätssicherung

Bei der Entwicklung von Softwaresystemen und Datenbanken spielt die Qualitätssicherung (QS) eine wichtige Rolle. Dabei wird nach der Norm IEEE 729 unter Qualitätssicherung die Gesamtheit aller planbaren Maßnahmen und Hilfsmittel verstanden, die bewusst dazu eingesetzt werden, um die Anforderungen an den Entwicklungs– und Wartungsprozess und damit an das Softwareprodukt selbst zu erfüllen. Es werden dabei folgende Teilaspekte unterschieden:

- *Konstruktive* QS fasst alle technischen, organisatorischen und psychologisch orientierten Maßnahmen zusammen, durch die Qualitätsmerkmale erzwungen oder wahrscheinlich und Qualitätsmessungen ermöglicht werden.
- *Analytische* QS zielt auf Maßnahmen zur Bewertung und Prüfung von Software und Softwaredokumentationen, die ggf. zu einer Verbesserung führen. Dabei wird zwischen *statischen* Prüfungen, d. h. ohne Programmausführung, und *dynamischen* Prüfungen mit Programmausführungen unterschieden[5].
- *Psychologisch–orientierte* QS fasst schließlich alle Maßnahmen zur Verbesserung der Teamarbeit, Kommunikation, Motivation usw. zusammen.

[5] In [Bal98] werden die statischen Prüfungen als *analysierende Verfahren* und die dynamischen Prüfungen als *testende Verfahren* bezeichnet.

Im Bereich der statischen Methoden der analytischen QS werden folgende drei wesentliche Verfahren unterschieden:

- *Inspektion* als formale Evaluationstechnik, in der Anforderungen an Software, Entwurfsentscheidungen oder Quelltext von einer Person oder einer Gruppe von Personen, die nicht Urheber des zu untersuchenden Objektes sind, detailliert geprüft werden, um Fehler, Verletzungen von Entwicklungsstandards oder andere Problemfelder zu entdecken [IEE83].
- *Review* als möglichst formalisierter Prozess zur Überprüfung von schriftlichen Dokumenten durch Gutachter, um Stärken und Schwächen des Dokuments festzustellen [Bal98].
- *Walkthrough* als abgeschwächte Form eines Review, bei dem ein Designer oder Programmierer ein oder mehrere Mitglieder des Entwicklungsteams durch einen von ihm entwickelten Quelltextabschnitt führt, während die anderen Mitglieder Fragen stellen und Kommentare geben zu Technik, Stil, möglichen Fehlern, Verletzung von Entwicklungsstandards und anderen Problemfeldern [IEE83].

Zusammenfassend kann man festhalten, dass die drei Verfahren in der hier aufgeführten Reihenfolge zunehmend informeller bzw. abgeschwächter werden.

### 5.3.2 Qualität konzeptioneller Schemata

Das Anwenden einer bestimmten Sprache bzw. Notation gewährleistet eine gewisse Qualität im Hinblick auf eine konstruktive Qualitätssicherung, allerdings sind bei Anwendung einer Sprache bzw. Notation immer noch unterschiedliche Interpretationen möglich. Dieses wird manchmal als „Modellierungswillkür“ bezeichnet [MS94]. Aus diesem Grunde kann zusätzlich eine analytische Qualitätssicherung in Form eines Reviews durchgeführt werden, die für ein Schema wünschenswerte Qualitätseigenschaften sichert. Zum Thema Qualität konzeptioneller Schemata existieren einige Ansätze (einen Überblick geben [Bur98, Sch99]), die im Folgenden kurz skizziert werden.

- In der bekannten Arbeit [BCN92] nennen die Autoren eine Reihe von Kriterien, die ein *hochqualitatives* Datenbankschema erfüllen muss. Es werden jedoch keine Metriken oder Messverfahren angegeben. Jedes genannte Kriterium (Vollständigkeit, Korrektheit, Minimalität, Ausdrucksstärke, Lesbarkeit, Selbsterklärbarkeit, Erweiterbarkeit und Normalisierung) wird informal beschrieben und ein erläuterndes Beispiel gegeben.
- In [Ris92] und [Ris93] werden die Ideen aus [BCN92] aufgegriffen und teilweise verfeinert.
- Die Veröffentlichungen [Bru91] und [RG94] nennen spezielle Regeln, um relationale Schemata zu entwerfen, die bestimmte Qualitätseigenschaften aufweisen. Der Fokus dieser Arbeiten ist die „Datenbankpraxis“, häufig auftretende Situationen sind untersucht und aus diesen einige konstruktive Vorgehensweisen abgeleitet worden.
- [MS94] definiert Qualitätsaspekte, die mit Hilfe einer Metrik gemessen und bewertet werden können. Um Schemata zu vergleichen, können die gemessenen Werte gewichtet werden. Ziel ist es hierbei, unter verschiedenen Entwurfsalternativen die beste Lösung im Sinne der gestellten Anforderungen zu finden. Konkrete Qualitätsaspekte sind Einfachheit, Vollständigkeit, Flexibilität, Integration, Verständlichkeit und Implementierbarkeit.
- [CM00] nennt einige Qualitätskriterien für DB–Schemata und fordert ein explizites Review, allerdings sind die genannten Ideen auf das E/R–Modell beschränkt.

- Im Projekt „Grundsätze ordnungsmäßiger Modellierung“ (GoM) [BRS95, BES98] wurden eine Reihe von notwendigen (Richtigkeit, Relevanz und Wirtschaftlichkeit) und ergänzenden (Klarheit, Vergleichbarkeit und systematischer Aufbau) Modellierungsgrundsätzen aufgestellt. Ziel war jedoch keine konkrete Schemabewertung im Sinne einer Messung, sondern eher die Erstellung eines allgemeinen Leitfadens zum Gestalten „guter“ Schemata.

Tabelle 5.1 führt nochmals die genannten Arbeiten auf und bewertet sie anhand der Kriterien[6]:

- Nennt die Arbeit eine Reihe von *Qualitätskriterien* für Datenschemata?
- Werden in der Arbeit *Metriken* und/oder *Messverfahren* zur Bewertung der Qualitätskriterien angegeben?
- Sieht der Ansatz ein explizites *Review* vor, in dem eine projektspezifische Bewertung des Schemas vorgenommen werden kann?
- Gibt es ein *Werkzeug*, das das vorgeschlagene Konzept implementiert?
- Welche *Datenmodelle* werden unterstützt?

| | **Qualitäts–kriterien** | **Messung/ Metriken** | **Reviews** | **Werk–zeug** | **Daten–modell** |
|---|---|---|---|---|---|
| **Bruce, Reingruber und Gregory ([Bru91, RG94])** | + | - | - | - | ER |
| **Batani, Ceri und Navathe ([BCN92])** | + | - | - | - | ER |
| **Rishe ([Ris92, Ris93])** | + | - | - | + | ER |
| **Moody/Shanks ([MS94])** | + | + | (+) | - | ER |
| **Becker, Rosemann und Schütte ([BRS95, BES98])** | + | - | - | (+) | ER[7] |

Tabelle 5.1: Vergleich einiger Arbeiten zur Qualität von Schemata

### 5.3.3 Präemptive Ansätze zum physischen Datenbankentwurf

In der Praxis werden Optimierungsmaßnahmen physischer Schemata häufig nach „Faustregeln“ vorgenommen, die der Hersteller des eingesetzten DBMS vorgibt oder die dem Erfahrungsschatz der

[6] Die Zeichen „+“ und „-“ geben nur an, ob der Aspekt behandelt worden ist oder nicht; sie spiegeln keine inhaltliche Wertung wider.

[7] Die aufgestellten Prinzipien sind nicht nur auf die Daten–, sondern auch auf die Prozesssicht bezogen; insbesondere werden auch EPKs (Ereignisorientierte Prozessketten) betrachtet.

Entwickler entstammen. Dieses führt häufig zu (lokal) guten Realisierungen, jedoch sind die Implementierungen für Dritte schwer nachvollziehbar und damit nicht sehr wartungsfreundlich. Bei auftretenden Performanzproblemen zieht dies hohe Kosten in Form aufwändiger Optimierungsmaßnahmen, leistungsfähigerer Hardware oder gar dem Re–Design des Systems nach sich. Dieser Abschnitt nennt einige Ansätze, die einen *präemptiven Weg* verfolgen, indem von einem Schema, einem zu erwartenden Zugriffsverhalten auf die DB und einigen Randbedingungen ausgehend Optimierungsmöglichkeiten ermittelt werden.

#### Das System DBDSGN

In [FST88] werden die am IBM San Jose Research Laboratory[8] entwickelten Konzepte der Implementierung von DBDSGN, einem Werkzeug zum physischen Entwurf relationaler DBen vorgestellt. Ausgehend von einem aus einer Menge von SQL–Anweisungen und deren Ausführungshäufigkeit bestehendem Workload für System R[9], schlägt DBDSGN physische Konfigurationen für eine performante Realisierung vor. Jede Konfiguration besteht aus einer Menge von Indizes und einer Sortierreihenfolge für jede Tabelle.

#### Arbeit von Rozen und Shasha

In [RS91] wird ein zweiphasiger Algorithmus für den physischen DB–Entwurf vorgestellt. Im ersten, als *Auswahlphase* bezeichneten Schritt wird für eine Menge von Anfragen an die DB aufgrund von vorher definierten Regeln eine Menge von Entwurfsentscheidungen wie z. B. Indizes bestimmt, die sich als vorteilhaft für die Ausführung der Anfragen erweisen. Daran schließt sich als zweiter Schritt die sog. *Kompromissphase* an, in der eine Teilmenge der im ersten Schritt ermittelten physischen Entwurfsentscheidungen selektiert wird, wobei die Kosten der Menge gewichteter Anfragen minimiert werden soll.

#### Performance Engineering

In den letzten Jahren hat sich innerhalb des Software Engineering mit dem sog. *Performance Engineering* [RS99, SS00] eine neue Teildisziplin entwickelt, deren Kernidee darin besteht, die Performanz eines Anwendungssystems bereits in den frühen Phasen der Softwareentwicklung zu berücksichtigen. Damit soll die Entwicklung von Softwaresystemen ermöglicht werden, die unter Angabe eines definierten Ressourcenverbrauchs und eines Lastmodells geforderte Leistungsattribute künftiger Anwender erfüllen können.

### 5.3.4 Erweiterungsmechanismen der UML

Bei Vorstellung der Entwurfsmethodik in Teil II der Arbeit kommt die UML (Unified Modeling Language) als Beschreibungsmittel zum Einsatz. Zum Verständnis notwendige grundlegende Kenntnisse der UML werden beim Leser vorausgesetzt[10], die eventuell nicht so weit verbreiteten *Erweiterungsmechanismen* werden in diesem Abschnitt kurz eingeführt.
Die UML versteht sich als offener Standard, was die Kombination mit bzw. Integration von weiteren

[8] Heute IBM Almaden Research Center.

[9] System R ist ein in den siebziger Jahren ebenfalls am IBM San Jose Research Laboratory entwickeltes DBS. System R realisierte SQL und zeigte darüber hinaus, dass relationale Systeme gute Performanz bez. Transaktionsverarbeitung bieten können.

[10] Oder können in z. B. [Alh98] nachgelesen werden.

Sprachen und Modellen betrifft. So kann die UML benutzerdefiniert erweitert bzw. modifiziert werden. Dabei stehen die beiden Mechanismen *Stereotypen* und *Elementeigenschaften* (*standardisierte Annotationen, Tagged Values*) zur Verfügung. Elementeigenschaften bestehen aus einem Schlüsselwort (dem *Tag*) und einem dazugehörigen Wert (dem *Value*). Neben einer Reihe von vordefinierten Schlüsselworten kann der Benutzer zusätzliche definieren. So besitzt z. B. das Modellierungskonstrukt „Klasse“ die Schlüsselworte „location“ und „persistence“, als Notation wäre beispielsweise

**class** Kunde {**location** = host
**persistence** = persistent}

möglich. Denkbare, selbstdefinierte Elementeigenschaften sind in diesem Zusammenhang „Ersteller“ oder „Erstellungsdatum“.
Ein *Stereotyp* ist ein Mechanismus zur benutzerdefinierten Erweiterung, Präzisierung oder Redefinition von Elementen der UML, wobei das Metamodell der Sprache nicht verändert wird. Als Notation für Stereotypen wird der Name des Stereotyps umschlossen von den Zeichenfolgen „«“ bzw. „»“ verwendet. Insgesamt werden vier Arten von Stereotypen unterschieden [Gli00]:

- *Dekorative Stereotypen* variieren die äußere Darstellung (Notation) der Sprache, womit benutzerdefinierte Symbole geschaffen werden.
- *Deskriptive Stereotypen* erweitern eine Sprache rein syntaktisch, wodurch zu den standardisierten Annotationen analoge Annotationen oder Sekundärklassifikationen geschaffen werden.
- *Restriktive Stereotypen* präzisieren bestehende oder schaffen neue Sprachkonstrukte. Anwendungen dieser Stereotypenklasse sind das Hinzufügen neuer Sprachkonstrukte oder das Überlagern bestehender Konstrukte mit zusätzlichen Bedeutungen.
- *Redefinierende Stereotypen* verändern die Bedeutung von Sprachkonstrukten, wodurch die Definition einer neuen Sprache auf der Grundlage einer bestehenden Sprache möglich ist.

## 5.4 Zusammenfassung

In diesem Kapitel wurden in Abschnitt 5.1 zunächst Grundbegriffe aus dem Entwurf von Informationssystemen eingeführt. Anschließend erfolgte in Abschnitt 5.2 die Vorstellung von Arbeiten mit DWH–Bezug. Diese behandeln entweder den gesamten Entwurfsprozess oder beziehen sich nur auf einen bestimmten Aspekt wie z. B. die Gewinnung des konzeptionellen Schemas. Auffallend ist, dass mehrere Ansätze eine Konstruktion des (initialen) konzeptionellen DWH–Schemas aus dem Schema der operativen Datenquelle(n) betrachten. Diese Vorgehensweise bringt aber drei wesentliche Probleme mit sich: Zum einen wird nur eine andere (nämlich analyseorientierte) Betrachtungsweise der Daten aus den operativen Systemen vorgenommen. Stattdessen sollte die konzeptionelle Modellierung unabhängig von jeglichen Restriktionen, zu denen insbesondere auch die Datenquellen zählen, durchgeführt werden. Ein zweites Problem sind fehlende Konzepte bei bzw. die Ausklammerung von sich überlappenden Quelldaten. Als drittes und letztes Manko verlangen die meisten der konstruierenden Ansätze bei der Ableitung eines analyseorientierten Schemas ein E/R–Schema als Vorgabe. Es kann in der Praxis jedoch nicht davon ausgegangen werden, dass von allen Datenquellen, insbesondere den externen, ein Schema in E/R–Notation vorliegt. Darüber hinaus kann den bestehenden Ansätzen bescheinigt werden, dass sie zwischen logischem und physischem Entwurf nicht exakt trennen bzw. den physischen Entwurf gar nicht betrachten, obwohl dieser gerade in einem DWH große Bedeutung besitzt. Ein Überblick über kommerzielle Werkzeuge kam schließlich zu dem Urteil, dass diese sich durch Verwendung der E/R–Notation schon frühzeitig auf ein Datenmodell festlegen bzw.

dass die DBMS–Hersteller–spezifischen Werkzeuge lediglich auf ihr eigenes System ausgelegt sind. Abschnitt 5.3 beschäftigte sich mit weiteren Aspekten des Software Engineering und insbesondere des DB–Entwurfs, die in Teil II der Arbeit eine Bedeutung haben werden.

# Teil II

# Entwurfsmethodik für Data Warehouses

# Überblick

Nachdem in Teil I neben einer Motivation für die Arbeit einige Grundbegriffe eingeführt und der State of the Art skizziert wurde, folgt in Teil II die Vorstellung der konzipierten Entwurfsmethodik. Diese Einleitung gibt einen Überblick über die wesentlichen Entwurfsentscheidungen der Methodik und ihren generellen Ablauf. Dabei wird jeweils auf die nachfolgenden Kapitel der Arbeit verwiesen, in denen die einzelnen Phasen im Detail behandelt werden.

## Entwurfskriterien

Der Methodik liegt der aus dem Entwurf konventioneller Datenbanken bekannte *Drei–Ebenen–Entwurf* zugrunde (siehe Abschnitt 5.1). Ergänzend sollen besondere *Aspekte eines DWH* berücksichtigt werden:

- Während der konzeptionellen Modellierung soll das multidimensionale Datenmodell Anwendung finden.
- Die Transformation in ein logisches, relationales Datenmodell erzeugt ein analyseorientiertes Schema.
- Existierende DBMS oder OLAP–Server erfordern häufig eine bestimmte physische Speicherungsform der Daten, z. B. Sternschemata, so dass eine entsprechende Umstrukturierung des Schemas möglich sein muss.
- Für analyseorientierte Datenbanken sind physische Optimierungsmaßnahmen wie Materialisierungen und Indexierungen von großer Bedeutung, so dass ein Framework zur integrierten Betrachtung dieser Aspekte entworfen werden soll.

Neben diesen DWH–spezifischen Gesichtspunkten liegen der Entwurfsmethodik einige *allgemeine Aspekte* zugrunde:

- Während der konzeptionellen Modellierung soll das objektorientierte Datenmodell Berücksichtigung finden, weil es aufgrund seiner Ausdrucksstärke eine natürliche Darstellung der Diskurswelt ermöglicht und auch gute Basis für eine implementationsunabhängige Modellierung ist.
- Nach Beendigung der konzeptionellen Modellierung soll der Entwurf teilautomatisiert fortgeführt werden. Dieses Vorgehen kombiniert die Vorteile des automatischen Entwurfs für wiederkehrende Vorgänge mit der Berücksichtigung von Benutzerinteraktionen an ausgewählten Stellen, an denen Kontextwissen des Designers nutzbringend eingebracht werden kann. Dieses Wissen kann sich auf das verwendete System beziehen und somit technischer Natur sein oder Domänenwissen betreffen.
- Weil aufgrund dieser Vorgehensweise die Phase der konzeptionellen Modellierung zentrale Bedeutung hat, wird sie um eine explizite Qualitätssicherung des erstellten Schemas sowie einen Leitfaden zur Gewinnung konzeptioneller Schemata ergänzt.
- Der Entwurfsprozess soll gut nachvollziehbar sein, indem alle Transformationsschritte und durch Benutzerinteraktion getroffene Entscheidungen dokumentiert werden.

- „Endprodukt“ des Entwurfsprozesses soll die Spezifikation der Implementierung sein, aus der ein automatisch ausführbares Datenbankskript erzeugt oder über eine Programmierschnittstelle das DB–Schema angelegt werden kann.

Unter Berücksichtigung dieser Aspekte ergibt sich der in Abbildung II.1 dargestellte Entwurfsprozess. Jeder Kasten stellt dabei einen Entwurfsschritt dar, die Zahlen in Klammern verweisen auf das Kapitel, in dem der Schritt behandelt wird, die kursiven Beschriftungen nennen den Beschreibungsformalismus des Entwurfsdokuments des abgeschlossenen Schrittes. Fettgedruckt am rechten Rand sind die drei Entwurfsebenen genannt.

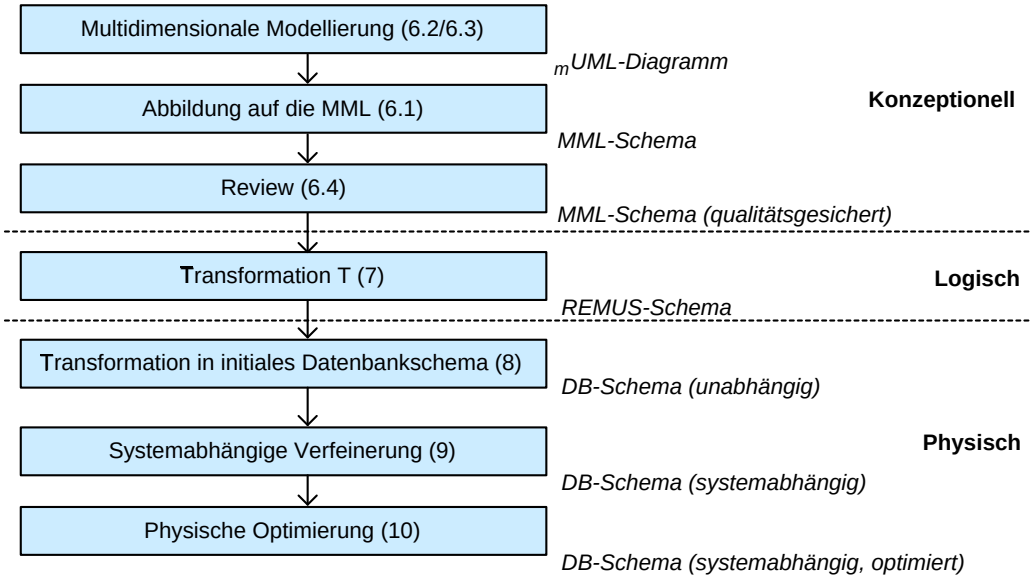



Abbildung II.1: Ablauf des Entwurfprozesses

Kapitel 6 widmet sich der konzeptionellen Entwurfsphase. Es wird zunächst in Abschnitt 6.1 mit der MML (Multidimensional Modeling Language) eine Metasprache eingeführt, die sowohl multidimensionale wie objektorientierte Aspekte enthält. Mit der mUML (multidimensional UML) wird in Abschnitt 6.2 eine dazugehörige graphische Notation vorgeschlagen. Wie mit Hilfe dieser Notation ein Schema erstellt werden kann, wird in Abschnitt 6.3 behandelt, in dem ein Leitfaden zur Gewinnung eines MML–Schemas präsentiert wird. Qualitätssicherung in Form eines Reviews sowie die Angabe zahlreicher Qualitätskriterien sind Gegenstand des Abschnitts 6.4.
Die Transformation von der konzeptionellen auf die logische Entwurfsebene wird in Kapitel 7 behandelt. Hierzu wird in Abschnitt 7.1 die Relationenschemaform REMUS (Relational Schema for Multidimensional Purpose) definiert, in 7.2 erfolgt dann die Spezifikation der eigentlichen Transformation.
In den Kapiteln 8 bis 10 wird der physische Datenbankentwurf behandelt. Nachdem zunächst in Kapitel 8 die Überführung von REMUS–Schemata in LCD (Lowest Common Denominator) of SQL–Schemata beschrieben wird, behandelt Kapitel 9 die Zielsystem–spezifische Umstrukturierung dieses Schemas.
Zum physischen Entwurf im engeren Sinne, der Optimierung des Schemas, wird in Kapitel 10 ein Framework für die integrierte Handhabung verschiedener Optimierungsmöglichkeiten unter Berücksichtigung von Umgebungs– und Extensionsparametern entworfen.
Die Präsentation der Konzepte wird von einem durchgängigen Beispiel „Handelswelt“ begleitet, dessen Vorstellung beim ersten Auftreten in Abschnitt 6.5 erfolgt.

# Kapitel 6

# Konzeptioneller Entwurf

Dieses Kapitel widmet sich dem konzeptionellen Entwurf. Dabei werden die in Abbildung 6.1 dunkel hinterlegten Schritte betrachtet. Zunächst wird in Abschnitt 6.1 mit der *Multidimensional Modeling Language* (MML) eine multidimensionale Sprache für die konzeptionelle Datenmodellierung von DWH vorgestellt. Anschließend wird in Abschnitt 6.2 mit der $_m$UML (multidimensional UML) eine graphische Notation für diese Sprache eingeführt. Als konstruktive Vorgehensweise, um mit Hilfe dieser Sprache bzw. Notation zu einem Schema zu gelangen, wird in Abschnitt 6.3 ein Leitfaden zum Erstellen von MML–Schemata vorgeschlagen. Ein Framework für die analytische Qualitätssicherung eines auf diese Weise erstellten Schemas ist Gegenstand von Abschnitt 6.4.
Abgerundet wird das Kapitel schließlich in Abschnitt 6.5 mit dem Beispiel „Handelswelt“, das für Teil II dieser Arbeit als durchgängiges Beispiel dient.

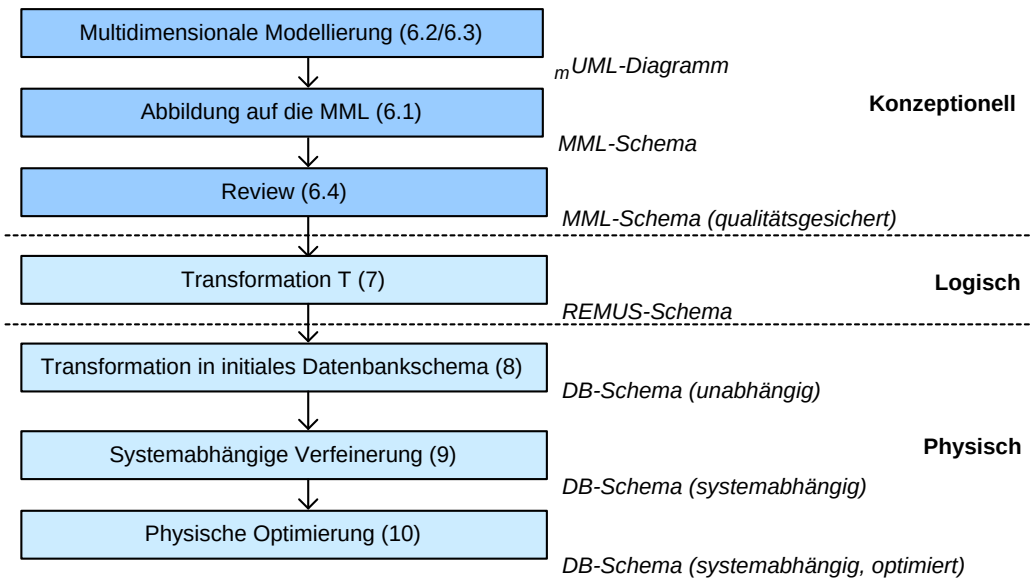



Abbildung 6.1: Einordnung des Schrittes in den Entwurfsprozess

## 6.1 MML: Multidimensional Modeling Language

Die in diesem Abschnitt eingeführte MML [Har99b, Har99a, HH99] ist eine multidimensionale Sprache für die konzeptionelle DWH–Datenmodellierung, die sich an den Anforderungen für ein konzep-

tionelles Datenschema aus Abschnitt 3.2 orientiert. Unter Einfluss der Ergebnisse des Abschnitts 3.4 lassen sich die Eigenschaften der MML wie folgt charakterisieren[1]:

- Die MML ist eine *objektorientierte Sprache*, d. h. sie kennt Konzepte wie Klasse, Vererbung etc. Dies bildet eine gute Basis für Erweiterbarkeit und Wiederverwendbarkeit mit ihr erstellter Schemata.
- Für die adäquate Beschreibung multidimensionaler Daten erfolgt in der MML die Unterscheidung *qualifizierender* (Dimensionen) und *quantifizierender* (Fakten) Datenobjekte.
- *Fakten* sind nicht auf ein einzelnes numerisches Datenelemente beschränkt, d. h. sie können eine innere Struktur mit verschiedenen Datenobjekten und weiteren Fakten aufweisen.
- Bei der Gestaltung von *Dimensionsstrukturen* bildet ein gerichteter, azyklischer Graph die Grundlage für die Modellierung, so dass dimensionale Klassen in verschiedenen Dimensionen genutzt und somit die Datenzusammenhänge angemessen und implemetationsunabhängig dargestellt werden können; ebenso sind auf diese Art und Weise alternative Verdichtungspfade darstellbar.
- Ist ein Datenobjekt durch eine Menge von Eigenschaften eindeutig identifiziert oder entsteht es aus anderen Datenobjekten durch die Anwendung einer *Berechnungsvorschrift*, so können auch diese Informationen explizit im Schema festgehalten werden.
- Die Anwendbarkeit von Verdichtungsoperatoren (*sum*, *avg*, *count* etc.) auf quantifizierenden Datenobjekten ist nicht nur von dem zugrundeliegenden Datentyp (z. B. nicht summierbare Textfelder) abhängig, sondern ergibt sich oftmals erst aus der Kombination des Datentyps mit der Bedeutung des Datenobjektes und den zugeordneten Dimensionen. Jedes Datenobjekt, das eine quantifizierende Eigenschaft darstellt, lässt sich in der MML mit den in bezug auf eine Dimension unterstützten Verdichtungsoperatoren markieren, so dass eine flexible Freigabemöglichkeit existiert[2]. Im Gegensatz zu Berechnungsvorschriften für abgeleitete Datenobjekte erfolgt in einem MML–Schema keine genaue Angabe, wie die Ausführung eines Verdichtungsoperators auf einem Datenobjekt definiert ist.
- Als Sprache für die konzeptionelle Modellierung ist die MML unabhängig von der logischen Entwurfsebene.

### 6.1.1 Begriffsbildung und Namenskonventionen

Um möglichen Begriffsverwirrungen vorzubeugen, erfolgt vor der detaillierten Vorstellung der MML eine Begriffsbildung bez. der Unterscheidung von MML–Konstrukten und Objekten eines MML–Datenschemas. Die Ursache möglicher Begriffsverwirrungen ergibt sich aus der Mehrfachverwendung des Begriffs *Klasse*. Einerseits stellt die MML objektorientierte Konzepte wie Klassen, Vererbung und Polymorphie für die multidimensionale Datenmodellierung zur Verfügung, andererseits wird die MML–Syntax selbst durch ein Klassendiagramm beschrieben. Aus Sicht der MML–Syntax sind daher die in einem Datenschema enthaltenen Klassen Instanzen von MML–Sprachelementen, also Objekte bestimmter MML–Klassen. In bezug auf die Datenmodellierung kann

[1] Im Gegensatz zur Originalarbeit der MML in [Har99b] ist in der hier beschriebenen Version der MML die Möglichkeit zum Modellieren von Schemaevolutionen ausgelassen, weil diese ausserhalb des Rahmens der Arbeit liegen. Verändert wurde die Funktion beim *SharedRollUp*–Operator, neu hinzugekommen ist das Konstrukt *DimensionalMapping*. Diese beiden Modifikationen begründen sich in einer grösseren Flexibilität der Modellierung.

[2] In der MML wird hierfür der Begriff *additivity* (=*Additivität*) gewählt, der im engeren Sinne nur die Anwendbarkeit des *sum*–Operators bezeichnet. Diese Wahl erfolgte, weil die Summierung in analytischen Applikationen die vorwiegend angewendete Verdichtungsoperation ist.

es sich jedoch in einem Datenschema um Klassen handeln, deren Objekte später die konkreten Daten sind. Um dieser Begriffsvermischung entgegenzuwirken, gelten im Folgenden die in Abbildung 6.2 aufgeführten Begriffe: MML–Klassen werden als (MML–)Metaklassen bezeichnet, Instanzen dieser Metaklassen als (Metaklassen–)Instanz, Schemaelement oder Klasse. Die konkreten Daten schließlich werden als (Schemaelement–)Instanz bzw. (Schemaelement–)Objekt bezeichnet.
Mit dieser Vereinbarung lassen sich Instanzen der Metaklassen gemäß der Datenmodellierungssicht weiterhin als Klassen bezeichnen. Darüber hinaus ist zu beachten, dass Instanzen einiger MML–Metaklassen nicht automatisch Klassen im Sinne der Objektorientierung sind, da diese Klasseneigenschaft durch eine spezielle Metaklasse bereitgestellt wird und explizit durch eine MML–Metaklasse geerbt werden muss. Daher werden Instanzen von Metaklassen auch allgemein als Schemaelemente bezeichnet.

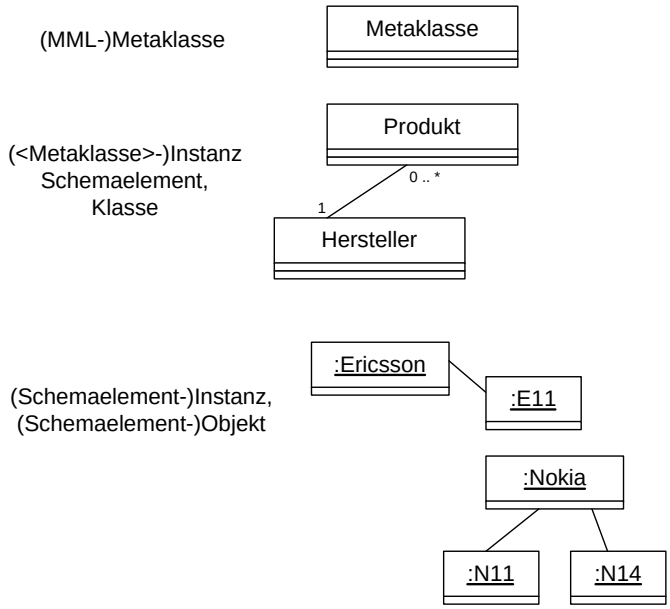



Abbildung 6.2: Begriffsbildung Metaklasse, Klasse, Objekt

Bezeichnungen für Klassen, Attribute etc. sind stets in Englisch gewählt. In der Wahl der Groß– bzw. Kleinschreibung wurde der UML–Spezifikation [Rat99a] gefolgt:

- Klassennamen und Datentypen beginnen mit einem Großbuchstaben, danach wird in Kleinbuchstaben fortgefahren, bei zusammengesetzten Begriffen bzw. einem wesentlichen weiteren Teil wird der Beginn dieses zweiten Wortteils durch Verwendung eines Großbuchstabens verdeutlicht.

- Attributnamen und Rollenbezeichnungen in Beziehungen beginnen mit einem Kleinbuchstaben, für zusammengesetzte Begriffe bzw. wesentliche weitere Wortteile gilt die gleiche Regel wie bei Klassennamen und Datentypen.

- Konstantennamen werden komplett in Großbuchstaben geschrieben.

- Bei allen Bezeichnungen werden nur Buchstaben und Ziffern verwendet, auf jegliche Art von Sonderzeichen wird verzichtet.

### 6.1.2 Das MML–Metaklassendiagramm

Bevor ab Abschnitt 6.1.3 eine detaillierte Beschreibung der einzelnen MML–Metaklassen erfolgt, soll an dieser Stelle ein Überblick über das gesamte MML–Metaklassendiagramm und seine wichtigsten Bereiche gegeben werden. Zur Verdeutlichung sind dazu zwei Darstellungen gewählt: während in Abbildung 6.4 das gesamte Metaklassendiagramm mit allen Verbindungen und Attributen dargestellt ist, zeigt Abbildung 6.3 die Vererbungshierarchie, wobei zur besseren Übersicht auf die Darstellung von Attributen und nicht generalisierenden Beziehungen verzichtet wurde. Die Metaklassen sind zu Bereichen zusammengefasst, wobei in der Abbildung jeder Bereich hinterlegt ist.

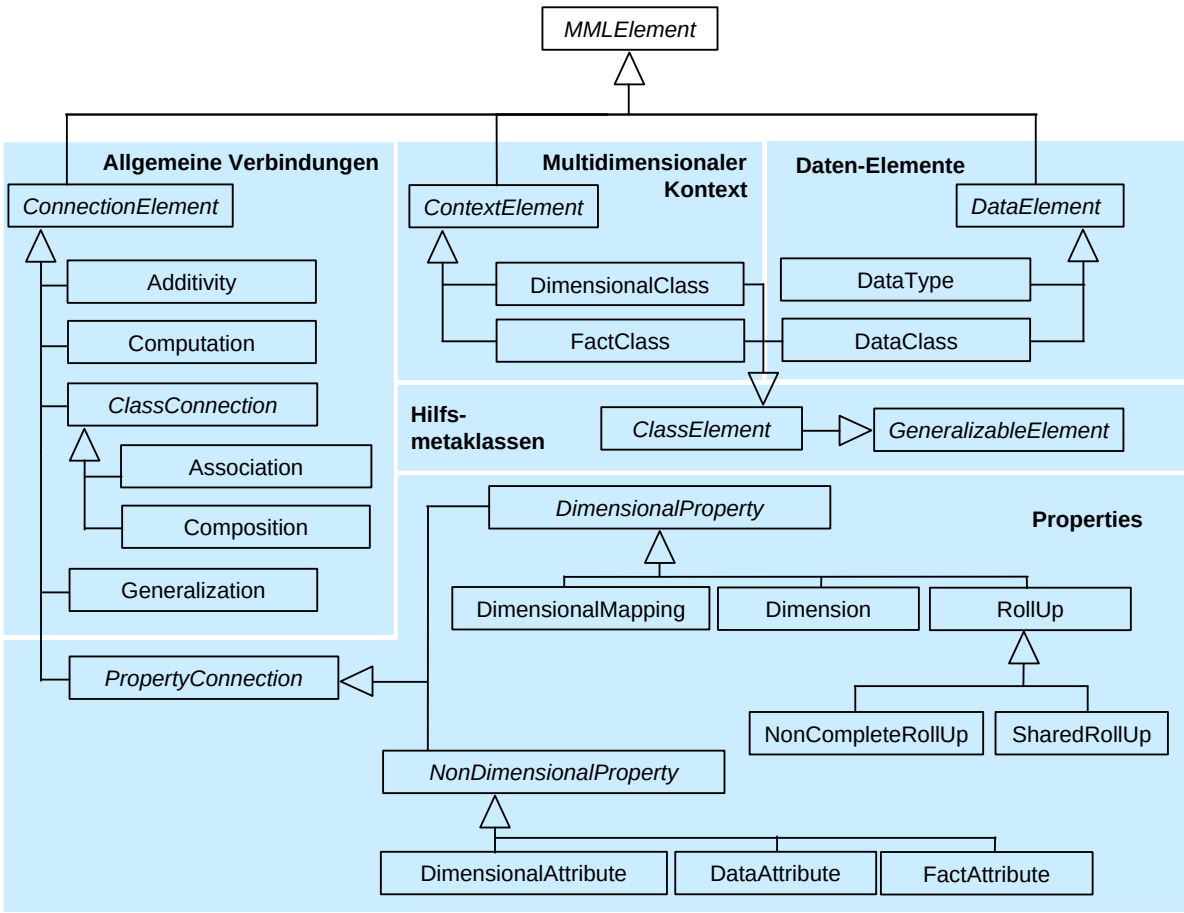



Abbildung 6.3: Vererbungshierarchie des MML–Metaklassendiagramms

Hauptcharakteristikum der MML ist die Differenzierung zwischen Daten, dem zugehörigen multidimensionalen Kontext und Elementen für die Beschreibung von Struktureigenschaften. Ursprung eines jeden MML–Konstrukts ist die abstrakte *MMLElement*–Metaklasse (siehe Abbildung 6.3), die jeder Klasse eine eindeutige Bezeichnung zuordnet. Das *MMLElement* bildet somit auch den Ursprung für die drei Bereiche *Allgemeine Verbindungen*, *Multidimensionaler Kontext* und *Daten–Elemente* mit ihren Hauptmetaklassen *ConnectionElement*, *ContextElement* und *DataElement*. Der Bereich *Daten–Elemente* beschreibt allgemein datenaufnehmende Komponenten eines MML–Schemas und entspricht damit einem Datentyp, wobei es sich um elementare (Metaklasse *DataType*) oder komplexe Typen (Metaklasse *DataClass*) handeln kann. Mit den im Bereich Hilfsmetaklassen aufgeführten Elementen wird der Begriff der Klasse (Metaklasse *ClassElement*) mit seiner Eigenschaft der Vererbbarkeit (Metaklasse *GeneralizableElement*) in die MML eingeführt.

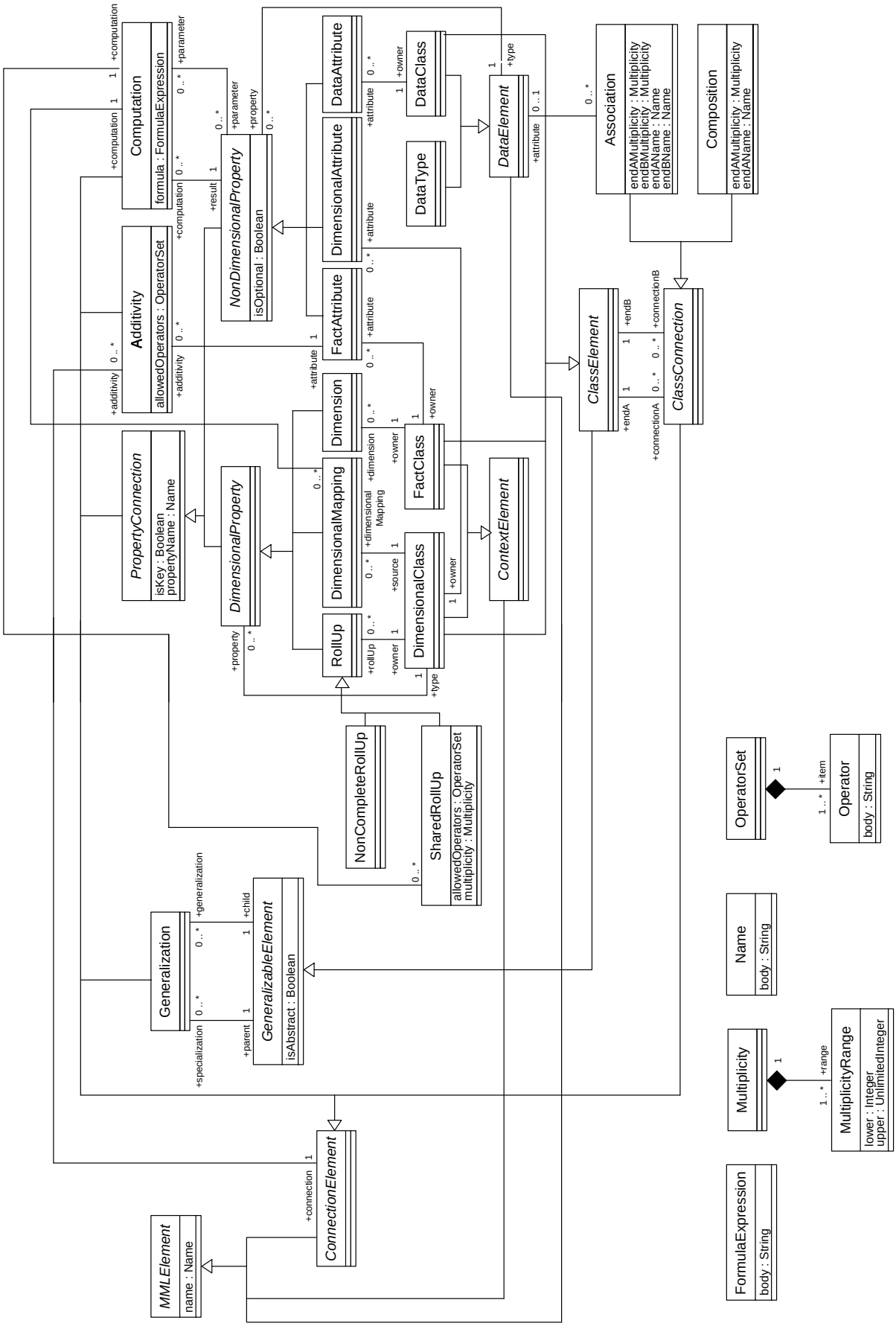



Abbildung 6.4: MML–Metaklassendiagramm

Im Bereich *Multidimensionaler Kontext* werden mit *FactClass* und *DimensionalClass* die für die multidimensionale Modellierung charakteristische Unterscheidung qualifizierender und quantifizierender Daten vorgenommen. Diese beiden Metaklassen sowie die komplexe Datentypen anbietende Metaklasse *DataClass* erben von der gemeinsamen Oberklasse *ClassElement* die Eigenschaft der Klasse und somit die Fähigkeit, Vererbungen aufbauen zu können.
Die Verknüpfungen zwischen gleich- oder verschiedenartigen Schemaelementen werden durch Instanzen von Unterklassen der *ConnectionElement*–Metaklasse realisiert. Hierbei wird nochmals zwischen *Allgemeinen Verbindungen* und auf den multidimensionalen Kontext ausgerichteten *Properties* unterschieden[3]. Während die *Allgemeinen Verbindungen* typische objektorientierte Konstrukte wie z. B. Generalisierung, Assoziation und Komposition zur Verfügung stellen, dienen die im Bereich *Properties* definierten Metaklassen der Darstellung multidimensionaler Sachverhalte, z. B. dem Aufbau von Hierarchiestrukturen.
Abbildung 6.4 zeigt neben den eigentlichen Metaklassen auch die zugehörigen Metadatentypen *FormulaExpression*, *Multiplicity*, *Name* und *OperatorSet*. Während *FormulaExpression* und *Name* lediglich Zeichenketten darstellen, enthält eine *Multiplicity*–Instanz eine Menge verschiedener Multiplizitätsangaben der Form *(min,max)*, die als „min .. max“ geschrieben werden. Die maximale Multiplizität darf auch unbeschränkt sein und wird dann durch das Symbol „*“ dargestellt. *OperatorSet* enthält eine Menge von Verdichtungsoperatoren.

### 6.1.3 Wurzelelement und Hilfsmetaklassen

Die in Abbildung 6.5 grau hinterlegten Metaklassen sind das Wurzelelement und die Hilfsmetaklassen.

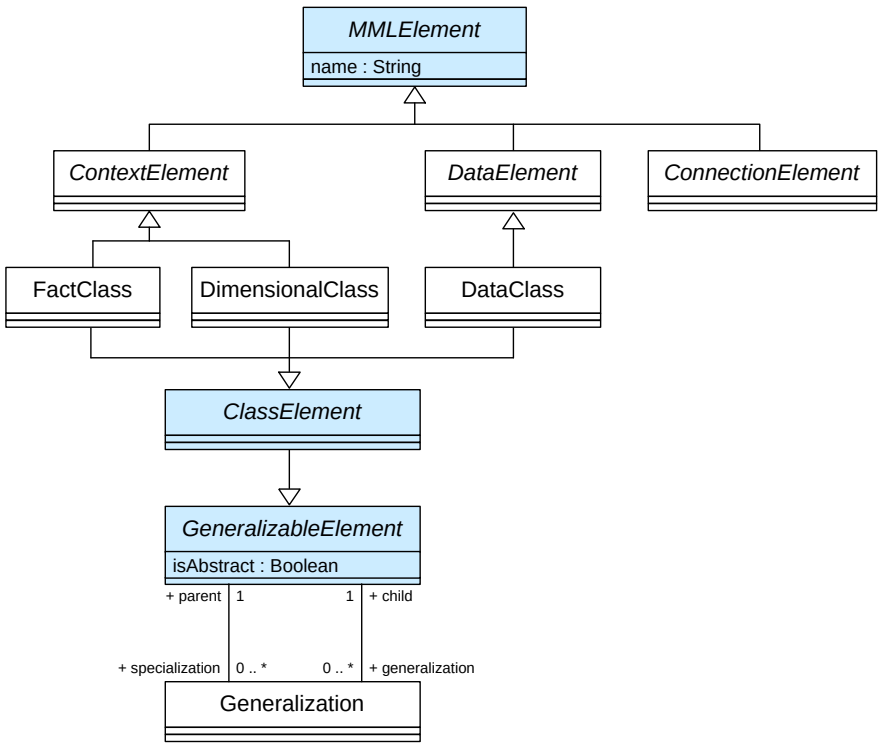



Abbildung 6.5: Wurzelelement und Hilfsmetaklassen *GeneralizableElement* und *ClassElement*

[3]Verbindung meint an dieser Stelle die Verbindung zwischen zwei MML–Instanzen unabhängig von ihrer späteren Darstellung. d. h. ein Verbindungselement muss im späteren Schema nicht unbedingt eine Kante sein.

Jede andere MML–Metaklasse erbt vom Wurzelelement *MMLElement* und erhält so das Attribut *name*, das der Klasse einen eindeutigen Namen gibt. In manchen Fällen besitzt eine Untermetaklasse ein zusätzliches Namensattribut, um unterschiedlichen Typen von Schemaelementen gleiche Namen geben zu können. Die beiden Hilfsmetaklassen *GeneralizableElement* und *ClassElement* werden für die vereinfachte Spezifikation von Beziehungen und Eigenschaften von MML–Metaklassen eingeführt.
Während der Datenmodellierung können Situationen auftreten, in denen die Zusammenfassung gleicher Eigenschaften von verschiedenen Schemaelementen die Modellierung vereinfachen und übersichtlicher gestalten. Diese gemeinsamen Eigenschaften können dann mit Hilfe eines verallgemeinerten Schemaelementes modelliert werden und sind für alle untergeordneten Schemaelemente gültig. Für die Erstellung dieser abstrahierenden Schemaelemente wird das aus der Objektorientierung bekannte Konzept der *Vererbung* übernommen, das als Abstraktionsmittel die *Generalisierung* für die Zusammenfassung von Eigenschaften in einem gemeinsamen Basisschemaelement und die *Spezialisierung* zur Modellierung von Unterschemaelementen mit ergänzenden Eigenschaften bereitstellt. Jede MML–Metaklasse, deren Instanzen Spezialisierungen bzw. Generalisierungen unterstützen sollen, muss eine Untermetaklasse der *GeneralizableElement*–Metaklasse sein. Nur bei Erfüllung dieser Voraussetzung kann in einem MML–Schema das Generalisierungskonstrukt *Generalization* verwendet werden.
Zur Unterscheidung zwischen künstlich eingeführten und in der zu modellierenden Diskurswelt „tatsächlich“ existierenden Basisschemaelementen besitzt die *GeneralizableElement*–Metaklasse das Attribut *isAbstract*. Ist dieses Attribut gesetzt, so dürfen zur betreffenden Klasse keine Objekte existieren. Als Nebenbedingung muss an dieser Stelle gelten, dass eine abstrakte Klasse nicht Blattelement innerhalb einer Vererbungshierarchie sein darf. Die beiden Referenzen *generalization* und *specialization* verweisen auf Instanzen der *Generalization*–Metaklasse, die das Schemaelement entsprechend generalisieren bzw. spezialisieren. Der Mechanismus der *Polymorphie* ermöglicht es, in einem MML–Schema an Stellen, an denen ein Basisschemaelement verwendet wurde, auf Instanzebene Datenobjekte der Unterschemaelemente zu nutzen. Die Metaklasse *ClassElement* dient der Unterstützung generalisierbarer Schemaelemente und führt damit das objektorientierte Konzept der *Klasse* in die MML ein. Sie dient zusätzlich der vereinfachten Definition der in Abschnitt 6.1.6 beschriebenen Metaklassen *Association* und *Composition* für die Verbindung von Schemaelementen.

### 6.1.4 Multidimensionaler Kontext

In Abbildung 6.6 werden mit der *FactClass* und der *DimensionalClass* die beiden wichtigsten Metaklassen der MML spezifiziert. Sie sorgen für die Einstufung von Daten als quantifizierend oder qualifizierend und bilden somit den multidimensionalen Kontext innerhalb der MML. Als gemeinsame Basismetaklasse ist das *ContextElement* definiert, das jedoch nur der Zusammenfassung dient und keine eigenen Eigenschaften aufweist.
Die Metaklasse *DimensionalClass* beschreibt Klassen, die innerhalb einer Dimension angeordnet sind, und entspricht dem Begriff *Hierarchieebene* der multidimensionalen Begriffswelt. Durch die *rollUp*– und *property*–Referenzen besteht die Möglichkeit, innerhalb einer Menge von *DimensionalClass*–Schemaelementen HierarchiepfadeVerdichtung!–spfad festzulegen. Die Referenz *dimensionalMapping* zur gleichnamigen Metaklasse ermöglicht die Abbildung von Instanzen einer Hierarchieebene auf Instanzen einer anderen Hierarchieebene, die außerhalb der gleichen Hierarchie liegt[4]. Die charakterisierenden Eigenschaften einer *DimensionalClass*–Instanz werden durch Attribute beschrieben, was sich in der MML durch die Referenz *attribute* zur Metaklasse *DimensionalAttribute* ausdrückt.

[4]Die Beschreibung von *DimensionalMapping* folgt auf Seite 85

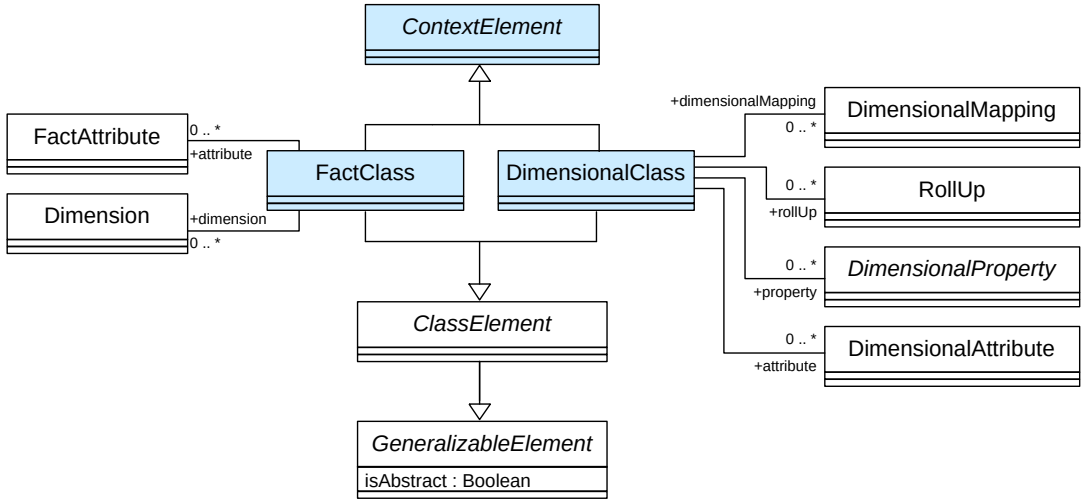



Abbildung 6.6: MML: Multidimensionaler Kontext

Als Untermetaklasse von *ClassElement* und damit transitiv auch Untermetaklasse der *GeneralizableElement*–Metaklasse wird die Erstellung von Unterklassen ermöglicht, so dass innerhalb einer Hierarchieebene verschiedene Objekttypen darstellbar sind. Die Unterklassen können ergänzende Eigenschaften wie z. B. *RollUp*–Beziehungen besitzen, die durch Instanzen der *RollUp*–Metaklasse dargestellt werden.
Fakten werden in der MML durch Instanzen der *FactClass*–Metaklasse dargestellt. Diese besitzt die Möglichkeit der Aufnahme von Beziehungen zu Datenelementen, die die *nicht–dimensionalen* Eigenschaften eines Faktes darstellen. Diese Beziehungen werden durch Instanzen der *FactAttribute*–Metaklasse hergestellt. In Ergänzung dieser *quantifizierenden* Eigenschaften kann jede *FactClass*–Instanz durch zugeordnete *DimensionalClass*–Schemaelemente qualifizierend beschrieben werden, was durch die *dimension*–Referenz zur gleichnamigen Metaklasse spezifiziert wird.

### 6.1.5 Datenelemente

Die in diesem Unterabschnitt vorgestellten Metaklassen dienen der Definition einfacher und komplexer Datentypen. Ihre Einordnung in das MML–Metaklassendiagramm ist in Abbildung 6.7 zu sehen.
Die *DataElement*–Metaklasse ist die Zusammenfassung der *DataClass*– und *DataType*–Metaklasse.
Jede *DataElement*–Instanz kann in verschiedenen Instanzen der *NonDimensionalProperty*–Metaklasse, die die Attribute von *FactClass*–, *DimensionalClass*– und *DataClass*–Schemaelementen repräsentieren, als (Daten–)Typ eingetragen werden.
Die *DataClass*–Metaklasse stellt ein *komplexes Datenelement* dar und erbt durch die Basismetaklasse *ClassElement* die Eigenschaften zum Aufbau komplex geschachtelter Datentypen. Im Gegensatz zur *DataClass*–Metaklasse werden mit Hilfe der *DataType*–MetaklasseDataType (MML)*DataType* (MML) *elementare Datentypen* festgelegt, so dass keine weiteren Strukturierungsmittel, wie beispielsweise Generalisierungen, erlaubt sind. Das von der *MMLElement*–Basismetaklasse geerbte *name*–Attribut enthält den Namen des Datentyps. Durch die Beziehung zur *Association*–Metaklasse kann einer Assoziation zwischen zwei *MMLElement*–Instanzen eine *DataElement*–Instanz für die Speicherung weiterer, das *Association*–Objekt ergänzender Daten genutzt werden.

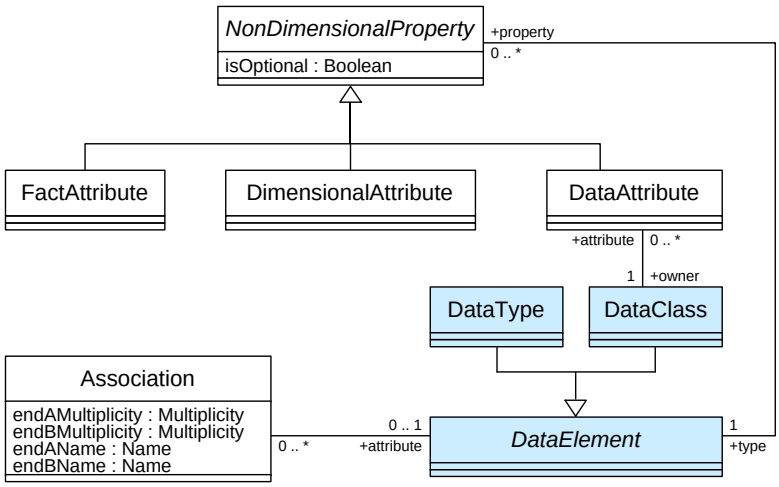



Abbildung 6.7: MML: Datenelemente

### 6.1.6 Allgemeine Verbindungen

Die *ConnectionElement*–Metaklasse bildet den Ausgangspunkt für alle Verbindungsmöglichkeiten zwischen Schemaelementen in einem MML–Schema. Der für diesen Abschnitt relevante Teil des MML–Metaklassendiagramms ist in Abbildung 6.8 zu sehen.

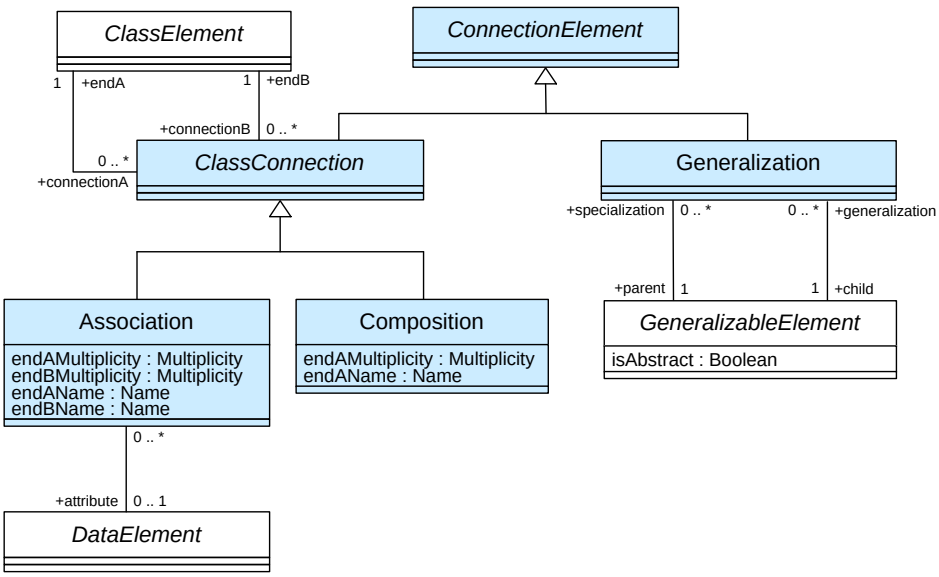



Abbildung 6.8: MML: Allgemeine Verbindungen

Die *ClassConnection*–Metaklasse legt fest, welche Beziehungstypen zwischen *ClassElement*–Instanzen erlaubt sind. Als Basismetaklasse für die *Association*– und die *Composition*–Metaklasse bietet sie zudem eine einheitliche Schnittstelle für den Zugriff auf vorhandene Beziehungen dieser Art einer *ClassElement*–Instanz. Auch wird hierdurch die Ergänzung der MML um neue Beziehungstypen bzgl. der *ClassElement*–Metaklasse erleichtert. *ClassElement* hat keine Attribute und besitzt die beiden Referenzen *endA* und *endB*, die auf die beiden in Beziehung stehenden *ClassElement*–Schemaelemente verweisen. Die Bedeutung der Enden wird durch die *ClassConnection*–Metaklasse nicht festgelegt.

Die allgemeinste Art von Beziehung zwischen zwei *ClassElement*–Instanzen wird durch die *Association*–Metaklasse bereitgestellt. Die Attribute der *Association*–Metaklasse erlauben neben Rollenangabe der beteiligten Schemaelemente in dieser Beziehung auch die Speicherung von Multiplizitätsangaben. Die Multiplizitäten dienen zur Einschränkung der minimalen und maximalen Anzahl in Beziehung stehender Objekte. Weitere, die Assoziation beschreibende Attribute werden über die *attribute*–Beziehung zur Metaklasse *DataElement* dargestellt.
Objektorientierte Modellierungssprachen wie UML bieten neben der Assoziation mit der Aggregation einen weiteren Beziehungstyp an. Die Aggregation verleiht einer Beziehung zwischen Klassen die Bedeutung einer *Teil–Ganzes*–Zugehörigkeit, entspricht jedoch prinzipiell einer Assoziation mit spezieller Multiplizität (*Eins*–zu–*Viele*) [Rat99b] und kann in der MML durch das *Association*–Konstrukt dargestellt werden. Eine besondere Unterform der Aggregation ist die *Komposition*, die neben der Zugehörigkeit zusätzlich die Existenzabhängigkeit der *Teil*– von der *Ganzes*–Klasse beinhaltet. Eine Instanz der *Teil*–Klasse kann daher nicht ohne ein zugeordnetes *Ganzes*–Objekt existieren. Dieses Konstrukt wird in der MML durch die *Composition*–Metaklasse bereitgestellt, die — als Untermetaklasse der *ClassConnection*–Metaklasse — Kompositionsverbindungen zwischen zwei *ClassElement*–Instanzen derselben Metaklasse herstellt.
Mit den beiden Attributen der *Composition*–Metaklasse lassen sich Rolle und Multiplizität der Klasse auf der *Teil*–Seite der Beziehung spezifizieren. Als Nebenbedingung ist zu berücksichtigen, dass die an den (geerbten) Beziehungen *endA* und *endB* beteiligten Schemaelemente Instanzen derselben Metaklasse sein müssen, d. h. es darf keine Kompositionen zwischen einer *FactClass*– und einer *DimensionalClass*–Instanz geben. Weiterhin dürfen keine zwei *DimensionalClass*–Instanzen über eine Komposition verbunden sein. Diese Restriktion ist darin begründet, dass Kompositionen zwischen *DimensionalClass*–Instanzen eine Hierarchie bilden. Hierfür aber stehen die speziellen *RollUp*– bzw. *NonCompleteRollUp*–Konstrukte zur Verfügung (siehe Seite 85). Das Schemaelement der Beziehung *endB* ist Besitzer des über *endA* verbundenen Schemaelements, so dass die Existenz der Objekte in *endA* durch *endB* bestimmt sind. Generalisierungsbeziehungen zwischen Instanzen der bereits vorgestellten *GeneralizableElement*–Hilfsmetaklasse werden mittels der *Generalization*–Metaklasse modelliert, die jeweils zwei Metaklasseninstanzen miteinander verbindet und über die *parent*– bzw. *child*–Referenz die Rolle als Basis– oder Unterklasse zuordnet. Eine Generalisierungsbeziehung kann nur zwischen zwei Schemaelementen derselben Untermetaklasse von *GeneralizableElement* etabliert werden, beispielsweise dürfen FactClass–Instanzen nur von anderen *FactClass*–Instanzen erben. Durch die in Abbildung 6.9 abgebildete *Computation*–Metaklasse werden in der MML Berechnungen unterstützt. Sie können für *abgeleitete Attribute* sowie *SharedRollUp*– (siehe Seite 86) und *DimensionalMapping*–Konstrukte (siehe Seite 85) verwendet werden.

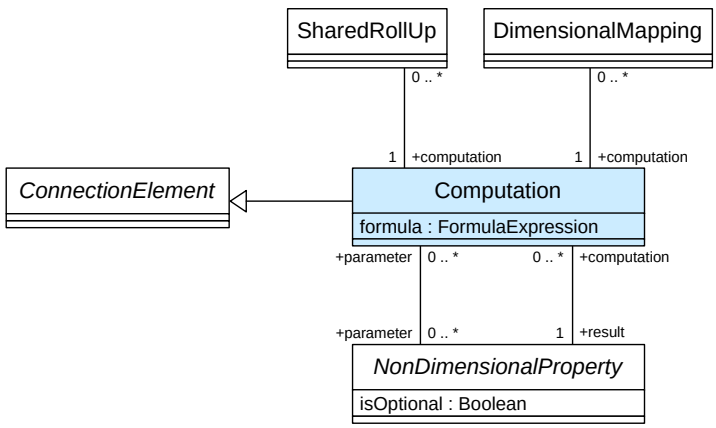



Abbildung 6.9: MML: *Computation*

Ein *Computation*–Schemaelement beinhaltet über die beiden Referenzen zur *NonDimensionalProperty* Informationen über die benötigten Parameter und das resultierende Attribut. Das Attribut *formula* spezifiziert die zu nutzende Berechnungsvorschrift. Das genaue Format der Formel ist nicht festgelegt, im Sinne einer völlig systemunabhängigen konzeptionellen Modellierung kann dies auch eine kurze verbale Beschreibung sein. Um eine *Computation*–Instanz für die Berechnung eines *SharedRollUp* oder eines *DimensionalMapping* zu verwenden, dienen die Referenzen auf die entsprechenden Metaklassen. An die Verwendung der Referenzen sind folgende Nebenbedingungen geknüpft:

- Sofern es sich um ein berechnetes Attribut handelt, d. h. das *Computation*–Objekt wird von keinem *SharedRollUp*– und keinem *DimensionalMapping*–Schemaelement refrenziert, müssen die *owner*–Beziehungen der als Parameter und Ergebnis beteiligten *NonDimensionalProperty*–Instanz auf dasselbe Schemaelement verweisen, d. h. sie müssen Attribute der gleichen Klasse sein.
- Im Falle der Verwendung für eine *SharedRollUp*–Instanz müssen die Parameter mit ihrer *owner*–Referenz auf den *Owner* des *SharedRollUp* und das Resultat muss auf das gleiche Element wie die *property*–Referenz des *SharedRollUp* verweisen.
- Wird das Objekt für ein *DimensionalMapping*–Schemaelement verwendet, müssen die Parameter mit dessen *source*–Referenz übereinstimmen, und das Resultat muss auf das gleiche Element verweisen wie die *property*–Referenz des *DimensionalMapping.*

Mit Hilfe von Instanzen der *Additivity*–Metaklasse können in einem MML–Schema für bestimmte *FactAttribute*–Instanzen und *ConnectionElement*–Schemaelemente anwendbare *Verdichtungsoperatoren* definiert werden. Fehlt bei einem *FactAttribute*–Schemaelement eine zugehörige *Additivity*–Instanz, so sind per Definition alle Operatoren erlaubt. Über die Art, wie ein Operator auf den durch die *FactAttribute*–Instanz verbundenen Datenelementen ausgeführt wird, erfolgt in einem MML–Schema keine Aussage.

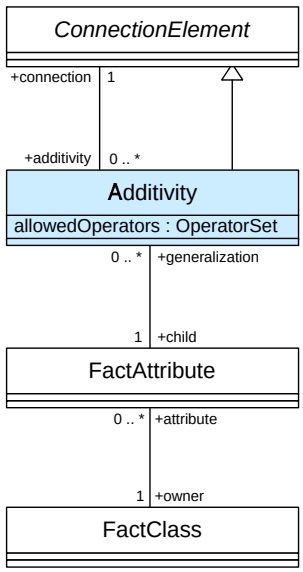



Abbildung 6.10: MML: *Additivity*

Die Angabe der Verdichtungsoperatoren ist auf *Dimension*–Beziehungen zwischen *FactClass* und *DimensionalClass*–Instanzen sowie auf Kompositionsbeziehungen zwischen zwei *FactClass*–Instanzen beschränkt.

### 6.1.7 Properties

Im Gegensatz zu den im letzten Unterabschnitt vorgestellten allgemeinen Verbindungsmöglichkeiten zwischen *MMLElement*–Instanzen ist die *PropertyConnection*–Metaklasse für Eigenschaften zuständig, die in direktem Zusammenhang mit der *multidimensionalen Sichtweise* stehen. Wie in Abbildung 6.11 zu sehen, ist sie Oberklasse der bereits erwähnten *NonDimensionalProperty*–Metaklasse und der für qualifizierende Eigenschaften bestimmten *DimensionalProperty*–Metaklasse.

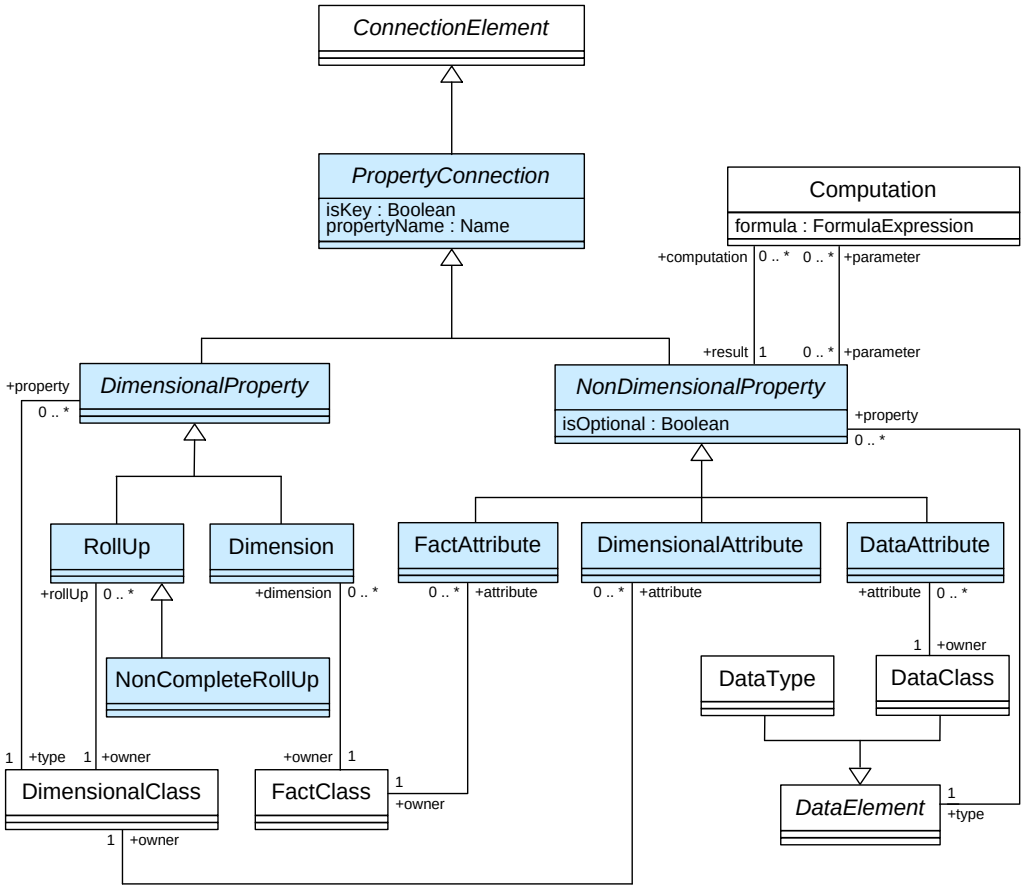



Abbildung 6.11: MML: *PropertyConnection*

Neben dem ein Schemaelement eindeutig identifizierenden, von *MMLElement* geerbten *name*–Attribut besitzt die *PropertyConnection*–Metaklasse ein zusätzliches *propertyName*–Attribut für die Aufnahme einer Beschreibung der repräsentierten Eigenschaft. Diese über alle *PropertyConnection*–Instanzen nicht eindeutige Beschreibung ermöglicht die Verwendung desselben Namens für unterschiedliche Typen von Schemaelementen, d. h. es darf in einem Schema z. B. eine *Dimension* und ein *FactAttribute* geben, die den gleichen Namen tragen, aber alle *Dimension*–Schemaobjekte müssen disjunkte Namen haben. Durch Setzen des *isKey*–Attributes auf den Wert „TRUE“ kann das die Eigenschaft besitzende Schemaelement als Identifikator markiert werden.
Für die Modellierung *nicht–dimensionaler* Eigenschaften von *ContextElement*– und *DataClass*–Instanzen bildet die *NonDimensionalProperty*–Metaklasse den Ausgangspunkt. Eine Instanz dieser Metaklasse steht immer in Verbindung mit einer *DataElement*–Instanz, die der nicht–dimensionalen Eigenschaft bzw. dem Attribut als (Daten–)Typ zugeordnet ist. Das boolesche Attribut *isOptional* gibt an, ob die dargestellte Eigenschaft optional sein darf. Dient ein *NonDimensionalProperty*–

Schemaelement als Parameter bzw. ist Resultat einer Berechnung, so wird dies durch die Beziehungen zur Metaklasse *Computation* modelliert. Die Metaklassen *DataAttribute*, *DimensionalAttribute* und *FactAttribute* sind als Untermetaklassen der *NonDimensionalProperty*–Metaklasse definiert und beschreiben *Attribute* von *DataClass*–, *DimensionalClass*– bzw. *FactClass*–Instanzen, die nicht dem Aufspannen des multidimensionalen Raumes dienen. Bei *DimensionalClass*–Schemaelementen entsprechen die Attribute *charakterisierenden* Eigenschaften, da sie im Gegensatz *RollUp*– bzw. *NonCompleteRollUp*–Verbindungen zusätzliche, beschreibende Daten enthalten. Für die *FactClass*–Schemaelemente sind Attribute hingegen die eigentlichen Komponenten zur Aufnahme *quantifizierender* Daten, d. h. die Kennzahlen. Jede dieser drei Metaklassen besitzt eine als *owner* bezeichnete Beziehung zur entsprechenden Metaklasse. Darüber hinaus hat die *FactAttribute*–Metaklasse eine Referenz auf *Additivity*.
Eine Instanz der *DimensionalProperty*–Metaklasse repräsentiert eine qualifizierende Eigenschaft. Sie ist immer mit genau einem *DimensionalClass*–Schemaelement verbunden, das entweder als übergeordnete Hierarchieebene einer anderen *DimensionalClass* oder als beschreibende Eigenschaft einer *FactClass*–Instanz angesehen werden kann. Die Interpretation unterliegt der jeweiligen Untermetaklasse. Die *DimensionalProperty*–Metaklasse besitzt kein Attribut zur Angabe einer Multiplizität für Objekte des über die *type*–Beziehung verbundenen *DimensionalClass*–Schemaelementes, so dass mit einer *DimensionalProperty*–Instanz immer genau eine *DimensionalClass*–Instanz verbunden sein muss. Die *DimensionalProperty*–Metaklasse vererbt an die Untermetaklassen *Dimension*, *RollUp* und *DimensionlMapping.* Instanzen der *Dimension*–Metaklasse verbinden eine *FactClass*–Instanz mit einer *DimensionalClass*–Instanz, was durch die entsprechenden Referenzen dargestellt wird. Semantisch wird durch ein solches Konstrukt der *FactClass*–Instanz eine in bezug auf die Multidimensionalität klassifizierende Bedeutung zugeordnet. Eine *Dimension*–Instanz bildet somit den Einstiegspunkt in eine über *DimensionalClass*–Instanzen definierte hierarchische Dimensionsstruktur. Mittels einer *Dimension*–Instanz kann ein Objekt der *DimensionalClass*–Instanz mit verschiedenen Objekten derselben *FactClass*–Instanz in Verbindung stehen, so dass implizit von einer Multiplizität der Form „0..*“ für die verbundene *FactClass*–Instanz ausgegangen wird. Instanzen der *RollUp*– bzw. *NonCompleteRollUp*–Metaklasse verbindet zwei *DimensionalClass*–Schemaelemente, wodurch Verdichtungspfade und hierarchische Strukturen innerhalb einer Dimension aufgebaut werden können.
Anzumerken ist an dieser Stelle, dass *RollUp*– und *Dimension*–Beziehungen besonderen *1–zu–Viele*–Assoziationen entsprechen, bei denen die mittels *type* referenzierte *DimensionalClass*–Instanz der *Eins*–Seite entspricht. Eine *NonCompleteRollUp*–Beziehung hingegen hat implizit die Multiplizität *0 .. 1–zu–Viele.* Dies bedeutet, dass bei einer Verdichtung einige Objekte der niedrigeren Hierarchieebene herausfallen; ab der höheren Ebene sind dann weitere Verdichtungen nur noch mit dieser *eingeschränkten* Ausgangsdatenmenge möglich.
Die bisher vorgestellten Modellierungskonstrukte gestatten lediglich die Bildung von Hierarchien innerhalb einer Dimension und die gemeinsame Nutzung dieser Strukturen durch unterschiedliche *FactClass*–Schemaelemente. Es gibt aber auch Fälle, in denen zwei mit verschiedenen *FactClass*–Instanzen verbundene Dimensionen semantisch Gleiches beschreiben, aber die Bildung einer gemeinsamen Hierarchieebene in Form einer *DimensionalClass*–Instanz nicht möglich ist. Dies kann z. B. bei zwei Ortsdimensionen der Fall sein: Die eine beschreibt Filialen und darauf aufbauende geschäftliche Hierarchien, die andere bildet Verwaltungseinheiten in Form einer Hierarchie Gemeinde–Landkreis–Bezirk ab. Um die Daten dennoch vergleichen zu können, fließt das in [Her99] vorgeschlagene *DimensionalMapping* in Form eines Modellierungskonstruktes in die MML ein. Ein *DimensionalMapping*–Schemaelement verweist über die *computation*–Referenz auf eine *Computation*–Instanz, welche die Berechnung der Abbildung beschreibt. Die Einordnung ins Metaklassendiagramm zeigt Abbildung 6.12.

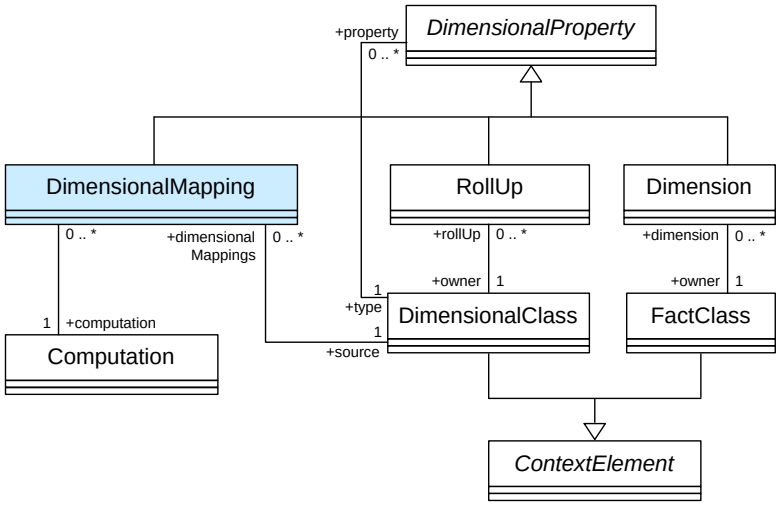



Abbildung 6.12: MML: *DimensionalMapping*

Als Spezialisierung einfacher *RollUp*–Beziehungen bietet die MML mit dem *SharedRollUp*–Konstrukt (siehe Abildung 6.13) eine Modellierungsmöglichkeit zur Darstellung *anteiliger Verrechnungen*. Anteilige Verrechnungen ermöglichen die Verknüpfung eines *DimensionalClass*–Objektes — unter Berücksichtigung einer Berechnungsvorschrift — zu mehreren Objekten einer übergeordneten Hierarchieebene.

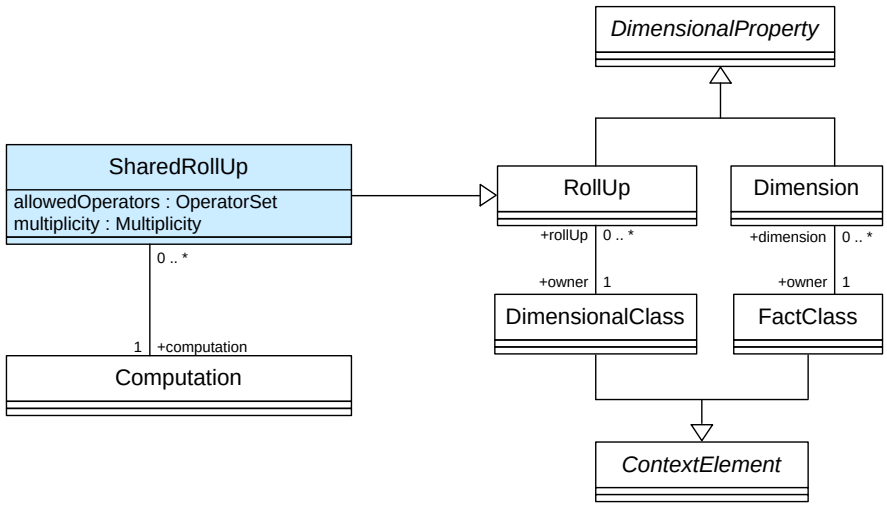



Abbildung 6.13: MML: *SharedRollUp*

Neben der Angabe zugehöriger Multiplizitäten auf Seiten der übergeordneten *DimensionalClass*–Instanz (der höheren Hierarchieebene) können die bei Nutzung des *SharedRollUp*–Pfades erlaubten Verdichtungsoperatoren angegeben werden. Operatoren, die ein fehlerhaftes Verhalten aufweisen würden, sollten auf diesem Weg von der Benutzung ausgeschlossen werden.

### 6.1.8 Wohlgeformtheitseigenschaften

Neben den im MML–Metamodell festgehaltenen Eigenschaften, z. B. der, dass Instanzen der Metaklasse *Dimension* nur zwischen Instanzen einer *FactClass* und *DimensionalClass* bestehen dürfen, werden in diesem Abschnitt weitere Regeln festgelegt, deren Einhaltung für ein *wohlgeformtes* MML–Schema notwendig ist. Teilweise werden hierbei bereits im letzten Abschnitt genannte oder durch das MML–Metaklassendiagramm sichergestellte Nebenbedingungen erneut aufgegriffen[5]. Die bereits durch das Metaklassendiagramm definierten Regeln sind durch ein „M“ gekennzeichnet.

**Allgemeines**

- (WF:A1) Jede nicht–abstrakte *DimensionalClass*–, *FactClass*– und *DataClass*–Instanz muss entweder selber mindestens ein Attribut haben oder durch Vererbung mindestens ein Attribut besitzen.
- (WF:A2) Jedes Attribut muss einen Typ haben.

**Verbindungstypen**

- (WF:VT1) Vererbungen dürfen nur zwischen Klassen gleichen Typs definiert werden.
- (WF:VT2) Kompositionen dürfen nur zwischen Klassen gleichen Typs definiert werden.
- (WF:VT3) Instanzen der Metaklasse *DimensionalClass* dürfen keine Kompositionen eingehen.
- (WF:VT4) Instanzen der Metaklasse *DataClass* dürfen keine Assoziationen eingehen.
- (WF:VT5) Wenn zwischen zwei Klassen mehr als eine Beziehung besteht, dann darf höchstens eine eine Nicht–Assoziation sein.
- (WF:VT6) Besteht zwischen zwei *DimensionalClass*–Instanzen ein *DimensionalMapping*, so dürfen die Instanzen nicht einer durch *RollUp*–, *NonCompleteRollUp*– oder *SharedRollUp*–Schemaelemente gebildeten Hierarchie angehören.
- (WF:VT7) *Association*–Schemaelemente dürfen nur zwischen zwei *DimensionalClass*–Instanzen oder zwischen einer *FactClass*–Instanz und einer *DimensionalClass*–Instanz verwendet werden. Im ersten Fall schränken sie den multidimensionalen Raum ein, im zweiten dienen sie der Konsistenzsicherung.

Tabelle 6.1 fasst nochmals die in einem MML–Schema zulässigen Verbindungen zusammen.

| | **DataClass** | **DimensionalClass** | **FactClass** |
|---|---|---|---|
| **DataClass** | Generalization<br>Composition | | |
| **DimensionalClass** | | Association<br>Generalization<br>RollUp (M)<br>NonCompleteRollUp (M)<br>SharedRollUp (M)<br>DimensionalMapping | Association |
| **FactClass** | | Association<br>Dimension (M) | Generalization<br>Composition |

Tabelle 6.1: MML: Erlaubte Verbindungstypen

[5] Diese Vorgehensweise begründet sich darin, um z. B. im Abschnitt *Verbindungstypen* eine Übersicht über alle möglichen Verbindungen zu haben.

**Zyklenfreiheit**

- (WF:ZF1) Bei Benutzung von *DataClass*–Instanzen als Attributtyp in anderen *Data Class*–Instanzen (Aufbau von geschachtelten Verbunden) muss diese Verwendung zyklenfrei sein.
- (WF:ZF2) Kompositionshierarchien dürfen keine Zyklen enthalten.
- (WF:ZF3) Vererbungshierarchien dürfen keine Zyklen enthalten.
- (WF:ZF4) Durch *RollUp*–, *NonCompleteRollUp*– und *SharedRollUp*–Instanzen aufgebaute Hierarchien von *DimensionalClass*–Instanzen dürfen keine Zyklen enthalten.

**Mehrfachvererbung**

- (WF:MV1) Mehrfachvererbungen sind erlaubt, gleichbenannte Attribute müssen aber auch das semantisch gleiche beschreiben, damit keine Namenskonflikte auftreten können.
- (WF:MV2) Erbt die Instanz einer *DimensionalClass*–Metaklasse mehrfach, so dürfen die Eltern in keiner *RollUp*–, *NonCompleteRollUp*– und *SharedRollUp*–Beziehung zueinander stehen (auch in keiner geerbten).
- (WF:MV3) Erbt die Instanz einer *FactClass*–Metaklasse mehrfach, so dürfen die Eltern in keiner *Composition*–Beziehung zueinander stehen (auch in keiner geerbten).

Auf die technische Realisierung zur Einhaltung der Regeln soll an dieser Stelle nicht eingegangen werden. Zur Entdeckung von Mehrfachvererbungen in Klassendiagrammen sei z. B. auf [FGM97, FGM98, BFM99] verwiesen, zur Entdeckung von Zyklen(freiheit) auf Arbeiten aus der Graphentheorie [Tur96, YG98].

## 6.2 $_m$UML : Graphische Notation

Mit der MML als Kern der konzeptionellen Entwurfsebene lassen sich verschiedene graphische Notationen verwenden. Dieses Konzept wurde gewählt, um in einer Organisation bereits etablierte Notationen weiterhin zu verwenden bzw. geeignet zu ergänzen oder in spezifischen Projekten spezielle Notationen verwenden zu können. Im Rahmen dieser Arbeit soll als graphische Notation die $_m$UML [Har99a, Har99b, HH99] dienen.
Ein $_m$UML–Diagramm wird mittels des *Klassendiagramms* (*static structure diagram*) der UML dargestellt und beinhaltet daher die statischen Eigenschaften von Klassen und Objekten wie beispielsweise Attribut- und Methodenangaben oder Beziehungen. Der weitere Aufbau dieses Abschnitts basiert dementsprechend auf den durch die UML bereitgestellen Modellierungskonstrukten für *Klassen*, *Attribute* und *Verbindungen*, ergänzt um den Bereich *Verdichtungsoperatoren*.

### 6.2.1 Klassen

Mittels des Klassenkonstruktes werden Instanzen der drei Metaklassen *DataClass*, *DimensionalClass* und *FactClass*FactClass ($_m$UML )@*FactClass* ($_m$UML ) modelliert, da diese die Untermetaklassen der Metaklasse *ClassElement* sind und daher innerhalb der MML den Notationselementen für verschiedenartige Klassen entsprechen. Für die Zugehörigkeit einer Klasse zu einer der drei Metaklassen werden Stereotypen definiert, die den Namen der *ClassElement*–Untermetaklasse tragen (siehe Abbildung 6.14). Klassen mit abweichendem oder fehlendem Stereotyp sind innerhalb eines

$_m$UML-Diagramms nicht erlaubt.

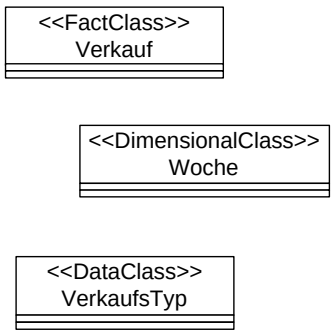



Abbildung 6.14: $_m$UML: Darstellung von Klassen

Die *ClassElement*–Metaklasse ist in der MML von der *GeneralizableElement*–Metaklasse abgeleitet und erbt dadurch die Eigenschaft zur Beschreibung eines abstrakten Schemaelements. Für Klassen wird durch die UML ebenfalls diese Abstraktionseigenschaft angeboten, so dass sie in der $_m$UML für die Darstellung der *isAbstract*–Eigenschaft verwendet wird. Abstrakte Klassen werden durch einen *kursiv* geschriebenen Klassennamen markiert. Zu beachten ist jedoch die durch die MML festgelegte Einschränkung bei der Definition abstrakter Schemaelemente: Besitzt ein Schemaelement keine Spezialisierungen (d. h. ist Blatt einer Vererbungshierarchie), so darf es nicht abstrakt sein.

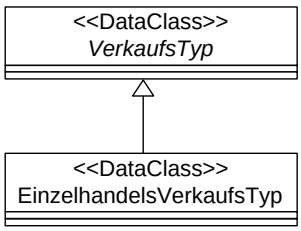



Abbildung 6.15: $_m$UML: Darstellung von abstrakten Klassen und Vererbung zwischen Klassen

Verhaltensbezogene Aspekte von Objekten bzw. Klassen, wie sie in der Objektorientierung betrachet werden, finden in der MML–Spezifikation keine Berücksichtigung, so dass der Methodenabschnitt einer $_m$UML–Klasse keine Angaben enthält.

### 6.2.2 Attribute

In der in Abschnitt 6.1 vorgestellten MML–Spezifikation werden die Attribute der drei Klassentypen *DataClass*, *DimensionalClass* und *FactClass* mittels zugehöriger Instanzen der *DataAttribute*–, *DimensionalAttribute*– bzw. *FactAttribute*–Metaklasse modelliert. In der UML werden Attribute in einem besonderen Abschnitt des Klassensymbols notiert, so dass keine weitere Kennzeichnung zur Unterscheidung des Typs eines Attributes notwendig ist: Attribute, die bei einer Klasse mit dem Stereotyp *FactClass* eingetragen sind, sind automatisch Instanzen der *FactAttribute*–Metaklasse, analoges gilt natürlich auch für *DimensionalClass* und *DataClass*. Der Attributname entspricht dabei der zugehörigen *propertyName*–Eigenschaft, die die *FactAttribute*–Metaklasse von der *PropertyConnection*–Metaklasse erbt. Abbildung 6.16 verdeutlicht diese Notation.


Abbildung 6.16: $_m$UML: Darstellung von Attributen

Der allgemeine Aufbau eines Attributes gemäß der UML–Notation [Rat97b] bildet den Ausgangspunkt für die Definition der $_m$UML–Besonderheiten:

Sichtbarkeit Attributname [Multiplizität] : Typ = Standardwert

Wie bereits erwähnt, wird der Attributname auf die *propertyName*–Eigenschaft abgebildet, die der Bezeichnung eines Attributs innerhalb der MML dient. Eine Multiplizitätsangabe der Form „0..1“ wird für die Darstellung der *isOptional*–Eigenschaft der *NonDimensionalProperty*–Metaklasse verwendet. Da die MML für Attribute keine allgemeinen Multiplizitätsangaben vorsieht, kommt diesem Attributzusatz keine andere Bedeutung zu[6]. Zudem entspricht diese Art der Kennzeichnung eines optionalen Attributes der üblichen UML–Notation. Der in der Definition angegebene Attributdatentyp *Typ* bezieht sich auf die *type*–Beziehung der *NonDimensionalProperty*– zur *DataElement*–Metaklasse, die einem Attribut eine *DataElement*–Instanz als Datentyp zuordnet. Gültige Werte für den *Typ*–Abschnitt sind folglich Namen von *DataClass*– und *DataType*–Instanzen innerhalb des $_m$UML–Diagramms. Während *DataClass*–Instanzen in einem $_m$UML–Diagramm als Klassen dargestellt werden, existiert für die *DataType*–Metaklasse keine vergleichbare Darstellungsform. *DataType*–Instanzen werden in einem $_m$UML–Diagramm nicht explizit modelliert, sondern sie ergeben sich aus den in Attributen referenzierten Typen. Jeder Datentyp, der in einem Attribut genutzt wird und nicht einer im Diagramm enthaltenen Klasse entspricht, ist per Definition eine Instanz der *DataType*–Metaklasse.
Die Attributsichtbarkeit — gültige Werte sind u. a. „+“ für *öffentlich*, „#“ für *geschützt* und „-“ für *privat* — wird in der $_m$UML–Notation nicht benutzt, da in der MML keine entsprechende Kennzeichnung für Attribute vorgesehen ist. Als Sichtbarkeit für $_m$UML–Attribute wird daher immer der Wert „+“ angenommen, so dass im objektorientierten Sinn der Zugriff auf die Attribute nicht eingeschränkt ist. Für Standardwerte von Attributen wird wie in der UML der Abschnitt „= Standardwert“ einer Attributdefinition verwendet.
Der Wert der Eigenschaft *isKey* der *PropertyConnection*–Metaklasse und somit die Angabe, ob ein Attribut eine Schlüsseleigenschaft der Klasse darstellt, erfolgt mittels des Erweiterungsmechanismus für Elementeigenschaften: Für Attribute einer $_m$UML–Klasse wird die neue Eigenschaft *isKey* mit dem Standardwert *false* eingeführt, der in Form einer Eigenschaftsspezifikation ein Wert zugewiesen werden kann. Für die Beschreibung von Eigenschaftsspezifikationen existiert in der UML eine verkürzte Schreibweise, die Anwendung finden kann, falls eine Eigenschaft einen booleschen Wert besitzt. Die Eigenschaftsspezifikation kann in diesem Fall auf den Namen der Eigenschaft reduziert werden. Es wird dann bei Angabe der Eigenschaft automatisch der Wert *true* angenommen. Fehlt die Angabe der Eigenschaft in den Eigenschaftsspezifikationen eines Modellelementes, so wird sie auf *false* gesetzt. Wird das Modellelement textuell in einem UML–Diagramm dargestellt, so können die in geschweiften Klammern geschriebenen Eigenschaftsspezifikationen der textuellen Darstellung angehängt werden [Rat97b]. Ein als Schlüsselattribut gekennzeichnetes $_m$UML–Attribut namens „Produkt–Nr.“ und dem Datentyp „Zahl“ wird somit als „Produkt-Nr. : Zahl {isKey}“ notiert, wie in Abbildung 6.17. dargestellt.

[6] Das Fehlen von Multiplizitätsangaben bei Attributen ist in der Modellierbarkeit dieser Attributbeziehungen durch die *Composition*–Metaklasse begründet.

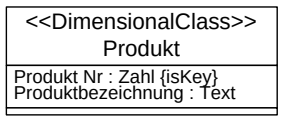



Abbildung 6.17: $_m$UML: Darstellung von Schlüsseln

Zur Darstellung *abgeleiteter Attribute* werden in einem $_m$UML–Diagramm die beiden Elementeigenschaften *formula* und *parameter* definiert. Die *formula*–Eigenschaft nimmt direkt die in der MML–Metaklasse *Computation* vorgesehene Berechnungsvorschrift auf. Die für die Durchführung der Berechnung als Parameter zugeordneten Attribute werden mit ihrem Namen in der Eigenschaft *parameter* in einer durch Kommata getrennten Auflistung gespeichert. Die UML bietet für abgeleitete Modellelemente (*derived elements*) eine Markierungsunterstützung in Form eines Schrägstriches, der dem jeweiligen Elementnamen vorangestellt wird [Rat97a]. Diese Notation wird für die $_m$UML übernommen, so dass abgeleitete Attribute in der Darstellung eines $_m$UML–Diagramms auch dann erkennbar sind, falls die zusätzlichen Elementeigenschaften nicht angezeigt werden.
Abbildung 6.18 zeigt diese Notation beispielhaft anhand einer *FactClass*–Klasse mit zwei normalen und einem abgeleiteten Attribut, dessen Wert sich aus dem Produkt der beiden anderen berechnet.

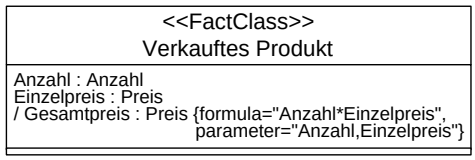



Abbildung 6.18: $_m$UML: *FactClass*–Instanz mit abgeleitetem Attribut

### 6.2.3 Verbindungen

Als Notationselemente für die Beschreibung von Verbindungen innerhalb eines Klassendiagramms bietet die UML die fünf Modellierungskonstrukte *Association, Composition, Link, Generalization* und *Dependency* [Rat97a]. Ein *Link* entspricht einer instantiierten Assoziation und kann folglich nicht als Notationselement für Klassen genutzt werden. Das *Dependency*–Konstrukt ist für Beziehungen zwischen Modellelementen ohne Berücksichtigung der zugehörigen Objekte vorgesehen. Aufgrund dieser Eigenschaften wird im Folgenden auf die Nutzung dieser zwei UML–Elemente verzichtet.
Der Name einer $_m$UML–Assoziation bzw. –Komposition wird dem Attribut *name* der *MMLElement*–Metaklasse zugeordnet, das die Verbindung innerhalb des MML–Diagramms eindeutig identifiziert. Auf die explizite Vergabe des Namens kann in einem $_m$UML–Diagramm verzichtet werden; in diesem Fall existiert implizit ein eindeutiger Name, der nicht dargestellt wird.

#### Assoziationen

Das Assoziationskonstrukt der UML wird für die Beschreibung von „normalen“ Beziehungen zwischen $_m$UML–Klassen übernommen. Assoziationen zwischen mehr als zwei Klassen, die in der UML durch ein Rautensymbol dargestellt werden, sind durch die MML–Metaklasse *Association* nicht definiert und dürfen daher in einem $_m$UML–Diagramm nicht verwendet werden.
Die Rollenbezeichnungen der zwei an der Assoziation beteiligten Klassen sind den zwei Attributen *endAName* und *endBName* der MML–Metaklasse zugeordnet, die Multiplizitätsangaben

entsprechend dem *endAMultiplicity*– bzw. *endBMultiplicity*–Attribut. Bei einer fehlenden Rollenbezeichnung wird der Name der referenzierten Klasse übernommen. Der Wert einer unspezifizierten Multiplizität ist „0..*“. Sichtbarkeitsangaben der Rollen werden analog der Sichtbarkeit von Attributen nicht berücksichtigt.
Jeder Assoziation kann in der MML eine *DataElement*–Instanz — also eine *DataClass*– bzw. eine *DataType*–Instanz — zur Aufnahme zusätzlicher Assoziationsdaten zugewiesen werden. Die UML bietet ein ähnliches Konstrukt, die sog. *Assoziationsklasse*, die die Eigenschaften des Klassen– mit denen des Assoziationskonstruktes vereint. Aufgrund dieser Kombination kann jedoch nur eine Assoziation zu einer Assoziationsklasse gehören, so dass dieses Modellierungskonstrukt für die Repräsentation des Zusatzattributes einer MML–Assoziation nicht verwendet werden kann. Statt der Assoziationsklasse existiert in der $_m$UML für Assoziationen die zusätzliche Elementeigenschaft *attribute*, deren Wert den Namen der als Attribut referenzierten *DataElement*–Instanz enthält. Fehlt diese Eigenschaft bei einer Assoziation, so können die Assoziationsobjekte neben den für die Verbindung notwendigen Daten keine Zusatzinformationen aufnehmen. Abbildung 6.19 verdeutlicht dies: Zwischen den beiden Klassen „Produkt“ und „Filiale“ existiert eine Assoziation, welches Produkt in welcher Filiale geführt wird. Diese Beziehung ist zeitabhängig, die Elementeigenschaft verweist auf den Datentyp „IntervallTyp“, der durch zwei Datumsangaben einen Zeitraum angibt.

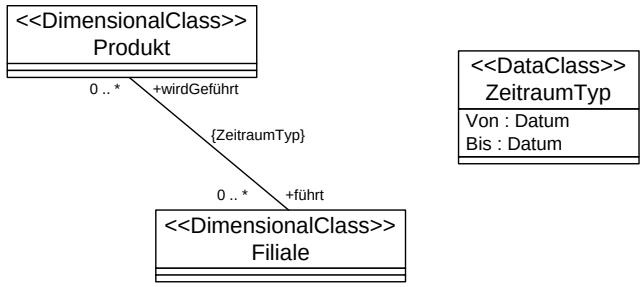



Abbildung 6.19: $_m$UML: Assoziationen zwischen zwei Klassen

**Dimensionshierarchien**

Das Assoziationskonstrukt bildet in der $_m$UML auch die Grundlage für die Darstellung der dimensionalen Eigenschaften *Dimension, RollUp, NonCompleteRollUp* und *SharedRollUp* sowie *DimensionalMapping,* die zwischen Klassen mit den Stereotypen *DimensionalClass* und *FactClass* bestehen können. Da es für die drei *RollUp*– und den *DimensionalMapping*–Beziehungstypen wichtig ist, in welcher Richtung die Kante zwischen zwei *DimensionalClass*–Klassen verläuft, wird für die Darstellung der Beziehungen die Navigierbarkeitseigenschaft der UML–Assoziation ausgenutzt. $_m$UML–*RollUp*– und $_m$UML–*NonCompleteRollUp*–Beziehungen sind somit gerichtete Beziehungen zwischen zwei *DimensionalClass*–Klassen, wobei die Beziehung nur in Richtung der als *type* betrachteten Klassen navigierbar ist. Entsprechendes gilt für die *Dimension*– und *DimensionalMapping*–Beziehungen. Zur Unterscheidung gegenüber normalen Assoziationen werden die fünf Stereotypen *Dimension, RollUp, NonCompleteRollUp, SharedRollUp* und *DimensionalMapping* verwendet. Aufgrund der Wohlgeformtheitseigenschaften der MML (vgl. Abschnitt 6.1.8) ist Zyklenfreiheit der gerichteten *RollUp–, NonCompleteRollUp*– und *SharedRollUp*–Beziehungen zu gewährleisten, so dass kein unendlicher Verdichtungspfad entstehen kann.
Bei dimensionalen Eigenschaften wird für die Benennung nicht das *name*–Attribut der *MMLElement*–Metaklasse verwendet, sondern der Wert der *propertyName*–Eigenschaft der *PropertyConnection*–Metaklasse zugeordnet. Die Bezeichnung muss daher innerhalb eines $_m$UML–Schemas nicht mehr eindeutig sein, um z. B. in verschiedenen Hierarchien gleichbezeichnete *RollUp*s darstellen

zu können. Das *name*–Attribut dient hingegen als interner Bezeichner und besitzt keine $_m$UML–Darstellung.
Für die dimensionalen Eigenschaften *Dimension*, *RollUp* sowie *NonCompleteRollUp* sind Multiplizitätsangaben und Rollenbezeichnungen nicht vorgesehen, weil auf der *owner*–Seite implizit von einer *1–zu–Viele*– bzw. *0..1–zu–Viele*–Multiplizität ausgegangen wird. Für die *type*–Seite, bei der immer genau ein *DimensionalClass*–Objekt referenziert werden muss, gilt folglich die Multiplizität „1..1“[7]. Anders verhält es sich hingegen bei den Metaklassen *SharedRollUp* und *DimensionalMapping*, die jeweils ein *multiplicity*–Attribut für die Aufnahme einer durch den Modellierer spezifizierbaren Multiplizitätsangabe für die *type*–Seite besitzen. Diese sollte nur explizit angegeben werden, wenn sie nicht „0..*“ ist, die Angabe dieses Standardwertes stellt aber keinen Verstoß gegen die $_m$UML–Notation dar. Da *Dimension*– und *RollUp*–Verbindungen auch die Schlüsseleigenschaft besitzen, wird die für die Attribute definierte Elementeigenschaft *isKey* übernommen. Zur Verdeutlichung der dimensionalen Eigenschaften von *DimensionalClass*– und *FactClass*–Objekten zeigt Abbildung 6.20 die *FactClass*–Klasse *Verkauf* und die zugeordneten Dimensionen *Zeit* und *Produkt*, die jeweils einen einfachen Hierarchiepfad aufweisen.

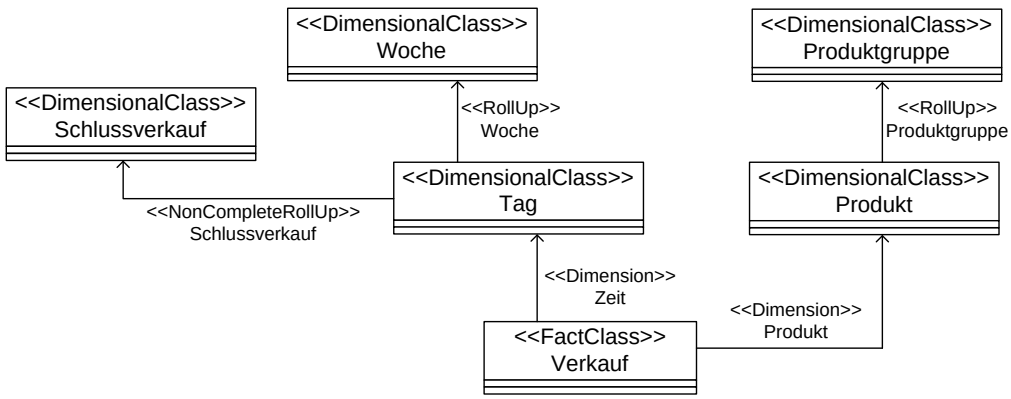



Abbildung 6.20: $_m$UML: *Dimension*- und *Roll-Up*-Beziehungen

*SharedRollUp*– *DimensionalMapping*–Verbindungen besitzen Berechnungsvorschriften. Diese werden analog zu den Berechnungsvorschriften abgeleiteter Attribute als Elementeigenschaft dargestellt.

### Kompositionen

Die UML sieht verschiedene Formen für die graphische Darstellung von Kompositionen [Rat97a] vor. Eine davon ist die attributbasierte Darstellung, die die Komposition anhand der dem Attributnamen folgenden Multiplizitätsangabe kennzeichnet. In der $_m$UML wird die Repräsentation einer Kompositionsbeziehung durch eine Linie zwischen dem besitzenden und dem abhängigen Modellelement bevorzugt, wobei auf der Seite des besitzenden Elementes eine schwarz gefüllte Raute gezeichnet wird. Aufgrund der Vorgaben durch die MML gilt für Kompositionen, dass diese nur zwischen Klassen erlaubt sind, die denselben Stereotyp tragen. Abbildung 6.21 verdeutlicht die graphische Darstellung einer *FactClass*–Klasse *Verkauf*, bei der jedem Objekt *Eins–bis–Viele* Objekte der Klasse *Verkauftes Produkt* zugeordnet sind.

[7] Für eine Multiplizitätsangabe, bei der der untere dem oberen Wert entspricht, kann die UML–Kurzform ohne Intervalldarstellung verwendet werden: Für das Intervall „1..1“ kann somit alternativ auch nur die Zahl „1“ notiert werden.

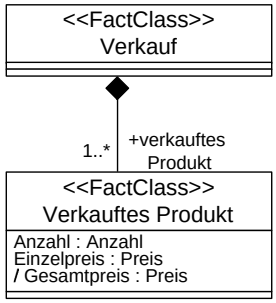



Abbildung 6.21: $_m$UML: Darstellung der Kompositionsbeziehung

Die Rollenbezeichnung auf Seite der abhängigen Klasse[8] ist der *endAName*–Eigenschaft der MML–Metaklasse *Composition* zugeordnet; die Bezeichnung auf der anderen Seite wird in der $_m$UML nicht genutzt. Bei Übereinstimmung des Namens der referenzierten Klasse mit der Bedeutung der Komposition, kann auf die Angabe der Rollenbezeichnung verzichtet werden. Entsprechend der Zugehörigkeit der Rollenbezeichnung wird die Multiplizitätsangabe in das *endAMultiplicity*–Attribut aufgenommen. Da jedes Objekt der abhängigen Klasse immer zu genau einem Objekt der besitzenden Klasse gehört, gilt für die Multiplizitätsangabe auf der Besitzerseite der implizite Wert „1“. Kompositionen dürfen gemäß der Wohlgeformtheitseigenschaft (WF:ZF2) (siehe Seite 87) nicht zyklisch sein.

**Generalisierungen**

Das UML–Generalisierungskonstrukt wird auch in der $_m$UML für die Beschreibung von Generalisierungen und Spezialisierungen verwendet. Das Konstrukt darf gemäß der Wohlgeformtheitseigenschaften aus Abschnitt 6.1.8 jedoch nur zwischen Klassen mit demselben Stereotyp genutzt werden, da es ansonsten möglich wäre, eine *FactClass*–Klasse von einer *DimensionalClass*–Klasse abzuleiten. Darüber hinaus müssen Generalisierungen zyklenfrei sein (Wohlgeformtheitseigenschaft (WF:ZF3)). Ein Beispiel war in Abbildung 6.15 zu sehen.

### 6.2.4 Verdichtungsoperatoren

Die Freigabemöglichkeit für Verdichtungsoperatoren, die in der MML die *allowedOperators*–Eigenschaft der *SharedRollUp*– und *Additivity*–Metaklasse bereitstellt, wird in der $_m$UML–Notation durch eine entsprechende Elementeigenschaft dargestellt. Der Wert dieser Eigenschaft enthält eine Liste von Attribut–Operator–Zuordnungen, die jeweils für ein Attribut die erlaubten Verdichtungsoperatoren festlegen und folgenden allgemeinen Aufbau besitzen:

$\text{Attributname} : \{ \text{Operator}_1, \text{Operator}_2, \ldots, \text{Operator}_n \}$

Für Kompositionsbeziehungen zwischen zwei *FactClass*–Klassen und die zu *FactClass*– Klassen gehörenden *Dimension*–Verbindungen kann die Elementeigenschaft Informationen darüber enthalten, welche Operatoren für welches Attribut freigegeben sind. Eine mögliche Zuweisung ist „allowedOperators = ” Attribut1 : {sum, max, min}, Attribut2 : {sum}, Attribut3 : {sum, max}” “, die u. a. für das Attribut *Attribut1* die Verdichtungsoperatoren *sum*, *max* und *min* freigibt.
Bei *SharedRollUp*–Verbindungen wird in der MML keine Differenzierung der Operatoren bzgl. der Anwendbarkeit für bestimmte Attribute vorgenommen, so dass in diesem Fall als Attributname ein Stern einzutragen ist, der als Platzhalter für alle Attribute steht. Soll beispielsweise bei einer anteiligen Verrechnung nur der *sum*–Operator erlaubt sein, ist bei der Beziehung für die *allowedOperators*–Eigenschaft der Wert ”* : {sum}” einzutragen.

[8] In Abbildung 6.21 ist dies der Text „verkauftes Produkt“; die Bezeichnung der Komposition fehlt in der Darstellung.

## 6.3 Leitfaden zum Erstellen eines Schemas

Nachdem die MML als Metasprache das Datenmodell zur Verfügung stellt und mit der $_m$UML eine darauf aufbauende graphische Notation eingeführt worden sind, soll in diesem Abschnitt die Frage „*Wie* komme ich mit diesen Beschreibungsmitteln zu einem Schema ?“ beantwortet werden. Dabei sollte der Leitfaden als *Kann*–Bestimmung aufgefasst werden und keinesfalls in jedem Projekt bzw. Kontext kategorisch eingesetzt werden. Zu empfehlen ist der Leitfaden vor allem solchen Modellierern, die in der multidimensionalen Modellierung weniger geübt sind.

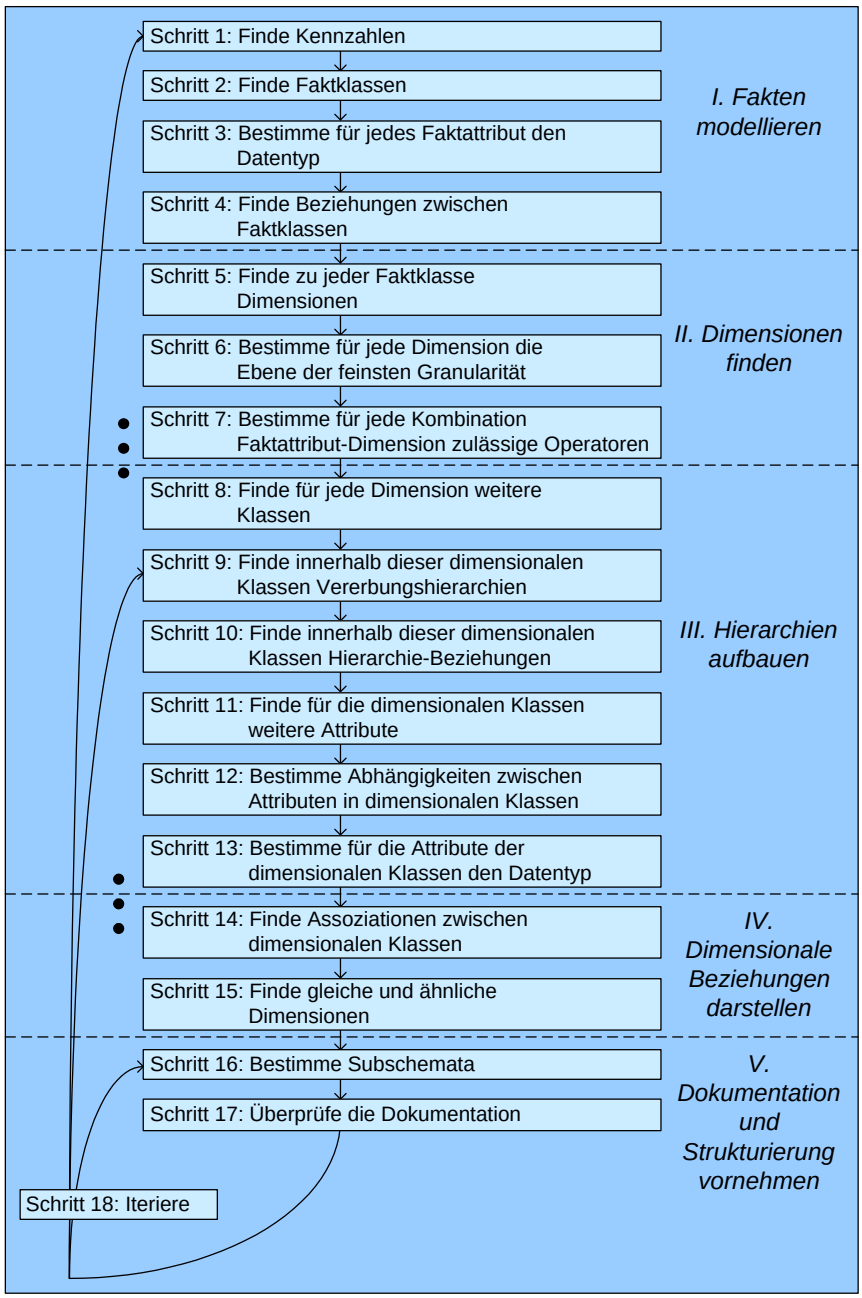



Abbildung 6.22: Leitfaden zum Erstellen eines MML–Schemas

Die Ursache hierfür liegt in der aus der (objektorientierten) Modellierung bekannten Tatsache, dass es in vielen Situationen konkurrierende Modellierungsmöglichkeiten gibt, es existiert nicht oder nur

in seltenen Ausnahmefällen „the one best way“ [Rum93b].
Die im Leitfaden vorgestellte Vorgehensweise ist im Sinne der Klassifikation in [ACPT99] eine *Bottom–Up*–Strategie. Sie umfasst die 18 in Abbildung 6.22 skizzierten Schritte, die sich in fünf Phasen unterteilen lassen: In der Phase I (Schritte 1–4) werden die Fakten identifiziert und Abhängigkeiten zwischen diesen festgehalten. Die Phase II (Schritte 5–7) dient dem Aufspüren der zu den Fakten passenden Dimensionen. Das Aufbauen von Hierarchiestrukturen, die für die spätere Datenanalyse relevant sind, ist die Aufgabe der Schritte 8 bis 13, die Phase III bilden. In Phase IV (Schritte 14 und 15) werden zwischendimensionale Beziehungen festgehalten. Die abschließenden Schritte 16 bis 18 bilden Phase V, die administrativen Aufgaben wie gute Strukturierung, Dokumentation und Verfeinerung des Schemas beinhaltet. Die Rückkopplungen von Schritt 18 zeigen, dass es sich nicht um einen sequentiellen Prozess handelt, sondern das das Schema durch Iterationen sukzessive verfeinert werden kann. Dabei kann der Rücksprung zu einem beliebigen Schritt durchgeführt werden.
Als alternativer Weg bietet sich an dieser Stelle eine von den Dimensionen ausgehende Vorgehensweise an [Kim96]. Allerdings wirkt dieses Vorgehen ein wenig unnatürlich und kann auch fehlerträchtig sein. So kann man u.U. dimensionale Strukturen aufbauen und dann gar keine Fakten haben, die gemäß dieser Dimensionen auszuwerten sind. Tabelle 6.2 beschreibt die einzelnen Schritte des Leitfadens im Detail.

## Leitfaden zum Erstellen eines Schemas

**Phase I: Fakten modellieren**

**Schritt 1: Finde Kennzahlen**

Alle für die Entscheidungsunterstützung relevanten Werte sind zu identifizieren und aufzulisten. Potenzielle Kandidaten hierfür sind vom zukünftigen Benutzer genannte oder in der Spezifikation gefundene numerische Attribute. Liegen Berichte in ausgedruckter Form vor, so sind alle Zahlenwerte, vor allem solche, auf denen addiert oder eine andere Gruppierungsfunktion ausgeführt wird, mögliche Kennzahlen. Liegt während dieser Phase das Datenschema von OLTP–Datenbanken vor, die als Datenquelle dienen, sind alle numerischen Attribute potenzielle Kennzahlen.

**Schritt 2: Finde Faktklassen**

Treten zwei (oder mehrere) der im ersten Schritt identifizierten Kennzahlen stets zusammen auf, so sind sie zu einer Klasse zusammenzufassen.

**Schritt 3: Bestimme für jedes Faktattribut den Datentyp**

Für jedes Faktattribut ist der Datentyp zu bestimmen, was an dieser Stelle im Sinne einer von jeglichen technischen Details abstrahierenden konzeptionellen Modellierung ein sprechender Name sein sollte und kein technischer wie z. B. „LongInteger“ oder „String(10)“.

**Schritt 4: Finde Beziehungen zwischen Faktklassen**

Beziehungen zwischen Faktklassen in Form von Generalisierungen und Kompositionen sind zu analysieren. Als Faustregel für das (Nicht–)Vorliegen einer Generalisierung kann dabei die Prüfung „B ist Unterklasse von A, wenn man „B *ist ein* A“ sagen kann“ herangezogen werden. Dieser *ist–ein*–Test ist nicht absolut zuverlässig, aber er identifiziert vor allem die negativen Fälle, d.h. er bewahrt vor Missbrauch der Vererbung [Rum93a]. Für Kompositionen kann analog ein ähnlicher Test durchgeführt werden: wenn man sagen kann, „B *enthält* As“ oder „B *besteht aus* As“ oder „A *ist Teil von* B“, so liegt (meistens) eine Komposition vor.

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

**Phase II: Dimensionen finden**

**Schritt 5: Finde zu jeder Faktklasse Dimensionen**

Zu jeder Faktklasse sind Dimensionen zu bestimmen, die die Werte der Attribute qualifizieren. Um alle Dimensionen einer Faktklasse vollständig zu erfassen, hilft es die *W–Fragewörter* zu beantworten: „Wo, wann, was, wie , ... treten die Faktattribute auf?“.

**Schritt 6: Bestimme für jede Dimension die Ebene der feinsten Granularität**

Für jede Dimension ist zu überprüfen, welche Anforderungen die potenziellen Analysen und Auswertungen an die Ebene der feinsten Granularität haben. Diese Ebene der feinsten Granularität ist entsprechend als *DimensionalClass* an der *Dimension*–Beziehung zur entsprechenden *FactClass* einzutragen.

**Schritt 7: Bestimme für jede Kombination Faktattribut–Dimension zulässige Operatoren**

Für jedes Attribut jeder Faktklasse ist bezüglich jeder Dimension zu überprüfen, welche Verdichtungsfunktionen für diese Kombination zugelassen sein sollen. Standardmäßig sind alle Operatoren zugelassen, dies kann jedoch in Abhängigkeit der Daten Probleme geben.

**Phase III: Hierarchien aufbauen**

**Schritt 8: Finde für jede Dimension weitere Klassen**

Für jede Dimension sind weitere für die potenzielle Analyse wichtige Klassen zu entdecken und als *DimensionalClass*–Objekte festzuhalten.

**Schritt 9: Finde innerhalb dieser dimensionalen Klassen Vererbungshierarchien**

Die im letzten Schritt gefundenen *DimensionalClasses* sind auf Spezialisierungen hin zu analysieren. Sind entsprechende Abhängigkeiten entdeckt worden, so sind sie durch eine *Generalization*–Beziehung festzuhalten. An dieser Stelle ist außerdem die Entscheidung zu fällen, ob und wenn ja, welche Klassen in dieser Hierarchie als abstrakt markiert werden sollen. Auch an dieser Stelle sollte ein Test bezüglich Vererbungsmissbrauch analog zu dem in Schritt 4 durchgeführt werden. Außerdem ist bei dimensionalen Klassen auf einen Vererbungsmissbrauch zur reinen Typunterscheidung zu achten. Es sollte deswegen konsequent die Regel gelten, ein neue Klasse nur dann einzuführen, wenn sie sich in mindestens einem Attribut von ihrer Generalisierung unterscheidet. Reine Typunterscheidung sollte durch ein Attribut mit entsprechendem Aufzählungstyp dargestellt werden [Rum93b]. Auch die korrekte Handhabung multipler Generalisierungen [Rum93b] spielt eine große Rolle: wird eine Klasse nach zwei orthogonalen Kriterien unterschieden, so ist dies durch Mehrfachvererbung zu realisieren.

**Schritt 10: Finde innerhalb dieser dimensionalen Klassen Hierarchie–Beziehungen**

Innerhalb der *DimensionalClass*–Instanzen einer Dimension sind Hierarchiepfade zu bilden. Dabei ist für jede Beziehung zu prüfen, ob sie als *RollUp* oder *SharedRollUp* zu realisieren ist. Im letzteren Fall ist eine geeignete Berechnungsvorschrift zu definieren.

**Schritt 11: Finde für die dimensionalen Klassen weitere Attribute**

Jede *DimensionalClass* ist in Hinblick auf weitere beschreibende Attribute zu untersuchen, die nicht unmittelbar für die Hierarchiebildung eingesetzt werden. Dabei sollte das allgemein in der Objektorientierung eingesetzte Prinzip der strengen Kohäsion (*Each attribute value should represent a fact about the object, the whole object, and nothing but the object.*) befolgt werden. Ein Spezialfall dieser Attribute sind abgeleitete Attribute. An dieser Stelle kann ein Re–Design von Schritt 9 notwendig werden, wenn man feststellt, dass sich zwei Klassen eventuell doch stärker bzw. weniger stark als ursprünglich angenommen unterscheiden.

**Schritt 12: Bestimme Abhängigkeiten zwischen Attributen in dimensionalen Klassen**

Im Gegensatz zu den abgeleiteten Attributen sind an dieser Stelle inhaltliche Abhängigkeiten zwischen Attributen zu identifizieren und in Form von Constraints in OCL–Notation festzuhalten.

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| |
|---|
| **Schritt 13: Bestimme für die Attribute der dimensionalen Klassen den Datentyp** |
| Für jedes Attribut jeder *DimensionalClass* ist der Datentyp zu bestimmen, wobei (analog zu den Faktattributen in Schritt 3) ein sprechender Bezeichner gewählt werden sollte. Für jede *DimensionalClass* sind die charakterisierenden Attribute mit der Schlüsseleigenschaft zu markieren. Ebenso kann in diesem Schritt vor allem für nicht an der Hierarchiebildung beteiligte Attribute ein Standardwert festgelegt werden. |
| **Phase IV: Dimensionale Beziehungen darstellen** |
| **Schritt 14: Finde Assoziationen zwischen dimensionalen Klassen** |
| Zwischen dimensionalen Klassen (vorwiegend der niedrigsten Hierarchieebene), die in unterschiedlichen Dimensionen sind, kann es Beziehungen geben, die durch eine Assoziation zu modellieren sind. Dies entspricht der Festlegung von Definitionslücken im multidimensionalen Raum. |
| **Schritt 15: Finde gleiche und ähnliche Dimensionen** |
| Durch paarweises Vergleichen von Dimensionen, die zu unterschiedlichen Fakten gehören, können gleiche und ähnliche identifiziert werden, die Verbindungen zu unterschiedlichen Faktklassen besitzen. *Gleich* bedeutet in diesem Falle, dass semantisch gleiche Dimensionen beschrieben werden und auch exakt die gleichen Hierarchiepfade vorliegen. Als *ähnlich* gelten Dimensionen, die zwar das gleiche beschreiben, aber verschiedene Hierarchiepfade besitzen oder unterschiedliche Ebenen der feinsten Granularität aufweisen. In diesem Fall kann durch Modellierung eines *DimensionalMapping*–Objektes diese Beziehung zwischen den Dimensionen realisiert werden, wenn in potenziellen Auswertungen Vergleiche der Fakten notwendig sind, mit denen sie verbunden sind. |
| **Phase V: Dokumentation und Strukturierung vornehmen** |
| **Schritt 16: Bestimme Subschemata** |
| Um die Übersichtlichkeit des Datenschemas zu gewährleisten, soll es durch Subschemata geeignet strukturiert werden. Es bietet sich insbesondere an, pro Dimension und zusammenhängender Faktklassen jeweils ein eigenes Teildiagramm zu bilden. |
| **Schritt 17: Überprüfe die Dokumentation** |
| Als letzter Schritt ist die Dokumentation der Klassen und Attribute zu überprüfen und gegebenenfalls zu berichtigen und ergänzen. |
| **Schritt 18: Iteriere** |
| Führe Iterationen durch und verfeinere das Schema. Bei jeder Iteration kann zu einem beliebigen Schritt des Leitfadens zurückgesprungen werden. |

Tabelle 6.2: Leitfaden zum Erstellen eines Schemas

## 6.4 Qualitätssicherung von MML–Schemata

Neben dem Vorliegen eines bez. Metamodell und Wohlgeformtheitsbedingungen korrekten MML–Schemas, ist zum Abschluss des konzeptionellen Entwurfs eine analytische Qualitätssicherung durch ein explizites Schema–Review vorgesehen, um zu überprüfen, ob auch *inhaltlich* korrekt modelliert wurde [Her01a]. Dabei sollen die zu untersuchenden Kriterien flexibel durch den Benutzer ausgewählt werden können, d. h. die Kriterien für ein Review können individuell bestimmt werden. Daher wird in Abschnitt 6.4.1 zunächst das Metamodell für dieses Review beschrieben, bevor in Abschnitt 6.4.2 die Anwendung dieses Framework verdeutlicht wird. Abschnitt 6.4.3 liefert schließlich eine Auflistung möglicher Kriterien für das Review; ebenso erfolgt die Nennung aus der Literatur bekannter Qualitätskriterien, die sich jedoch nicht für ein Review anbieten.

### 6.4.1 Metamodell für Qualitäts–Review

Das Metamodell für ein Qualitäts–Review ist in Abbildung 6.23 dargestellt.

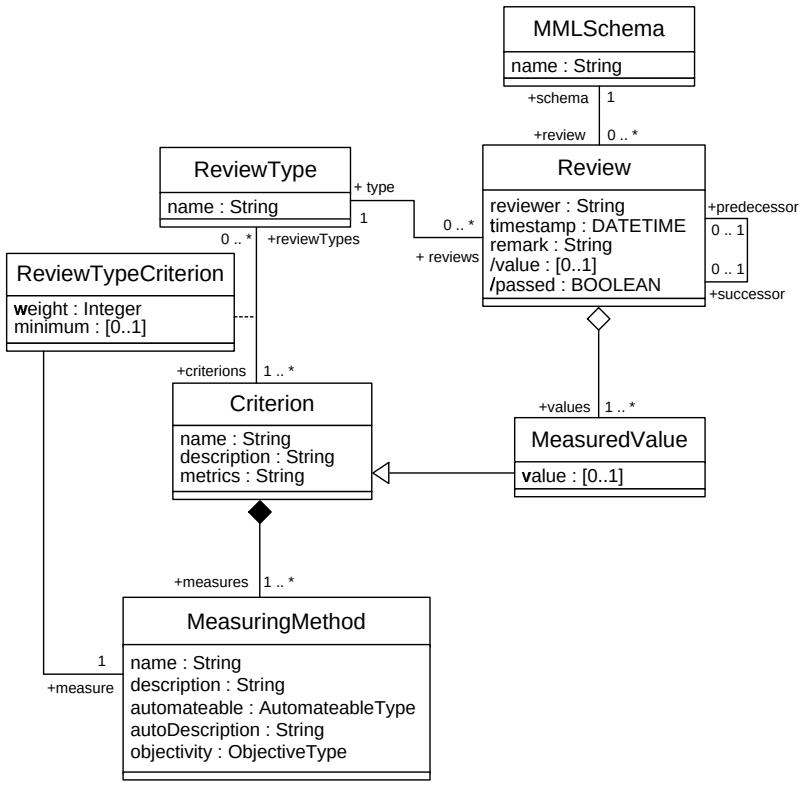



Abbildung 6.23: Metamodell für Reviews

Die Bewertung eines Schemas wird anhand von Kriterien (Objekte der Klasse *Criterion*) vorgenommen, die durch ihren Namen, eine Beschreibung und Angabe einer Metrik definiert sind. Jedes Kriterium kann durch ein oder mehrere Messverfahren (Objekte der Klasse *MeasuringMethod*) bestimmt werden. Ein *MeasuringMethod*–Objekt ist durch seinen Namen und eine Beschreibung definiert und enthält zusätzlich die Attribute *automateable* (gibt an, zu welchem Grad das Verfahren automatisierbar ist), *autoDescription* (gibt im Falle der Automatisierbarkeit das Vorgehen an) und *objectivity* (bewertet, wie objektiv/subjektiv dieses Messverfahren ist).
Zulässige Werte zur Beschreibung des Automatisierbarkeitsgrades sind „Ja“, „Nein“ und „Bedingt“. Der Grad der Objektivität kann mit einem der vier Werte „Objektiv“, „Relativ Objektiv“, „Subjektiv“ und „Relativ Subjektiv“ bewertet werden.
Aus einem oder mehreren Kriterien setzt sich ein *ReviewType* zusammen. Hierbei ist mittels der Assoziationsklasse *ReviewTypeCriterion* für jedes der Kriterien ein Gewicht (Attribut *weight*), das gewählte Messverfahren (Referenz *measure* auf die Klasse *MeasuringMethod*) und optional ein minimaler Schwellwert (Attribut *minimum*) zu definieren. Ein Unterschreiten dieses Wertes soll ein Scheitern des gesamten Reviews zur Folge haben, d. h. es handelt sich hierbei um ein K.O.–Kriterium. Ein solcher *ReviewType* kann als Basis mehrerer *Reviews* dienen, die konkreten Messwerte werden als Objekte der Klasse *MeasuredValue* verwaltet. Die rekursive Beziehung der *Review*–Metaklasse ermöglicht es, eine Folge von Reviews festzulegen. Untersuchungsobjekt eines Reviews ist ein MML–Schema, was durch die Klasse *MMLSchema* modelliert ist.

### 6.4.2 Konfigurieren und Durchführen von Reviews

Die praktische Handhabung des Metaschemas ist in Abbildung 6.24 dokumentiert: Zunächst werden im ersten Schritt Kriterien vor dem Hintergrund des konkreten Projektes bzw. der konkreten Domäne zu einem *ReviewType* zusammengefasst. In diese Auswahl fließen konkrete Projektziele ein. Soll z. B. nur ein Prototyp realisiert werden, so ist Vollständigkeit sicherlich kein Kriterium eines Reviews. Der im ersten Schritt bestimmte Review–Typ wird dann im zweiten Schritt für ein MML–Schema ausgeführt, bevor im dritten Schritt das Schema verbessert werden kann. Die Schritte 2 und 3 können dabei mehrfach iterieren, bis das Resultat des Review das gewünschte Ergebnis, d. h. das MML–Schema die geforderte Qualität, besitzt.

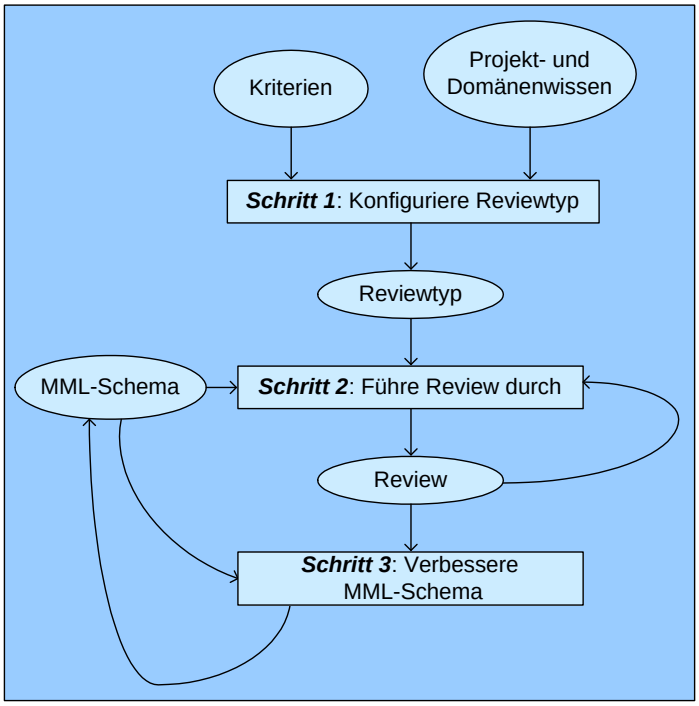



Abbildung 6.24: Vorgehen Konfiguration und Durchführung von Reviews

### 6.4.3 Qualitätskriterien für MML–Schemata

In diesem Abschnitt werden Qualitätskriterien für MML–Schemata anhand folgender Merkmale beschrieben:

- *Kriterium* : <Name des Kriteriums>
- *Beschreibung* : <Kurze Erläuterung>
- *Relevanz* : <Bedeutung des Kriteriums im Kontext von MML–Schema–Reviews>
- *Messung* : <Vorschlag eines Messverfahrens für das Kriterium>
- *Metrik* : <Maß für das Kriterium>
- *Automatisierbarkeit* : <Aussage zur Automatisierbarkeit des Messens>
- *Objektivität* : <Aussage zur Objektivität/Subjektivität des Kriteriums bzw. Verfahrens>

Die in Tabelle 6.3 aufgeführten Kriterien können als *Criterion*–Objekte im Metaschema aus Abbildung 6.23 eingetragen werden. Dabei wird für jedes Kriterium nur ein Messverfahren genannt, im Metamodell in Abbildung 6.23 war eine flexible *Eins–zu–Viele*–Beziehung zwischen Kriterien und Messverfahren definiert worden, um später eventuell weitere Messverfahren für ein Kriterium definieren zu können.

## Mögliche Qualitätskriterien für Reviews von MML–Schemata

| | |
|---|---|
| **Kriterium** | (Fachliche) Korrektheit |
| **Beschreibung** | Sind die Schemaaussagen korrekt in Bezug auf fachliche Anforderungen, Begriffe bzw. Sachverhalte im Anwendungsbereich? |
| **Relevanz** | Kriterium ist essentiell für die Bedeutung eines Schemas. |
| **Messung** | Durch Review eines Fachvertreters. Überprüfung jedes Sachverhaltes (Fakt, Dimension, Hierarchiebildung, freigegebene Operatoren etc.) auf fachliche Korrektheit. |
| **Metrik** | Quote fachlich (nicht) korrekter Sachverhalte. |
| **Automatisierbarkeit** | Nein. |
| **Objektivität** | Objektiv. |
| **Kriterium** | (Fachliche) Konsistenz |
| **Beschreibung** | Sind die Schemaaussagen widerspruchsfrei? |
| **Relevanz** | Kriterium ist zusammen mit der Korrektheit essentiell für die Bedeutung und Akzeptanz eines Schemas. |
| **Messung** | Durch Review eines Fachvertreters. Überprüfen jedes Sachverhaltes (Fakt, Dimension, Hierarchiebildung, freigegebene Operatoren etc.) und anschließender Vergleich auf fachliche Widersprüche oder Ungenauigkeiten. |
| **Metrik** | Quote fachlich (nicht) widersprüchlicher Sachverhalte. |
| **Automatisierbarkeit** | Nein. |
| **Objektivität** | Objektiv. |
| **Kriterium** | (Fachliche) Relevanz |
| **Beschreibung** | Sind die Schemaaussagen fachlich relevant, d.h. werden wirklich für die Datenanalyse relevante Objekte beschrieben? |
| **Relevanz** | Kriterium bedeutend im Hinblick auf spätere Benutzerakzeptanz. |
| **Messung** | Durch Review eines Fachvertreters. Feststellen aller relevanten Sachverhalte und der durch Fakten und Dimensionen des Schemas abgedeckten. |
| **Metrik** | Quote fachlich (nicht) relevanter Sachverhalte. |
| **Automatisierbarkeit** | Nein. |
| **Objektivität** | Objektiv. |
| **Kriterium** | Umfang |
| **Beschreibung** | Ist die Schemabreite angemessen? |
| **Relevanz** | Kriterium bedeutend im Hinblick auf spätere Benutzerakzeptanz. Der Umfang eines Schemas ist immer relativ im Hinblick auf die Zielsetzung. Soll z. B. ein Data–Mart–Prototyp realisiert werden, dann sind die Anforderungen an dieses Kriterium natürlich geringer als bei einem Referenzschema. |
| **Messung** | Durch Review eines Fachvertreters. Feststellen aller relevanten Sachverhalte und aller modellierten Sachverhalte. |
| **Metrik** | Quote (nicht) abgedeckter Sachverhalte bez. der Anforderungsdefinition oder dem Domänenwissen eines Fachvertreters. |
| **Automatisierbarkeit** | Nein. |
| **Objektivität** | Objektiv. |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | |
|---|---|
| **Kriterium** | Detaillierungsgrad |
| **Beschreibung** | Ist die Schematiefe angemessen? |
| **Relevanz** | Kriterium bedeutend im Hinblick auf spätere Benutzerakzeptanz. Bei den potenziellen Analysen ist es wichtig, einen (vom Benutzer vorgegebenen, kontextabhängigen) Detaillierungsgrad zu erreichen, der eine gute Entscheidungsgrundlage bietet. |
| **Messung** | Durch Review eines Fachvertreters. Feststellen des benötigten Detaillierungsgrades für jedes Fakt und jede Dimension, Angemessenheit der „Sprünge“ zwischen den Hierarchie–Ebenen. |
| **Metrik** | Quote (nicht) ausreichend detaillierter Sachverhalte. |
| **Automatisierbarkeit** | Nein. |
| **Objektivität** | Objektiv. |
| **Kriterium** | Vollständigkeit |
| **Beschreibung** | Ist das Schema vollständig (bez. der gestellten Anforderungen)? |
| **Relevanz** | Kriterium ist sehr wichtig, denn Ziel der konzeptionellen Modellierung ist ein vollständiges Schema. |
| **Messung** | Folgt aus (den beiden zuvor diskutierten Kriterien) Umfang und Detaillierungsgrad. |
| **Metrik** | Quote fehlender Anforderungen bez. der Anforderungsdefinition oder dem Domänenwissen eines Fachvertreters. |
| **Automatisierbarkeit** | Nein. |
| **Objektivität** | Objektiv. |
| **Kriterium** | Minimalität |
| **Beschreibung** | Ist das Schema kompakt genug beschrieben? |
| **Relevanz** | Kriterium kann wichtig sein, denn Ziel der konzeptionellen Modellierung ist ein möglichst genaues Schema. |
| **Messung** | Durch Review eines Fachvertreters. Feststellen, ob gewisse Sachverhalte nicht kompakter modelliert werden können, z. B. verwandte Fakten. |
| **Metrik** | Quote (nicht) redundanter Strukturen. |
| **Automatisierbarkeit** | Nein. |
| **Objektivität** | Objektiv. |
| **Kriterium** | Integrationsfähigkeit |
| **Beschreibung** | Ist das Schema für organisationsübergreifende Standardisierungen/Modellierungen geeignet? |
| **Relevanz** | Abhängig vom Kontext der Organisation, für die das Schema entsteht. Je stärker die Einbindung in einen wirtschaftlichen Verbund, desto relevanter wird dieses Kriterium, ebenso je stärker die internationale Orientierung. |
| **Messung** | Durch Review eines Fachvertreters. Feststellen, ob jeder für ein Attribut- oder Klassennamen gewählte Begriff branchenüblich oder international anerkannt ist. |
| **Metrik** | Quote des (Nicht-)Vorhandenseins solcher Begriffe. |
| **Automatisierbarkeit** | Bedingt, z. B. durch Einsatz eines Thesaurus. |
| **Objektivität** | Objektiv. |
| **Kriterium** | Dokumentation |
| **Beschreibung** | Sind alle im Schema benutzten Begriffe bzw. Fachworte dokumentiert? |
| **Relevanz** | Dokumentation ist sehr wichtig aufgrund der Lesbarkeit und Verständlichkeit des Schemas, insbesondere gegenüber nicht an der Modellierung beteiligten Personen und im Hinblick auf eine langlebige Wartbarkeit. |
| **Messung** | Per Review. Feststellen, ob zu jedem Attribut- und Klassennamen ein aussagekräftiger Kommentar existiert. |
| **Metrik** | Quote des (Nicht-)Vorhandenseins von fachlich korrekten, in der Länge angemessenen Kommentaren. |
| **Automatisierbarkeit** | Teilweise möglich, im Hinblick auf die Existenz von Kommentaren, nicht bez. ihrer Qualität. |
| **Objektivität** | (Relativ) Objektiv. |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | |
|---|---|
| **Kriterium** | Namenskonventionen |
| **Beschreibung** | Sind für Bezeichnungen alle organisations-, domänen- bzw. projektspezifischen Namenskonventionen eingehalten worden? |
| **Relevanz** | Einhaltung der Namenskonventionen ist wichtig bez. der Lesbarkeit, insbesondere gegenüber nicht an der Modellierung beteiligten Personen. |
| **Messung** | Review. Abgleich aller Attribut- und Klassennamen mit den Konventionen. |
| **Metrik** | Quote der (Nicht-)Erfüllung von Namenskonventionen. |
| **Automatisierbarkeit** | Teilweise, z. B. die Forderung nach bestimmten Präfixen. Teilweise wird das Kriterium auch durch die Methodik aus Abschnitt 6.3 unterstützt, z. B. Schritt 2. |
| **Objektivität** | Objektiv. |

Tabelle 6.3: Qualitätskriterien für MML–Schemata

## Für Reviews ungeeignete Qualitätskriterien

| | |
|---|---|
| **Kriterium** | Erweiterbarkeit |
| **Beschreibung** | Sind neue Anforderungen leicht in das Schema zu integrieren? |
| **Grund** | Kriterium ist zwar für jedes Schema relevant, denn 80% aller Softwarekosten sind Wartungskosten. Dennoch kann Erweiterbarkeit nicht an einem konzeptionellen Schema gemessen werden, vielmehr kann man festhalten, dass durch die MML als objektorientierte Sprache eine gute Basis für Erweiterbarkeit gegeben ist. |
| **Kriterium** | Wiederverwendbarkeit |
| **Beschreibung** | Sind das Schema bzw. Teile davon für spätere Modellierungen nutzbar? |
| **Grund** | Obwohl Wiederverwendbarkeit im Hinblick auf Aufwandsersparnis in späteren Projekten bedeutend ist, ist es als Kriterium in einem Review nicht geeignet. Analog zur Erweiterbarkeit bietet der Einsatz von Objektorientierung auf der konzeptionellen Entwurfsebene eine gute Basis für Wiederverwendbarkeit. |
| **Kriterium** | Schemamanagement |
| **Beschreibung** | Ist eine geregelte Administration, Nutzung und Fortschreibung des Schemas gewährleistet? |
| **Grund** | Relevanz ist zwar hoch, begründet vor allem in der Nachvollziehbarkeit der Entwicklung. Dennoch ist das Schemamanagement nicht Aspekt des konzepionellen Schemas, sondern sollte vom Werkzeug oder verwendeten Repository bereitgestellt werden. Während eines Reviews können höchstens korrekte Versionsnummerverwendung überprüft werden. |
| **Kriterium** | Umsetzbarkeit |
| **Beschreibung** | Ist das vorliegende Schema implementierbar bzw. auf die logische Entwurfsebene tranformierbar? |
| **Grund** | Das Kriterium ist wichtig, denn nur ein konzeptionelles Schema, das sich auch umsetzen lässt, hat einen Nutzen. Ansonsten würde es der reinen Dokumentation dienen. Die Umsetzbarkeit kann dennoch nicht durch ein Review überprüft werden, sondern muss durch einen Transformationsalgorithmus gewährleistet werden, der zur Methode gehört. |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| **Kriterium** | Normalisierung |
|---|---|
| **Beschreibung** | Ist das Schema „normalisiert“? |
| **Grund** | In der Welt relationaler Datenbanken eines der Hauptentwurfsziele, im multidimensionalen Kontext jedoch nicht/kaum relevant, weil keine allgemein akzeptierten multidimensionalen Normalformen existieren. |

Tabelle 6.4: Für Reviews ungeeignete Qualitätskriterien

### 6.4.4 Exkurs: Normalformen

Die Nicht–Relevanz der Normalisierung als Qualitätskriterium für konzeptionelle multdimensionale Schemata ist sicherlich ein überraschendes Resultat, insbesondere vor dem Hintergrund der Bedeutung dieses Kriteriums in der relationalen Welt. Aus diesem Grunde soll in diesem Exkurs eine weitergehende Betrachtung erfolgen. Dabei wird zunächst als Grundlage die (einzige) in der Literatur existierende Arbeit zu diesem Thema vorgestellt und im Anschluss daran die vorgestellten Aspekte kritisch beleuchtet.

#### Multidimensionale Normalformen

Während Normalformen in der relationalen Welt sich hauptsächlich auf Probleme bei Datenabhängigkeiten beziehen, Update–Anomalien auflösen und relationale Strukturen vereinfachen, sollen die in [ALW98] vorgeschlagenen Schemaeigenschaften die Korrektheit analytischer Berechnungen auf multidimensionalen Daten (zu)sichern.
Weil [ALW98] auf existierende Arbeiten *statistischer Normalformen* [Gho91, LS97] aufbaut, wird dort als Grundlage die Terminologie statistischer Datenbanken verwendet. Im folgenden werden die dort vorgenommenen Überlegungen und Definitionen auf die MML–Terminolgie übertragen.
Dazu wird zunächst in (6.1) und (6.2) der aus dem Relationenmodell bekannte Begriff der *(schwachen) funktionalen Abhängigkeit* [CBS98, HS00] zwischen Attributen auf Klassen übertragen.

$$\begin{aligned} &\text{Seien } A \text{ und } B \text{ zwei Klassen.} \\ &B \text{ ist } \textit{funktional abhängig von } A \text{ (als Symbol } A \rightarrow B) \\ \overset{def}{\Longleftrightarrow} &\text{Für jede Instanz } a \text{ von } A : \exists^1 \text{ eine Instanz } b \text{ von } B. \end{aligned} \tag{6.1}$$

$$\begin{aligned} &\text{Seien } A \text{ und } B \text{ zwei Klassen.} \\ &B \text{ ist } \textit{schwach funktional abhängig von } A \text{ (als Symbol } A \Rightarrow B) \\ \overset{def}{\Longleftrightarrow} &\text{Für jede Instanz } a \text{ von } A : \exists \text{ maximal eine Instanz } b \text{ von } B. \end{aligned} \tag{6.2}$$

Hierauf aufbauend wird in (6.3) der Begriff des *Dimensionsschemas* definiert, der durch eine Menge von *RollUp*–Beziehungen verbundener *DimensionalClasses* charakterisiert ist.

$$\begin{aligned} &\text{Eine Menge } \mathcal{D} = \{D_1, \ldots, D_n\} \text{ von } \textit{DimensionalClasses} \text{ heißt } \textit{Dimensionsschema} \\ \overset{def}{\Longleftrightarrow} &\forall D_i \in \mathcal{D} : (\exists D_j \in \mathcal{D} - \{D_i\} \text{ mit } D_i \Rightarrow D_j) \vee (\exists D_k \in \mathcal{D} - \{D_i\} \text{ mit } D_k \Rightarrow D_i). \end{aligned} \tag{6.3}$$

Das Einstiegselement in ein Dimensionsschema wird in (6.4) als *Wurzelelement* definiert.

$$\begin{aligned} &\text{Sei } \mathcal{D} \text{ ein Dimensionsschema.} \\ &D \in \mathcal{D} \text{ heißt } \textit{Wurzelelement} \overset{def}{\Longleftrightarrow} \nexists D_j \in \mathcal{D} : D_j \Rightarrow D. \end{aligned} \tag{6.4}$$

Nach diesen Vorbereitungen wird in (6.5) die *dimensionale Normalform* für ein Dimensionsschema definiert.

Sei $\mathcal{D}$ ein Dimensionsschema.

$\mathcal{D}$ befindet sich in *dimensionaler Normalform* (DNF)

$\overset{def}{\Longleftrightarrow}$ (DNF1) $\exists!$ Wurzelelement $D_t \in \mathcal{D}$.

(DNF2) Der Wertebereich der Klasse $D_t$ ist *vollständig.*

(DNF3) $\forall D_i, D_j \in \mathcal{D}$ mit $i \neq j$ und $D_i \Rightarrow D_j : D_i \rightarrow D_j$.

„Alle zwischen zwei *DimensionalClasses* bestehenden Abhängigkeiten sind nicht schwach."

(6.5)

Zur Erweiterung auf den multidimensionalen Fall wird in (6.6) ein *multidimensionales Schema* als Kombination aus Dimensionsschemata und Fakten definiert.

Ein *multidimensionales Schema* $\mathcal{M} = (\{\mathcal{D}_1, \ldots, \mathcal{D}_m\}, \mathcal{F})$ besteht aus einer endlichen Menge von Dimensionsschemata $\{\mathcal{D}_1, \ldots, \mathcal{D}_m\}$ und einer Faktklasse $\mathcal{F}$ sowie der schwachen funktionalen Abhängigkeit $\{\mathcal{D}_1, \ldots, \mathcal{D}_m\} \Rightarrow \mathcal{F}$.

(6.6)

Die Ausweitung des Begriffes *Normalform* auf den multidimensionalen Fall geschieht in (6.7).

Sei $\mathcal{M}$ ein multidimensionales Schema.

$\mathcal{M}$ ist in *multidimensionaler Normalform* (MNF)

$\overset{def}{\Longleftrightarrow}$ (M1) $\forall D \in \mathcal{D}_i : D$ ist in dimensionaler Normalform.

(M2) Alle Dimensionsschemata sind paarweise orthogonal zueinander,

d. h. $\forall C \in \mathcal{D}_i, E \in \mathcal{D}_j$ mit $i \neq j : \neg(C \Rightarrow E)$.

(M3) Die Menge der Dimensionsschemata $\{\mathcal{D}_1, \ldots, \mathcal{D}_m\}$ ist *minimal,*

d. h. $\{\mathcal{D}_1, \ldots, \mathcal{D}_m\} \Rightarrow \mathcal{F} \wedge \forall D \in \{\mathcal{D}_1, \ldots, \mathcal{D}_m\} : \{\mathcal{D}_1, \ldots, \mathcal{D}_m\} - \{D\} \not\Rightarrow \mathcal{F}$.

„Insbesondere: Das Fakt ist durch die Gesamtheit der Wurzelelemente bestimmt".

(6.7)

Eine Erweiterung auf die in [ALW98] dargestellte generalisierte multidimensionale Normalform macht in der MML–Terminologie keinen Sinn, denn die dort als *Gültigkeitskontext* geforderte Beschreibung für das Auftreten optionaler Attribute ist in der MML durch die Zugehörigkeit eines Attributes zu einer *DimensionalClass* gegeben, d. h. die Klasse ist der Gültigkeitskontext des Attributes.

**Kritische Einschätzung**

Die Forderung nach Eindeutigkeit des Wurzelelementes innerhalb einer Dimension (DNF1) und Vollständigkeit des Wertebereiches (DNF2) sind nachvollziehbar. In der MML wird ersteres durch die Eindeutigkeit der Kante vom Typ *Dimension* bez. einer *FactClass* im Metaklassendiagramm sichergestellt. Für die Vollständigkeit des Wertebereiches ist der Modellierer verantwortlich, insbesondere muss er in Schritt 13 des in Abschnitt 6.3 vorgestellten Leitfadens darauf achten, korrekte Datentypen auszuwählen, d. h. solche, die es ermöglichen, jedes auftretende Objekt der Realwelt zu beschreiben.

Die dritte Bedingung (DNF3) ist dagegen von zweifelhaftem Wert. Beispielsweise verstößt die in Abbildung 6.25 dargestellte Situation gegen (DNF3), aber die Aufsummierung zum Schlussverkauf soll gerade nur die Daten dieser Periode erfassen und die anderen unberücksichtigt lassen.

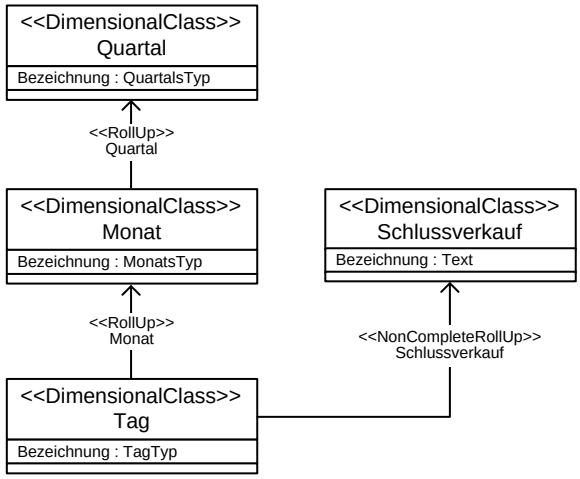



Abbildung 6.25: Dimension mit schwach abhängigen Klassen

Auch für die beiden Kriterien (M2) und (M3) der in (6.7) definierten multidimensionalen Normalformen lassen sich Gegenbeispiele finden, die diese als zu restriktiv erscheinen lassen. So kann die in M2 geforderte Orthogonalität von Dimensionen z. B. in folgendem Szenario verletzt werden: Ein Geschäft hat eine saisonabhängige Angebotspalette: Während im Winterhalbjahr Skiartikel und Zubehör verkauft werden, werden im Sommerhalbjahr Wanderartikel nebst Zubehör veräußert. In diesem Falle lässt sich aus dem Datum der Artikel bestimmen (und umgekehrt). Die Forderung nach Minimalität der Dimensionen (M3) kann in folgendem Szenario hinderlich sein: Es werden personenbezogene Fakten erhoben, eine Dimension „Person“ ist durch ein Attribut „Personalkennziffer“ charakterisiert, das auch in codierter Form das Geschlecht der Person enthält. Dennoch kann es sinnvoll sein, eine weitere Dimension „Geschlecht“ zu modellieren, um potenzielle Auswertungen einfacher gestalten zu können. Zusammenfassend kann festgehalten werden, dass die in [ALW98] geforderten multidimensionalen Normalformen aufgrund der hier aufgeführten Gegenbeispiele weitgehend unbrauchbar sind.

## 6.5 Beispiel: Handelswelt

Zur Verdeutlichung der bisher in diesem Kapitel vorgestellten Konzepte (MML, $_m$UML und Leitfaden) wird in diesem Abschnitt eine Fallstudie vorgestellt, die auch zur Demonstration der weiteren Entwurfsschritte in den Kapiteln 7 bis 10 dienen wird.

### 6.5.1 Szenario

Das in diesem Abschnitt vorgestellte Szenario soll als durchgängiges Beispiel für Teil II dieser Arbeit dienen. Aus der Fülle möglicher Beispielszenarien wurde das betriebswirtschaftliche Szenario „Handelswelt“ gewählt. Dieses Beispiel hat sich (mit einigen Variationen) in der wissenschaftlichen DWH–Literatur [BG01] quasi zum Standardbeispiel entwickelt, weil es einerseits leicht verständlich und überschaubar ist, andererseits aber auch ausreichend komplex, um z. B. alle Modellierungskonstrukte zu demonstrieren. Bei der Wahl der betriebswirtschaftlichen Fachbegriffe wurde ferner auf Übereinstimmung mit [BS96] geachtet.
Das Szenario sei wie folgt definiert: Eine in Deutschland und der Schweiz tätige Lebensmittelkette, die sowohl eigene Filialen betreibt als auch in Kaufhäusern tätig ist, ist an einer Verkaufszahlenanalyse interessiert. Man ist an der Auswertung abgesetzter Produkte nach verschiedenen räumlichen und

zeitlichen Kriterien interessiert. Es ist bekannt, dass unterschiedliche Abteilungen Verkaufszahlen nach unterschiedlichen zeitlichen Perioden benötigen. Produkte bilden immer Produktgruppen, diese Produktfamilien, diese wiederum Produktkategorien.
Auf der räumlichen Ebene ist jede Filiale bzw. jedes Kaufhaus einer Stadt zugeordnet, diese wiederum einem Bundesland (in Deutschland) bzw. einem Kanton (in der Schweiz) und diese schließlich einem Staat. Parallel dazu gibt es für die deutschen Niederlassungen Zusammenfassungen zu Verkaufsbezirken, welche ihrerseits wiederum einem Bundesland zugeordnet sind. Die Niederlassungen in der Schweiz kennen keine vergleichbare Struktur. Die Filialen werden für interne Auswertungen zu Filialkategorien und Filialoberkategorien zusammengefasst.
Weiterhin interessiert sich die Marketingabteilung für die Kombination von Verkäufen (sog. „Basket Analysis“) und es soll ein Vergleich zwischen dem Absatz einer Filiale und der Kaufkraft ihres Einzugbereiches hergestellt werden. Hierfür liegen von einer Konsumforschungsgesellschaft Daten auf der Ebene von Straßenbereichen vor, die vierteljährlich geliefert werden.

### 6.5.2 Modellierung

Auf das Szenario „Handelswelt“ soll nun der Leitfaden aus Abschnitt 6.3 angewendet werden.

#### Schritt 1: Finde Kennzahlen

Als Kennzahlen lassen sich von einem Bon direkt „Anzahl“ sowie der „Einzelpreis“ eines Artikels identifizieren. Um der Anforderung der Marketingabteilung nach Kombinationen von Verkäufen genüge zu tun, ist auch der „Gesamtwert“ eines Verkaufs eine relevante Kennzahl. Um schließlich die Auswertungen auf der Produktpalette durchführen zu können, muss die „Anzahl verkaufter Artikel“ festgehalten werden. Für die extern bezogenen Daten ist „Einkommen“ eine Kennzahl.

#### Schritt 2: Finde Faktklassen

Die „Anzahl“ eines Produktes sowie dessen „Einzelpreis“ treten stets zusammen auf und sind somit Attribute einer *FactClass*. Da zusätzlich der Gesamtpreis einer Position des Bons für Auswertungen relevant ist, wird auch dieses Attribut in die Klasse aufgenommen.
Die drei anderen Kenngzahlen „Gesamtwert“ eines Verkaufs, „Anzahl verkaufter Artikel“ und „Einkommen“ sind jeweils in einer eigenen Klasse anzusiedeln, so dass sich sich das in Abbildung 6.26 dargestellte Bild ergibt.

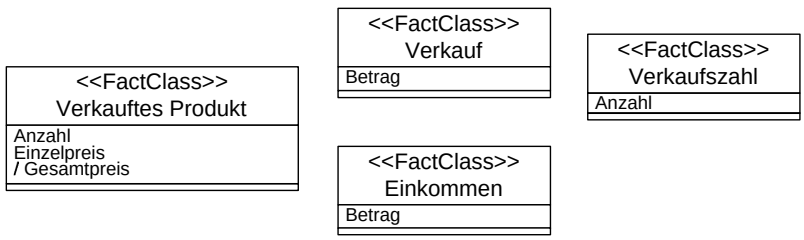



Abbildung 6.26: $_m$UML Beispiel – Ergebnis Schritt 2: Faktklassen

**Schritt 3: Bestimme für jedes Faktattribut den Datentyp**

Die Attribute bekommen ihren Datentyp und das abgeleitete Attribut zusätzlich eine Berechnungsvorschrift und die Parameter, was zum Resultat in Abbildung 6.27 führt.

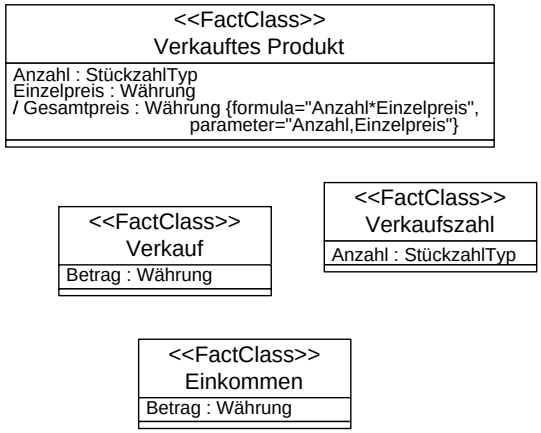



Abbildung 6.27: $_m$UML Beispiel – Ergebnis Schritt 3: Faktattribute mit Datentyp

**Schritt 4: Finde Beziehungen zwischen Faktklassen**

Ein „Verkauf“ setzt sich aus mehreren verkauften Artikeln zusammen, was sich in einer Komposition zwischen den beiden Klassen widerspiegelt. Das Resultat ist in Abbildung 6.28 zu sehen.

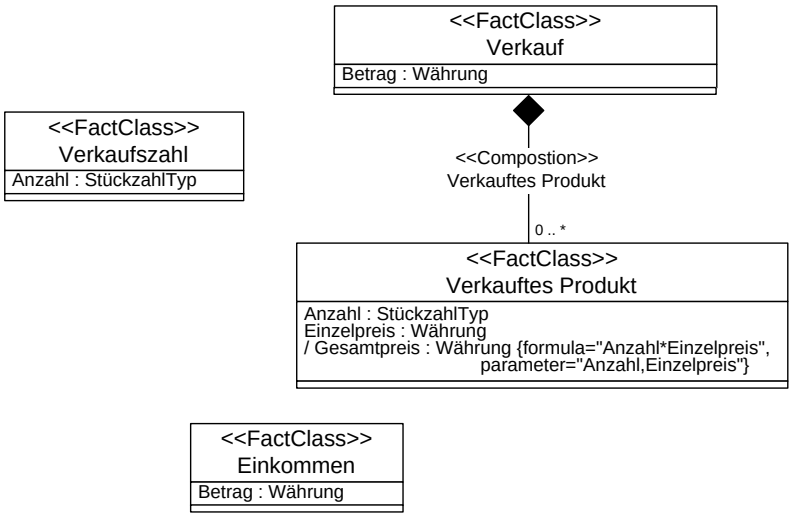



Abbildung 6.28: $_m$UML Beispiel – Ergebnis Schritt 4: Beziehungen zwischen Faktklassen

**Schritt 5: Finde zu jeder Faktklasse Dimensionen**

Aus den Anforderungen ergibt sich, dass das „Verkaufte Produkt“ durch das „Produkt“ und der „Verkauf“ durch den „Ort“ und die „Zeit“ qualifizierend beschrieben werden. Die Anzahl verkaufter Produkte erhält die Dimensionen „Zeit“, „Ort“ und „Produkt“. Das Einkommen wird durch den „Ort“ und die „Zeit“ genauer beschrieben, so dass sich zusammenfassend die in Abbildung 6.29 gezeigten Dimensionen ergeben. In dieser Abbildung sind die Dimensionen pro Faktklasse dargestellt. Ob tatsächlich eine gemeinsame „Zeit“– oder „Ort“–Dimension benutzt wird, wird erst im folgenden Schritt entschieden. Dann werden auch die Namenskonflikte in Form mehrerer Dimensionen mit gleichem Namen aufgelöst.

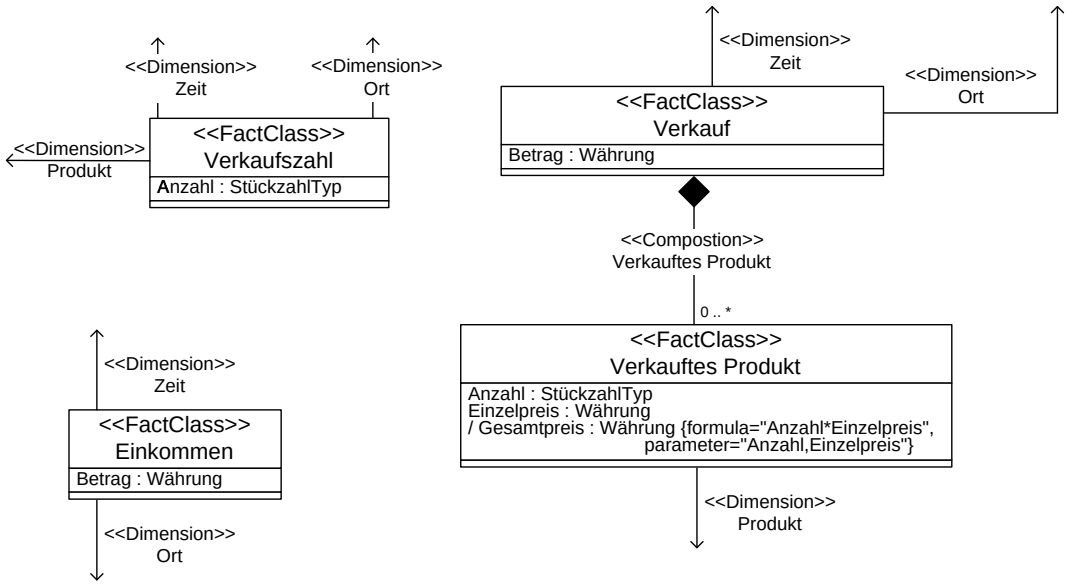



Abbildung 6.29: $_m$UML Beispiel – Ergebnis Schritt 5: Dimensionen

**Schritt 6: Bestimme für jede Dimension die Ebene der feinsten Granularität**

Die Ebenen der feinsten Granularität ergeben sich aus den in Abschnitt 6.5.1 beschriebenen Anforderungen. Für die drei Faktklassen „Verkauf“, „Verkauftes Produkt“ und „Verkaufszahl“ können die feingranularsten Dimensionsklassen gemeinsam genutzt werden. Die Faktklasse „Einkommen“ hingegen bildet hier eine Ausnahme.
Um im Schema eindeutige Namen zu erreichen, sind die Dimensionsbeziehungen im Gegensatz zum fünften Schritt umzubenennen. Die Tatsache, dass sowohl Filialen wie auch Abteilungen in Kaufhäusern existieren, führt schließlich dazu, dass auf der Ortshierarchie eine abstrakte Klasse als „Ort des Verkaufs“ modelliert wird. Das Ergebnis von Schritt 6 zeigt Abbildung 6.30.

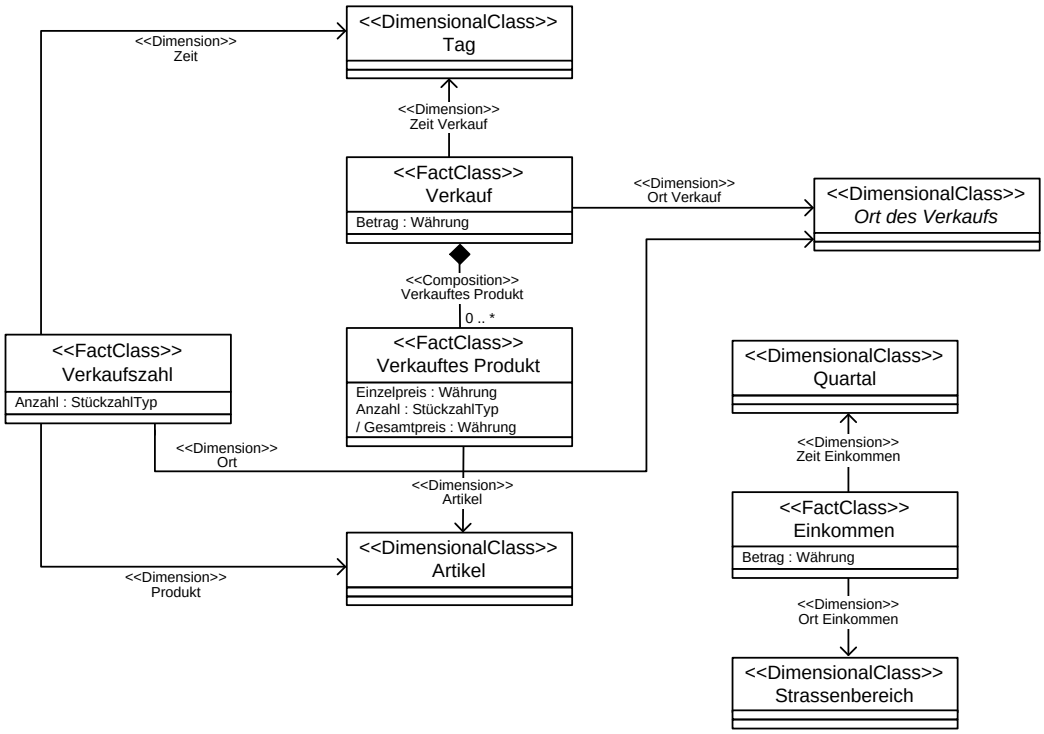



Abbildung 6.30: $_m$UML Beispiel – Ergebnis Schritt 6: Ebenen der feinsten Granularität

**Schritt 7: Bestimme für jede Kombination Faktattribut–Dimension zulässige Operatoren**

In diesem Beispiel ist es sinnvoll die Operationen „SUM“, „MIN“, „MAX“ und „AVG“ zuzulassen, weil sie für jedes Faktattribut bez. jeder Dimension semantisch sinnvolle Auswertungen liefern.

**Schritt 8: Finde für jede Dimension weitere Klassen**

Aufgrund der Anforderung „ ... nach verschiedenen räumlichen und zeitlichen Kriterien ...“ ergeben sich die in Abbildung 6.31 aufgelisteten weiteren Klassen.

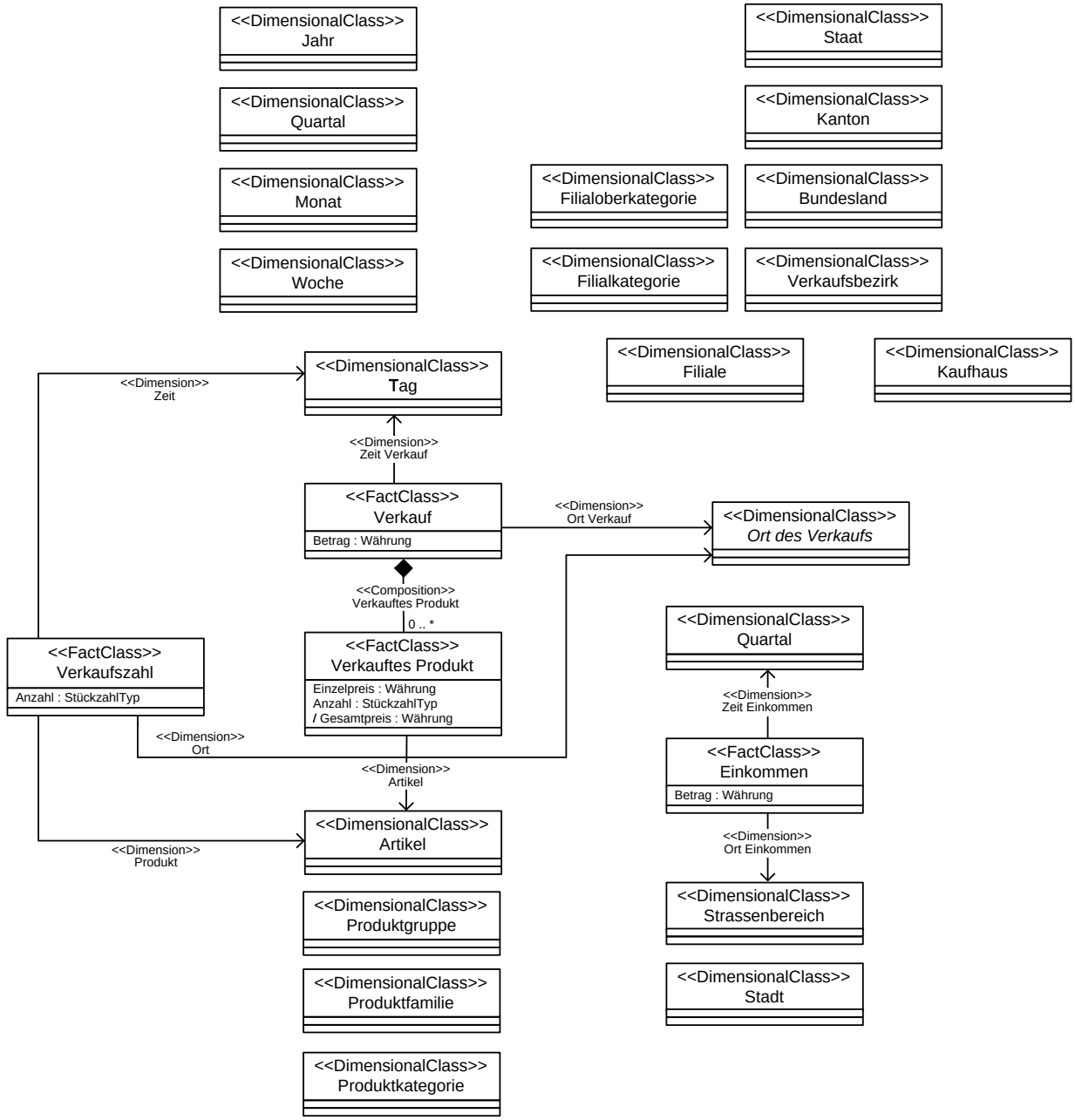



Abbildung 6.31: $_m$UML Beispiel – Ergebnis Schritt 8: Dimensionale Klassen

**Schritt 9: Finde innerhalb dieser dimensionalen Klassen Vererbungshierarchien**

Offenbar sind „Filiale“ und „Kaufhaus“ auf der untersten Ebene der Ortsdimension angesiedelte Klassen. Unter der Annahme, dass sich die beiden Klassen in ihren beschreibenden Attributen unterscheiden, kann hier Vererbung mit einer abstrakten Oberklasse zum Einsatz kommen (siehe Abbildung 6.32).

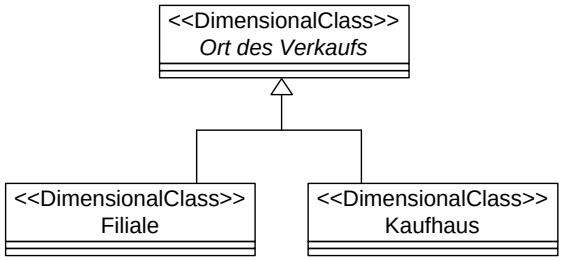



Abbildung 6.32: $_m$UML Beispiel – Ergebnis Schritt 9: Vererbung zwischen dimensionalen Klassen

**Schritt 10: Finde innerhalb dieser dimensionalen Klassen Hierarchie–Beziehungen**

Die Hierarchie der Produktdimension ist aufgrund der Vorgaben relativ einfach zu finden, in der Zeitdimension spielt die „Woche“ eine Sonderrolle, denn sie lässt sich nicht eindeutig auf eine Klasse gröberer Granularität abbilden, so dass hier der *SharedRollUp*–Operator zum Einsatz kommt. In der Ortsdimension schließlich tritt der Fall einer unbalancierten Hierarchie auf, weil aufgrund der Problembeschreibung in Abschnitt 6.5.1 in der Schweiz die Ebene der „Verkaufsregion“ nicht existiert. Eine wesentliche Entwurfsentscheidung an dieser Stelle ist die Frage, ob „Kanton“ und „Bundesland“ in separaten Hierarchiepfaden (Abbildung 6.33(a)) oder zusammengefasst modelliert werden sollen. Im letzten Fall besteht wiederum die Alternative zwischen einer Klasse mit aufzählendem Typ–Attribut (Abbildung 6.33(b)) und dem Einsatz der Vererbung (Abbildung 6.33(c)).

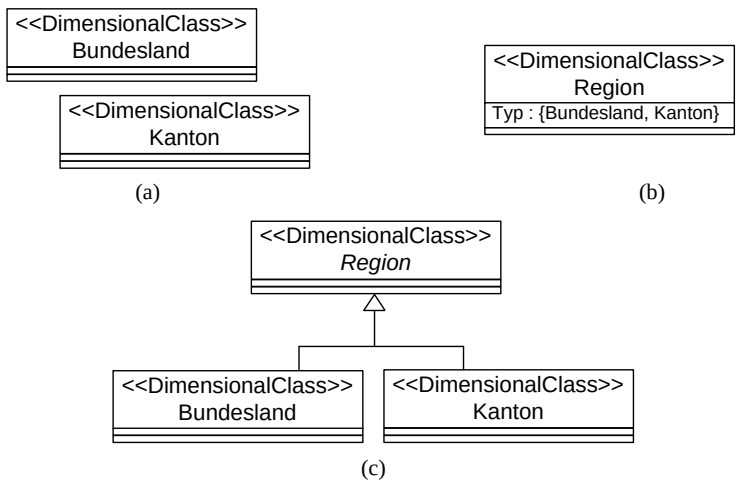



Abbildung 6.33: $_m$UML Beispiel – Schritt 10: Alternativen

Die Entscheidung hängt vom Wissen des Modellierers über den Typ der späteren Analysen zum Zeitpunkt der konzeptionellen Modellierung ab. Im Falle von (a) können die Verkaufszahlen aus Kantonen und Bundesländer nur schwer gegenübergestellt werden (vielleicht will man das aufgrund zu großer Unterschiede oder organisatorischer Richtlinien aber auch gerade verhindern), bei (b) und (c) ist dies hingegen möglich. Bei Auswahl zwischen (b) und (c) spielen wieder die in Abschnitt 6.3 unter Schritt 9 genannten Argumente zum „Vererbungsmissbrauch“ eine Rolle. Die Entscheidung soll im Beispiel zugunsten von Variante (b) fallen, weil die Daten miteinander vergleichbar sein sollen und diese Lösung die beste Erweiterungsmöglichkeit bietet. Das Resultat der bisherigen Modellierung ist in Abbildung 6.34 zu sehen.

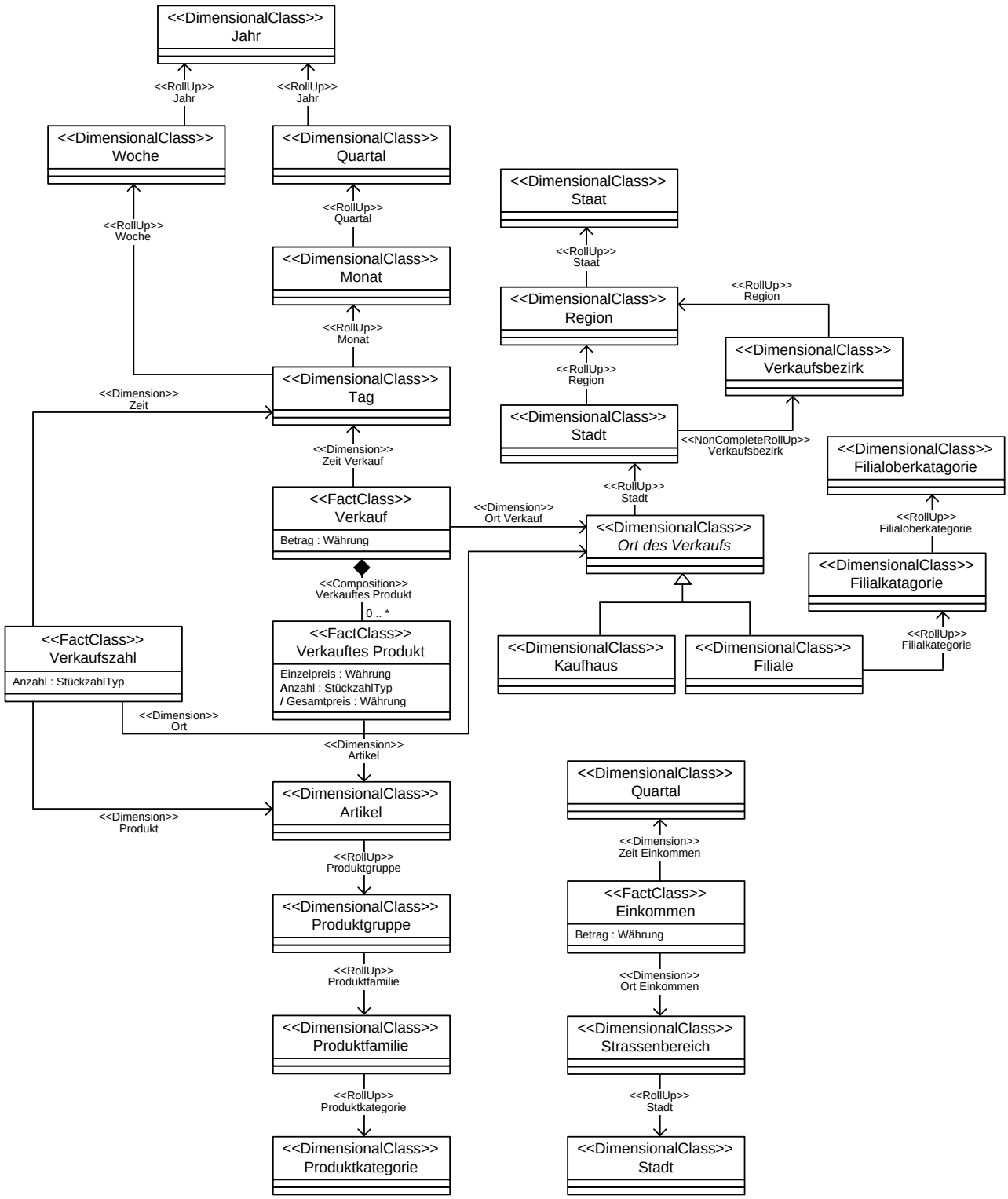



Abbildung 6.34: $_m$UML Beispiel – Ergebnis Schritt 10: Hierarchiepfade

**Schritt 11: Finde für die dimensionalen Klassen weitere Attribute**

Jedes *DimensionalClass*–Schemaelement bekommt ein Attribut „Bezeichnung“ vom Typ „Text“, das auch jeweils die Schlüsseleigenschaft erhält. Zusätzliche beschreibende Attribute werden teilweise eingefügt, z. B. bei der „Stadt“ die „PLZ“ (siehe Abbildung 6.35(a)).
Eine Besonderheit bildet die Beschreibung einer „Filiale“, die sich aus dem „Filialleiter“ und der „Filialart“ zusammensetzt. Hierfür wird eine eigenes *DataClass*–Objekt angelegt, das dann von einem entsprechenden Attribut in der dimensionalen Klasse „Filiale“ als Datentyp verwendet wird (siehe Abbildung 6.35(b)).

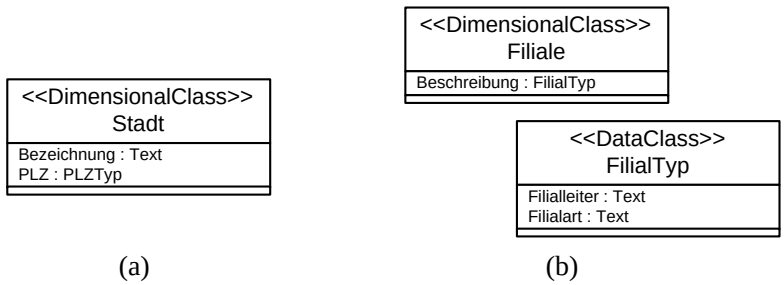



Abbildung 6.35: $_m$UML Beispiel – Schritt 11: Datenklasse

**Schritt 12: Bestimme Abhängigkeiten zwischen Attributen in dimensionalen Klassen**

Aufgrund der in Schritt 10 getroffenen Entscheidung der Modellierung von Bundesländern und Kantonen kann es notwendig sein, für den einen oder anderen Typ spezifische Attribute festzuhalten, z. B. könnten für Bundesländer spezielle Informationen vorliegen, die es für Kantone nicht gibt.

**Schritt 13: Bestimme für die Attribute der dimensionalen Klassen den Datentyp**

Dieses war im Beispiel schon implizit in den Schritten 11 bzw. 12 erfolgt.

**Schritt 14: Finde Assoziationen zwischen dimensionalen Klassen**

Hier spielt die Anforderung eine Rolle, dass nicht jedes Produkt an jedem Ort erhältlich ist. Dieses wird durch eine *Viele–zu–Viele*–Assoziation „Geführtes Produkt“ zwischen „Artikel“ und „Ort des Verkaufs“ zum Ausdruck gebracht, wie in Abbildung 6.36 dargestellt.

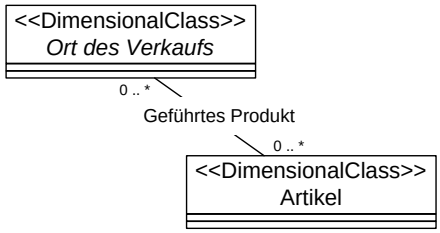



Abbildung 6.36: $_m$UML Beispiel – Ergebnis Schritt 14: Assoziation zwischen dimensionalen Klassen

**Schritt 15: Finde gleiche und ähnliche Dimensionen**

Während gleiche Dimensionen nicht zu finden sind bzw. die „Zeit“– und „Produkt“–Dimension schon mit mehreren *FactClasses*–Instanzen verbunden sind, sind die beiden Ortsdimensionen gemäß der Definition in Abschnitt 6.3 Schritt 15 *ähnlich*. Es ist also an geeigneter Stelle, d. h. auf einer für potenzielle Analysen sinnvollen Granularität eine Verbindung in Form eines *DimensionalMapping*–Objekt darzustellen (siehe Abbildung 6.37).

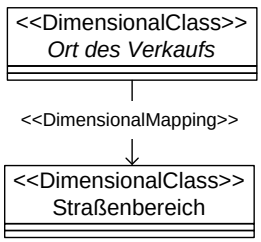



Abbildung 6.37: $_m$UML Beispiel – Ergebnis Schritt 15: *DimensionalMapping* zwischen Ortsdimensionen

**Schritt 16: Bestimme Subschemata**

Zur Erhaltung der Übersichtlichkeit des Schemas wird für jede dimensionale Struktur ein Subschema gebildet. Das resultierende Schema ist in Anhang A zu sehen.

**Schritt 17: Überprüfung der Dokumentation**

Ebenfalls in Anhang A ist eine vervollständigte Dokumentation zu finden.

**Schritt 18: Iteration**

Zusätzliche Iterationen sind nicht notwendig, weil die in Abschnitt 6.5.1 definierten Anforderungen bereits alle im Schema erfüllt sind.

## 6.6 Zusammenfassung

Kapitel 6 hat die konzeptionelle Entwurfsebene behandelt. Dazu wurde in Abschnitt 6.1 mit der MML eine Sprache eingeführt, die sowohl multidimensionale wie auch objektorientierte Konstrukte enthält. In die Konzeption der MML sind die in Abschnitt 3.2 herausgearbeiteten Datenmodellanforderungen eingeflossen. Inwiefern die einzelnen Aspekte berücksichtigt worden sind, ist in den Tabellen 6.5 und 6.6 wiedergegeben.

**Anforderungen an Fakten**

| **Anforderung** | **Realisierung in der MML** |
|---|---|
| Ein (Daten–)Würfel bzw. ein Schema sollte *beliebig viele Fakten* enthalten dürfen. | Ein $_m$UML–Diagramm darf beliebig viele Faktklassen enthalten. |
| Ein (Daten–)Würfel bzw. ein Fakt sollte *beliebig viele Kennzahlen* enthalten dürfen. | Jede Faktklasse darf in der MML beleibig viele Attribute umfassen. |
| *Beziehungen zwischen Fakten*, wie z. B. Spezialisierungen oder Aggregationen, sollten explizit dargestellt werden können. | Zwischen Faktklassen ist die Definition von Vererbungen und Aggregationen möglich. |
| Ein Fakt sollte eine *innere Struktur* besitzen, denn Fakten bestehen nur in Ausnahmefällen aus einer einzigen numerischen Kennzahl. Ein Fakt kann einerseits sowohl mehrere Attribute besitzen, die beispielsweise in Form einer Verbundstruktur angeordnet sind, und zum anderen können auch textuelle „Kennzahlen“ oder Eigenschaften, deren Domäne ein Aufzählungstyp ist, existieren. | Der Datentyp von Faktattributen muss nicht unbedingt numerisch sein, ebenso können hier mit Hilfe der Klasse *DataClass*–Objekten komplexe Datentypen verwendet werden. |
| Die Markierung *abgeleiteter Attribute* und die Angabe der Berechnungsvorschrift sollten im Datenschema möglich sein. | Die $_m$UML nutzt hier die Möglichkeit der UML, berechnete Attribute mit einem Schrägstrich zu kennzeichen und die Berechnungsvorschrift in Form einer Annotation vorzunehmen. |
| Die *Additivität von Kennzahlen* sollte im Schema explizit angegeben werden können. | Hierzu existiert in der MML die Metaklasse *Additivity*, die es ermöglicht, jeder Faktattribut–Dimension–Kombination eine Menge von Verdichtungsoperatoren zuzuweisen. |
| Auch Kennzahlen sollten eine *innere Struktur* besitzen können. | Hierzu können Faktattribute einen komplexen Datentyp besitzen. |

Tabelle 6.5: Datenmodellanforderungen an Fakten und ihre Erfüllung in der MML

## Anforderungen an Dimensionen

| Anforderung | Realisierung in der MML |
|---|---|
| Die Modellierung von *Hierarchieebenen* sollte möglich sein. | In der MML werden Hierarchieebenen durch *DimensionalClass*–Schemaelemente dargestellt. |
| Zwischen Hierarchieebenen sollte die Darstellung von *Verdichtungspfaden* möglich sein. | Mit den Konstrukten *RollUp*, *NonCompleteRollUp* und *SharedRollUp* stehen in der MML drei Möglichkeiten zur Verfügung, unterschiedliche Typen von Verdichtungspfaden zu modellieren. |
| Die Struktur sollte ein *gerichteter azyklischer Graph* sein. | Die Dimensionsstruktur kann ein solcher Graph sein, wodurch die Zusammenführung von Verdichtungspfaden und die gemeinsame Nutzung von Hierarchieebenen möglich ist. |
| Darüber hinaus können zwischen Hierarchieebenen weitere Beziehungen wie *Generalisierungen* und *Assoziationen* bestehen. , die ebenfalls explizit formuliert werden sollten. | Zwischen *DimensionalClass*–Schemaelementen sind sowohl Generalisierungen als auch Assoziationen möglich. |
| In der Hierarchiestruktur sollten *Mehrfachhierarchien* möglich sein. | Mit Hilfe von *RollUp*–Verbindungen zwischen *DimensionalClass*–Objekten können Mehrfachhierarchien realisiert werden. |
| In der Hierarchiestruktur sollten *alternative Verdichtungspfade* möglich sein. | Mit Hilfe von *RollUp*–Verbindungen zwischen *DimensionalClass*–Objekten können alternative Verdichtungspfade realisiert werden. |
| In der Hierarchiestruktur sollten *unbalancierte Hierarchien* möglich sein. | Mit Hilfe von *RollUp*–Verbindungen zwischen *DimensionalClass*–Objekten können Hierarchien realisiert werden, die dann später auf Instanzebene unbalanciert sind. |
| Bei der Verdichtung sollte die Darstellung *anteiliger Verrechnung* möglich sein. | Hierfür stellt die MML das *SharedRollUp*–Konstrukt zur Verfügung, mit dem eine die anteilige Verrechnung spezifizierende Berechnungsvorschrift verknüpft ist. |
| Die Modellierungssprache sollte ein Konzept zur Handhabung *nicht–vollständiger Verdichtungen* anbieten. | In der MML werden nicht–vollständige Verdichtungen mittels eines *NonCompleteRollUp*–Schemaelementes gekennzeichnet. |
| Für jede Hierarchieebene sollten darüber hinaus *zusätzliche Attribute* definierbar sein, die nicht unmittelbar der Hierarchiebildung dienen. | Jede dimensionale Klasse kann beliebig viele weitere Attribute besitzen, die nicht unmittelbar an der Hierarchiebildung beteiligt sind. |
| Für jede Hierarchieebene sollten *Schlüsselattribute* definiert werden können. | Attribute eines *DimensionalClass*–Schemaelements können als Schlüssel spezifiziert werden. |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | |
|---|---|
| Innerhalb einer Hierarchieebene können Datenobjekte mit unterschiedlichen Dimensionsattributen existieren, was insbesondere bei Hierarchiebildungen Probleme nach sich ziehen kann, so dass die Modellierung *unterschiedlicher Datenobjekttypen* unterstützt werden sollte. | Durch Generalisierungen können unterschiedliche Typen realisiert werden. Da auch Unterklassen Ausgangspunkt einer Verdichtung sein können, sind spezielle Verdichtungen für spezielle Typen möglich. |
| Insbesondere sollten auch *optionale Dimensionsattribute* definiert werden können. | Attribute können als optional gekennzeichnet werden. |

Tabelle 6.6: Datenmodellanforderungen an Dimensionen und ihre Erfüllung in der MML

Darüber hinaus ist die MML eine objektorientierte Sprache, was folgende Vorteile mit sich bringt:

- Durch den Einsatz der Objektorientierung wird *größtmögliche Unabhängigkeit* vom Zielsystem erreicht.
- Durch die Vielfalt der objektorientierten Konstrukte lassen sich viele Aspekte in *natürlicher*, dem Benutzerdenken nahen *Weise* modellieren.
- Objektorientierte Schemata lassen sich einfacher in verteilte Objekt– und Komponentenframeworks einer *organisationsweiten Entscheidungsunterstützung* integrieren.
- Es bleibt die Möglichkeit offen, weitere Methoden der UML, wie z. B. *Use Cases* oder *Interaktionsdiagramme* in die Modellierung einzubeziehen.

Aufbauend auf der MML können verschiedene graphische Notationen Anwendung finden, wobei in Abschnitt 6.2 mit der $_m$UML exemplarisch eine Erweiterung der UML vorgenommmen wurde, wobei als wesentliches Instrument zur Erweiterung das der UML inhärente Konzept der Stereotype verwendet wurde. Als Hilfsmittel zur korrekten Anwendung von MML und $_m$UML wurde in Abschnitt 6.3 eine konstruktive Vorgehensweise in Form eines konkreten Leitfadens vorgeschlagen, der in 18 Schritten zu einem MML–Schema führt. Dieser Leitfaden realisiert eine Bottom–Up–Strategie und hilft von den Fakten ausgehend systematisch ein multidimensionales Schema zu modellieren. Abschnitt 6.4 behandelte die analytische Qualitätssicherung eines konzeptionellen Schemas mittels Reviews. Hierbei wurde zunächst ein Metamodell für den Review–Schritt vorgeschlagen und der Ablauf eines Reviews skizziert. Anschließend wurde eine Reihe möglicher Kriterien inkl. möglicher Messverfahren ebenso aufgezählt wie Kriterien, die für ein Review eher ungeeignet sind. Als vielleicht etwas überraschendes Resultat wird festgestellt, dass die im relationalen Modell wichtigen Normalformen im multidimensionalen Fall keine große Bedeutung haben. Im abschließenden Abschnitt 6.5 wurde die Anwendungswelt „Handelswelt“ eingeführt, die dem gesamten Teil II als durchgängiges Beispiel dienen wird. Schließlich wurde mit Hilfe des Leitfadens ein konzeptionelles Schema für dieses Szenario erstellt.

# Kapitel 7

# Logischer Entwurf

Dieses Kapitel widmet sich dem logischen Entwurfsschritt, dessen Einordnung in den Entwurfsprozess in Abbildung 7.1 dargestellt ist.

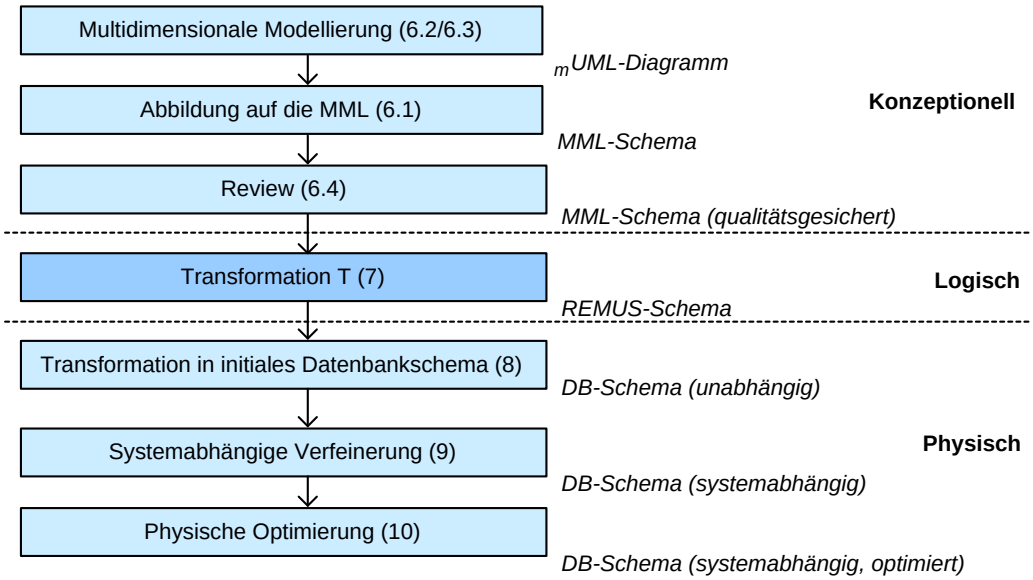



Abbildung 7.1: Einordnung des Schrittes in den Entwurfsprozess

Aufgrund des Fazits von Abschnitt 4.1 besitzen relationale Implementierungen für DWH die größte Bedeutung, weil ROLAP– und HOLAP–Systeme die erfolgversprechendsten Realisierungsformen sind.
Zielschema des logischen Entwurfs ist eine erweiterte Form von Relationenschema. Dieser als *REMUS* (Relational Schema for Multidimensional Purpose) bezeichnete Schematyp wird in Abschnitt 7.1 eingeführt, Abschnitt 7.2 beschreibt den Transformationsalgorithmus und demonstriert ihn anhand von Ausschnitten des in Abschnitt 6.5.2 eingeführten Beispiels „Handelswelt“. Abschnitt 7.3 skizziert kurz die Transformation in nicht–relationale Datenmodelle, bevor das Kapitel mit einer Zusammenfassung in Abschnitt 7.4 schließt.

## 7.1 REMUS: Relational Schema for Multidimensional Purpose

In den folgenden beiden Unterabschnitten werden mit Objekten und Attributen (Abschnitt 7.1.1) sowie den unterschiedlichen Metadatentypen (Abschnitt 7.1.2) die Bestandteile eines REMUS–Schemas eingeführt.

### 7.1.1 Schema, Objekte und Attribute

Ein REMUS–Schema besteht in Ergänzung zu herkömmlichen Relationenschemata vor allem aus vielfältigen Metadaten, die die multidimensionalen und objektorientierten Sachverhalte des konzeptionellen Entwurfs festhalten. Außerdem wird in einem REMUS–Schema das Konzept der *Relation* durch den allgemeineren Oberbegriff *Objekt* ersetzt. Dies ist im Hinblick auf eine spätere Erweiterbarkeit der Methodik geschehen.
Damit ergibt sich das in Abbildung 7.2 dargestellte Metaschema: Ein benanntes REMUS–Schema setzt sich aus einer Menge von Objekten, Attributen und Metadaten zusammen. Die Metadaten werden weiter in Kategorie A– und Kategorie B–Metadaten unterschieden. Kategorie A–Metadaten beschreiben dabei Eigenschaften jeweils einzelner Objekte oder Attribute, während Kategorie B–Metadaten Eigenschaften beschreiben, die sich auf jeweils zwei Objekte beziehen. Objekte werden in REMUS durch ihren Namen beschrieben, Attribute durch Angabe von Namen und Datentyp.

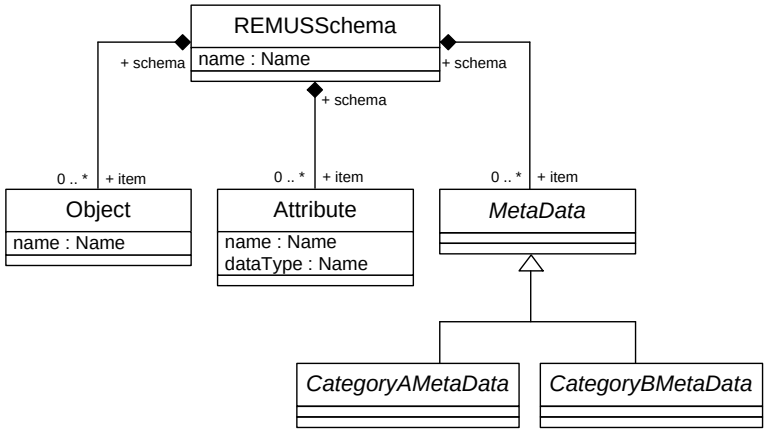



Abbildung 7.2: REMUS–Metaschema

### 7.1.2 Metadatentypen

Zur Darstellung der Metadaten wird an dieser Stelle eine Tupelnotation verwendet. Die unterschiedlichen Typen von Kategorie A–Metadaten sind in Tabelle 7.1 mit Erläuterungen aufgeführt. Eine ausführliche Spezifikation mit Metaklassendiagramm und Klassenbeschreibungen kann in [Her01c] nachgelesen werden.

**REMUS: Kategorie A–Metadaten**

| **AggregatedAttribute** |
|---|
| $(R, \text{AggregatedAttribute}, (A, M))$<br>Das Attribut $A$ der Relation $R$ ist ein aggregiertes Attribut und hat die Multiplizität $M$. |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

**Computation**

$(C, \text{Computation}, ((P_1, ..., P_n), F, R))$
$C$ ist der Name einer Berechnungsvorschrift, die die Parameter $P_1$ bis $P_n$ hat und deren Resultat in $R$ gespeichert wird. $F$ gibt die Formel an, z. B. in Form eines Methodennamens.

**ConceptualKey**

$(R, \text{ConceptualKey}, (A))$
Das Attribut $A$ ist konzeptioneller Schlüssel der Relation $R$. Der konzeptionelle Schlüssel setzt sich aus der Menge ConceptualKey–Metadaten zusammen, die $R$ referenzieren.

**Identifier**

$(R, \text{Identifier}, (A))$
In der Relation $R$ ist das Attribut $A$ diskriminierendes Attribut.

**IdentifierValue**

$(A, \text{IdentifierValue}, (\{V_1, ..., V_n\}))$
Das als Identifier gekennzeichnete Attribut $A$ hat die zulässigen Werte $V_1$ bis $V_n$.

**Multiplicity**

$(R, \text{Multiplicity}, (\{A_1, ..., A_n\}, M))$
In der Relation $R$ darf die Attributkombination $\{A_1, ..., A_n\}$ so oft auftreten wie die Multiplizität $M$ angibt.

**ObjectType**

$(O, \text{ObjectType}, (T, D))$
Das ObjectType–Metadatum beschreibt das Objekt $O$ genauer. $T$ gibt den Typ an, $D$ eine Beschreibung. Zulässige Werte für $T$ sind „Relation“ und „DataType“. Im ersten Fall sind als Werte für $D$ „Fact“, „Dimension“ und „Data“ möglich, im zweiten Fall bleibt $D$ ohne Bedeutung.

**Optional**

$(R, \text{Optional}, (A))$
Das Attribut $A$ in der Relation $R$ ist optional.

**PrimaryKey**

$(R, \text{PrimaryKey}, (A))$
Das Attribut $A$ ist Primärschlüsselattribut in der Relation $R$. Der gesamte Primärschlüssel von $R$ setzt sich aus allen PrimaryKey–Metadaten zusammen, die $R$ referenzieren.

**Reference**

$(A_1, \text{Reference}, (A_2))$
Das Attribut $A_1$ wird als Fremdschlüsselattribut von $A_2$ referenziert. $A_2$ muss durch ein entsprechendes Metadatum als Primärschlüsselattribut gekennzeichnet sein.

**Valid**

$(A, \text{Valid}, (I, \{V_1, ..., V_n\}))$
Das Attribut $A$ ist nur gültig, wenn das diskriminierende Attribut $I$ einen der Werte $V_1$ bis $V_n$ besitzt.

Tabelle 7.1: REMUS: Kategorie A–Metadaten

Analog werden in Tabelle 7.2 die Kategorie B–Metadaten in einer Tupelnotation eingeführt.

## REMUS: Kategorie B–Metadaten

**Additivity**

$(A, R, \text{Additivity}, (D, F, O))$

Das Faktattribut $A$ der Faktrelation $F$ hat bezüglich der dimensionalen Relation $R$, mit der $F$ mittels der Dimension $D$ in Beziehung steht, die Menge $O$ als zulässige Verdichtungsoperatoren.

**Association**

$(R_1, R_2, \text{Association}, (R_3, L_1, L_2, T_1, T_2, (P_{R_1}), (F_{R_1}), (P_{R_2}), (F_{R_2})))$

Zwischen den beiden Relationen $R_1$ und $R_2$ ist eine Assoziation definiert, die mittels der Zwischenrelation $R_3$ aufgelöst worden ist. $L_1$ und $L_2$ beschreiben die Rollen der beiden Relationen $R_1$ und $R_2$ innerhalb der Assoziation, die Mengen $T_1$ und $T_2$ zählen die zulässigen Typen auf. $P_{R_1}$ und $F_{R_1}$ sind Primär– und Fremdschlüssel von Relation $R_1$ zur Zwischenrelation, $P_{R_2}$ und $F_{R_2}$ entsprechend für $R_2$.

**Composition**

$(R_1, R_2, \text{Composition}, (C, M))$

Zwischen den Relationen $R_1$ und $R_2$ besteht eine Komposition $C$ mit der Multiplizität $M$, $R_2$ ist die Detailrelation dieser Komposition.

**Dimension**

$(R_1, R_2, \text{Dimension}, (D, T_1, T_2, F, P))$

Zwischen der Faktrelation $R_1$ und der dimensionalen Relation $R_2$ existiert eine Dimension $D$. Die Mengen $T_1$ und $T_2$ zählen die zulässigen Typen der Relationen $R_1$ bzw. $R_2$ auf. $F$ und $P$ sind Primär– und Fremdschlüssel, über die die Beziehung definiert ist.

**DimensionalMapping**

$(R_1, R_2, \text{DimensionalMapping}, (D, T_1, T_2, C))$

Zwischen den beiden dimensionalen Relationen $R_1$ und $R_2$ ist ein DimensionalMapping mit dem Namen $D$ definiert. Die Mengen $T_1$ und $T_2$ zählen die zulässigen Typen der Relationen $R_1$ bzw. $R_2$ auf. $C$ gibt die Berechnungsvorschrift an.

**RollUp**

$(R_1, R_2, \text{RollUp}, (R, T_1, T_2, F, P, S))$

Zwischen den dimensionalen Relationen $R_1$ und $R_2$ ist ein Verdichtungspfad mit dem Namen $R$ definiert. Die Mengen $T_1$ und $T_2$ zählen die zulässigen Typen der Relationen $R_1$ bzw. $R_2$ auf. $F$ und $P$ sind Primär– und Fremdschlüssel, über die die Beziehung definiert ist. $S$ gibt an, ob es sich um eine (nicht–)vollständige Verdichtung handelt.

**SharedRollUp**

$(R_1, R_2, \text{SharedRollUp}, (S, T_1, T_2, C, O))$

Zwischen den dimensionalen Relationen $R_1$ und $R_2$ ist ein Verdichtungspfad mit anteiliger Verrechnung mit dem Namen $S$ definiert. Die Mengen $T_1$ und $T_2$ zählen die zulässigen Typen der Relationen $R_1$ bzw. $R_2$ auf. $C$ gibt die Berechnungsvorschrift an, die Menge $O$ die zulässigen Verdichtungsoperatoren.

Tabelle 7.2: REMUS: Kategorie B–Metadaten

## 7.2 Transformationsalgorithmus

Hauptziel der Transformation eines MML– ein REMUS–Schema ist das Erhalten während der konzeptionellen Modellierung verwendeter objektorientierter und multidimensionaler Aspekte. Während der Transformation muss jeder MML–Schemaelementtyp behandelt werden, wobei die in Abbildung 7.3 dargestellte Reihenfolge der Abarbeitung gewählt wird. Zunächst werden in den beiden ersten Schritten Datentypen und –klassen behandelt, dann folgt in den Schritten 3 bis 7 die Transformation von dimensionalen Klassen mit ihren Attributen und Hierarchiestrukturen. Schritt 8 sorgt für das Übertragen der Faktklassen inklusive Faktattributen, bevor die letzten beiden Schritte 9 und 10 durch Transformieren der Dimensionseinstiege und zulässigen Verdichtungsoperatoren für die Verbindung von dimensionalen Relationen und Faktrelationen im erzeugten REMUS–Schema sorgen.

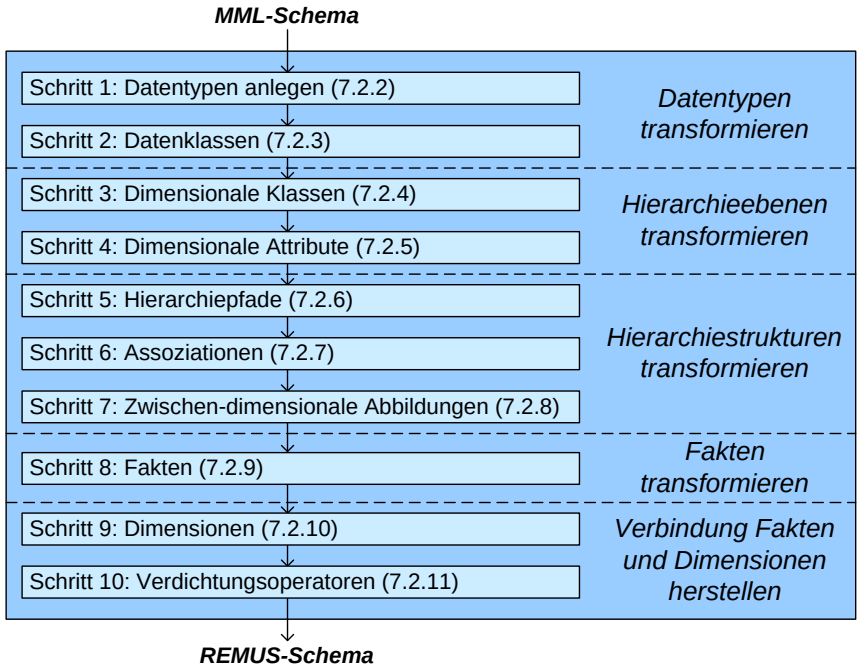



Abbildung 7.3: Ablauf der Transformation von MML nach REMUS

Weil neben einer verbalen auch eine formale Beschreibung der Transformation in einer mengenorientierten Notation erfolgen soll, werden in Abschnitt 7.2.1 zunächst einige Definitionen vorgenommen. Jeder der Abschnitte 7.2.2 bis 7.2.11 behandelt einen Schritt der Transformation. Hierbei wird für jeden Teilschritt zunächst eine informale Beschreibung gegeben, zur Illustration dient jeweils ein Ausschnitt (evtl. in leichter Variation) aus dem Beispiel „Handelswelt“. Dabei werden Skizzen gezeigt, die wie Abbildung 7.4 aufgebaut sind: Das Teilschema, das die zu transformierenden Elemente enthält, ist zur besseren Lesbarkeit sowohl in $_m$UML –Notation als auch in Form von MML–Schemaelementen dargestellt. Die im betreffenden Schritt transformierten Elemente sind jeweils grau hinterlegt. Ein Pfeil symbolisiert den Transformationsschritt, am Ende des Pfeiles werden die in diesem Schritt erzeugten REMUS–Schemaelemente aufgeführt. Fließen Informationen aus früheren Schritten in den Transformationsschritt ein, so werden diese am Pfeil notiert. Im Anschluss an diese Beschreibung erfolgt innerhalb eines jeden Teilschritts die formale Definition der jeweiligen Transformation.

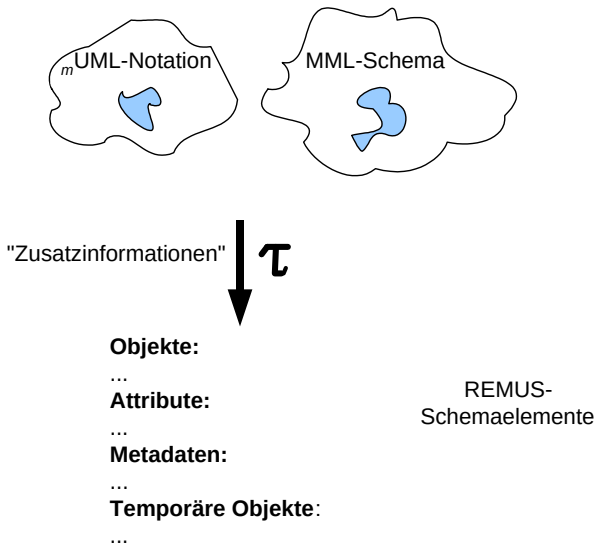



Abbildung 7.4: Darstellung einzelner Transformationsschritte

### 7.2.1 Definitionen und Notationsvereinbarungen

In diesem Teilabschnitt werden mit MML– und REMUS–Schema Ausgangs– und Zielschema sowie einige während der Transformation benötigte Hilfsabbildungen definiert.

**MML**

In (7.1) erfolgt zunächst die Definition eines MML–Schemas.

Ein *MML–Schema* ist ein Paar $M = (S, type)$ mit
(i) $S$ ist eine endliche, nichtleere Menge von Schemaelementen,
(ii) $type$ ist eine partielle Funktion, die jedem $s \in S$ seinen Typ zuweist, d. h.

$$\begin{aligned} type : S \rightarrow \{&"FactClass", "DimensionalClass", "DataClass",\\ &"FactAttribute", "DimensionalAttribute",\\ &"DataAttribute", "DataType", "Dimension",\\ &"RollUp", "SharedRollUp", "DimensionalMapping",\\ &"Additivity", "Computation", "Generalization",\\ &"Association", "Composition"\} \end{aligned} \tag{7.1}$$

$type(s) \stackrel{def}{=}$ "Objekttyp von $s$".

Als Eingabe der Transformation sind nur MML–Schemata zugelassen, die der Spezifikation gemäß Metaklassendiagramm und den Wohlgeformtheitseigenschaften genügen. Aus diesem Grunde wird in (7.2) die Eigenschaft *Gültigkeit* für MML–Schemata eingeführt.

Sei $M$ ein MML–Schema.

$M$ heißt *gültig* $\overset{def}{\Longleftrightarrow}$

(i) $M$ genügt der Spezifikation des Metaklassendiagramms in Abschnitt 6.1 (Seiten 73 bis 86),

(ii) $M$ erfüllt die Wohlgeformtheitseigenschaften aus Abschnitt 6.1.8 (Seiten 87 bis 88).

$\mathcal{M}$ sei die Menge aller gültigen MML–Schemata. (7.2)

Von dieser Stelle an werden für den Rest des Kapitels nur noch gültige MML–Schemata betrachtet, auch wenn dies nicht jedesmal explizit angegeben wird.
Für die Transformation notwendige Mengen bestimmter Schemaelementtypen werden in (7.3) festgelegt.

Sei $t \in Wertebereich(type)$.
$\mathcal{M}_{<t>}$ sei die Menge aller Schemaelemente vom Typ $t$. (7.3)

So beschreibt beispielsweise $\mathcal{M}_{FactClass}$ die Menge aller Faktklassen (aller gültigen MML–Schemata).
Eine Schreibweise für den Zugriff auf alle Schemaelemente eines bestimmten Typs in einem speziellen MML–Schema wird in (7.4) festgelegt.

Sei $M = (S, type) \in \mathcal{M}$ ein MML–Schema. Sei $t \in Wertebereich(type)$.
Dann sei $M_{<t>} \overset{def}{=} \{s \in S \mid type(S) = t\}$ die Menge aller Schemaelemente vom Typ $t$. (7.4)

So ist beispielsweise $M_{FactClass}$ die Menge aller Faktklassen innerhalb des Schemas $M$. Der Zugriff von Schemaelementen auf Attribute erfolgt über die in der Objektorientierung üblichen Punktnotation der Form „<Klassenname>.<Attributname>“, ebenso geschieht der Zugriff über Referenzen auf verbundene Komponenten und deren Attribute in der Form „<Klassenname>.<Referenzname>.<Attributname>“. Aus diesem Grunde sei für den MML–Datentyp *Name* o. B. d. A. der Punkt als Zeichen nicht erlaubt. Für einen speziellen Schemaelementtyp sind genau die Attribute und Referenzen definiert wie sie im MML–Metaschema auf den Seiten 73 bis 86 spezifiziert sind.

**REMUS**

Entsprechend den Definitionen für MML–Schemata werden in (7.5) REMUS–Schemata definiert.

Ein *REMUS–Schema* ist ein Tripel $R = (O, A, M)$ mit
(i) $O$ ist eine endliche, nichtleere Menge von Objekten
(ii) $A$ ist eine endliche, nichtleere Menge von Attributen
(In der Form von Paaren $(a, b)$, wobei $a$ der Name des Attributs und $b$ der des zugehörigen Datentyps ist)
(iii) $M$ ist eine endliche, nichtleere Menge von Metadaten
(In der in Abschnitt 7.1.2 eingeführten Tupelnotation).
$\mathcal{R}$ sei die Menge aller REMUS–Schemata. (7.5)

Auch die Menge der für die Transformation relevanten REMUS–Schemata wird auf *gültige* beschränkt, was in (7.6) geschieht.

$$\begin{aligned}&\text{Sei } R = (O, A, M) \text{ ein REMUS–Schema.}\\&R \text{ heißt } \textit{gültig} \overset{def}{\Longleftrightarrow}\\&\text{(i) } \forall m \in M : m \text{ bezieht sich nur auf Objekte } o \in O \cup A\\&\text{(ii) } \forall m \in M : m \text{ erfüllt eine der Tupelnotationen aus den Tabellen 7.1 bzw. 7.2}\\&\mathcal{R}_{Valid} \text{ sei die Menge aller gültigen REMUS–Schemata.}\end{aligned} \tag{7.6}$$

Das leere REMUS–Schema $R = (\emptyset, \emptyset, \emptyset)$ soll mit $R_{Empty}$ bezeichnet werden. In (7.7) wird die Vereinigung zweier REMUS–Schemata als komponentenweise Vereinigung definiert.

$$\begin{aligned}&\text{Seien } R_1 = (O_1, A_1, M_1), R_2 = (O_2, A_2, M_2) \in \mathcal{R}\\&\text{Die } \textit{Vereinigung} \text{ von } R_1 \text{ und } R_2 \text{ ist definiert als}\\&R_1 \cup R_2 \overset{def}{=\!=} (O_1 \cup O_2, A_1 \cup A_2, M_1 \cup M_2).\end{aligned} \tag{7.7}$$

**Transformation**

Zur übersichtlicheren Darstellung der im folgenden Abschnitt beschriebenen Transformation seien folgende Funktionen definiert:

- Eine *deterministische* Funktion $f_{det_{<x>}}$, die einigen Entwurfsentscheidungen wie z. B. der Auflösung von Namenskonflikten dient, an dieser Stelle aber noch nicht weiter spezifiziert wird und später in der Implementierung durch Benutzerinteraktion realisiert werden soll.
- Eine Funktion $\pi$, die zu einer Klasse aus dem MML–Schema die Relation des REMUS–Schemas angibt, auf die sie abgebildet worden ist.
- Eine Funktion $\phi$, die zu einer Klasse innerhalb der Vererbungshierarchie die Menge ihrer Nachfolger einschließlich der Klasse selbst ermittelt.
- Eine Funktion $\psi$, die die Konkatenation von Zeichenketten beschreibt.

Bei der Beschreibung der Transformationen werden Zeichenketten durch eine Schreibweise in Anführungszeichen hervorgehoben.

### 7.2.2 Schritt 1: Transformation von *DataType*–Schemaelementen

Zu jeder *DataType*–Instanz wird ein Objekt und ein Metadatum angelegt, wie exemplarisch für den elementaren Datentyp „Text“ in Abbildung 7.5 gezeigt.

Außerdem werden einmalig die speziellen Datentypen *KeyType* und *ForeignKeyType* für Primär- und Fremdschlüsseleinträge sowie der Typ *IdentifierValueType* (zur Verwendung siehe Schritt 3 in Abschnitt 7.2.4) definiert. Die Abbildungsvorschrift wird in (7.8) definiert.

$$\begin{aligned}&\tau_{DataType} : \mathcal{M}_{DataType} \times \mathcal{R} \to \mathcal{R}\\&\tau_{DataType}(d, R) \overset{def}{=\!=} (O \cup \{d.name\}, A, M \cup \{(d.name, \mathsf{ObjectType}, (\text{"}DataType\text{"}, \text{""}))\}).\end{aligned} \tag{7.8}$$

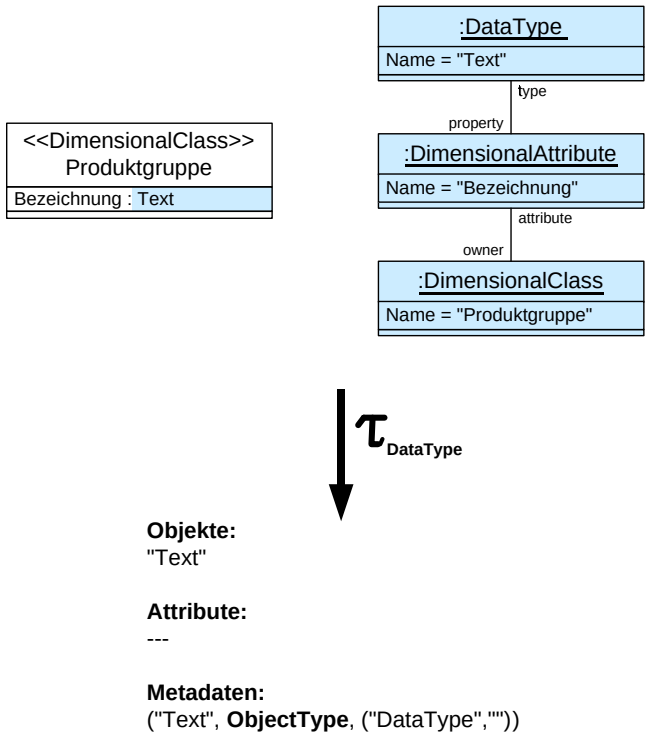



Abbildung 7.5: Transformation von *DataType*–Instanzen

Die Transformation aller Datentypen eines Schemas geschieht mittels der Vorschrift in (7.9).

$$
\begin{aligned}
&\mathcal{T}_{DataType} : \mathcal{M} \times \mathcal{R} \rightarrow \mathcal{R} \\
&\mathcal{T}_{DataType}(M, R) \stackrel{def}{=\!=} \bigcup_{d \in M_{DataType}} \tau_{DataType}(d, R) \\
&\cup \quad (\{"KeyType", "ForeignKeyType", "IdentifierValueType"\}, \\
&\quad \emptyset, \\
&\quad \{("KeyType", \text{ObjectType}, ("DataType", "")), \\
&\quad\ ("ForeignKeyType", \text{ObjectType}, ("DataType", "")), \\
&\quad\ ("IdentifierValueType", \text{ObjectType}, ("DataType", ""))\}).
\end{aligned}
\tag{7.9}
$$

### 7.2.3 Schritt 2: Transformation von *DataClass*–Schemaelementen

*DataClass*–Instanzen stellen komplexe Datentypen in Form geschachtelter Verbunde zur Verfügung. Um die erste Normalform zu erfüllen, müssen diese Strukturen aufgelöst werden, so dass als Resultat dieses Transformationsschrittes einem komplexen Datentyp eine Menge von Attributen mit zugehörigem elementarem Datentyp zugeordnet wird. Diese Zuordnung wird später bei der Transformation von Attributen in den Schritten 4 und 8 verwendet, um einem Attribut mit komplexem Datentyp die entsprechende Attributmenge zuordnen zu können. Abbildung 7.6 verdeutlicht diese Vorgehensweise an einem Beispiel: Das Attribut „Beschreibung“ der dimensionalen Klasse „Filiale“ hat den komplexen Datentyp „FilialTyp“. Durch diesen Transformationsschritt wird dem Datentyp „FilialTyp“ die Menge $\{(\text{„Filialleiter“}, \text{Text}), (\text{„Filialart“}, \text{Text})\}$ zugewiesen. Später in Schritt 4 bekommt die dimensionale Klasse „Filiale“ anstelle des einzelnen Attributes „Beschreibung“ diese Menge an Attributen zugewiesen.

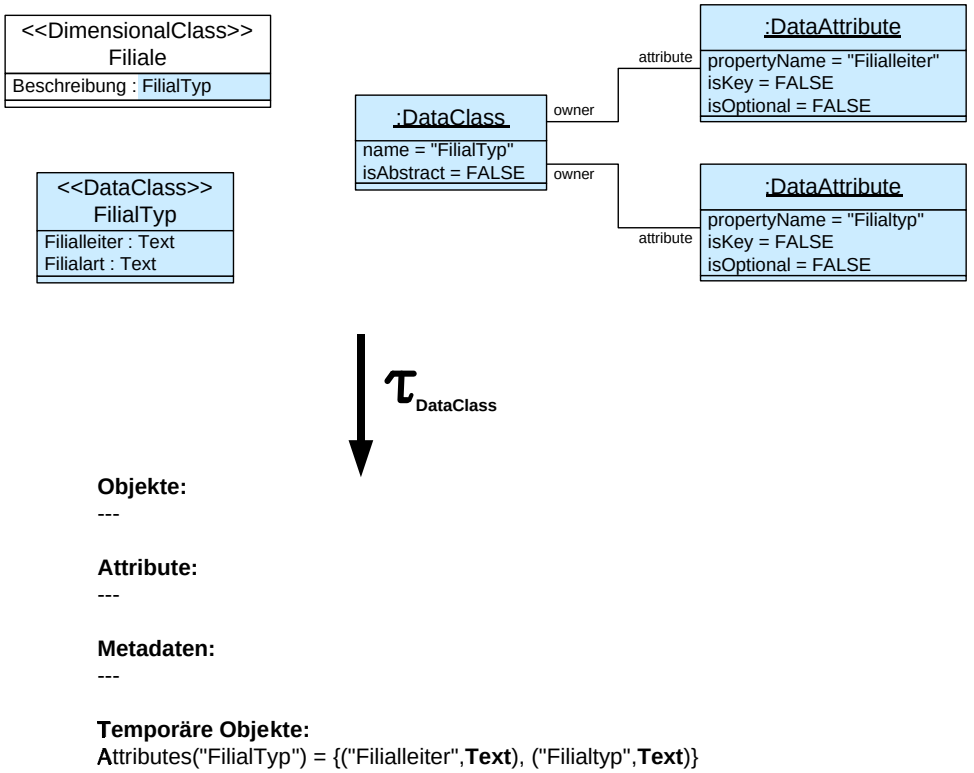



Abbildung 7.6: Transformation von *DataClass*–Instanzen

Als Hilfsstruktur zur Beschreibung der Transformationsvorschrift wird in (7.10) zunächst eine Struktur zur Verwaltung der Menge der Attribut–Datentyp–Paare von *DataClass*–Schemalementen definiert.

$$\begin{array}{l}\text{Sei } d \in \mathcal{M}_{DataClass}.\\ Attributes(d) \subseteq Name \times Name \text{ bezeichne die Menge der Attribut–Datentyp–Paare}\\ \text{von } d \text{ (und ihren Vorgängern in der Vererbungshierarchie).}\\ \text{Die Menge aller } \textit{Attributes}\text{–Mengen sei mit } \mathcal{M}_{DataClass_{Attributes}} \text{ bezeichnet.}\end{array} \tag{7.10}$$

Die Menge der Elemente in $Attributes(d)$ ergibt sich aus den Attributen der Datenklasse $d$ selber sowie von allen ihren Vorgängern in der Vererbungshierarchie. Um diese geerbten Attribute „einsammeln“ zu können, wird in (7.11) eine Abbildung definiert, die Schachtelungen von *DataClass*–Instanzen auflöst. Die Abbildung ist rekursiv, ihre Terminierung ist durch die Wohlgeformtheitseigenschaft (WF:ZF1) (siehe Seite 87) gegeben, die Zyklen innerhalb von *DataClass*–Schemaelementen unterbindet.

$$\begin{array}{l}CalcAttributes : M_{DataAttribute} \cup M_{DataClass} \rightarrow \mathcal{M}_{DataClass_{Attributes}}\\ CalcAttributes(s)\\ \stackrel{def}{=\!=} \begin{cases} (s.name, s.type.name) & \text{falls } s \in M_{DataAttribute} \wedge s.type \notin M_{DataClass} \\ CalcAttributes(s.type) & \text{falls } s \in M_{DataAttribute} \wedge s.type \in M_{DataClass} \\ \bigcup\limits_{a \in s.attribute} CalcAttributes(a) & \text{falls } s \in M_{DataClass}. \end{cases}\end{array} \tag{7.11}$$

Mit Hilfe der Abbildung $CalcAttributes$ kann in (7.12) die Auflösung von Generalisierungsstrukturen zwischen *DataClass*–Schemaelementen beschrieben werden, indem für eine *DataClass*–Instanz die Attributmenge der Klasse selbst und all ihrer Oberklassen berechnet wird.

$$\begin{aligned} &\tau_{DataClass} : \mathcal{M}_{DataClass} \rightarrow \mathcal{M}_{DataClass_{Attributes}} \\ &\tau_{DataClass}(d) \stackrel{def}{=\!=} CalcAttributesClass(d). \end{aligned} \tag{7.12}$$

Um alle *DataClass*–Instanzen eines Schemas aufzulösen, muss die in (7.12) definierte Abbildung für jedes *DataClass*–Schemaelement aufgerufen werden, was in (7.13) geschieht. Diese Transformationsvorschrift benötigt das REMUS–Schema nicht als Eingabe, weil sie im Zuge der Gesamttransformation in Abschnitt 7.2.12 nicht in der „normalen“ Reihenfolge aufgerufen wird, sondern in den Transformationen der Schritte 4 und 8 verwendet wird.

$$\begin{aligned} &\mathcal{T}_{DataClass} : \mathcal{M} \rightarrow \mathcal{M}_{DataClass_{Attributes}} \\ &\mathcal{T}_{DataClass}(M) \stackrel{def}{=\!=} \bigcup_{d \in M_{DataClass}} \tau_{DataClass}(d). \end{aligned} \tag{7.13}$$

### 7.2.4 Schritt 3: Transformation von *DimensionalClass*–Schemaelementen

Zur Transformation der *DimensionalClass*–Instanzen wird die Menge aller dimensionalen Klassen eines Schemas zunächst nach dem Kriterium der Vererbungshierarchien zerlegt, d. h. zwei *DimensionalClass*–Instanzen sind in derselben Teilmenge einer Zerlegung g. d. w. sie in einer (möglicherweise mehrstufigen) Generalisierungsbeziehung zueinander stehen. Aufgrund der in Abschnitt 6.1.8 auf Seite 87 definierten Wohlgeformtheitseigenschaft (WF:ZF3), die die Zyklenfreiheit von Vererbungsstrukturen fordert, ist eine solche Zerlegung eindeutig. Jedes Element der Zerlegung entspricht anschaulich einer Hierarchieebene, die einzelnen Klassen eines Zerlegungselementes entsprechen den unterschiedlichen Objekttypen der Hierarchieebene. Das Resultat für das Beispiel „Handelswelt“ ist in Abbildung 7.7 zu sehen.

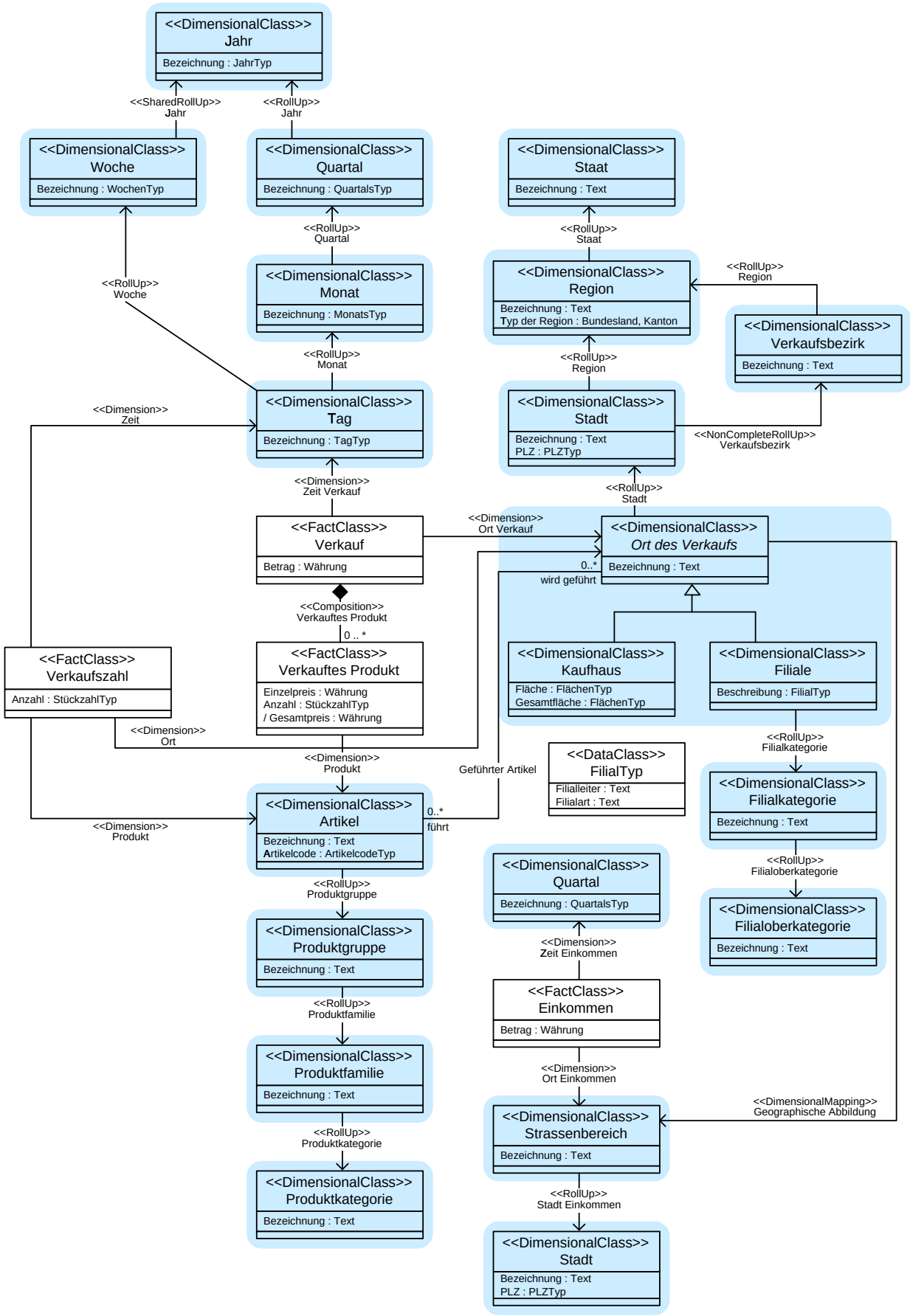



Abbildung 7.7: Zerlegung der Menge von *DimensionalClass*–Instanzen

Für jedes Element dieser Zerlegung wird im REMUS–Schema eine Relation angelegt, d. h. Vererbungsstrukturen zwischen dimensionalen Klassen werden durch Nestung aufgelöst. Dabei wird der Name durch die deterministische Funktion $f_{det_{ClassName}}$ festgelegt. Bei einelementigen Zerlegungselementen bietet sich als Funktionswert der Klassennamen an, bei mehrelementigen kann es beispielsweise der Name des Wurzelelements sein. Dementsprechend soll im Beispiel auch für alle einelementigen Zerlegungselemente der Klassenname gewählt werden z. B. $f_{det_{ClassName}}(\text{"Woche"}) \stackrel{def}{=} \text{"Woche"}$. Für die dreielementige Teilmenge mit der Vererbungshierarchie bestimmt mit
$f_{det_{ClassName}}(\{\text{"Filiale"}, \text{"Kaufhaus"}, \text{"Ort des Verkaufs"}\}) \stackrel{def}{=} \text{"Ort des Verkaufs"}$
das Wurzelelement den Namen der Hierarchieebene.
Jeder angelegten Relation werden die Attribute „ID“ vom Typ „KeyType“ und, im Falle der Zugehörigkeit zu einem mehrelementigen Element der Zerlegung, „Type“ vom Datentyp „IdentifierValueType“ zugeordnet.
In den Metadaten wird festgehalten, dass die Relation ein Objekt vom Typ *Dimension* ist, das Attribut „ID“ wird als Primärschlüssel und das „Type“–Attribut als diskriminierendes Attribut (*Identifier*) markiert. Die zulässigen Werte für dieses Attribut werden im Metadatum *IdentifierValue* festgehalten, indem als gültige Werte die Namen aller an dem Zerlegungselement beteiligten nicht–abstrakten *DimensionalClass*–Instanzen, die Nachfolger der ursprünglichen Klasse des Attributs sind, eingetragen werden. Abbildung 7.8 verdeutlicht die Transformation für die einelementige Teilmenge „Woche“ und die mehrelementige Teilmenge „Ort des Verkaufs“.

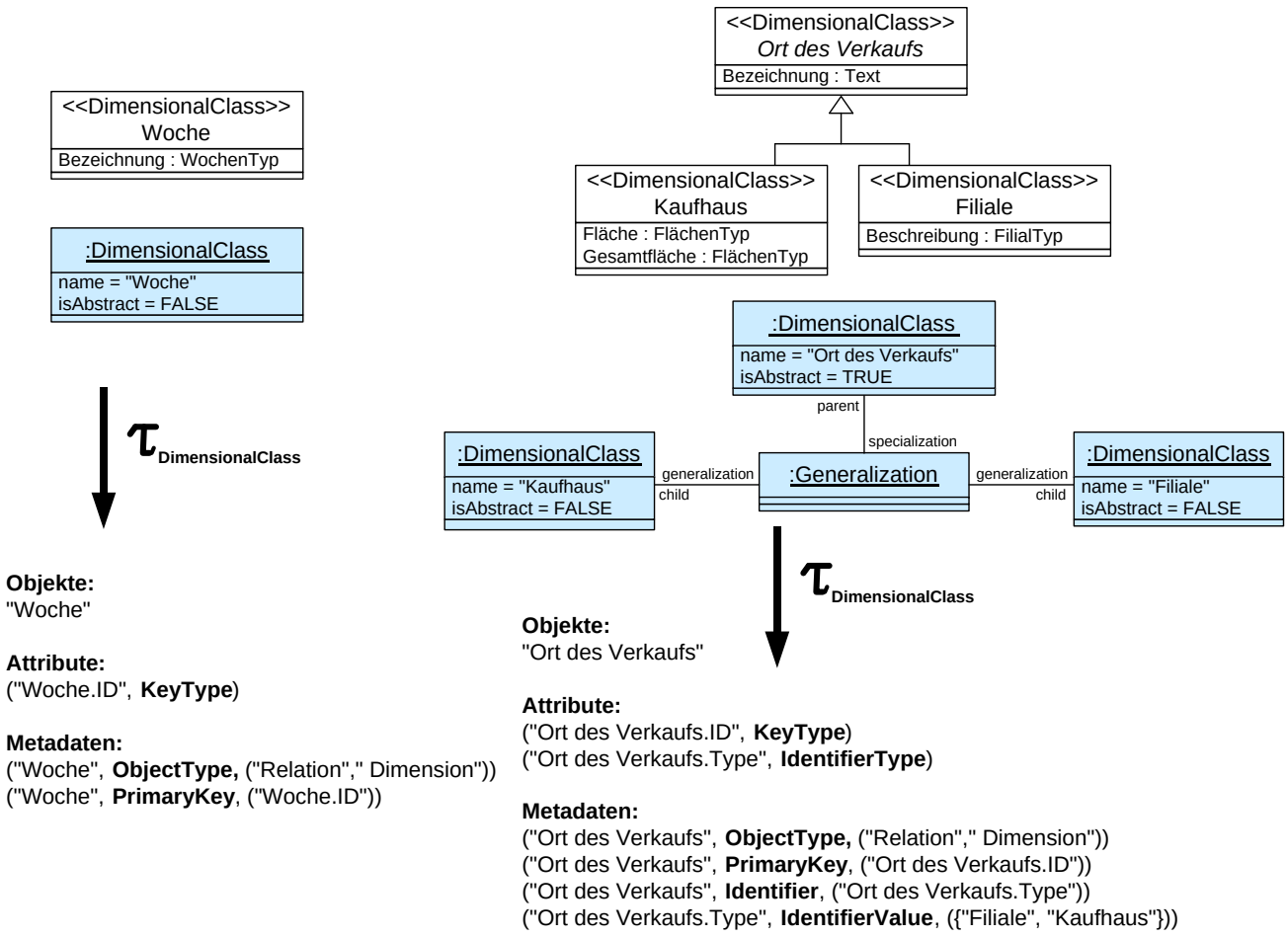



Abbildung 7.8: Transformation von *DimensionalClass*–Instanzen

Zur Definition der Abbildungsvorschrift wird in (7.14) zunächst der Begriff des Vorgängers definiert.

$$\begin{aligned}
&\text{Sei } M \in \mathcal{M} \text{ ein MML–Schema.}\\
&\text{Seien } d_1, d_2 \in M_{DimensionalClass}.\\
&d_1 \text{ heisst } \textit{Vorgänger} \text{ von } d_2\\
&\overset{def}{\Longleftrightarrow} \exists g_1, ..., g_n \in M_{Generalization}, \exists d'_1, ..., d'_{n-1} \in M_{DimensionalClass} :\\
&\qquad g_1.parent = d_1 \wedge g_1.child = d'_1\\
&\wedge\ g_2.parent = d'_1 \wedge g_2.child = d'_2\\
&\qquad\qquad \vdots\\
&\wedge\ g_n.parent = d'_{n-1} \wedge g_n.child = d_2.
\end{aligned} \tag{7.14}$$

Darauf aufbauend kann in (7.15) „Zerlegung“ formal erfasst werden, indem zwei dimensionale Klassen eines Schemas einem Zerlegungselement zugeordnet werden, falls sie einen gemeinsamen Vorgänger haben.

$$\begin{aligned}
&\text{Sei } M \in \mathcal{M} \text{ ein MML–Schema.}\\
&P = \{P_1, ..., P_k\} \text{ ist eine } \textit{Zerlegung} \text{ von } M_{DimensionalClass}\\
&\overset{def}{\Longleftrightarrow} \forall d_1, d_2 \in P_i \text{ mit } i \in \{1, ..., k\}\ : \exists d \in P_i :\\
&\qquad d \text{ ist Vorgänger von } d_1 \text{ und } d_2.
\end{aligned} \tag{7.15}$$

Damit lässt sich in (7.16) dieser Teilschritt für ein Element der Zerlegung darstellen.

$$\begin{aligned}
&\tau_{DimensionalClass} : Pot(\mathcal{M}_{DimensionalClass}) \times \mathcal{R} \to \mathcal{R}\\
&\tau_{DimensionalClass}(P, R) \overset{def}{=} (O \cup \{f_{det_{ClassName}}(P)\},\\
&\quad A \cup \{(\psi(f_{det_{ClassName}}(P), \text{".}ID\text{"}), \text{"}KeyType\text{"})\}\\
&\qquad \cup \{(\psi(f_{det_{ClassName}}(P), \text{".}Type\text{"}), \text{"}IdentifierValueType\text{"}) \mid \mid P \mid > 1\},\\
&\quad M \cup \{(f_{det_{ClassName}}(P), \mathsf{ObjectType}, (\text{"}Relation\text{"}, \text{"}Dimension\text{"}))\}\\
&\qquad \cup \{(f_{det_{ClassName}}(P), \mathsf{PrimaryKey}, (\psi(f_{det_{ClassName}}(P), \text{".}ID\text{"})))\}\\
&\qquad \cup \{(f_{det_{ClassName}}(P), \mathsf{Identifier}, (\psi(f_{det_{ClassName}}(P), \text{".}Type\text{"}))) \mid \mid P \mid > 1\}\\
&\qquad \cup \{(\psi(f_{det_{ClassName}}(P), \text{".}Type\text{"}), \mathsf{IdentifierValue},\\
&\qquad\quad (\{p.name \mid p \in P \wedge p.isAbstract = FALSE\}))\}).
\end{aligned} \tag{7.16}$$

Das Transformieren der gesamten Zerlegung ist in (7.18) definiert. Zweck dieser Hilfsfunktion ist die Kapselung, so dass die in (7.18) definierte Transformation die gleiche Signatur wie die anderen Transformationsschritte tragen kann.

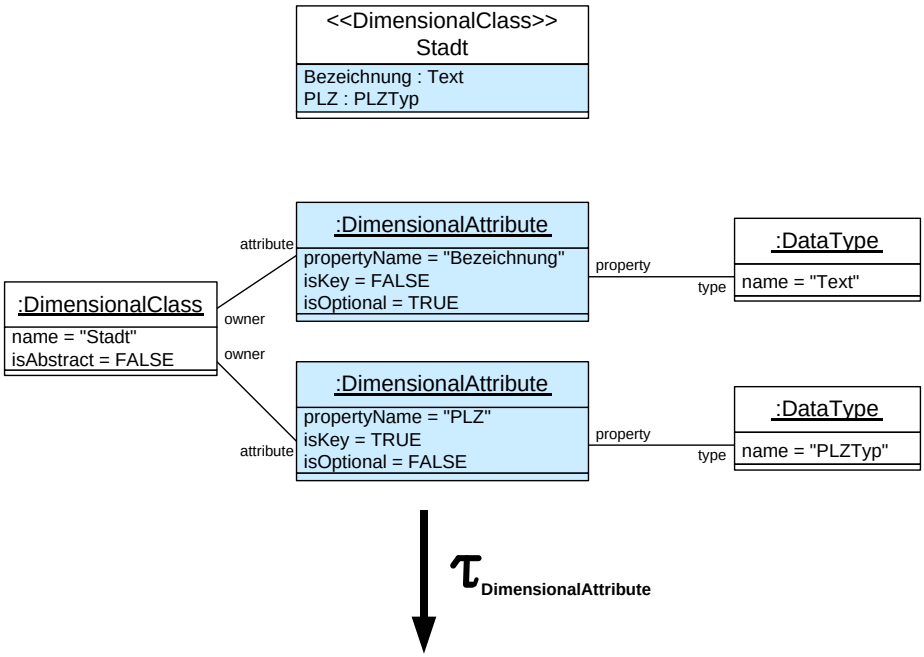
$$\begin{aligned}
&\text{Sei } P \in \mathcal{P} \text{ die in (7.15) definierte Zerlegung von } M_{DimensionalClass}.\\
&\mathcal{T}_{DimensionalClass\,Help} : \mathcal{M} \times \mathcal{R} \to \mathcal{R}\\
&\mathcal{T}_{DimensionalClass\,Help}(M, R) \overset{def}{=} \bigcup_{P_i \in P} \tau_{DimensionalClass}(P_i, R).
\end{aligned} \tag{7.17}$$

$$\begin{aligned}
&\mathcal{T}_{DimensionalClass} : \mathcal{M} \times \mathcal{R} \to \mathcal{R}\\
&\mathcal{T}_{DimensionalClass}(M, R) \overset{def}{=} \mathcal{T}_{DimensionalClass\,Help}(M, R).
\end{aligned} \tag{7.18}$$

Die auf Seite 126 eingeführte Funktion $\pi$, die einer Klasse eines MML–Schemas die Relation des REMUS–Schemas zuordnet, auf die sie abgebildet wird, lässt sich an dieser Stelle als $\pi(d) \overset{def}{=} f_{det_{ClassName}}(P_i)$ für $d \in P_i$ definieren.

### 7.2.5 Schritt 4: Transformation von *DimensionalAttribute*–Schemaelementen

Alle *DimensionalAttribute*–Instanzen werden in das REMUS–Schema übertragen und bekommen dabei, getrennt durch das Zeichen „.“, als Präfix den Namen der Relation zugeordnet, auf die die Klasse des Attributs abgebildet wird. Im Beispiel in Abbildung 7.9 hat die *DimensionalClass*–Instanz „Stadt“ zwei Attribute, die dementsprechend als Präfix den Namen der Relation, auf die „Stadt“ abgebildet wird, d. h. $\pi(\text{"Stadt"}) = f_{det_{ClassName}}(\text{"Stadt"}) = \text{"Stadt"}$, als Präfix bekommen. Die bei der Nestung „hochgezogenen“ Attribute erhalten entsprechend als Präfix neben der Relation auch den Namen ihrer ursprünglichen Klasse, z. B. wird das Attribut „Fläche“ aus der Klasse „Kaufhaus“ unter dem Namen „Ort des Verkaufs.Kaufhaus.Fläche“ angelegt. Ist ein Attribut, wie im Beispiel „PLZ“, als Schlüssel gekennzeichnet, so wird ein *ConceptualKey*–Metadatum angelegt. Ist das Attribut als optional markiert (wie im Beispiel das Attribut „Bezeichnung“), wird entsprechend ein *Optional*–Metadatum erzeugt.




**Objekte:**

**Attribute:**
("Stadt.Bezeichnung", **"Text"**)
("Stadt.Farbe", **"PLZTyp"**)

**Metadaten:**
("Stadt", **ConceptualKey,** ("Stadt.PLZ"))
("Stadt", **Optional**, ("Stadt.Bezeichnung"))

Abbildung 7.9: Transformation von *DimensionalAttribute*–Instanzen

Um die Transformationsfunktion für ein *DimensionalAttribute*–Schemaelement zu formulieren, wird in (7.19) zunächst die Hilfsfunktion $CalcDimensionalAttributes$ gebildet, die einem dimensionalen Attribut im Falle eines elementaren Datentyps den Namen und Typ des Attributs zuweist. Bei Attributen mit komplexem Datentyp wird unter Verwendung des in (7.13) in Schritt 2 definierten

Transformationsschrittes entsprechend eine Menge von Attributnamen–Datentyp–Kombinationen zugewiesen.

$$
\begin{aligned}
&CalcDimensionalAttributes : \mathcal{M}_{DimensionalAttribute} \times M \rightarrow Name \times Name \\
&CalcDimensionalAttributes(a, M) \\
&\quad \stackrel{def}{=\!=} \begin{cases} (a.propertyName, a.type.name) & \text{falls } a.type \notin M_{DataClass} \\ \mathcal{T}_{DataClass}(a.type) & \text{sonst.} \end{cases}
\end{aligned}
\tag{7.19}
$$

Unter Verwendung von $CalcDimensionalAttributes$ kann in (7.20) die Transformation für dimensionale Attribute mit einfachem und komplexem Datentyp einheitlich formuliert werden.

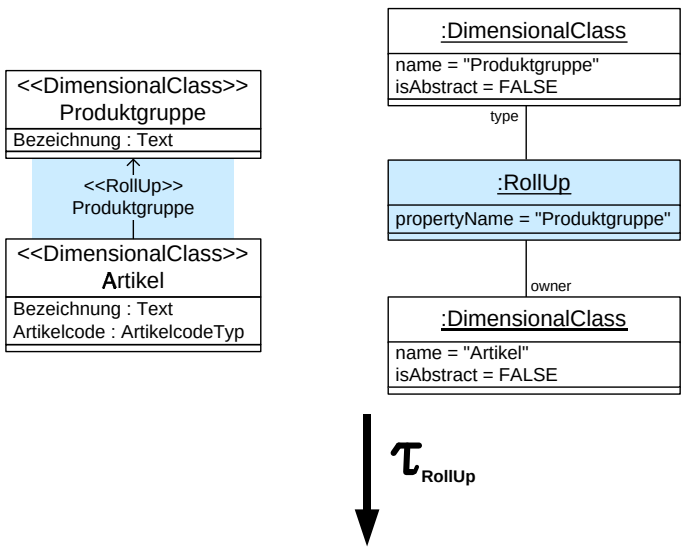
$$
\begin{aligned}
&\tau_{DimensionalAttribute} : \mathcal{M}_{DimensionalAttribute} \times \mathcal{M} \times \mathcal{R} \rightarrow \mathcal{R} \\
&\tau_{DimensionalAttribute}(a, M, R) \stackrel{def}{=\!=} (O, A \cup CalcDimensionalAttributes(a, M), \\
&\quad M \cup \bigcup_{b \in CalcDimensionalAttributes(a,M)} \{(\pi(b.owner.name), \text{ConceptualKey}, \\
&\qquad (\psi(\pi(b.owner.name), ".", b.propertyName))) \mid b.isKey = TRUE\} \\
&\quad \cup \bigcup_{b \in CalcDimensionalAttributes(a,M)} \{(\pi(b.owner.name), \text{Optional}, \\
&\qquad (\psi(\pi(b.owner.name), ".", propertyName))) \mid b.isOptional = TRUE\} \\
&\quad \cup \bigcup_{b \in CalcDimensionalAttributes(a,M)} \{(\psi(\text{"Berechnung von Attribut "}, \pi(b.owner.name)), \\
&\qquad \text{Computation}, ((b.computation.parameter), \pi(b.owner.name))) \mid b.computation \neq \emptyset\}). \\
&\quad \cup \bigcup_{b \in CalcDimensionalAttributes(a,M)} \{(\pi(b.owner.name), \text{Valid}, \\
&\qquad (\psi(\pi(b.owner.name), ".", propertyName), \phi(b.owner.name))) \\
&\qquad \mid \phi(b.owner.name) \neq \{b.owner.name\}\}
\end{aligned}
\tag{7.20}
$$

Zur Transformation aller *DimensionalAttribute*–Instanzen eines Schemas dient schließlich die Abbildung (7.21).

$$
\begin{aligned}
&\mathcal{T}_{DimensionalAttribute} : \mathcal{M} \times \mathcal{R} \rightarrow \mathcal{R} \\
&\mathcal{T}_{DimensionalAttribute}(M, R) \stackrel{def}{=\!=} \bigcup_{a \in M_{DimensionalAttribute}} \tau_{DimensionalAttribute}(a, M, R).
\end{aligned}
\tag{7.21}
$$

### 7.2.6 Schritt 5: Transformation von Hierarchiepfaden (*RollUp*–, *NonCompleteRollUp*– und *SharedRollUp*–Schemaelementen)

*RollUp*–Instanzen werden wie in Abbildung 7.10 dargestellt transformiert: Der Primärschlüssel der höheren Hierarchieebene (im Beispiel „Produktgruppe“) wird in die Relation der niedrigeren Ebene „Artikel“ als Fremdschlüssel eingetragen und diese Abhängigkeit in den Metadaten festgehalten. Die Multiplizität zwischen zwei Hierarchieebenen ist per Definition für einen *RollUp*–Operator immer *1–zu–Viele*, für einen *NonCompleteRollUp*–Operator hingegen *0 .. 1–zu–Viele*. Dies wird ebenso in den Metadaten festgehalten wie die Menge der zulässigen Typen für die Verbindung, die mit Hilfe der Funktion $\phi$ (zur Definition siehe Seite 126) ermittelt werden. Ist das Resultat von $\phi$ nur die Klasse selbst, so wird für die zulässigen Typen die Konstante „ALL_TYPES“ eingetragen. Bedeutung hat dies bei der *RollUp*–Instanz „Filialkategorie“, weil diese ihren Ursprung in einer Unterklasse innerhalb einer Vererbungshierarchie dimensionaler Klassen hatte. Die *RollUp*–Operation ist somit nur für „Filialen“, nicht aber für Einträge vom Typ „Kaufhaus“ möglich. Daher wird nicht „ALL_TYPES“ verwendet, sondern das Metadatum
(”Ort des Verkaufs”,”Filialkategorie”,RollUp, (”Filialkategorie”,{”Filiale”},ALL_TYPES,
(”Ort des Verkaufs.Filialkategorie.ForeignID”),(”Filialkategorie.ID”),COMPLETE))
erzeugt. Schließlich wird im Metadatum der Typ der Verdichtung festgehalten. Dabei ist zu beachten, dass dieser Eintrag nicht nur vom ursprünglich spezifizierten Typ abhängt. Auch wenn die Spezifikation ursprünglich eine vollständige Verdichtung war, die ausgehende dimensionale Klasse aber eine Oberklasse besitzt, so ist durch diese Verdichtung durch die Nestung als nicht–vollständig zu deklarieren.




**Objekte:**

**Attribute:**
("Artikel.Produktgruppe.ForeignID", **ForeignKeyType**)

**Metadaten:**
("Artikel.Produktgruppe.ForeignID", **Reference**, ("Produktgruppe.ID"))
("Artikel", **Multiplicity,** ({"Artikel.Produktgruppe.ForeignID"}, {1..*}))
("Artikel","Produktgruppe", **RollUp**, ("Produktgruppe", ALL_TYPES, ALL_TYPES,
({"Artikel.Produktgruppe.ForeignID"}), ({"Produktgruppe.ID"}), COMPLETE))

Abbildung 7.10: Transformation von *RollUp*–Instanzen

Aufgrund der sich nur geringfügig unterscheidenden Transformationsvorschriften werden *RollUp*– und *NonCompleteRollUp*–Instanzen einheitlich behandelt werden. Als Voraussetzung dafür werden in (7.22) die zwei Hilfsfunktionen $Calculate_{Multiplicity}$ und $Calculate_{RollUpType}$ definiert.

$$
\begin{aligned}
&Calculate_{RollUpType} : (\mathcal{M}_{RollUp} \cup \mathcal{M}_{NonCompleteRollUp}) \rightarrow \{\text{COMPLETE}, \text{NONCOMPLETE}\}\\
&Calculate_{RollUpType}(r) \stackrel{def}{=} \begin{cases} \text{COMPLETE} & \text{falls } r \text{ ist vom Typ } \textit{RollUp} \\ & \quad \wedge \phi(r.owner) = \{r.owner\} \\ \text{NONCOMPLETE} & \text{sonst.} \end{cases}\\
&Calculate_{Multiplicity} : (\mathcal{M}_{RollUp} \cup \mathcal{M}_{NonCompleteRollUp}) \rightarrow \{\text{"0..*"}, \text{"1..*"}\}\\
&Calculate_{Multiplicity}(r) \stackrel{def}{=} \begin{cases} \text{"1..*"} & \text{falls } Calculate_{RollUpType}(r) = COMPLETE \\ \text{"0..*"} & \text{sonst.} \end{cases}
\end{aligned}
\tag{7.22}
$$

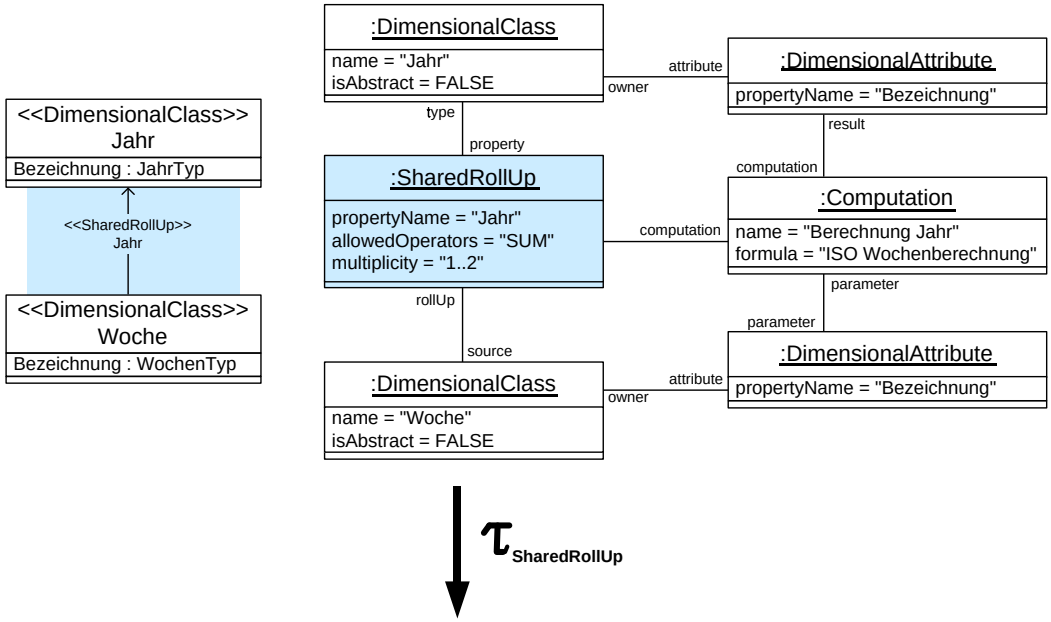
Dieses führt zur Transformationsvorschrift (7.23).

$$
\begin{aligned}
&\tau_{RollUp} : (\mathcal{M}_{RollUp} \cup \mathcal{M}_{NonCompleteRollUp}) \times \mathcal{R} \rightarrow \mathcal{R}\\
&\tau_{RollUp}(r, R) \stackrel{def}{=} (O,\\
&\qquad A \cup \{(\psi(\pi(r.owner.name), \text{"."}, \pi(r.type.name), \text{".}ForeignID\text{"}),\\
&\qquad\qquad \text{".}ForeignKeyType\text{"})\},\\
&\qquad M \cup \{(\psi(\pi(r.owner.name), \text{".}ID\text{"}), \textsf{Reference}, (\psi(\pi(r.type.name), \text{".}ID\text{"})))\}\\
&\qquad\quad \cup \{(\pi(r.owner.name), \textsf{Multiplicity},\\
&\qquad\qquad (\psi(\pi(r.owner.name), \text{"."}, \pi(r.type.name), \text{".}ForeignID\text{"}),\\
&\qquad\qquad Calculate_{Multiplicity}(r)))\}\\
&\qquad\quad \cup \{(\pi(r.owner.name), \pi(r.type.name), \textsf{RollUp},\\
&\qquad\qquad (r.propertyName, \phi(r.owner.name), \phi(r.type.name),\\
&\qquad\qquad\quad (\psi(\pi(r.owner.name), \text{"."}, \pi(r.type.name), \text{".}ForeignID\text{"})),\\
&\qquad\qquad\quad (\psi(\pi(r.type.name), \text{".}ID\text{"})), Calculate_{RollUpType}(r)))\}).
\end{aligned}
\tag{7.23}
$$

Die Transformation aller *RollUp*–Elemente eines Schemas ist in (7.24) defininert.

$$
\begin{aligned}
&\mathcal{T}_{RollUp} : \mathcal{M} \times \mathcal{R} \rightarrow \mathcal{R}\\
&\mathcal{T}_{RollUp}(M, R) \stackrel{def}{=} \bigcup_{r \in M_{RollUp} \cup M_{NonCompleteRollUp}} \tau_{RollUp}(r, R).
\end{aligned}
\tag{7.24}
$$

Beim Transformieren einer *SharedRollUp*–Instanz (siehe Abbildung 7.11) kann der Primärschlüssel der höheren Hierarchieebene nicht als Fremdschlüssel in die dimensionale Relation der niedrigeren Hierarchieebene eingetragen werden. Es handelt sich um ein *Viele–zu–Viele*–Beziehung, deren natürliche Darstellung im relationalen Modell durch eine Verbindungsrelation ausgedrückt wird. Dieses würde jedoch die spätere Navigierbarkeit des Schemas erschweren. Folglich wird die vorhandene Beziehung zwischen den beiden dimensionalen Relationen im REMUS–Schema durch zwei Metadaten dokumentiert. Das *SharedRollUp*–Metadatum enthält neben den beiden dimensionalen Relationen auch die zulässigen Typen, verweist auf die Berechnungsvorschrift und erhält darüber hinaus als zusätzlichen Eintrag die Menge von Verdichtungsoperatoren, die mit der Berechnungsvorschrift „verträglich“ sind, d. h. die beim Navigieren entlang dieses Verdichtungsfades angewendnet werden dürfen, ohne dass fehlerhafte Berechnungen auftreten.




**Objekte:**

**Attribute:**

**Metadaten:**
("Berechnung Jahr", **Computation**, (("Woche.Bezeichnung"),
"ISO Wochenberechnung", "Jahr.Bezeichnung"))
("Woche","Jahr", **SharedRollUp**, ("Jahr", ALL_TYPES, ALL_TYPES,
"Berechnung Jahr", {"SUM"}))

Abbildung 7.11: Transformation von *SharedRollUp*–Instanzen

Die zugehörige Transformationsvorschrift ist in (7.25) festgelegt.

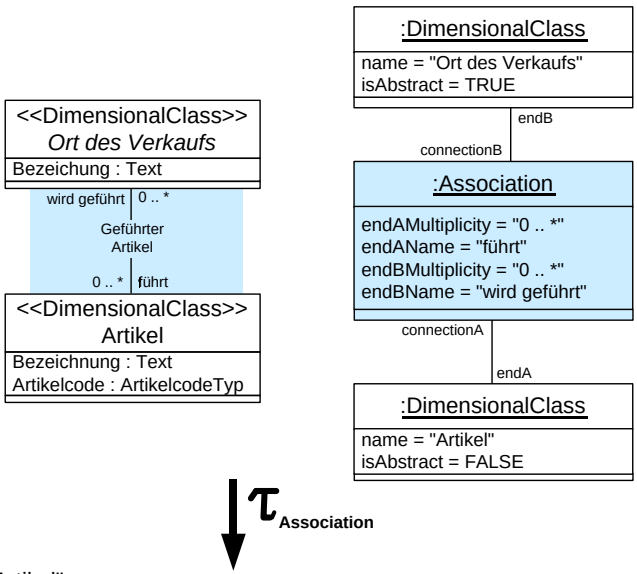
$$
\begin{aligned}
&\tau_{SharedRollUp} : \mathcal{M}_{SharedRollUp} \times \mathcal{R} \to \mathcal{R} \\
&\tau_{SharedRollUp}(r, R) \stackrel{def}{=\!=} (O, A, \\
&\qquad M \cup \{(\pi(r.owner.name), \pi(r.type.name), \mathsf{SharedRollUp}, \\
&\qquad\qquad (r.propertyName, \phi(r.owner.name), \phi(r.type.name), \\
&\qquad\qquad\quad r.computation.name, r.allowedOperators))\} \\
&\qquad \cup \{(r.computation.name, \mathsf{Computation}, ((r.computation.parameters), \\
&\qquad\qquad r.computation.formula, r.computation.result))\}).
\end{aligned}
\tag{7.25}
$$

Die Transformationsvorschrift für alle *SharedRollUp*–Elemente eines Schemas ist in (7.26) festgelegt.

$$
\begin{aligned}
&\mathcal{T}_{SharedRollUp} : \mathcal{M} \times \mathcal{R} \to \mathcal{R} \\
&\mathcal{T}_{SharedRollUp}(M, R) \stackrel{def}{=\!=} \bigcup_{r \in M_{SharedRollUp}} \tau_{SharedRollUp}(r, R).
\end{aligned}
\tag{7.26}
$$

### 7.2.7 Schritt 6: Transformation von Inter–Hierarchiebeziehungen (*Association*–Schemaelementen)

Ist zwischen zwei *DimensionalClass*–Instanzen eine Assoziation definiert, so wirddiese grundsätzlich in eine separate Relation abgebildet. Bei einfachen Multiplizitäten könnte auch ein Fremdschlüsseleintrag vorgenommen werden, aber diese Vorgehensweise wurde an dieser Stelle nicht gewählt, weil Assoziationen zwischen *DimensionalClass*–Schemaelementen die Konsistenz sichern, indem Definitionslücken im Datenwürfel festgelegt werden. Dieser Sachverhalt ist inhaltlich von den multidimensionalen Daten zu trennen, was durch die separate Relation zum Ausdruck gebracht wird.




**Objekte:**
"Ort des VerkaufsArtikel"

**Attribute:**
("Ort des VerkaufsArtikel.ID", **PrimaryKeyType**)
("Ort des VerkaufsArtikel.Ort des Verkaufs.ForeignID", **ForeignKeyType**)
("Ort des VerkaufsArtikel.Artikel.ForeignID", **ForeignKeyType**)

**Metadaten:**
("Ort des VerkaufsArtikel", **PrimaryKey**,("Ort des VerkaufsArtikel.ID"))
("Ort des VerkaufsArtikel", **ConceptualKey**, ("Ort des VerkaufsArtikel.Artikel.ForeignID"))
("Ort des VerkaufsArtikel", **ConceptualKey**, ("Ort des VerkaufsArtikel.Ort des Verkaufs.ForeignID"))
("Ort des VerkaufsArtikel.Artikel.ForeignID", **Reference**, ("Artikel.ID"))
("Ort des VerkaufsArtikel.Ort des Verkaufs.ForeignID", **Reference**, ("Ort des Verkaufs.ID"))
("Ort des VerkaufsArtikel", **Multiplicity,** ({"Ort des VerkaufsArtikel.Ort des Verkaufs.ForeignID"}, {"0 .. *"}))
("Ort des VerkaufsArtikel", **Multiplicity**, ({"Ort des VerkaufsArtikel.Artikel.ForeignID"}, {"0 .. *"}))
("Ort des Verkaufs","Artikel", **Association**, ("Geführter Artikel", "wird geführt", "führt",
ALL_TYPES, ALL_TYPES,
("Ort des Verkaufs.ID"), ("Ort des VerkaufsArtikel.Ort des Verkaufs.ForeignID"),
("Artikel.ID"), ("Ort des VerkaufsArtikel.Artikel.ForeignID")))

Abbildung 7.12: Transformation von *Association*–Instanzen zwischen *DimensionalClasses*

Wie in Abbildung 7.12 dargestellt, setzt sich der Name der Verbindungsrelation aus den Namen der beiden beteiligten *DimensionalClass*–Instanzen zusammen. Die Verbindungsrelation erhält einen künstlichen Primärschlüssel sowie die Primärschlüssel der beiden an der Assoziation beteiligten Klassen als Fremdschlüsseleinträge. In Form von Metadaten werden die Primärschlüsseleigenschaft, die beiden Fremdschlüsseleinträge als konzeptioneller Schlüssel, die Referenzen sowie die Multiplizitäten der *Association*–Instanz festgehalten. Das *Association*–Metadatum fasst diese Informationen nochmals zusammen. Die Transformationsvorschrift ist in (7.27) formuliert.

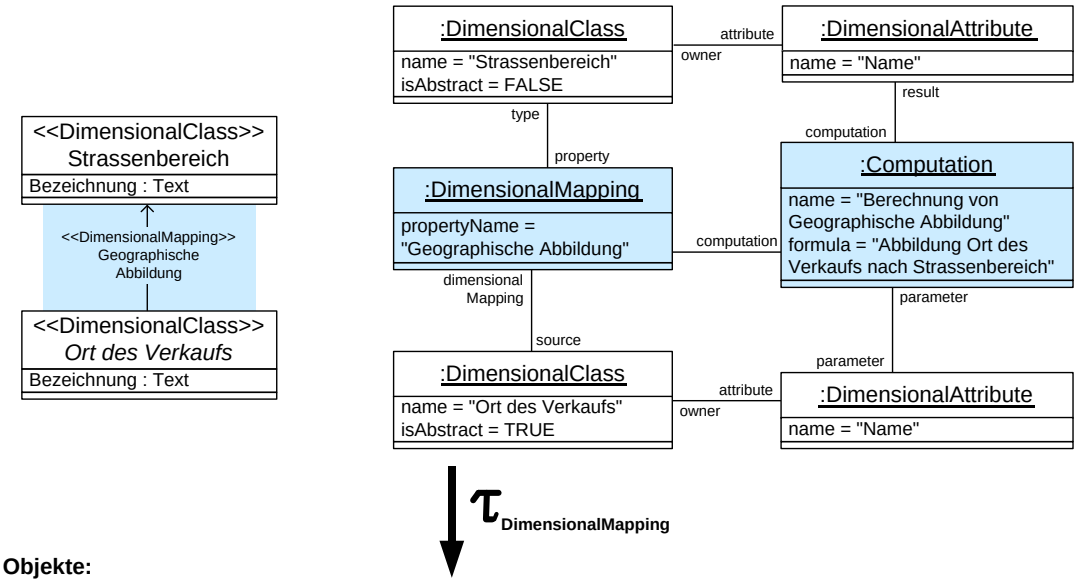
$$
\begin{aligned}
&\tau_{Association} : \mathcal{M}_{Association} \times \mathcal{R} \to \mathcal{R} \\
&\tau_{Association}(a, R) \\
&\stackrel{def}{=} (O \cup \{\psi(\pi(a.endA.name), \pi(a.endB.name))\}, \\
&\quad A \cup \{(\psi(\pi(a.endA.name), \pi(a.endB.name), ".ID", "PrimaryKeyType"), \\
&\qquad (\psi(\pi(a.endA.name), \pi(a.endB.name), ".", \pi(a.endB.name), \\
&\qquad\quad ".ForeignID"), "ForeignKeyType"), \\
&\qquad (\psi(\pi(a.endA.name), \pi(a.endB.name), ".", \pi(a.endA.name), \\
&\qquad\quad ".ForeignID"), "ForeignKeyType")\}, \\
&\quad M \cup \{(\psi(\pi(a.endA.name), \pi(a.endB.name)), \mathsf{PrimaryKey}, \\
&\qquad (\psi(\pi(a.endA.name), \pi(a.endB.name), ".ID"))), \\
&\qquad (\psi(\pi(a.endA.name), \pi(a.endB.name)), \mathsf{ConceptualKey}, \\
&\qquad\quad (\psi(\pi(a.endA.name), \pi(a.endB.name), ".", \\
&\qquad\quad \pi(a.endA.name), ".ForeignID"))), \\
&\qquad (\psi(\pi(a.endA.name), \pi(a.endB.name)), \mathsf{ConceptualKey}, \\
&\qquad\quad (\psi(\pi(a.endA.name), \pi(a.endB.name), ".", \\
&\qquad\quad \pi(a.endB.name), ".ForeignID"))), \\
&\qquad (\psi(\pi(a.endA.name), \pi(a.endB.name), ".", \pi(a.endA.name), ".ForeignID"), \\
&\qquad\quad \mathsf{Reference}, (\psi(\pi(a.endA.name), ".ID"))), \\
&\qquad (\psi(\pi(a.endA.name), \pi(a.endB.name), ".", \pi(a.endB.name), ".ForeignID"), \\
&\qquad\quad \mathsf{Reference}, (\psi(\pi(a.endB.name), ".ID"))), \\
&\qquad (\psi(\pi(a.endA.name), \pi(a.endB.name)), \mathsf{Multiplicity}, \\
&\qquad\quad (\psi(\pi(a.endA.name), ".", \pi(a.endB.name), ".", \pi(a.endA.name), \\
&\qquad\quad ".ForeignID"), a.endBMultiplicity)), \\
&\qquad (\psi(\pi(a.endA.name), \pi(a.endB.name)), \mathsf{Multiplicity}, \\
&\qquad\quad (\psi(\pi(a.endA.name), ".", \pi(a.endB.name), ".", \pi(a.endB.name), \\
&\qquad\quad ".ForeignID"), a.endAMultiplicity)), \\
&\qquad (\pi(a.endA.name), \pi(a.endB.name), \mathsf{Association}, \\
&\qquad\quad (\psi(\pi(a.endA.name), \pi(a.endB.name)), a.endBName, a.endAName, \\
&\qquad\quad \phi(a.endA.name), \phi(a.endB.name), \\
&\qquad\quad (\psi(\pi(a.endA.name), ".ID")), \\
&\qquad\quad (\psi(\pi(a.endA.name), \pi(a.endB.name), ".", \\
&\qquad\qquad \pi(a.endA.name), ".ForeignID")), \\
&\qquad\quad (\psi(\pi(a.endB.name), ".ID")), \\
&\qquad\quad (\psi(\pi(a.endA.name), \pi(a.endB.name), ".", \\
&\qquad\qquad \pi(a.endB.name), ".ForeignID"))))\}).
\end{aligned}
\tag{7.27}
$$

Um alle Assoziationen eines Schemas zu transformieren, ruft (7.28) die in (7.27) definierte Transformationsvorschrift für alle *Association*–Elemente auf.

$$
\begin{aligned}
&\mathcal{T}_{Association} : \mathcal{M} \times \mathcal{R} \to \mathcal{R} \\
&\mathcal{T}_{Association}(M, R) \stackrel{def}{=} \bigcup_{a \in M_{Association}} \tau_{Association}(a, R).
\end{aligned}
\tag{7.28}
$$

### 7.2.8 Schritt 7: Transformation von Inter–Hierarchiebeziehungen (*DimensionalMapping*–Schemaelementen)

Ein *DimensionalMapping*–Schemaelement wird, wie in Abbildung 7.13 dargestellt, durch Anlegen eines Metadatums für die Berechnungsvorschrift und eines speziellen Metadatums für das *DimensionalMapping* an sich festgehalten. Objekte oder Attribute werden in diesem Schritt nicht angelegt.




**Objekte:**

**Attribute:**

**Metadaten:**
("Berechnung von Geographische Abbildung", **Computation**, (("Ort des Verkaufs.Bezeichnung"), "Geographische Abbildung", "Strassenbereich.Bezeichnung"))
("Ort des Verkaufs", "Strassenbereich", **DimensionalMapping**,
("Abbildung Ort des Verkaufs nach Strassenbereich", ALL_TYPES, ALL_TYPES,
"Berechnung von Geographische Abbildung"))

Abbildung 7.13: Transformation von *DimensionalMapping*–Instanzen zwischen dimensionalen Klassen

Die Berechnungsvorschrift für diesen Schritt wird in (7.29) festgelegt.

$$
\begin{aligned}
&\tau_{DimensionalMapping} : \mathcal{M}_{DimensionalMapping} \times \mathcal{R} \rightarrow \mathcal{R} \\
&\tau_{DimensionalMapping}(d, R) \stackrel{def}{=\!=} (O, A, \\
&\quad M \cup \{(d.computation.name, \text{Computation}, \\
&\qquad (\psi(\pi(d.source.name), ".", d.computation.parameter.name) \\
&\qquad\ , d.computation.formula, d.computation.result)), \\
&\qquad (\pi(d.source.name), \pi(d.type.name), \text{DimensionalMapping}, \\
&\qquad\ (d.propertyName, \phi(d.source.name), \phi(d.type.name), \\
&\qquad\ d.computation.name))\}).
\end{aligned}
\tag{7.29}
$$

Zur Transformation aller *DimensionalMappings* dient Transformationsvorschrift (7.30).

$$
\begin{aligned}
&\mathcal{T}_{DimensionalMapping} : \mathcal{M} \times \mathcal{R} \rightarrow \mathcal{R} \\
&\mathcal{T}_{DimensionalMapping}(M, R) \stackrel{def}{=\!=} \bigcup_{d \in M_{DimensionalMapping}} \tau_{DimensionalMapping}(d, R).
\end{aligned}
\tag{7.30}
$$

### 7.2.9 Schritt 8: Transformation von *FactClass* und *FactAttribute*–Schemaelementen

Die Transformation von *FactClass*–Instanzen wird mehrstufig durchgeführt: Zunächst werden in Schritt 8a Vererbungsbeziehungen mittels *Weiterreichen* von Attributen, Dimensionen und Kompositionsbeziehungen an ihre Unterklassen aufgelöst. Anschließend werden in Schritt 8b die Kompositionen aufgelöst. Kompositionen mit einfacher Multiplizität werden dabei durch Einbettung aufgelöst, bei Kompositionen mit komplexer Multiplizität hingegen wird in Abhängigkeit von der Semantik der konkreten Daten zwischen vier verschiedenen Umsetzungen unterschieden. Nachdem diese vorbereitenden Schritte geschehen sind, werden in Schritt 8c schließlich die Faktrelationen und –attribute erzeugt.

#### Schritt 8a: Auflösen von Generalisierungen

Die Vererbungen werden aufgelöst, indem von den Oberklassen alle *FactAttribute*–, *Dimension*– und *Composition*–Schemaelemente an ihre Unterklassen übertragen werden. Zur Illustration dient in Abbildung 7.14 eine Variante aus dem Beispiel „Handelswelt“: Im ersten Schritt reicht die Klasse „Produkt“ ihre Dimension „Herkunftsland“ an ihre Unterklasse weiter. Danach braucht sie nicht weiter betrachtet zu werden, weil es sich um eine abstrakte Klasse handelt. Im zweiten Schritt in der Abbildung reicht die Klasse „Verkauftes Produkt“ ihre beiden Dimensionen, ihre beiden Attribute und ihre Komposition zur Klasse „Verkauf“ an die Unterklasse „Ökologisches Produkt“ weiter, womit dieser erste Teilschritt abgeschlossen ist.

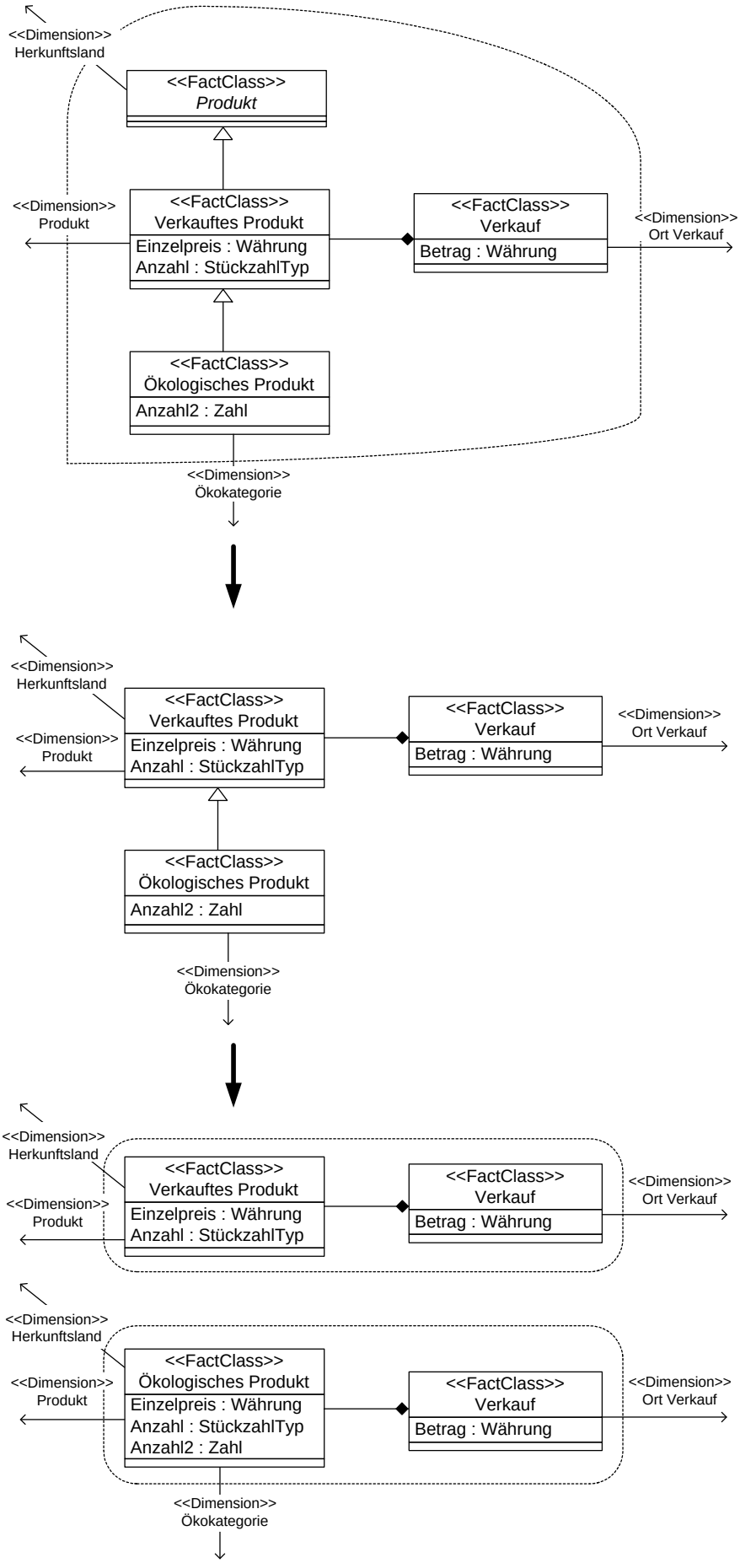



Abbildung 7.14: Auflösen von Vererbung in *FactClass*–Zusammenhangskomponenten

Wie anhand der Abbildung 7.14 zu erkennen ist, kann durch das Vererben die Attributmenge einer *FactClass*–Instanz über die in der MML spezifizierten Attribute hinausgehende Attribute umfassen, ebenso können die Dimensionen Verbindungen zu mehr als einer Faktklasse besitzen und die Kompositionen auf mehr als ein *FactClass*–Schemaelement auf Detailseite bzw. aggregierter Seite verweisen. Um Manipulationsoperationen am ursprünglichen MML–Schema zu vermeiden, werden zur Beschreibung der Transformationsvorschrift in (7.31) Hilfsstrukturen definiert.

Sei $d \in \mathcal{M}_{Dimension}$.
$Owner(d)$ bezeichne die Menge der Faktklassen, für die Dimension $d$ festgelegt ist.
Die Menge aller *Owner*–Mengen sei mit $\mathcal{M}_{Owner}$ bezeichnet.
Sei $c \in \mathcal{M}_{Composition}$.
$Detail(c)$ bezeichne die Menge der Faktklassen, die Komposition $c$ auf der Detailseite besitzt.
Die Menge aller *Detail*–Mengen sei mit $\mathcal{M}_{Detail}$ bezeichnet.
Sei $c \in \mathcal{M}_{Composition}$.
$Aggregated(c)$ bezeichne die Menge der Faktklassen, die Komposition $c$ auf der aggregierten Seite besitzt.
Die Menge aller *Aggregated*–Mengen sei mit $\mathcal{M}_{Aggregated}$ bezeichnet.
Sei $f \in \mathcal{M}_{FactClass}$.
$FactClassAttributes(f)$ bezeichne die Menge der Attribute, die Faktklasse $f$ besitzt.
Jedes $a \in FactClassAttributes$ ist ein Quintupel (n,t,o,m,a), wobei
$n$ der Name des Attributs ist, $t$ den Datentyp, o die Optionalität,
$m$ die Multiplizität angibt und $a$ beschreibt, ob es sich um ein aggregiertes Attribut handelt.
Die Menge aller *FactClassAttributes*–Mengen sei mit $\mathcal{M}_{FactClass_{Attributes}}$ bezeichnet. (7.31)

Ferner ist in (7.32) die Hilfsabbildung $Predecessors(f)$ für Faktklassen definiert, die der Faktklasse $f$ die Menge ihrer Oberklassen zuordnet.

Sei $M \in \mathcal{M}$.

$$
\begin{aligned}
&Predecessors(f) : M_{FactClass} \rightarrow Pot(M_{FactClass}) \\
&Predecessors(f) \stackrel{def}{=} \{ \\
&s \in M_{FactClass} \mid \exists g_1, ..., g_n \in M_{Generalization}, f_1, ..., f_{n-1} \in M_{FactClass} : \\
&\qquad g_1.parent = s \wedge g_1.child = f_1 \\
&\quad \wedge g_2.parent = f_1 \wedge g_2.child = f_2 \\
&\qquad\qquad\qquad \vdots \\
&\quad \wedge g_n.parent = f_{n-1} \wedge g_n.child = f\}.
\end{aligned}
\tag{7.32}
$$

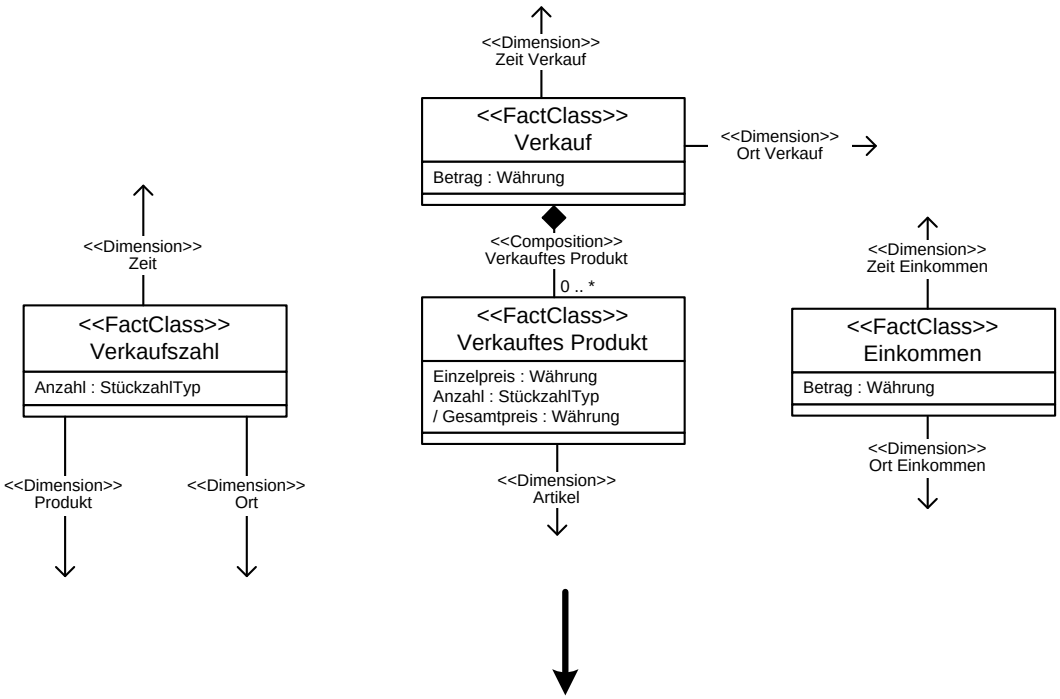
Analog erfolgt in (7.33) die Definition der Hilfsabbildung $Successors(f)$, die einer Faktklasse die Menge ihrer nicht–abstrakten Unterklassen zuordnet.

$$
\begin{aligned}
&\text{Sei } M \in \mathcal{M}.\\
&Successors(f) : M_{FactClass} \rightarrow Pot(M_{FactClass})\\
&Successors(f) \stackrel{def}{=} \{\\
&s \in M_{FactClass} \mid \exists g_1, ..., g_n \in M_{Generalization}, f_1, ..., f_{n-1} \in M_{FactClass} :\\
&\qquad g_1.parent = f \wedge g_1.child = f_1\\
&\quad \wedge g_2.parent = f_1 \wedge g_2.child = f_2\\
&\qquad\qquad \vdots\\
&\quad \wedge g_n.parent = f_{n-1} \wedge g_n.child = s\\
&\quad \wedge s.isAbstract = FALSE\}.
\end{aligned}
\tag{7.33}
$$

Die Transformationsvorschrift geht nicht wie in Abbildung 7.14 schrittweise jede einzelne Generalisierung durch, weil dies eine komplizierte Formulierung der Abarbeitungsreihenfolge verursachen würde. Stattdessen „sammelt“ jede *FactClass*–Instanz alle benötigten Attribute bei ihren Vorgängern ein, und jede Dimension bzw. Komposition ermittelt alle Nachfolger der ursprünglich referenzierten Faktklasse, was dem gleichen Resultat entspricht. (7.34) zeigt die Berechnungen für die Mengen.

$$
\begin{aligned}
&CalcOwner : \mathcal{M}_{Dimension} \rightarrow \mathcal{M}_{Owner}\\
&CalcOwner(d) \stackrel{def}{=} Successors(d.owner).\\
&CalcDetail : \mathcal{M}_{Composition} \rightarrow \mathcal{M}_{Detail}\\
&CalcDetail(c) \stackrel{def}{=} Successors(c.endB).\\
&CalcAggregated : \mathcal{M}_{Composition} \rightarrow \mathcal{M}_{Aggregated}\\
&CalcAggregated(c) \stackrel{def}{=} Successors(c.endA).\\
&CalcFactAttributes : \mathcal{M}_{FactAttribute} \cup \mathcal{M}_{DataClass} \rightarrow \mathcal{M}_{FactClass_{Attributes}}\\
&CalcFactAttributes(s)\\
&\stackrel{def}{=} \begin{cases}
(s.name, s.type.name, s.isOptional, \text{"0..*"}, FALSE)\\
\qquad \text{falls } s \in M_{FactAttribute} \wedge s.type \notin M_{DataClass}\\
CalcFactAttributes(s.type)\\
\qquad \text{falls } s \in M_{FactAttribute} \wedge s.type \in M_{DataClass}\\
\bigcup\limits_{a \in s.attribute} CalcFactAttributes(a)\\
\qquad \text{falls } s \in M_{DataClass}.
\end{cases}
\end{aligned}
\tag{7.34}
$$

Beim Berechnen der $FactClassAttributes$–Mengen werden zunächst die Attribute mit komplexen Datentypen in eine Menge von Attributen verwandelt. Dies geschieht analog zu der in (7.11) für *DimensionalAttribute*–Instanzen definierten Funktion. Abbildung 7.15 zeigt die ermittelten $Owner$–, $Detail$–, $Aggregated$– und $Attributes$–Mengen für die Beispielwelt.




**Objekte:**

**Attribute:**

**Metadaten:**

**Temporäre Objekte:**

```
Owner("Artikel") = {"Verkauftes Produkt"}
Owner("Ort") = {"Verkaufszahl"}
Owner("Ort Einkommen") = {"Einkommen"}
Owner("Ort Verkauf") = {"Verkauf"}
Owner("Produkt") = {"Verkaufszahl"}
Owner("Zeit") = {"Verkaufszahl"}
Owner("Zeit Einkommen") = {"Einkommen"}
Owner("Zeit Verkauf") = {"Verkauf"}
Detail("Verkauftes Produkt") = {"Verkauftes Produkt"}
Detail("Verkaufszahl") = {"Verkauftes Produkt"}
Attributes("Einkommen") = {("Betrag","Währung",FALSE,"{0..*}",FALSE)}
Attributes("Verkauf") = {("Betrag","Währung",FALSE,"{0..*}",FALSE)}
Attributes("Verkaufszahl") = {("Anzahl","StückzahlTyp",FALSE,"{0..*}",FALSE)}
Attributes("Verkauftes Produkt")= {("Anzahl","StückzahlTyp",FALSE,"{0..*}",FALSE),
                                   ("Einzelpreis","Währung",FALSE,"{0..*}",FALSE),
                                   ("Gesamtpreis","Währung",FALSE,"{0..*}",FALSE)}
```

Abbildung 7.15: Berechnung der Hilfsstrukturen für die Faktklassen

**Schritt 8b: Auflösen von Kompositionsbeziehungen**

Für die Behandlung der Kompositionen sind diese zunächst aufgrund ihrer Multiplizität zu unterscheiden. Während *Composition*–Schemaelemente einfacher Multiplizität („0..1“ oder „1“) stets durch Einbettung aufgelöst werden, sind bei Vorliegen komplexer Multiplizität in Abhängigkeit von den konkreten Daten Fallunterscheidungen notwendig. Zur Differenzierung wird in (7.35) die boolesche Funktion $Complex$ definiert, die einer Komposition den Wert „TRUE“ bzw. „FALSE“ in Abhängigkeit von der festgelegten Multiplizität zuordnet.

$$Complex : \mathcal{M}_{Composition} \rightarrow BOOLEAN$$
$$Complex(c) \stackrel{def}{=} \begin{cases} FALSE & \text{falls } c.endAMultiplicity \in \{\text{”0..1”}, \text{”1”}\} \\ TRUE & \text{sonst.} \end{cases} \tag{7.35}$$

Während des Vorgangs der Kompositionsauflösung werden die *Owner*– und *FactClassAttributes*–Mengen entsprechend erweitert sowie eventuell weitere Metadaten angelegt. In diese Aktualisierungsberechnungen fließen die *Detail*– und *Aggregated*–Mengen ein.

**Kompositionsbeziehungen mit einfacher Multiplizität**

Als Beispiel soll die in Abbildung 7.16 dargestellte Variante aus dem Beispiel „Handelswelt“ dienen, in der die Multiplizität der Komposition verändert wurde.

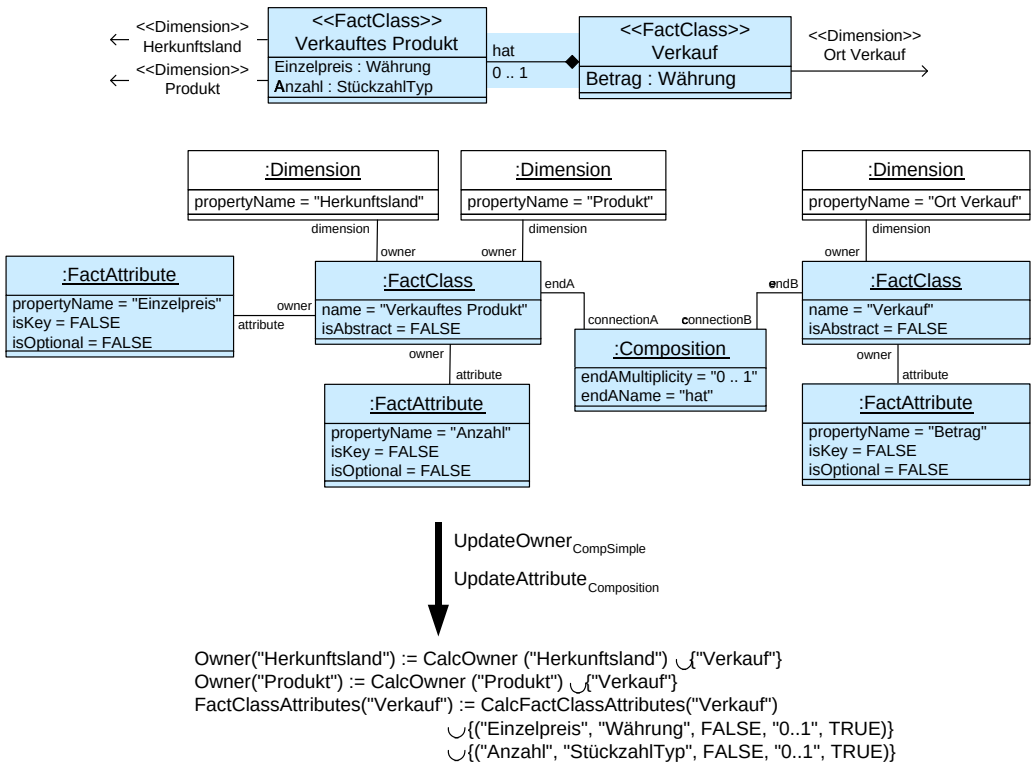



Abbildung 7.16: Auflösen von Kompositionsbeziehungen mit einfacher Multiplizität zwischen *FactClasses*

Die Attribute und Dimensionen der aggregierten Seite werden hierbei auf die Detailseite übertragen, die Attribute werden entsprechend als optional gekennzeichnet und mit der Multiplizität der Komposition versehen. Die Information, welche *Dimension*–Instanz im MML–Schema mit welcher *FactClass*–Instanz verbunden war, braucht nicht mitgeführt zu werden, denn bei potenziellen Auswertungen kann die entsprechende Dimension einfach unberücksichtigt bleiben.
Für die Transformationsvorschrift wird in (7.36) die Hilfsfunktion $UpdateOwner_{CompSimple}$ festgelegt, die durch Manipulation der $Owner$–Menge die Dimension von der Detailklasse auf die aggregierte Klasse „umhängt“, d. h. die Detailklasse der $Owner$–Menge hinzufügt, die aggregierte Klasse aus ihr entfernt.

$$
\begin{aligned}
&UpdateOwner_{CompSimple} : \mathcal{M}_{Composition} \times \mathcal{M}_{Owner} \rightarrow \mathcal{M}_{Owner} \\
&UpdateOwner_{CompSimple}(c, O) \\
&\stackrel{def}{=\!=} \begin{cases} O \cup \{f.name \mid f \in CalcAggregated(c)\} \setminus \{c.endA.name\} & \text{falls } c.endA.name \in O \\ O & \text{sonst.} \end{cases}
\end{aligned}
\tag{7.36}
$$

In analoger Art und Weise wird in (7.37) das Übertragen der Attribute definiert.

$$
\begin{aligned}
&UpdateAttribute_{Composition} : \mathcal{M}_{Composition} \times \mathcal{M}_{FactClass_{Attributes}} \rightarrow \mathcal{M}_{FactClass_{Attributes}} \\
&UpdateAttribute_{Composition}(c, A) \\
&\stackrel{def}{=\!=} \begin{cases} A \cup \bigcup\limits_{a \in CalcFactAttributes(c.endA)} \{(a.n, a.t, TRUE, c.endAMultiplicity, TRUE)\} \\ \quad \text{falls } f \in CalcAggregated(c) \\ A \quad \text{sonst.} \end{cases}
\end{aligned}
\tag{7.37}
$$

Die Funktion (7.37) trägt im Gegensatz zu (7.36) als Bezeichner „Composition“ statt „CompSimple“, weil sie auch für Kompositionen mit komplexer Multiplizität verwendet wird, für die $Owner$–Mengen jeodch in (7.40) die Definition einer neue Funktion notwendig wird.

### Kompositionsbeziehungen mit komplexer Multiplizität

Im Falle komplexer Multiplizität werden stets zwei Faktrelationen angelegt. Zusätzlich entscheidet die deterministische Funktion $f_{det_{Composition}}$ in Abhängigkeit vom modellierten Kontext über das Übertragen von Dimensionen und/oder Attributen, wobei die vier im Folgenden erläuterten Möglichkeiten existieren.
Unabhängig von der Auswahl wird eine Komposition mit komplexer Multiplizität durch ein Metadatum im REMUS–Schema dokumentiert, das in (7.38) für eine Komposition erzeugt wird.

$$
\begin{aligned}
&\tau_{FactClass_{Composition}} : \mathcal{M}_{Composition} \times \mathcal{R} \rightarrow \mathcal{R} \\
&\tau_{FactClass_{Composition}}(c, R) \stackrel{def}{=\!=} (O, A, \\
&\quad M \cup \{(g, d, \text{Composition}, \\
&\qquad (g.endANname, c.endAMultiplicity)) \mid g \in CalcAggregated(c), d \in CalcDetail(c)\}).
\end{aligned}
\tag{7.38}
$$

(7.39) erweitert dies für alle komplexen Kompositionen des Schemas.

$$
\begin{aligned}
&\text{Sei } M \in \mathcal{M}. \\
&M_{Composition\ Complex} \stackrel{def}{=\!=} \{c \in M_{Composition} \mid Complex(c) = TRUE\}. \\
&\mathcal{T}_{FactClass_{Composition}} : \mathcal{M} \times \mathcal{R} \rightarrow \mathcal{R} \\
&\mathcal{T}_{FactClass_{Composition}}(M, R) \stackrel{def}{=\!=} \bigcup_{c \in M_{Composition\ Complex}} \tau_{FactClass_{Composition}}(c, R).
\end{aligned}
\tag{7.39}
$$

**Möglichkeit I: Vernachlässigung der Komposition**

Im Falle der Vernachlässigung der Komposition bleiben die *Attributes*– und *Owner*–Mengen unverändert, die modellierte *Composition*–Instanz wird lediglich in Form eines Metadatums festgehalten. Dies kann sinnvoll sein, wenn die Komposition auf der konzeptionellen Ebene zur besseren Verständlichkeit beigetragen hat, die Beziehung der Fakten untereinander aber für potenzielle Auswertungen nicht relevant ist. Durch das Metadatum wird aber die Abhängikeit der Daten festgehalten und kann so z. B. von einem Ladewerkzeug ausgenutzt werden.

**Möglichkeit II: Übertragung der Dimensionen**

Bei dieser Möglichkeit wird der Komposition Rechnung getragen, indem die Dimensionen der aggregierten Faktklasse auch auf die Faktklasse auf der Detailseite übertragen werden. Dabei entstehen neue Faktattribut–Dimension–Kombinationen, weshalb für jedes Attribut der Detailseite für jede weitergegebene Dimension ein *Additivity*–Metadatum angelegt werden muss (siehe Schritt 10 auf Seite 156), das die erlaubten Operatoren beschreibt. Häufig sind hier nicht so viele Einschränkungen nötig, wie im Falle der Möglichkeiten III und IV (siehe Bemerkung auf Seite 154 am Ende von Schritt 8).
Für die Transformationsvorschrift wird in (7.40) eine Hilfsfunktionen zum Aktualisieren der *Owner*–Mengen definiert. In diesem Falle wird aber, im Gegensatz zu (7.36) für Kompositionen mit einfacher Komplexität, die ursprüngliche Faktklasse nicht aus der *Owner*–Menge entfernt.

**Möglichkeit III: Übertragung der Attribute**

Bei dieser Möglichkeit werden statt der Dimensionen die Attribute der aggregierten Faktklasse auf die Faktklasse der Detailseite übertragen. Die bei der Komposition vereinbarte Multiplizität wird in einem *AggregateAttribute*–Metadatum festgehalten. Neben der Tatsache, dass das betreffende Attribut aggregiert ist, wird auch dokumentiert, wieviele Datensätze die Komposition eingehen dürfen.
Auch hier müssen, wie bei Möglichkeit II, aufgrund der neu entstehenden Faktattribut–Dimension–Kombinationen wieder *Additivity*–Metadaten angelegt werden, was wiederum in Schritt 10 geschieht. Restriktionen sind hier nötig, wenn sowohl die aggregierte Klasse als auch die Detailklasse die gleichen Dimensionen besitzen (siehe Anmerkung auf Seite 154 am Ende von Schritt 8).
Das Übertragen der Attribute entspricht der Einbettung bei einfacher Multiplizität, so dass die in (7.37) definierte Aktualisierungsfunktion für Attributmengen wiederverwendet werden kann.

**Möglichkeit IV: Übertragung von Attributen und Dimensionen**

Dieser Ansatz stellt eine Kombination der Möglichkeiten II und III dar. Es werden sowohl die Attribute wie auch die Dimensionen der aggregierten Seite auf die Detailseite übertragen.

Um das Übertragen der Dimensionen bei den Möglichkeiten II und IV zu ermöglichen, wird in (7.40) die Funktion (7.36) dahingehend modifiziert, dass die Faktklasse der Detailseite nicht aus der $Owner$–Menge entfernt wird.

$$
\begin{aligned}
&UpdateOwner_{CompComplex} : \mathcal{M}_{Composition} \times \mathcal{M}_{Owner} \rightarrow \mathcal{M}_{Owner} \\
&UpdateOwner_{CompComplex}(c, O) \\
&\stackrel{def}{=\!=} \begin{cases} O \cup \{f.name \mid f \in CalcAggregated(c)\} & \text{falls } c.endA.name \in O \\ O & \text{sonst.} \end{cases}
\end{aligned}
\tag{7.40}
$$

Nach diesen Vorarbeiten kann die Transformation von einfachen wie komplexen Kompositionen wie in (7.41) und (7.42) beschrieben vorgenommen werden. Hierbei gibt $f_{det_{Composition}}$ im Falle komplexer Multiplizität an, welche der vier dargestellten Möglichkeiten gewählt werden soll.

$$
\begin{aligned}
&UpdateOwner : \mathcal{M}_{Composition} \times \mathcal{M}_{Owner} \rightarrow \mathcal{M}_{Owner} \\
&UpdateOwner(c, O) \\
&\stackrel{def}{=\!=} \begin{cases} UpdateOwner_{Comp\ Simple}(c, O) & \text{falls } Complex(c) = FALSE \\ UpdateOwner_{Comp\ Complex}(c, O) & \text{falls} f_{det_{Composition}}(c) \in \{II, IV\} \\ O & \text{sonst.} \end{cases}
\end{aligned} \tag{7.41}
$$

$$
\begin{aligned}
&UpdateAttributes : \mathcal{M}_{Composition} \times \mathcal{M}_{FactClass_{Attributes}} \rightarrow \mathcal{M}_{FactClass_{Attributes}} \\
&UpdateAttributes(c, A) \\
&\stackrel{def}{=\!=} \begin{cases} UpdateAttribute_{Composition}(c, A) & \text{falls } Complex(c) = FALSE \\ & \quad \vee f_{det_{Composition}}(c) \in \{III, IV\} \\ A & \text{sonst.} \end{cases}
\end{aligned} \tag{7.42}
$$

**Abarbeitungsreihenfolge**

Bei Berechnung der $Owner$– und $Attributes$–Mengen können die Funktionen in (7.42) und (7.41) für die Kompositionen eines Schemas nicht in beliebiger Reihenfolge aufgerufen werden. Die Behandlung bei mehrstufigen Kompositionen ist immer mit einer *Composition*–Instanz zu beginnen, deren aggregierte Klasse nicht Detailklasse einer weiteren Komposition ist. Dazu muss eine auf der Menge der Kompositionen des MML–Schemas vorliegende Halbordnung definiert werden.
Abbildung 7.17 zeigt ein Beispiel einer solchen Halbordnung.

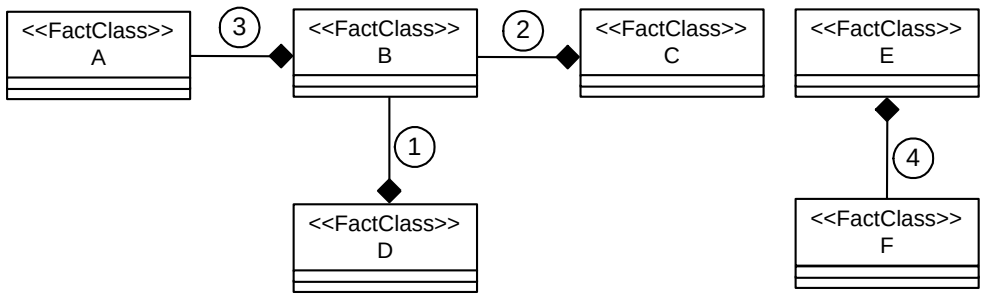



Abbildung 7.17: Abarbeitungsreihenfolge von *Composition*

Aufgrund der in Abschnitt 6.1.8 auf Seite 88 definierten Wohlgeformtheitseigenschaft (WF:ZF2), die die Zyklenfreiheit von Kompositionshierarchien fordert, muss eine solche Halbordnung existieren. Gibt es mehrere *Composition*–Instanzen mit der Eigenschaft, dass ihre aggregierte Klasse nicht Detailklasse einer weiteren Komposition ist, so ist innerhalb dieser Teilmenge die Abarbeitungsreihenfolge beliebig. In Abbildung 7.17 ist die Reihenfolge der Behandlung der Kompositionen 1 und 2 beliebig. Ebenso ist die Reihenfolge beliebig, wenn zwei Kompositionen nicht über Faktklassen (auch nicht mehrstufig) verbunden sind. So ist im Beispiel die Abarbeitung von Komposition 4 an beliebiger Stelle möglich.

Der Begriff der Halbordnung wird in 7.43 formal eingeführt.

$$\begin{aligned}&\text{Sei } M \in \mathcal{M}.\\&\text{Dann ist auf } M_{Composition} \text{ eine } \textit{Halbordnung} \text{ definiert:}\\&\forall c_1, c_2 \in M_{Composition} : c_1 < c_2 \overset{def}{\Longleftrightarrow} c_2.endB.name = c_1.endA.name\\&(\text{"Detailklasse von } c_1 \text{ ist aggregierte Klasse von } c_2\text{"}).\\&ord : M_{Composition} \rightarrow \{1, ..., \mid M_{Composition} \mid\}, ord \text{ bijektiv :}\\&\text{mit} \quad c_i < c_j \implies ord(c_i) < ord(c_j).\end{aligned} \tag{7.43}$$

Unter Verwendung der bisherigen Hilfsfunktionen und der Halbordnung werden in (7.44) die $Owner$– und in (7.45) die $FactAttributes$–Mengen berechnet.

$$\begin{aligned}&\text{Sei } M \in \mathcal{M}. \text{ Sei } < c_1, ..., c_n > \text{ eine Halbordnung auf } M_{Composition}.\\&Owner : M_{Dimension} \rightarrow M_{Owner}\\&Owner(d) \overset{def}{=\!=} UpdateOwner(c_n,\\&\qquad UpdateOwner(c_{n-1},\\&\qquad\qquad \vdots\\&\qquad UpdateOwner(c_2,\\&\qquad\quad UpdateOwner(c_1, CalcOwner(d))\\&\qquad )...)).\end{aligned} \tag{7.44}$$

$$\begin{aligned}&\text{Sei } M \in \mathcal{M}. \text{ Sei } < c_1, ..., c_n > \text{ eine Halbordnung auf } M_{Composition}.\\&FactClassAttributes : M_{FactClass} \rightarrow M_{Attributes}\\&FactClassAttributes(f) \overset{def}{=\!=} UpdateAttribute(c_n,\\&\qquad UpdateAttribute(c_{n-1},\\&\qquad\qquad \vdots\\&\qquad UpdateAttribute(c_2,\\&\qquad\quad UpdateAttribute(c_1, CalcFactClassAttributes(f))\\&\qquad )...)).\end{aligned} \tag{7.45}$$

**Schritt 8c: Faktrelationen und –attribute anlegen**

Nachdem nun alle Generalisierungen und alle Kompositionen aufgelöst sind, können Faktrelationen und — unter Ausnutzung der in den *FactClassAttributes*–Mengen gespeicherten Informationen — die zugehörigen Attribute angelegt werden. Sei dazu für alle $f \in M_{FactClass}$ eines MML–Schemas $M \in \mathcal{M}$ mit $Relation(f)$ eine boolesche Funktion definiert, deren Wert bestimmt, ob für $f$ eine eigene Relation angelegt wird oder nicht. $Relation(f)$ liefert genau dann FALSE, wenn $f$ abstrakt ist oder wenn $f$ als einzige Verbindung in Kompositionen auf der De-

tailseite mit einfacher Multiplizität erscheint. Das Anlegen einer einzelnen Faktrelation ist in (7.46) definiert.

$$\begin{aligned}
&\tau_{FactClass} : \mathcal{M}_{FactClass} \times \mathcal{R} \to \mathcal{R} \\
&\tau_{FactClass}(f, R) \\
&\stackrel{def}{=\!=} \begin{cases} ( \quad O \cup \{f.name\}, A, & \\ \quad M \cup \{(f.name, \text{ObjectType}, (\text{"}Relation\text{"}, \text{"}Fact\text{"}))\}) & \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{falls } Relation(f) = TRUE & \\ \quad R \qquad\qquad\qquad\qquad\qquad\qquad \text{sonst.} & \end{cases}
\end{aligned} \tag{7.46}$$

Alle Faktrelationen werden mittels (7.47) angelegt.

$$\begin{aligned}
&\mathcal{T}_{FactClass} : \mathcal{M} \times \mathcal{R} \to \mathcal{R} \\
&\mathcal{T}_{FactClass}(M, R) \stackrel{def}{=\!=} \bigcup_{f \in M_{FactClass}} \tau_{FactClass}(f, R).
\end{aligned} \tag{7.47}$$

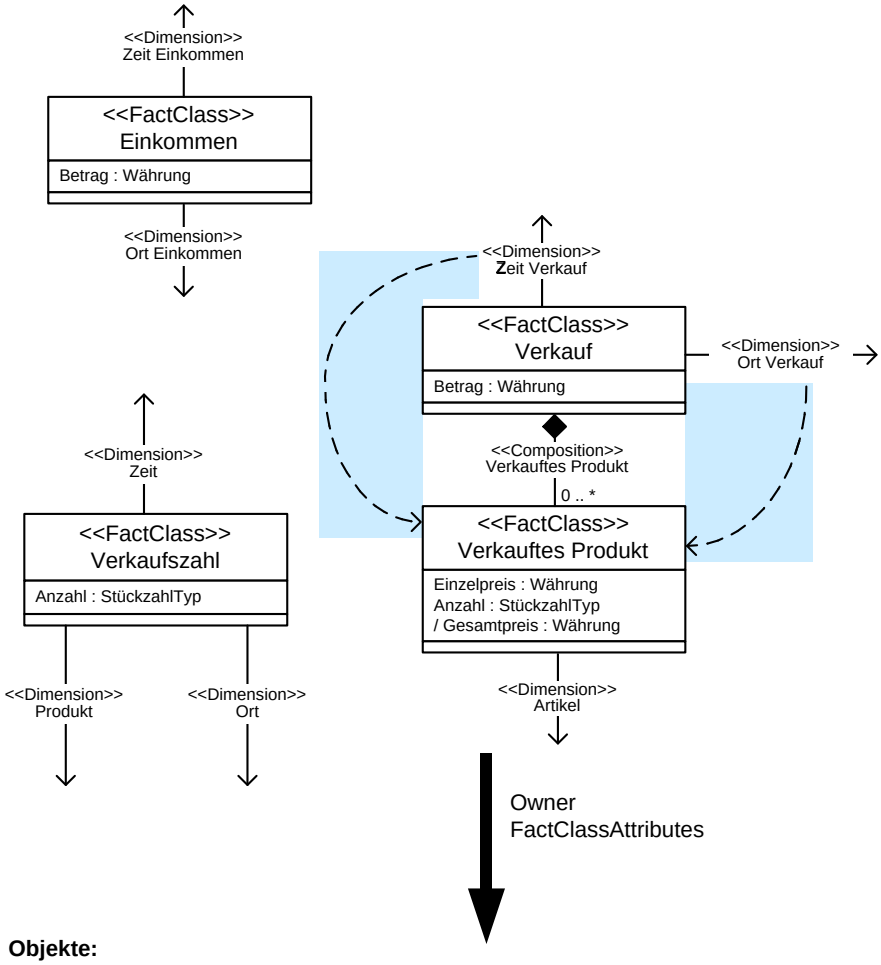
Nun kann in (7.48) die Abbildung für das Anlegen eines einzelnen Faktattributes definiert werden. Dabei sind als zusätzliche Argumente der Name der Klasse, die Angabe über Optionalität des Attributes und über die Multiplizität notwendig, da diese Informationen in den $FactClassAttributes$–Mengen abgespeichert sind.

$$\begin{aligned}
&\tau_{FactAttribute} : \mathcal{M}_{FactClass} \times \mathcal{M}_{FactClass_{Attributes}} \times \mathcal{R} \to \mathcal{R} \\
&\tau_{FactAttribute}(f, a, R) \stackrel{def}{=\!=} \\
&\quad (O, A \cup \{(\psi(f.name, \text{"."}, a.n)), a.t)\}, \\
&\quad M \cup \{(f.name, \text{Optional}, (\psi(f.name, \text{"."}, a.n))) \mid a.o = TRUE\} \\
&\qquad \cup \{(f.name, \text{AggregatedAttribute}, (\psi(f.name, \text{"."}, a.n), a.m)) \\
&\qquad\qquad \mid a.a = TRUE\}) \\
&\qquad \cup \{(\psi(\text{"Berechnung von Attribut "}, a.n), \text{Computation}, \\
&\qquad\qquad ((a.computation.parameter), f.name)) \mid a.computation \neq \emptyset\}).
\end{aligned} \tag{7.48}$$

Das endgültige Anlegen aller Faktattribute im Schema wird mittels (7.49) realisiert.

$$\begin{aligned}
&\mathcal{T}_{FactAttribute} : \mathcal{M} \times \mathcal{R} \to \mathcal{R} \\
&\mathcal{T}_{FactAttribute}(M, R, A) \stackrel{def}{=\!=} \bigcup_{f \in M_{FactClass}} \Big( \bigcup_{a \in FactClassAttributes(f)} \tau_{FactAttribute}(f, a, R) \Big).
\end{aligned} \tag{7.49}$$

Für das Beispiel „Handelswelt“ muss für die Komposition „Verkauftes Produkt“ entschieden werden, welche der vier auf Seite 148 vorgestellten Möglichkeiten gewählt werden soll. Aufgrund der Anforderungen an das System und der damit verbundenen zu erwartenden Auswertungen sollen hierbei die beiden Dimensionen „Ort Verkauf“ und „Zeit Verkauf“ an die Detailklasse übertragen werden (Möglichkeit II). Dazu werden die Mengen $Owner(\text{"Ort Verkauf"})$ und $Owner(\text{"Zeit Verkauf"})$, wie in Abbildung 7.18 zu sehen, jeweils um das Element "Verkauftes Produkt" erweitert.




**Objekte:**

**Attribute:**

**Metadaten:**
("Verkauf", "Verkauftes Produkt", **Composition**, ("Verkaufszahl",{0..*}))

**Temporäre Objeke**:
Owner("Artikel") = {"Verkauftes Produkt"}
Owner("Ort") = {"Verkaufszahl"}
Owner("Ort Einkommen") = {"Einkommen"}
Owner("Ort Verkauf") = {"Verkauf", "Verkauftes Produkt"}
Owner("Produkt") = {"Verkaufszahl"}
Owner("Zeit") = {"Verkaufszahl"}
Owner("Zeit Einkommen") = {"Einkommen"}
Owner("Zeit Verkauf") = {"Verkauf", "Verkauftes Produkt"}
Detail("Verkauftes Produkt") = {"Verkauftes Produkt"}
Detail("Verkaufszahl") = {"Verkauftes Produkt"}
FactClassAttributes("Einkommen") = {("Betrag","Währung",TRUE,"{0..*}",FALSE)}
FactClassAttributes("Verkauf") = {("Betrag","Währung",TRUE,"{0..*}",FALSE)}
FactClassAttributes("Verkaufszahl") = {("Anzahl","StückzahlTyp",TRUE,"{0..*}",FALSE)}
FactClassAttributes("Verkauftes Produkt")= {("Anzahl","StückzahlTyp",TRUE,"{0..*}",FALSE),
("Einzelpreis","Währung",TRUE,"{0..*}",FALSE),
("Gesamtpreis","Währung",TRUE,"{0..*}",FALSE)}

Abbildung 7.18: Aktualisierung der *Owner*– und *FactClassAttributes*–Mengen

Anschließend werden für die vier Faktklassen des Schemas Relationen angelegt und jeweils durch ein Metadatum dokumentiert, wie in Abbildung 7.19 exemplarisch für „Verkaufszahl“ gezeigt.

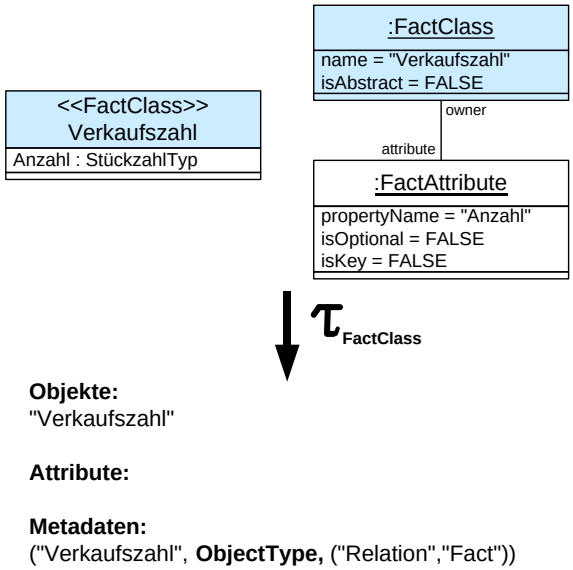



Abbildung 7.19: Transformation von *FactClass*–Schemaelementen

Schließlich werden die zugehörigen $FactClassAttributes$–Mengen abgearbeitet und die entsprechenden Faktattribute angelegt (siehe Beispiel in Abbildung 7.20). Zugehörige *Optional*– oder *Multiplicity*–Metadaten fallen im Beispielschema nicht an, weil die Bedingungen nicht erfüllt sind.

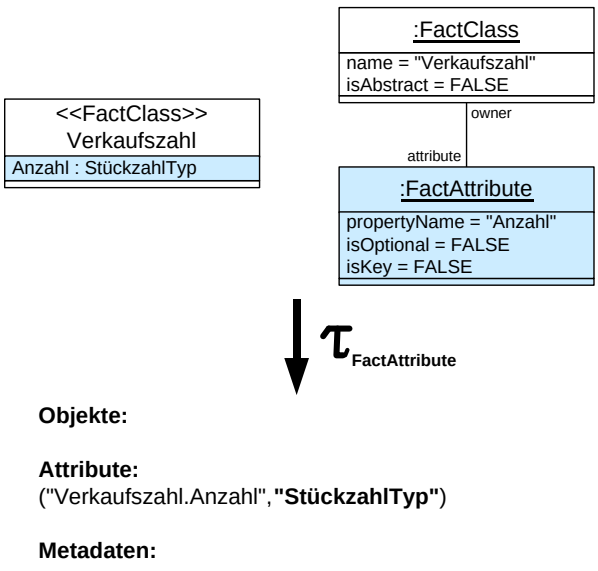


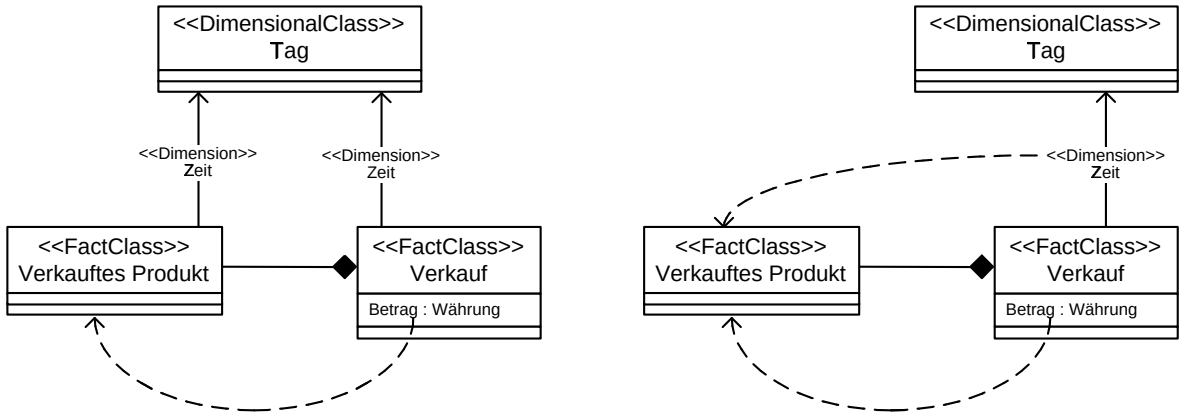

Abbildung 7.20: Transformation von *FactAttribute*–Schemaelementen

**Anmerkung zu den neuen *Additivity*–Metadaten**

Bei den Transformationsalternativen II bis IV entstehen neue Kombinationen von Faktattributen und Dimensionen, für die entsprechende *Additivity*–Metadaten angelegt werden müssen. Während Fall II meistens nicht kritisch ist, können in den in Abbildung 7.21 dargestellten Szenarien (die dem gleichen Resultat entsprechenden) Probleme auftreten:

- Bei Möglichkeit III können sowohl aggregierte Klasse als auch Detailseite eine *Dimension* zur selben *DimensionalClass*–Instanz besitzen (Abbildung 7.21(a)).
- Bei Möglichkeit IV können sowohl aggregiertes Attribut als auch eine Dimension auf die Detailseite übertragen werden (Abbildung 7.21(b)).

Die von der aggregierten Seite übertragenen Faktattribute dürfen in der Regel nur die Operatoren *MAX* und *MIN* zur Verdichtung besitzen, denn nur diese sind unabhängig davon, wie oft der gleiche Wert in die Berechnung eingeht. Bei der Summierung beispielsweise würden falsche Ergebnisse erzielt werden, weil das aggregierte Attribut so oft in die Berechnung eingehen würde, wie es die in der Komposition modellierte Multiplizität beschreibt.



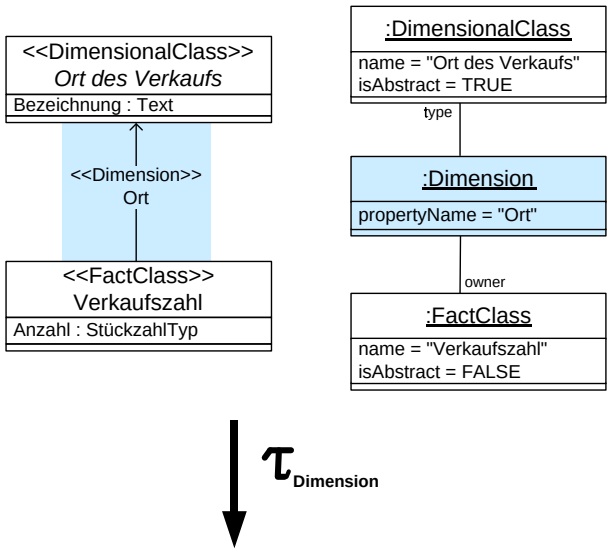

(a) Übertragung Attribut bei gemeinsam genutzter Dimension

(b) Gleichzeitige Übertragung von Attribut und Dimension

Abbildung 7.21: Neue *Additivity*–Metadaten

### 7.2.10 Schritt 9: Transformation von *Dimension*–Schemaelementen

Für jede *Dimension*–Instanz des Schemas wird mit Hilfe der *Owner*–Funktion festgestellt, für welche Faktklassen sie gültig ist. Neben den im MML–Schema modellierten Verbindungen können dies zusätzlich in Schritt 8 durch Kompositionsauflösung entstandene Verbindungen sein. Abbildung 7.22 zeigt exemplarisch das Vorgehen beim Transformieren der Dimensionen: Der Primärschlüssel der aus der *DimensionalClass*–Instanz hervorgegangenen Relation wird in die aus der *FactClass*–Instanz hervorgegangenen Relation als Fremdschlüssel eingetragen. Die Referenz sowie ihr Typ werden als Metadaten festgehalten. Weil aufgrund der Transformation in Schritt 8 $Owner(\text{"Ort"}) = \{\text{"Verkaufszahl"}\}$ gilt, braucht dieses im Beispiel nur einmal durchgeführt werden. Durch das Eintragen des Fremdschlüssels als *PrimaryKey*–Metadatum wird durch Auflösen der einzelnen Dimensionen einer Faktrelation sukzessive der Primärschlüssel dieser Faktrelation




**Objekte:**

**Attribute:**
("Verkaufszahl.Ort des Verkaufs.ForeignID", **ForeignKeyType**)

**Metadaten:**
("Verkaufszahl.Ort des Verkaufs.ForeignID", **Reference**, ("Ort des Verkaufs.ID"))
("Verkaufszahl", **PrimaryKey**, ("Verkaufszahl.Ort des Verkaufs.ForeignID"))
("Verkaufszahl", **Multiplicity**, ({"Verkaufszahl.Ort des Verkaufs.ForeignID"}, {"0 .. *"}))
("Verkaufszahl", "Ort des Verkaufs", **Dimension**, ("Ort", ALL_TYPES, ALL_TYPES,
("Verkaufszahl.Ort des Verkaufs.ForeignID"), ("Ort des Verkaufs.ID")))

Abbildung 7.22: Auflösen von *Dimensions*

angelegt.

Die Transformation einer Dimension bez. einer Faktklasse ist in (7.50) definiert.

$$
\begin{aligned}
&\tau_{Dimension} : \mathcal{M}_{Dimension} \times \mathcal{M}_{FactClass} \times \mathcal{R} \to \mathcal{R} \\
&\tau_{Dimension}(d, f, R) \stackrel{def}{=\!=} (O, \\
&\quad A \cup \{(\psi(f.name, ".", \pi(d.name), ".ForeignID"), \mathsf{ForeignKeyType})\} \\
&\quad M \cup \{(\psi(f.name, ".", \pi(d.name)), \mathsf{Reference}, (\psi(\pi(d.name), ".ID")))\} \\
&\qquad \cup \{(f.name, \mathsf{PimaryKey}, (\psi(f.name, ".", \pi(d.name)".ForeignID")))\} \\
&\qquad \cup \{(f.name, \mathsf{Multiplicity}, (\psi(f.name, ".", \pi(d.name)".ForeignID"), \{0..*\}))\} \\
&\qquad \cup \{(f.name, \pi(d.type.name), \mathsf{Dimension}, (d.propertyName, \phi(f), \phi(d.type), \\
&\qquad\quad (\psi(\pi(d.type.name), ".ForeignID")), \psi(\pi(d.type.name), ".ID")))\}).
\end{aligned}
\tag{7.50}
$$

Für das Auflösen einer *Owner*–Menge wird Transformation (7.50) in (7.51) auf die gesamte $Owner$–Menge (zur Berechnung siehe (7.44)) erweitert.

$$
\begin{aligned}
&\tau_{DimensionSet} : \mathcal{M}_{Dimension} \times \mathcal{R} \to \mathcal{R} \\
&\tau_{DimensionSet}(d, R) \stackrel{def}{=\!=} \bigcup_{f \in Owner(d)} \tau_{Dimension}(d, f, R).
\end{aligned}
\tag{7.51}
$$

Alle *Dimension*–Instanzen eines Schemas werden schließlich mittels (7.52) konvertiert.

$$\mathcal{T}_{Dimension} : \mathcal{M} \times \mathcal{R} \to \mathcal{R}$$
$$\mathcal{T}_{Dimension}(M, R) \stackrel{def}{=\!=} \bigcup_{d \in M_{Dimension}} \tau_{DimensionSet}(d, R). \tag{7.52}$$

### 7.2.11 Schritt 10: Transformation von *Additivity*–Schemaelementen

Für jede Faktattribut–Dimension–Kombination wird im REMUS–Schema ein Metadatum angelegt. Die Menge der zulässigen Verdichtungsoperatoren wird durch die deterministische Funktion $f_{det_{Additivity}}$ bestimmt. Abbildung 7.23 zeigt das Vorgehen exemplarisch für das Attribut „Anzahl“ bez. der Dimension „Zeit“.

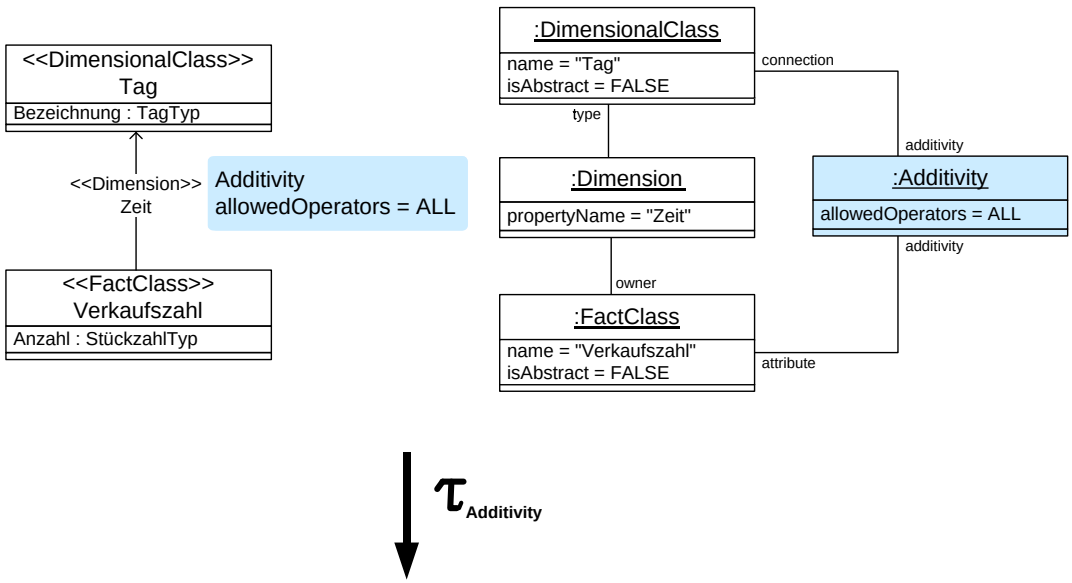



Abbildung 7.23: Transformation der Additivität

Die Funktion $f_{det_{Additivity}}$ sollte bereits im MML–Schema definierte Verdichtungsoperatoren als Ergebnis liefern. Davon soll auch in unserem Beispiel ausgegangen werden. Bedeutung bekommt die Funktion vor allem für die in Schritt 8 neu entstandenen Faktattribut–Dimension–Kombinationen. Für das Beispiel sollen
$f_{det_{Additivity}}(\text{”Verkauftes Produkt.*”}, \text{”Ort Verkauf”}) = \{\text{”SUM”}, \text{”MIN”}, \text{”MAX”}, \text{”AVG”}\}$
und $f_{det_{Additivity}}(\text{”Verkauftes Produkt.*”}, \text{”Zeit Verkauf”}) = \{\text{”ALL”}\}$

gelten.
Formal ist das Anlegen eines *Additivity*–Metadatums in (7.53) definiert.

$$\begin{aligned}
&\tau_{Additivity} : \mathcal{M}_{Dimension} \times \mathcal{M}_{FactAttribute} \times \mathcal{R} \to \mathcal{R} \\
&\tau_{Additivity}(d, a, R) \stackrel{def}{=\!=} (O, A, \\
&\quad M \cup \{(f.name, \pi(d.name), \mathsf{Additivity}, \\
&\qquad (d.propertyName, f.relation.name, f_{det_{Additivity}}(d, a)))\}).
\end{aligned} \tag{7.53}$$

Erweitert auf Mengen ergibt sich Transformation (7.54).

$$\begin{aligned}
&\tau_{AdditivitySet} : \mathcal{M}_{Dimension} \times \mathcal{M} \times \mathcal{R} \to \mathcal{R} \\
&\tau_{AdditivitySet}(d, M, R) \stackrel{def}{=\!=} \bigcup_{f \in Owner(d)} \Big( \bigcup_{a \in FactClassAttributes(f)} \tau_{Additivity}(d, a, R) \Big).
\end{aligned} \tag{7.54}$$

Schliesslich wird durch (7.55) das Anlegen aller benötigten *Additivity*–Metadaten eines Schemas erreicht.

$$\begin{aligned}
&\mathcal{T}_{Additivity} : \mathcal{M} \times \mathcal{R} \to \mathcal{R} \\
&\mathcal{T}_{Additivity}(M, R) \stackrel{def}{=\!=} \bigcup_{d \in M_{Dimension}} \tau_{AdditivitySet}(d, R).
\end{aligned} \tag{7.55}$$

### 7.2.12 Gesamttransformation

Unter Anwendung der in den Abschnitten 7.2.2 bis 7.2.11 definierten Abbildungen wird in (7.56) die Transformation eines MML–Schemas in ein REMUS–Schema definiert.

Sei $M \in \mathcal{M}, R_{Empty}$ ein leeres REMUS–Schema,
Dann ist die Abbildung $\mathcal{T}_{Schema} : \mathcal{M} \to \mathcal{R}_{Valid}$ wie folgt definiert:

$$\begin{aligned}
&\mathcal{T}_{Schema} : \mathcal{M} \times \mathcal{R}_{Valid} \to \mathcal{R}_{Valid} \\
&\mathcal{T}_{Schema}(M, R) \stackrel{def}{=\!=} \\
&\quad \mathcal{T}_{Additivity}(M, \\
&\qquad \mathcal{T}_{Dimension}(M, \\
&\qquad\quad \mathcal{T}_{FactAttribute}(M, \\
&\qquad\qquad \mathcal{T}_{FactClass}(M, \\
&\qquad\qquad\quad \mathcal{T}_{FactClass_{Composition}}(M, \\
&\qquad\qquad\qquad \mathcal{T}_{FactClass_{Generalization}}(M, \\
&\qquad\qquad\qquad\quad \mathcal{T}_{DimensionalMapping}(M \\
&\qquad\qquad\qquad\qquad \mathcal{T}_{Association}(M, \\
&\qquad\qquad\qquad\qquad\quad \mathcal{T}_{SharedRollUp}(M, \\
&\qquad\qquad\qquad\qquad\qquad \mathcal{T}_{RollUp}(M, \\
&\qquad\qquad\qquad\qquad\qquad\quad \mathcal{T}_{DimensionalAttribute}(M, \\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathcal{T}_{DimensionalClass}(M, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathcal{T}_{DataType}(M, R_{Empty}) \\
&))))))))))).
\end{aligned} \tag{7.56}$$

## 7.3 Nicht–relationale Transformationen

In diesem Abschnitt sollen Transformationsalternativen für nicht–relationale „Zielwelten“ kurz skizziert werden. Berücksichtigt werden sollen dabei das multidimensionale Datenmodell (Abschnitt 7.3.1), das objektorientierte Datenmodell (Abschnitt 7.3.2) und das objektrelationale Datenmodell (Abschnitt 7.3.3).

### 7.3.1 Multidimensionales Zielsystem

Die Transformation in eine multidimensionale Zielwelt basiert auf einem logischen Metamodell, das die in Abschnitt 3.1.1 beschriebenen statischen Aspekte der multidimensionalen Welt enthält. Die Form der Implementierung im Sinne der Klassifikation in Abschnitt 4.1 entspricht einer reinen MOLAP–Realisierung.
Die während der Transformation durchzuführenden Schritte lassen sich in drei Gruppen einteilen:

- Zunächst sind die objektorientierten Aspekte wie Vererbungen und Kompositionen aufzulösen, denn diese besitzen keine Gegenstücke im logischen, multidimensionalen Modell. Das Vorgehen kann dabei ähnlich wie in der relationalen Transformation in Abschnitt 7.2 geschehen: Vererbungen zwischen *DimensionalClass*–Instanzen können durch Nestung aufgelöst werden (Schritt 3; Abschnitt 7.2.4), für Faktklassen kann das Auflösen der Vererbungen und Kompositionen durch Weiterreichen bzw. Übertragen von Attributen und Dimensionen geschehen (Schritt 8; Abschnitt 7.2.9).
- Die multidimensionalen Aspekte des MML–Schemas können relativ kanonisch auf ihre entsprechenden Gegenstücke im logischen, multidimensionalen Modell abgebildet werden. Faktklassen mit ihren Faktattributen werden zu Kennzahlen, die *Dimension*–Instanzen kennzeichnen Dimensionen, die dimensionalen Klassen beschreiben die Hierarchieebenen und die *RollUp*–, *NonCompleteRollUp*– bzw. *SharedRollUp*–Schemaelemente geben den Dimensionen Struktur in Form von Verdichtungspfaden.
- Schließlich fallen eine Reihe von Metadaten an, die für den späteren Ladeprozess und die Aufbereitung (siehe Abschnitt 4.1) des Datenwürfels genutzt werden können. Diese sind im logischen Modell ebenfalls festzuhalten. Zu diesen Metadaten zählen z. B. durch das MML–Konstrukt *Computation* anfallende Berechnungsvorschriften oder während der Auflösung der objektorientierten Konstrukte im ersten Punkt entstandene Zusatzinformationen wie Gültigkeitsregeln für Attribute.

Als wichtiger Unterschied zur Transformation in ein relationales (oder auch objektrelationales bzw. –orientiertes) Modell bleibt festzuhalten, dass die Informationen über Verdichtungspfade nicht als Metadaten festgehalten werden, sondern in ihre „direkten Gegenstücke“ des logischen Modells transformiert werden können.

### 7.3.2 Objektorientiertes Zielsystem

Ein logisches objektorientiertes Modell stellt die „typischen“ objektorientierten Konstrukte, etwa im Umfang der UML, zur Verfügung. Aus diesem Grunde können die objektorientierten Aspekte der MML während der Transformation erhalten bleiben, d. h. Klassen, Generalisierungen etc. werden „1:1“ übernommen. Nun muss „lediglich“ noch den multidimensionalen Aspekten Rechnung getragen werden. Dabei wird für jede Teilmenge von Klassen des MML–Schemas, die über multidimensionale Kontrukte wie z. B. *Dimension*– oder *RollUp*–Instanzen verbunden sind, eine Instanz der

abstrakten Klasse „Cube“ angelegt werden (siehe Abbildung 7.25). Diese besitzt keine Attribute, aber eine Reihe von Methoden, die die in Abschnitt 3.1.2 vorgestellten dynamischen Aspekte des multidimensionalen Datenmodells realisieren.
Bei diesem Vorgehen wird durch das Definieren der „Cube“–Klasse die gleichzeitige Behandlung von Daten und Funktionen im objektorientierten Modell deutlich. Die mit „Metainformation“ bezeichnete Klasse muss im Zuge der Transformation die Informationen über die Verdichtungspfade (im linken Teil von Abbildung 7.25 die grau hinterlegten Teile) zugewiesen bekommen, damit die Methoden der Nachfolger der Klasse „Cube“ nur (gemäß der konzeptionellen Modellierung) zulässige Methodenparameter akzeptieren. So darf die Methode *rollUp* besipielsweise nur mit Dimensionen parametrisiert werden, die für diesen Würfel definiert sind.
Grundsätzlich anders als bei der relationalen bzw. multidimensionalen Transformation können Berechnungsvorschriften behandelt werden, denn diese können direkt als Methoden der Klassen realisiert werden. Abbildung 7.24 zeigt dies anhand eines *SharedRollUp*–Schemaelementes.

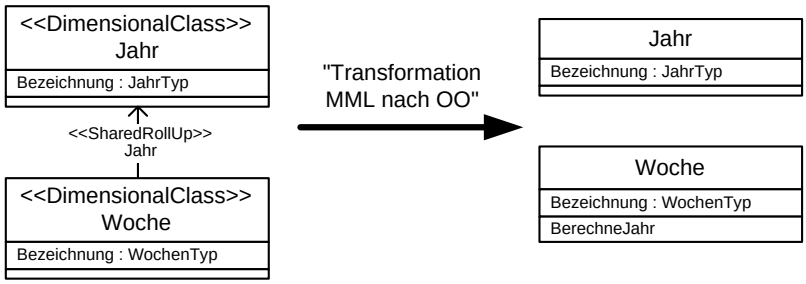



Abbildung 7.24: Transformation von Berechnungsvorschriften in ein objekorientiertes Zielsystem

### 7.3.3 Objektrelationales Zielsystem

Ein logisches objektrelationales Datenmodell [SBM99] erweitert das relationale Modell um die Konzepte zur Definition erweiterbarer, vordefinierter und benutzerdefinierter Datentypen, Funktionen und Methoden, Typkonstruktoren, Objektidentität und Referenzen.
Als Transformation sind zwei extreme Varianten denkbar: Einerseits kann bei Verzicht auf die Erweiterungen die relationale Transformation $\mathcal{T}$ verwendet werden, andererseits kann die im letzten Abschnitt skizzierte Idee für die Transformation in eine objektorientierte Zielwelt verfolgt werden. Daneben kann als „Zwischenweg“ eine Kombination der beiden Ansätze gewählt werden, wozu die einzelnen objektrelationalen Konzepte auf ihre Verwendung bez. der Realisierung eines DWH–Schemas zu bewerten sind.
Bei Verwenden von Objektidentitäten (OID) wird jedem erzeugten Tupel oder Objekt beim Anlegen ein eindeutiger Identifikator zugewiesen, mit dessen Hilfe Referenzen realisiert werden können. Im wesentlichen entspricht dieses dem Eintragen von Fremdschlüsseln, allerdings sind Referenzen über OIDen eine „natürlichere“ Form zur Darstellung der Beziehung, indem sie z. B. das direkte Navigieren zwischen verschiedenen Objekten bzw. Tupeln ermöglicht. Somit werden komplexe Anfragen u. U. einfacher, weil komplexe Verbundanfragen leichter formuliert werden können. Aus diesem Grunde sollten in den Transformationsschritten, in denen Fremdschlüsselattribute eingetragen werden, stattdessen die Navigierbarkeit anzeigende Referenzen erzeugt werden. Das Verwenden von benutzerdefinierten Typen könnte genutzt werden, um die als Datenklassen modellierten, komplexen Datentypen direkt abzubilden, d. h. das Ermitteln von Attributmengen in Transformation (7.11) in Schritt 2 und in Transformation (7.45) in Schritt 8 könnte dann entfallen. Ebenso könnten Berechnungsvorschriften wie im Abschnitt 7.3.2 beschrieben als benutzerdefinierte Methode realisiert werden.

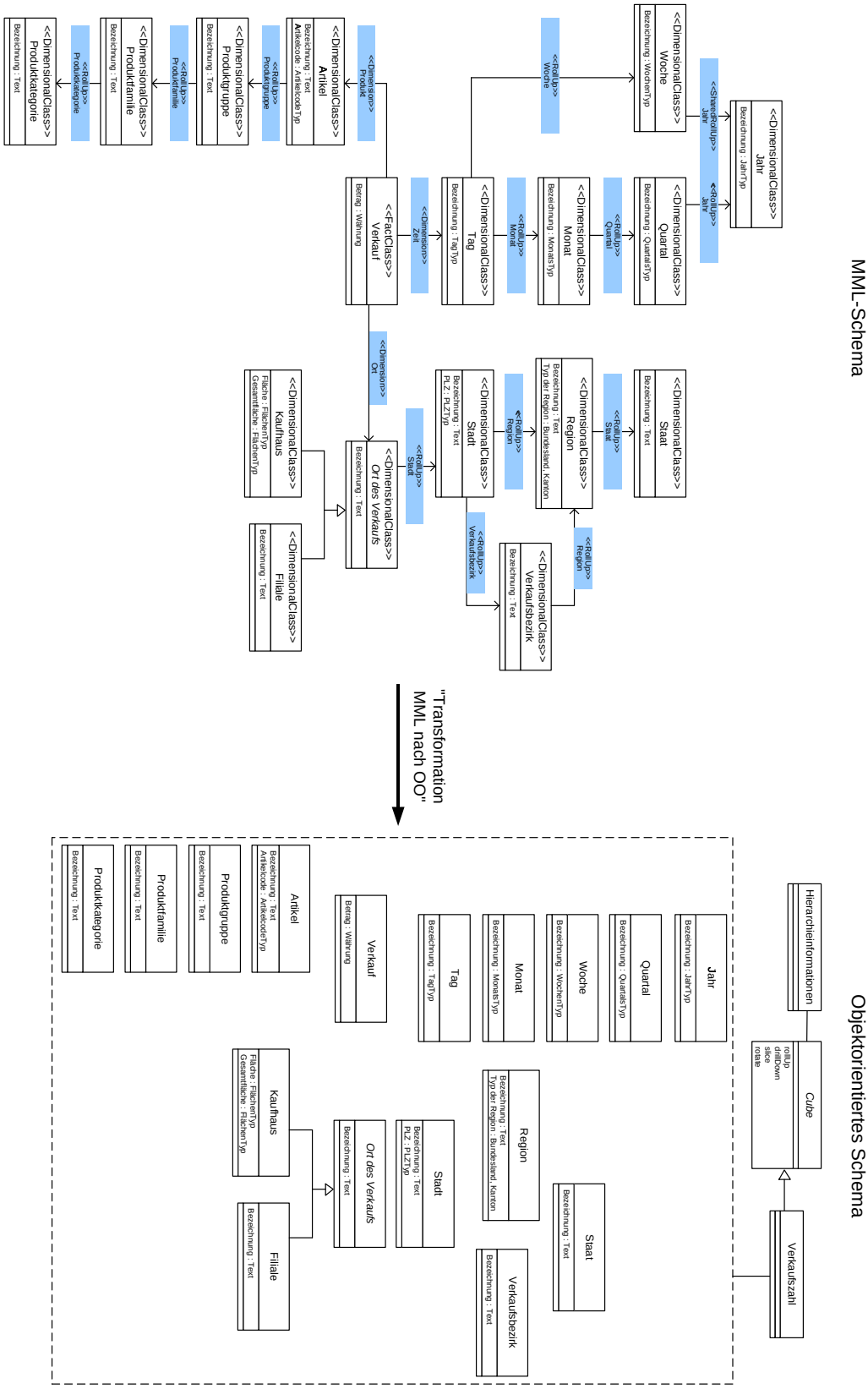



Abbildung 7.25: Transformation in ein objekorientiertes Zielsystem

## 7.4 Zusammenfassung

In diesem Kapitel wurde die Abbildung von der konzeptionellen auf die logische Entwurfsebene behandelt. Dabei wurde zunächst in Abschnitt 7.1 das logische relationale Metamodell REMUS definiert. Der eigentliche Transformationsalgorithmus wurde in Abschnitt 7.2 vorgestellt, seine Wirkungsweise wurde anhand des Beispiels „Handelswelt“ demonstriert, eine Auflistung aller erzeugten REMUS–Objekte für das Beispiel befindet sich in Anhang A.2. Der Algorithmus handelt die einzelnen MML–Konstrukte sukzessive ab und transformiert sie in entsprechende REMUS–Objekte, wobei an einigen Stellen eine deterministische Funktion $f_{det_{<X>}}$ gewisse Transformationsentscheidungen übernimmt. Diese sollte im Hinblick auf eine Implementierung als Benutzerinteraktion realisiert werden.
Als kompliziertester Teilschritt der Transformation hat sich das Auflösen von Generalisierungen und Kompositionen zwischen Faktklassen erwiesen (siehe Abschnitt 7.2.9). Weil hierbei sowohl das Übertragen von Dimensionen wie auch Attributen von der aggregierten Faktklasse zur Faktklasse auf der Detailseite sinnvoll sein kann, ist eine Reihe von Fallunterscheidungen vorzunehmen.
Tabelle 7.3 gibt einen Überblick, welche Schritte des Transformationsalgorithmus welche *REMUS*–Objekte erzeugen, welche deterministischen Funktionen benutzen und welche temporären Hilfsstrukturen verwenden.
Als Resultat ist ein logisches Schema entstanden, welches in den nächsten Schritten als Eingabe für den physischen Entwurfsprozess dient.
Abschnitt 7.3 schließlich skizziert die Transformation in nicht–relationale Datenmodelle. Hierbei wurden lediglich rudimentäre Ideenskizzen vorgestellt. Zur Verfeinerung müssen hier für jede der drei Zielwelten Metamodelle (analog zu REMUS in der relationalen Welt) definiert und dann jeweils Transformationen von MML in ein solches Metamodell (vergleichbar zur Transformation $\mathcal{T}$ aus Abschnitt 7.2) definiert werden.

| | | | **Transformation $\mathcal{T}_{Schema}$** | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Schritt 1: *DataType* | Schritt 2: *DataClass* | Schritt 3: *DimensionalClass* | Schritt 4: *DimensionalAttribute* | Schritt 5: *(Shared)RollUp* | Schritt 6: *Association* | Schritt 7: *DimensionalMapping* | Schritt 8: *FactClass* | Schritt 9: *Dimension* | Schritt 10: *Additivity* |
| **REMUS** | **Objekte** | DataTypes | • | | | | | | | | | |
| | | Dimensionale Relationen | | | • | | | | | | | |
| | | Faktrelationen | | | | | | | | • | | |
| | | Attribute | | | | • | • | | | | | |
| | **Kategorie A–Metadaten** | AggregatedAttribute | | | | | | | | • | | |
| | | Computation | | | | • | • | | • | • | | |
| | | ConceptualKey | | | | • | | • | | | | |
| | | Identifier | | | • | | | | | | | |
| | | IdentifierValue | | | • | | | | | | | |
| | | Multiplicity | | | | | • | • | | | • | |
| | | ObjectType | • | | • | | | | | • | | |
| | | Optional | | | | • | | | | • | | |
| | | PrimaryKey | | | • | | | • | | | | |
| | | Reference | | | | | • | • | | | • | |
| | | Valid | | | | • | | | | | | |
| | **Kategorie B–Metadaten** | Additivity | | | | | | | | | | • |
| | | Association | | | | | | • | | | | |
| | | Composition | | | | | | | | • | • | |
| | | Dimension | | | | | | | | | • | |
| | | DimensionalMapping | | | | | | | • | | | |
| | | RollUp | | | | | • | | | | | |
| | | SharedRollUp | | | | | • | | | | | |
| | **Funktion $f_{det}$** | ClassName | | | • | | | | | | | |
| | | Composition | | | | | | | | • | | |
| | | Additivity | | | | | | | | | | • |
| | **Temporäre Objekte** | Attributes | | • | | | | | | • | | |
| | | Aggregated | | | | | | | | • | | |
| | | Detail | | | | | | | | • | | |
| | | Owner | | | | | | | | • | • | • |

Tabelle 7.3: Transformationsschritte und erzeugte *REMUS*–Objekte

# Kapitel 8

# Relationaler Entwurf

## 8.1 Einleitung

In diesem Abschnitt wird die Transformation eines REMUS–Schemas in ein „initiales“ SQL–Schema beschrieben. Initial bedeutet in diesem Kontext, dass das aus diesem Schritt resultierende Entwurfsdokument den Ausgangspunkt für weitere Umformungen bildet. Die Einordnung dieses Schrittes in den Entwicklungsprozess ist in Abbildung 8.1 zu sehen.

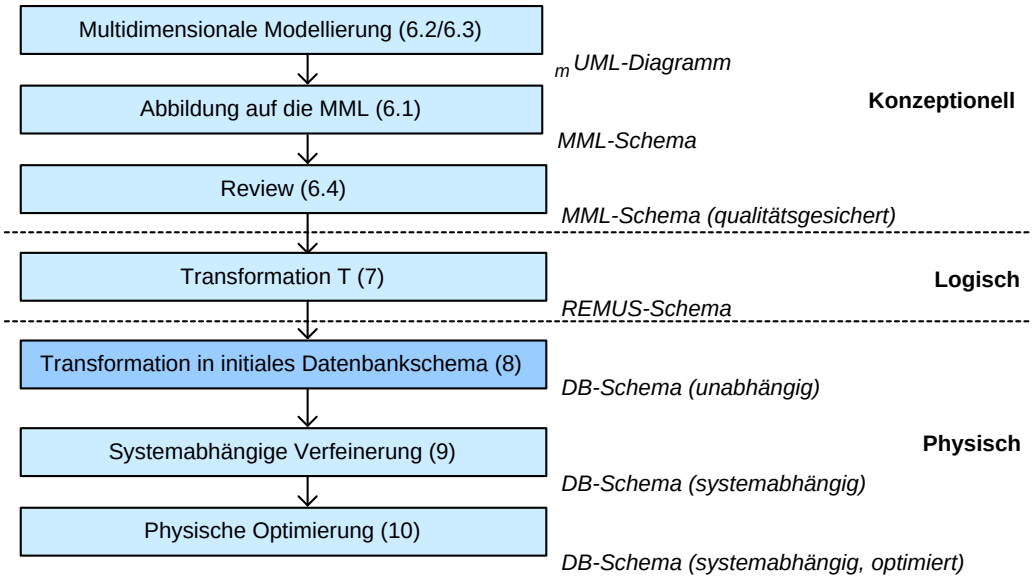



Abbildung 8.1: Einordnung des Schrittes in den Entwurfsprozess

Zunächst wird im folgenden Abschnitt 8.2 das relationale Metamodell *LCD of SQL* (Lowest Common Denominator of SQL[1]) eingeführt. Dies berücksichtigt einen gemeinsamen Kern des SQL–Standards und von bedeutenden kommerziellen Systemen[2] implementierte Konzepte. Bei der Spezifikation des *LCD of SQL*–Metamodells erfolgte eine enge Orientierung am Informationsmodell *Database and Warehousing: Database Schema* des *Open Information Model* (OIM) der *Meta Data Coalition* (siehe Abschnitt 4.4.1). Abschnitt 8.3 beschreibt die einzelnen Abbildungsschritte im Detail, illustriert anhand des Beispiels „Handelswelt“.

[1] Lowest Common Denominator = Kleinster gemeinsamer Nenner.
[2] Nach [Dat00] sind dies Oracle, IBM DB/2, Microsoft SQL Server, Informix und Sybase.

## 8.2 Das relationale Metamodell LCD of SQL

### 8.2.1 Überblick LCD of SQL

Das relationale Metamodell *LCD of SQL* ist eng an das Teilmodell *Database and Warehousing: Database Schema* des OIM angelehnt,einige der in REMUS vorhandenen Informationen haben kein direktes Gegenstück im OIM, so dass in diesem Falle entsprechende Ergänzungen notwendig sind. Zur Unterscheidung übernommener und ergänzter Elemente sei als Konvention vereinbart, dass zusätzliche Klassen und Attribute durch das Zeichen „*“ gekennzeichnet sind, sofern es sich um beschreibende Attribute handelt. Handelt es sich um Metadaten, dann wird die Endung „META“ verwendet. Das *LCD of SQL*–Metaklassendiagramm ist in Abbildung 8.2 dargestellt.

Das zentrale Element *Schema* besitzt „organisierenden“ Charakter, indem es mehrere andere Objekt zusammenfasst. Die Klassen dieser Objekte lassen sich in folgende Bereiche unterteilen:

- *Relational Basics* (siehe Abschnitt 8.2.2) stellt mit Tabellen, Attributen und Datentypen die fundamentalen Elemente des relationalen Modells zur Verfügung.
- Die im Bereich *Keys* (siehe Abschnitt 8.2.3) zusammengefassten Klassen bieten die Möglichkeit, Eindeutigkeitsschlüssel, insb. Primärschlüssel, und Fremdschlüssel zu definieren.
- Unter Verwendung dieser Schlüssel besteht die Möglichkeit, mit Hilfe von Instanzen der Klassen im Bereich *Referential Integrity* (siehe Abschnitt 8.2.4) referentielle Integritätsregeln festzulegen.
- Darüber hinausgehende Integritätsregeln werden durch Instanzen der Klassen aus dem Bereich *Constraints* (siehe Abschnitt 8.2.5) festgelegt, wobei je nach Granularität des *Constraint* zwischen Datenbank–, Tabellen– und Spalten–Constraint unterschieden wird.
- Im Bereich *Meta Data* (siehe Abschnitt 8.2.6) sind Metaklassen angeordnet, deren Objekte Informationen aufnehmen, die sich nicht als Constraints in der Datenbank repräsentieren lassen.
- Die im Bereich *Data Types* (siehe Abschnitt 8.2.7) befindlichen Metaklassen stellen Datentypen zur Verfügung, die von Attributen anderer Metaklassen verwendet werden.

In den folgenden Abschnitten werden die Metaklassen der einzelnen Bereiche detailliert vorgestellt, eine Beschreibung der einzelnen *LCD of SQL*–Metaklassen kann in [Her01b] nachgelesen werden. Alle Klassen erben von *Element* bzw. *ModelElement* aus dem *UML–Core* das Attribut *name*, mit dem der Name einer Instanz festgelegt wird. Aus Gründen der Übersichtlichkeit wurde auf das Einzeichnen dieser gemeinsamen Basisklassen in Abbildung 8.2 verzichtet; das geerbte Attribut ist jedoch bei den einzelnen Metaklassen aufgeführt. Auch sind bei den Auflistungen von Oberklassen nur diejenigen aus diesem Schema dargestellt und nicht die aus anderen Informationsmodellen des OIM bzw. der UML geerbten. An den Stellen, an denen aus diesen Quellen wichtige Attribute oder Beziehungen vererbt werden, werden diese jedoch explizit genannt.

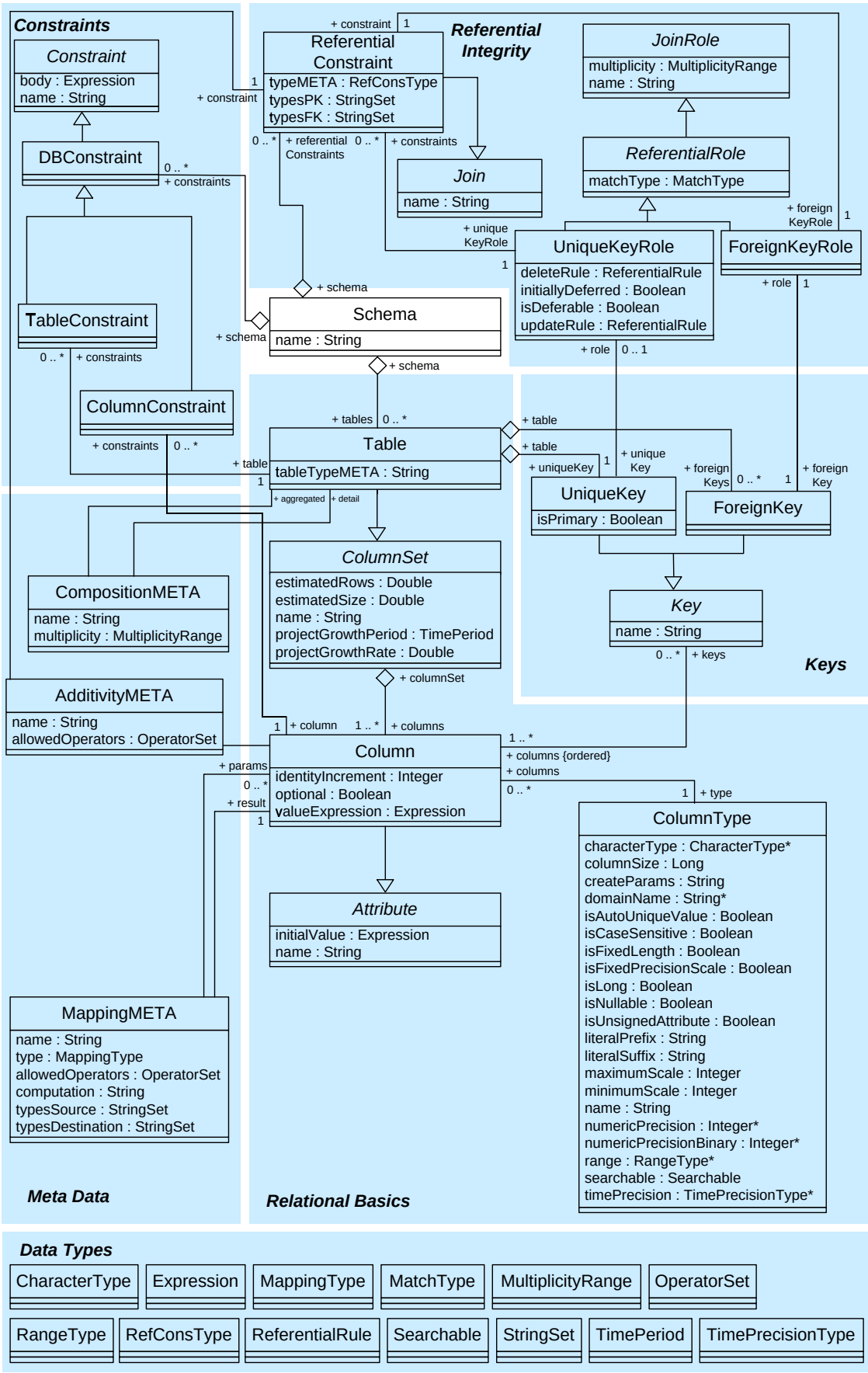



Abbildung 8.2: LCD of SQL–Metamodell

### 8.2.2 Relational Basics

Der in Abbildung 8.3 dargestellte Bereich *Relational Basics* stellt mit Tabellen, Attributen und Datentypen die fundamentalen Elemente des relationalen Modells zur Verfügung.

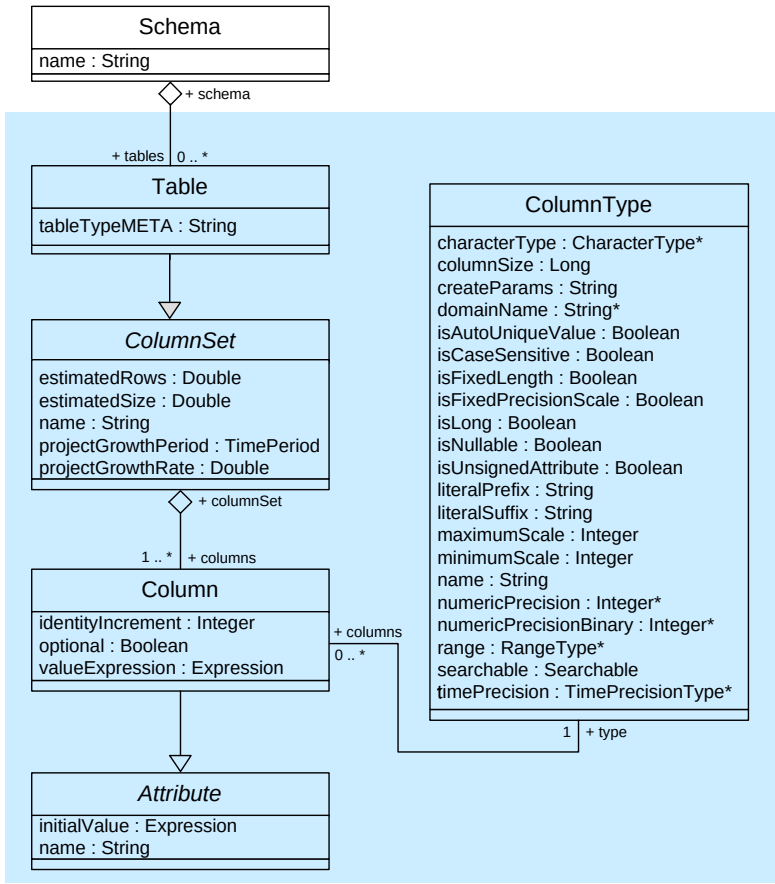



Abbildung 8.3: LCD of SQL–Metamodell: Bereich *Relational Basics*

Mit Hilfe von Instanzen der Metaklasse *ColumnType* werden die Datentypen detailliert beschrieben. Tabelle 8.2 gibt an, welche Bedeutung und zulässigen Werte die einzelnen Attribute besitzen.

**Attribute der Metaklasse *ColumnType***

| | |
|---|---|
| *characterType* : CharacterType* | Dieses Attribut spezifiziert den verwendeten Zeichensatz. |
| *columnSize* : Long | Die Länge eines nicht–numerischen Typs, die entweder das Maximum oder die festgelegte Länge dieses Typs beschreibt. Für Zeichenketten ist dieser Wert das Maximum oder die este Länge in Zeichen. Für Zeitdatentypen ist es die Länge der Zeichenkettenrepräsentation, unter Annahme der maximal erlaubten Präzision wie sie im Attribut *timePrecision* angegeben wird. Ist der Datentyp numerisch, so gibt ColumnSize die Obergrenze der maximalen Genauigkeit an. |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | |
|---|---|
| *createParams* : String | Benötigte Parameter zum Erzeugen einer Spalte dieses Typs. Die Parameter werden als eine durch Kommata getrennte Liste in der Reihenfolge ihres Auftretens genannt, ohne umschließende Anführungszeichen. Sind die Parameter Länge, maximale Länge, Genauigkeit und Skalierung, so sollten die Bezeichnungen „length“, „max length“, „precision“ und „scale“ verwendet werden, bei anderen Parametern sind entsprechende hersteller–abhängige Bezeichner zu wählen. Benötigt der Datentyp Parameter, so sollte „()“ im Typnamen erscheinen, um die Position, an der die Parameter eingesetzt werden sollen, anzuzeigen. Enthält der Typname keine Teilzeichenkette „()“, so werden die Parameter in Klammern eingeschlossen an das Ende des Datentyps angehängt. Beispiel: Der Datentypname ist „DECIMAL()“ und das Attribut *CreateParams* bekommt den Wert „precision,scale“. Beim Erzeugen einer Spalte dieses Typs wird „precision,scale“ in den Klammerausdruck der Datentypnamen eingesetzt, also „DECIMAL(precision,scale)“. |
| *domainName* : String* | Im Gegensatz zum technisch geprägten *name*–Attribut enthält dieses Attribut einen logischen Namen. |
| *isAutoUniqueValue* : Boolean | Gibt an, ob der Datentyp automatisch inkrementiert wird. |
| *isCaseSensitive* : Boolean | Definiert den Datentyp als Zeichenkette, in der Groß– und Kleinschreibung unterschieden wird. |
| *isFixedLength* : Boolean | Gibt an, ob Spalten dieses Datentyps eine feste Länge haben. |
| *isFixedPrecisionScale* : Boolean | Gibt für numerische Datentypen an, ob sie feste Vor- und Nachkommastellen haben. |
| *isLong* : Boolean | Gibt an, ob es sich um einen Binärdatentyp oder einen sehr langen Textdatentyp handelt. Was unter einem sehr langen Textdatentyp zu verstehen ist, ist systemabhängig. |
| *isNullable* Boolean | Gibt an, ob für Werte der Spalte dieses Datentyps NULL–Werte möglich sein sollen. |
| *isUnsignedAttribute* Boolean | Gibt an, ob der Datentyp vorzeichenlos ist. |
| *literalPrefix* String | Die Zeichenfolge, die ein Literal dieses Typs als Präfix erhält. |
| *literalSuffix* String | Die Zeichenfolge, die ein Literal dieses Typs als Suffix erhält. |
| *maximumScale* Integer | Gibt bei einem numerischen Datentyp die maximale Anzahl an Nachkommastellen an. |
| *minimumScale* Integer | Gibt bei einem numerischen Datentyp die minimale Anzahl an Nachkommastellen an. |
| *name* String | Die Bezeichnung des Datentyps, im Gegensatz zum Attribut *domainName* steht hier ein technischer Name. |
| *numericPrecision* : Integer* | Maximale Anzahl an Stellen zur Basis 10, die für ein numerisches Attribut gespeichert werden können. |
| *numericPrecisionBinary* : Integer* | Maximale Anzahl an Stellen zur Basis 2, die für ein numerisches Attribut gespeichert werden können. |
| *range* : RangeType* | Liste von Intervallen, dient der Darstellung von Aufzählunstypen oder eingeschränkten Domänen, z. B. positive Integer. |
| *searchable* : Searchable* | Zeigt an, ob auf diesem Datentyp gesucht werden kann; ist dies nicht der Fall, dann ist der Wert NULL. |
| *timePrecision* : TimePrecisionType* | Gibt den Granularitätsgrad eines Zeit–Datentyps an. Gilt nur für Time und Unterklassen. |

Tabelle 8.2: LCD of SQL: Attribute der Metaklasse *ColumnType*

Die abstrakte Klasse *Attribute* stammt aus der UML bzw. aus dem OIM–Teilmodell *UML Extensions* und repräsentiert elementare Attribute, die einen Namen (Attribut *name*) und optional einen initialen

Standardwert (Attribut *initialValue*) besitzen. Diese Metaklasse wird zur Metaklasse *Column* spezialisiert; jede Instanz von *Column* repräsentiert eine Spalte des relationalen Schemas, d. h. alle ihre Werte müssen vom gleichen Datentyp bzw. der gleichen Domäne sein; der Wert einer Spalte ist die kleinste Dateneinheit, die abgefragt bzw. aktualisiert werden kann. Das Attribut *optional* charakterisiert eine Spalte als (nicht–)optional, mittels *valueExpression* kann für Spalten eine Berechnungsvorschrift angegeben werden. Der Wert des Attributes *identityIncrement* gibt das Inkrement an, um das eine Identitätsspalte bei Einfügen eines neuen Datensatzes erhöht wird. Ist dieser Wert größer als 0, dann wird diese Spalte als Identitätsspalte interpretiert.
Eine Menge von *Column*–Objekten bildet einen *ColumnSet*, der neben einem eindeutigen Namen einige statistische Informationen enthält. *estimatedRows* gibt die geschätzte Anzahl an Zeilen des *ColumnSets* an, *estimatedSize* die geschätzte Größe eines *ColumnSets*. Mit *projectGrowthPeriod* wird die Zeitspanne mit einem ganzzahligen numerischen Wert, der den Zuwachszeitraum in Tagen angibt, beschrieben, in der der im Attribut *ProjectGrowthRate* angegebene Zuwachs erwartet wird, während das Attribut *projectGrowthRate* den erwarteten Zuwachs des *ColumnSets* angibt. In Verbindung mit dem Attribut *ProjectGrowthPeriod* kann hieraus die Zuwachsrate berechnet werden[3]. Als Spezialisierung der Metaklasse *ColumnSet* wird die Metaklasse*Table (LCD of SQL)@Table (LCD of SQL)* eingeführt. Jede Instanz beschreibt eine Tabelle, wie sie aus der relationalen Welt bekannt ist. Als zusätzliches Metadatum wird das Attribut *tableTypeMETA** festgelegt, das den Typ der Tabelle angibt; zulässige Werte sind „FACT“ und „DIMENSION“.

### 8.2.3 Keys

Im Bereich *Keys* werden Schlüssel festgelegt. Der relevante Schemaausschnitt ist in Abbildung 8.4 zu sehen.

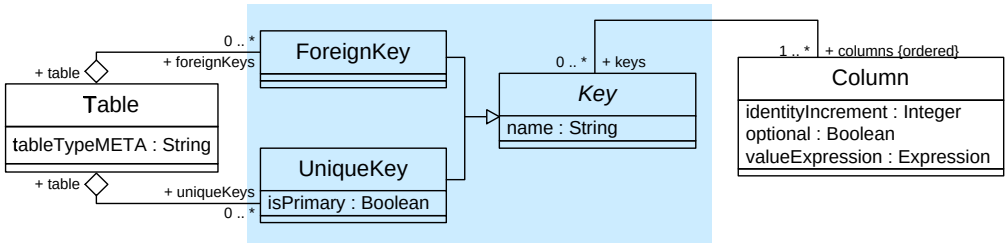



Abbildung 8.4: LCD of SQL–Metamodell: Bereich *Keys*

Ausgangspunkt ist die abstrakte Metaklasse *Key*, die einen Schlüssel als eine geordnete Menge von Spalten festlegt. Sie vererbt an ihre beiden Unterklassen *ForeignKey* und *UniqueKey*. Instanzen der *UniqueKey*–Metaklasse legen eindeutige Schlüssel fest, d. h. sie verweisen auf eine Teilmenge von Spalten, deren Werte eindeutig sein müssen. Trägt das Attribut *isPrimary* den Wert „TRUE“, handelt es sich um einen Primärschlüssel. Instanzen der *ForeignKey*–Metaklasse beschreiben Fremdschlüsseleinträge, d. h. eine geordnete Menge von Spalten, die die Primärschlüsselspalten einer anderen Tabelle referenzieren.

### 8.2.4 Referential Integrity

Die Metaklassen des Bereichs *Referential Integrity* und ihre Instanzen ermöglichen unter Nutzung der in Abschnitt 8.2.3 definierten Schlüssel die Festlegung referentieller Integritäten innerhalb eines

[3] Auch wenn diese vier Attribute in diesem Entwurfsschritt noch nicht benötigt werden, werden sie wegen der Anlehnung von *LCD of SQL* an das *OIM* hier schon genannt.

Schemas. Der Schemaausschnitt ist in Abbildung 8.5 wiedergegeben.

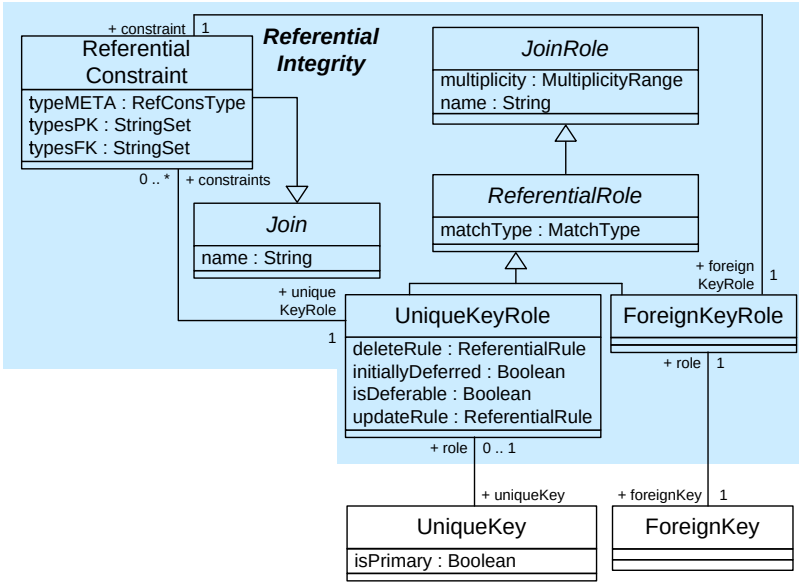



Abbildung 8.5: LCD of SQL–Metamodell: Bereich *Referential Integrity*

Die abstrakte Metaklasse *JoinRole* definiert allgemein die Rolle einer Verbindung, die durch einen Namen und die Angabe der Multiplizität der Beziehung gekennzeichnet ist. Die Metaklasse *ReferentialRole* spezialisiert die abstrakte Metaklasse *JoinRole* um das Attribut *matchType*, welches die Art der Beziehung spezifiziert, indem angegeben wird, ob jede referenzierende Spalte jeder referenzierten entsprechen muss. Zulässige Werte sind „MATCHTYPE_FULL_MATCH“ und „MATCHTYPE_PARTIAL_MATCH“. Durch Spezialisierung von *ReferentialRole* ensteht die Metaklasse *UniqueKeyRole*, die das Verhalten eines Primärschlüssels spezifiziert. Dies geschieht mittels der vier Attribute *deleteRule* (gibt das Verhalten im Falle des Löschens an), *initiallyDeferred* (gibt an, ob das Änderungsverhalten anfangs als verzögert definiert ist), *isDeferrable* (gibt an, ob das Änderungsverhalten als verzögert definiert werden kann) sowie *updateRule* (gibt das Verhalten im Falle einer Änderung an). Zulässige Werte für die Attribute *deleteRule* und *updateRule* sind „REFERENTIALRULE_CASCADE“ (Änderung oder Löschung wird an referenzierten Datensatz übertragen), „REFERENTIALRULE_SET_NULL“ (die entsprechenden Spalten im referenzierten Datensatz werden auf „NULL“ gesetzt), „REFERENTIALRULE_SET_DEFAULT“ (die entsprechenden Spalten im referenzierten Datensatz werden auf ihren Standardwert gesetzt), „REFERENTIALRULE_NO_ACTION“ (der referenzierte Datensatz bleibt unverändert) und „REFERENTIALRULE_RESTRICT*“ (Änderungen und Löschungen sind nicht erlaubt, wenn noch referenzierte Datensätze existieren).
Auch die Metaklasse *ForeignKeyRole* spezialisiert die abstrakte Metaklasse *ReferentialRole*, hat keine eigenen Attribute, sondern lediglich einen Verweis auf das/die *ForeignKey*–Attribut/e. Sowohl die *UniqueKeyRole*– wie auch die *ForeignKeyRole*–Metaklasse referenzieren die Metaklasse *ReferentialConstraint*. Diese ist von der abstrakten Metaklasse *Join* abgeleitet, die eine allgemeine Verbindung zwischen zwei Tabellen unter Nutzung eines Schlüssels von jeder Tabelle festlegt. Neben dem von dieser Klasse geerbten Attribut *name*, besitzt *ReferentialConstraint* drei weitere Metadaten, die die referentielle Beziehung hinsichtlich multidimensionaler Eigenschaften genauer beschreiben. Während *typeMETA* den Typ des ursprünglichen Verbindungskonstruktes angibt (zulässige Werte sind „ASSOCIATION“, „DIMENSION“ und „ROLL_UP“ und „NON_STRICT_ROLL_UP“), werden in *typesPK* bzw. *typesFK* die zulässigen Typen auf der Seite des Primär– bzw. Fremdschlüssels festgehalten.

### 8.2.5 Constraints

Objekte der im Bereich *Constraints* definierten Metaklassen, die im Metamodellausschnitt in Abbildung 8.6 zu sehen sind, ermöglichen die Beschreibung über die referentielle Integrität hinausgehender Integritätsbedingungen.

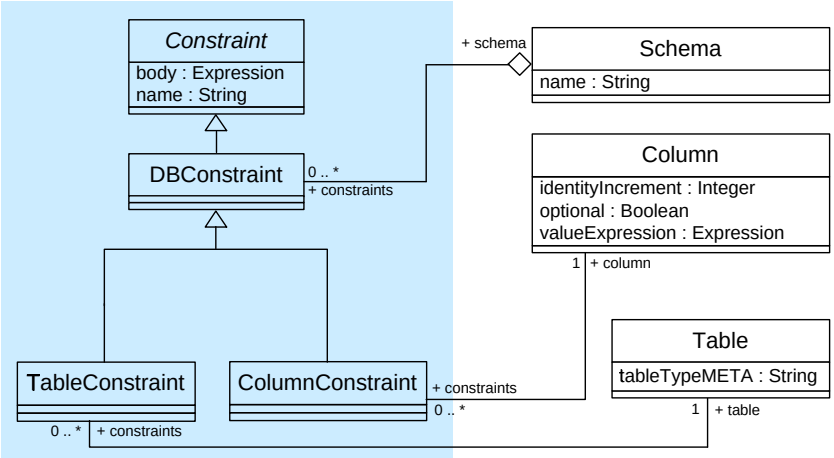



Abbildung 8.6: LCD of SQL–Metamodell: Bereich *Constraint*

Ausgangspunkt ist die abstrakte Metaklasse *Constraint*, die aus dem OIM–Teilmodell UML–Core stammt. Während *name* der eindeutige Bezeichner der Integritätsregel ist, definiert *body* die Integritätsregel. Auf eine genauere Spezifikation, wie diese Beschreibung auszusehen hat, wird verzichtet. Sie kann z. B. in Boolescher Logik oder SQL–Pseudocode erfolgen.
Je nach Typ des Bezugsobjektes der Integritätsregel gibt es drei Unterklassen von *Constraint*: *DBConstraint*, *TableConstraint* und *ColumnConstraint*. Diese haben keine weiteren Attribute, sie unterscheiden sich lediglich in den unterschiedlichen Referenzen zu ihren Bezugsobjekten.

### 8.2.6 Meta Data

In den Objekten der Metaklassen des Bereichs *Meta Data* werden Information festgehalten, die sich nicht mit anderen *LCD of SQL*–Objekten beschreiben lassen. Der Ausschnitt aus dem Klassendiagramm ist in Abbildung 8.7 dargestellt. Im einzelnen sind dies Metainformationen über die Additivität von Kennzahlen bez. Dimensionen, die Multiplizität aufgelöster Kompositionen und berechnete Beziehungen zwischen Dimensionen, die sich aus den MML–Konstrukten *SharedRollUp* und *DimensionalMapping* ergeben können. Diese Informationen werden nicht als *ReferentialConstraint* oder *DBConstraint* gespeichert, weil sie auswertungsbezogene (bzw. die Informationen über die Kompositionen für den Ladevorgang des DWH relevante), für die spätere Verwendung der Daten nützliche Metainformationen sind. Sie dienen somit einem OLAP–Server bzw. einem Ladewerkzeug als Zusatzinformationen. Objekte der beiden Metaklassen *ReferentialConstraint* und *DBConstraint* stellen hingegen die (statische) Datenintegrität sicher und sind keine Informationen, die von auf die DB zugreifenden Komponenten genutzt werden. Für die Wahl einer eigenen Metaklasse zur Speicherung der Additivitätseigenschaft gilt noch zusätzlich das Argument, dass ein *ReferentialConstraint* die Beziehung von zwei Tabellen, Additivität aber eine Beziehung zwischen Faktattribut und einer dimensionalen Relation beschreibt.
Die Klasse *AdditivityMETA* wird durch die Bezeichnung der Addivität (Attribut *name*) und eine Menge zulässiger Verdichtungsoperatoren (Attribut *allowedOperators*) definiert. Die beiden Referenzen

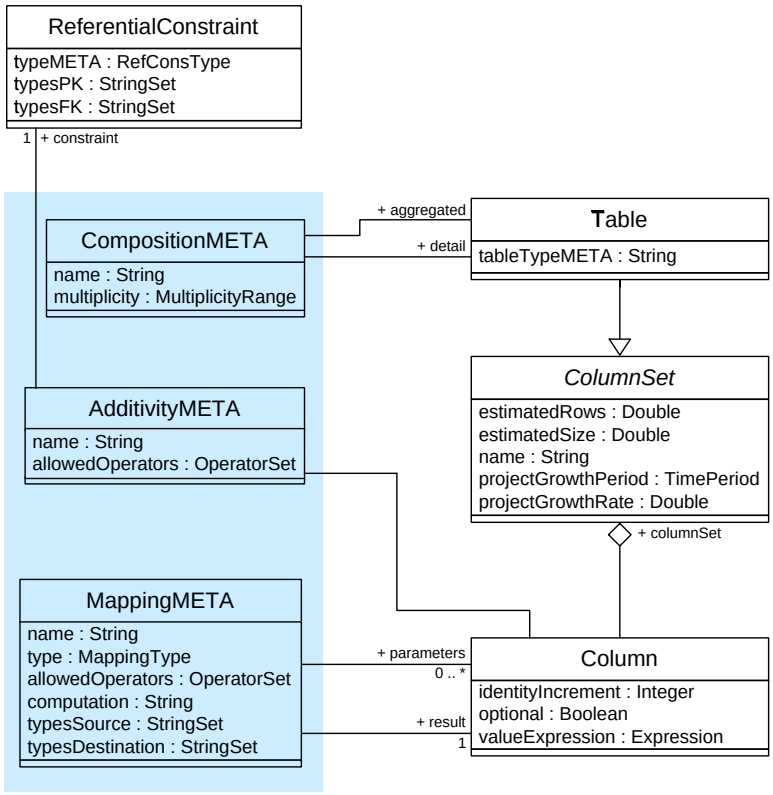



Abbildung 8.7: LCD of SQL–Metamodell: Bereich *Meta Data*

dieser Metaklasse verweisen auf das Faktattribut und das die Dimension realisierende *Referential-Constraint*–Objekt. Objekte der Metaklasse *CompositionMETA* beschreiben ursprünglich im Schema vorhandene Kompositionsbeziehungen mittels deren Bezeichnung (Attribut *name*) und der Multiplizität der an der Komposition beteiligten Objekte (Attribut *multiplicity*). Referenziert werden die beiden Tabellen, auf die die Faktklasse der aggregierten Seite bzw. die Faktklasse der Detailseite abgebildet wurden. *MappingMETA* hält Eigenschaften von Berechnungsvorschriften fest, die sich bei der Auflösung der Beziehungstypen *SharedRollUps* und *DimensionalMappings* zwischen Dimensionen, ergeben. Dabei wird über die Referenz *parameters* auf die Spalten verwiesen, die in die Berechnung als Parameter eingehen, und über die Referenz *result* die Spalte, die das Ergebnis der Berechnung enthält. Eine solche Berechnung ist darüber hinaus durch die Attribute *name* (Bezeichnung der Abbildungsbeziehung), *type* (gibt die Herkunft für der Beziehung an; zulässige Werte sind „SHARED_ROLL_UP“ und „DIMENSIONAL_MAPPING“), *allowedOperators* (gibt die zulässigen Verdichtungsoperatoren an, die mit dieser Abbildungsbeziehung verträglich sind), *computation* (gibt die Berechnungsvorschrift an) sowie *typesSource* und *typesDestination*, die gültigen Typen für die die Parameter bzw. das Resultat enthaltenden Tabelle angeben.

### 8.2.7 Datentypen

Während in einem *REMUS*–Schema als Datentypen die aus dem *MML*–Schema übernommenen sprechenden Bezeichner verwendet wurden (siehe Transformationsschritt 1 auf Seite 126), sollen in einem *LCD of SQL*–Schema die *Common Data Types* des OIM–Teilmodells *Analysis and Design: Common Data Types* Anwendung finden. Diese sind in Abbildung 8.8 dargestellt.

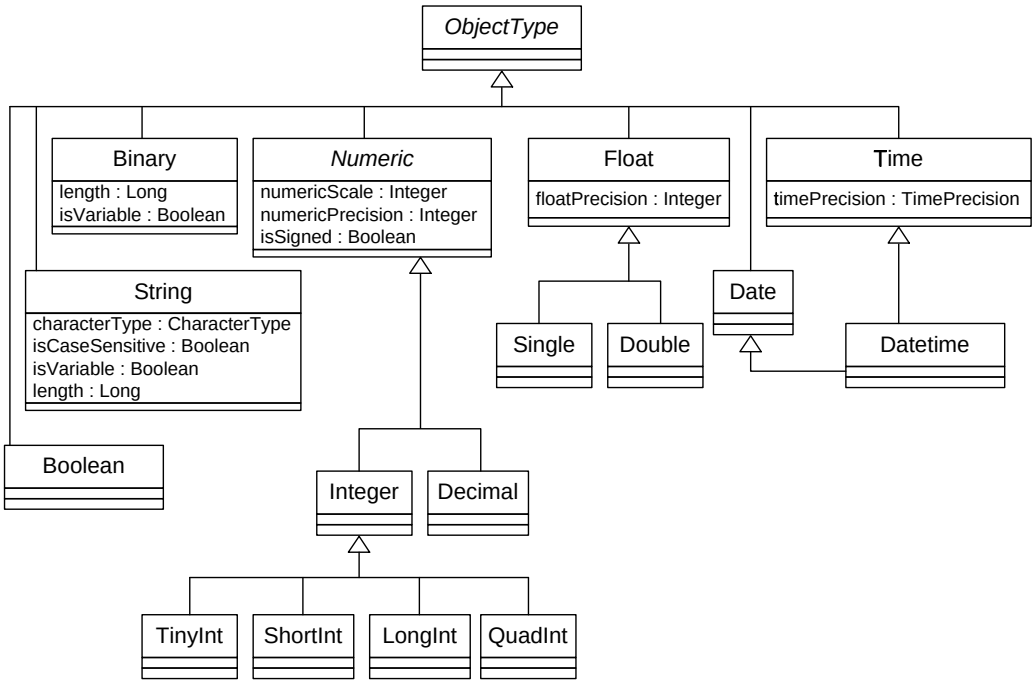



Abbildung 8.8: *Common Data Types* des OIM

Weil die dort definierten Datentypen bzw. die Attributbezeichnungen aber nicht exakt denen im OIM–Teilmodell *Database and Warehousing: Database Schema* entsprechen, ist eine Abbildung der Datentypen notwendig. Auf welche Art und Weise dies geschieht, ist in Tabelle 8.4 zu festgehalten.

## Abbildung der Datentypen

| **Common Data Types** | | | **LCD of SQL** |
|---|---|---|---|
| **Type** | **Attribute** | **Mögliche Werte** | **Attribut der Klasse** *ColumnType* |
| Autoincrement | — | — | isAutoUniqueValue=TRUE |
| Binary | length | — | columnSize |
| | isVariable | — | isFixedLength |
| Boolean | — | — | — |
| Date | — | — | — |
| Datetime | timePrecision | 0–11 | timePrecision |
| Decimal | numericScale | >=0 | maximumScale |
| | numericPrecision | >=0 | numericPrecision |
| | isSigned | TRUE,FALSE | isUnsignedAttribute = NOT isSigned |
| | floatPrecision | >=0 | numericPrecision |
| Double | — | — | isUnsignedAttribute=FALSE |
| | floatPrecision | 53 | numericPrecision=53 |
| Integer | numericScale | 0 | maximumScale |
| | numericPrecision | >=0 | numericPrecision |
| | isSigned | TRUE,FALSE | isUnsignedAttribute = NOT isSigned |
| Float | — | — | isUnsignedAttribute=FALSE |
| | floatPrecision | >=0 | numericPrecision |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | | | |
|---|---|---|---|
| LongInt (DoubleWord 4 Byte) | NumericScale | 0 | maximumScale |
| | NumericPrecision | 0–10 | numericPrecision |
| | isSigned | TRUE,FALSE | isUnsignedAttribute = NOT isSigned |
| QuadInt (QuadWord 8 Byte) | numericScale | 0 | maximumScale |
| | numericPrecision | 0–19 (signed) 20 (unsigned) | numericPrecision |
| | isSigned | TRUE,FALSE | IsUnsignedAttribute = NOT isSigned |
| ShortInt (Word 2 Byte) | numericScale | 0 | maximumScale |
| | numericPrecision | 0–5 | numericPrecision |
| | isSigned | TRUE,FALSE | isUnsignedAttribute |
| Single | — | — | IsUnsignedAttribute=FALSE |
| | floatPrecision | 24 | NumericPrecision =24 |
| String | isVariable | — | isFixedLength = NOT isVariable |
| | length | — | columnSize |
| | isCaseSensitive | — | isCaseSensitive |
| | characterType | — | characterType |
| Time | timePrecision | 0–11 | timePrecision |
| TinyInt (HalfWord 1 Byte) | numericScale | 0 | maximumScale |
| | numericPrecision | 0–3 | numericPrecision |
| | isSigned | TRUE,FALSE | isUnsignedAttribute = NOT isSigned |

Tabelle 8.4: Abbildung der Datentypen

Die Tabelle ist folgendermaßen zu lesen: Soll z. B. der Datentyp „Integer“ verwendet werden, so ist für das *ColumnType*–Objekt das Attribut *maximumScale* auf „0“ und *numericPrecision* auf die gewünschte Genauigkeit zu setzen, während *isUnsignedAttribute* bestimmt, ob es sich um einen Typen mit oder ohne Vorzeichen handelt.

## 8.3 Abbildung von REMUS nach *LCD of SQL*

### 8.3.1 Vorgehensweise

Die prinzipielle Vorgehensweise bei der Überführung eines REMUS– in ein *LCD of SQL*–Schema ist in Abbildung 8.9 dargestellt. Der dreiphasige Entwurfsschritt beginnt mit dem Übertragen der Objekte (Schritte 1 und 2) und Attribute (Schritt 3) des REMUS–Schemas. In den beiden weiteren Phasen (Abarbeitung der Kategorie A– bzw. Kategorie B–Metadaten) schließt sich eine Metadaten–orientierte Vorgehensweise an. In der zweiten Phase, die die Schritte 4 bis 9 umfasst, werden zunächst die Kategorie A–Metadaten übertragen, wobei referentielle Integritäten und *Constraints* auf Tabellen abgebildet bzw. Spalten angelegt werden. Die letzte Phase, die die Schritte 10 bis 15 umfasst, sorgt für die Übertragung der Kategorie B–Metadaten. Hierbei werden neben dem Anlegen von *DBConstraint*–Objekten auch weitere Metadaten erzeugt.

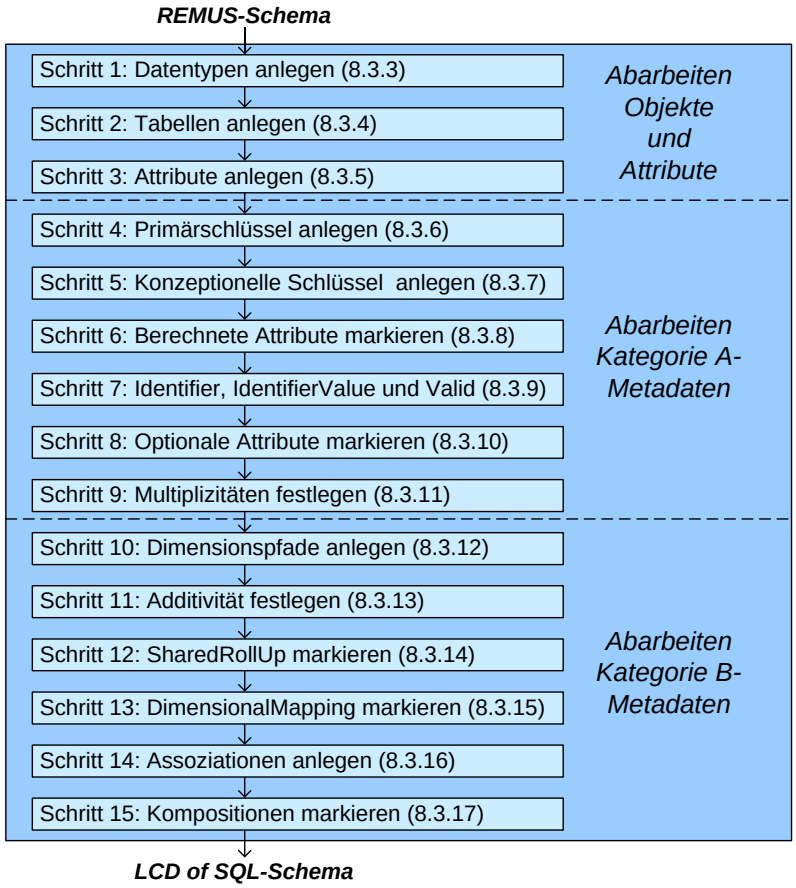



Abbildung 8.9: Vorgehensweise der Abbildung von REMUS nach *LCD of SQL*

### 8.3.2 Notationsvereinbarungen

**REMUS**

Analog zu der in Kapitel 7 beschriebenen Transformation eines MML– in ein REMUS–Schema wird auch das Übertragen in ein *LCD of SQL*–Schema neben einer informalen Beschreibung nebst Erklärung mit Hilfe von Abbildungen durch eine formale Beschreibung mittels mengenwertiger Abbildungsvorschriften begleitet. Dazu gelten für REMUS–Schemata die auf Seite 125 vorgenommenen Definitionen (7.5) bis (7.7). Zusätzlich werden in (8.1) einzelne REMUS–Metadatentypen definiert.

$$\begin{aligned}\text{Sei } t \in \{&\text{"}AggregatedAttribute\text{"}, \text{"}Computation\text{"}, \text{"}ConceptualKey\text{"},\\ &\text{"}Identifier\text{"}, \text{"}IdentifierValue\text{"}, \text{"}Multiplicity\text{"}, \text{"}ObjectType\text{"},\\ &\text{"}Optional\text{"}, \text{"}PrimaryKey\text{"}, \text{"}Reference\text{"}, \text{"}Valid\text{"},\\ &\text{"}Additivity\text{"}, \text{"}Association\text{"}, \text{"}Composition\text{"}, \text{"}Dimension\text{"},\\ &\text{"}DimensionalMapping\text{"}, \text{"}RollUp\text{"}, \text{"}SharedRollUp\text{"}\}.\end{aligned} \tag{8.1}$$

$META_{<t>}$ sei die Menge aller REMUS–Metadaten des Typs $t$.

So bezeichnet z. B. $META_{Dimension}$ die Menge aller *Dimension*–Metadaten.
An manchen Stellen wird innerhalb eines Schemas der Zugriff nur auf bestimmte Metadaten(typen) notwendig sein. Daher werden in (8.2) *Einschränkungen* definiert.

Sei $R = (O, A, M)$ ein REMUS–Schema. Sei $t \in Wertebereich(type)$.
Dann sei die *Einschränkung* $M_{\downarrow(a,t,(b))}$ definiert als

$$M_{\downarrow(a,t,(b))} \stackrel{def}{=} \{m \in M \mid type(m) = t \wedge m.a = a \wedge m.b = b\}. \quad (8.2)$$

a und b können durch das „*“–Symbol als Platzhalter für alle Einträge dieser Komponente ersetzt werden.

Beispielsweise bezeichnet $M_{\downarrow(\text{"Verkaufszahlen"},\text{PrimaryKey},(*))}$ die Primärschlüsselattribute der Relation „Verkaufszahlen“ und $M_{\downarrow(*,\text{PrimaryKey},(*))}$ alle *PrimaryKey*–Metadaten eines Schemas.
Zur besseren „Lesbarkeit“ der Transformationsabbildungen im folgenden Abschnitt wird die im letzten Kapitel in den Tabellen 7.1 und 7.2 festgelegte Tupelnotation in den Tabelle 8.5 und 8.6 auf sprechende Bezeichner abgebildet.

## REMUS: Kategorie A–Metadaten, Langform

| **AggregatedAttribute** | |
|---|---|
| $(R$, AggregatedAttribute, $(A, M))$ | $(relation.name$, AggregatedAttribute, $(attribute.name, multiplicity))$ |
| **Computation** | |
| $(C$, Computation, $((P_1, ..., P_n), F, R))$ | $(computation$, Computation, $((parameters), formula, result))$ |
| **ConceptualKey** | |
| $(R$, ConceptualKey, $(A))$ | $(relation.name$, ConceptualKey, $(attribute.name))$ |
| **Identifier** | |
| $(R$, Identifier, $(A))$ | $(relation.name$, Identifier, $(attribute.name))$ |
| **IdentifierValue** | |
| $(A$, IdentifierValue, $(\{V_1, ..., V_n\}))$ | $(attribute.name$, IdentifierValue, $(values))$ |
| **Multiplicity** | |
| $(R$, Multiplicity, $(\{A_1, ..., A_n\}, M))$ | $(relation.name$, Multiplicity, $(attributes, multiplicity))$ |
| **ObjectType** | |
| $(O$, ObjectType, $(T, D))$ | $(object.name$, ObjectType, $(type, description))$ |
| **Optional** | |
| $(R$, Optional, $(A))$ | $(relation.name$, Optional, $(attribute.name))$ |
| **PrimaryKey** | |
| $(R$, PrimaryKey, $(A))$ | $(relation.name$, PrimaryKey, $(attribute.name))$ |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| **Reference** | |
|---|---|
| $(A_1$, Reference, $(A_2))$ | $(fkAttribute.name$, Reference, $(pkAttribute.name))$ |
| **Valid** | |
| $(A$, Valid, $(I, \{V_1, ..., V_n\}))$ | $(attribute.name$, Valid, $(identifier.name, values))$ |

Tabelle 8.5: REMUS: Langform Kategorie A–Metadaten

## REMUS: Kategorie B–Metadaten, Langform

| **Additivity** | |
|---|---|
| $(A, R$, Additivity, $(D, F, O))$ | $(factAttribute.name, dimensionalRelation.name,$ Additivity, $(dimension, factRelation.name, allowedOperators))$ |
| **Association** | |
| $(R_1, R_2$, Association, $(R_3, L_1, L_2, T_1, T_2,$ $(P_{R_1}), (F_{R_1}), (P_{R_2}), (F_{R_2})))$ | $(dimensionalRelationA.name,$ $dimensionalRelationB.name,$ Association, $(intermediateRelation.name,$ $roleA, roleB, validTypesA, validTypesB,$ $(primaryKeyA), (foreignKeyA),$ $(primaryKeyB), (foreignKeyB)))$ |
| **Composition** | |
| $(R_1, R_2$, Composition, $(C, M))$ | $(factRelationAggregated.name,$ $factRelationDetail.name$ Composition, $(name, multiplicity))$ |
| **Dimension** | |
| $(R_1, R_2$, Dimension, $(D, T_1, T_2, F, P))$ | $(factRelation.name,$ $dimensionalRelation.name,$ Dimension, $(name,$ $validTypesFact, validTypesDimension,$ $foreignKey.name, primaryKey.name))$ |
| **DimensionalMapping** | |
| $(R_1, R_2$, DimensionalMapping, $(D, T_1, T_2, C))$ | $(dimensionalRelationSource.name,$ $dimensionalRelationDestination.name,$ DimensionalMapping, $(name,$ $validTypesSource, validTypesDestination,$ $computation.name))$ |
| **RollUp** | |
| $(R_1, R_2$, RollUp, $(R, T_1, T_2, F, P, S))$ | $(dimensionalRelationLower.name,$ $dimensionalRelationHigher.name,$ RollUp, $(name,$ $validTypesLowerLevel, validTypesHigherLevel,$ $foreignKey.name, primaryKey.name,$ $type))$ |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| **SharedRollUp** | |
|---|---|
| $(R_1, R_2,$ SharedRollUp, $(S, T_1, T_2, C, O))$ | $(dimensionalRelationLower.name,$ $dimensionalRelationHigher.name,$ SharedRollUp, $(name,$ $validTypesLowerLevel, validTypesHigherLevel,$ $computation.name, allowedOperators))$ |

Tabelle 8.6: REMUS: Langform Kategorie B–Metadaten

**LCD of SQL**

Ein *LCD of SQL*–Schema wird in (8.3) definiert.

Ein *LCD of SQL*–Schema ist ein Paar $L = (O, type)$ mit
(i) $O$ ist eine endliche, nichtleere Menge von Objekten
(ii) $type$ ist die Funktion, die jedem $o \in O$ seinen Typ zuweist, d. h.

$$type : O \to \{"AdditivityMETA", "Column", "ColumnConstraint", "ColumnType", "DatabaseConstraint", "ForeignKey", "MappingMETA", "ReferentialRole", "Table", "TableConstraint", "UniqueKey"\} \tag{8.3}$$

$type(o) \stackrel{def}{=}$ "Objekttyp von $o$".
$\mathcal{L}$ sei die Menge aller *LCD of SQL*–Schemata.

Für die Abbildung notwendige Mengen bestimmter Schemaelementtypen werden in (8.4) festgelegt.

$$\begin{aligned} &\text{Sei } t \in Wertebereich(type).\\ &\mathcal{L}_{<t>} \text{ sei die Menge aller Objekte vom Typ } t. \end{aligned} \tag{8.4}$$

So beschreibt beispielsweise $\mathcal{L}_{Table}$ die Menge aller Tabellen (aller *LCD of SQL*–Schemata).
Eine Schreibweise für den Zugriff auf alle Objekte eines bestimmten Typs in einem speziellen *LCD of SQL*–Schema wird in (8.5) festgelegt.

$$\begin{aligned} &\text{Sei } L = (O, type) \in \mathcal{L} \text{ ein } \textit{LCD of SQL}\text{–Schema. Sei } t \in Wertebereich(type).\\ &\text{Dann sei } O_{<t>} \stackrel{def}{=} \{o \in O \mid type(o) = t\} \text{ die Menge aller Objekte vom Typ } t. \end{aligned} \tag{8.5}$$

So ist beispielsweise $O_{AdditivityMETA}$ die Menge aller Additivitätsmetadaten innerhalb des Schemas. Wie schon für MML– und REMUS–Schemata in Abschnitt 7.2.1 wird auch für *LCD of SQL*–Schemata *Gültigkeit* definiert, was in (8.6) geschieht.

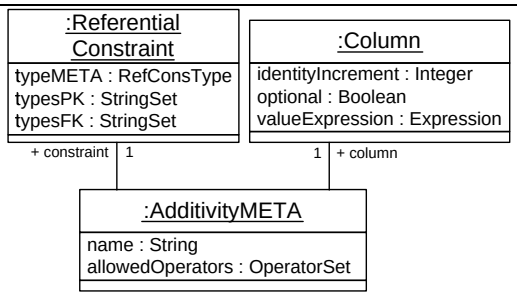
$$\begin{aligned} &\text{Sei } L \text{ ein } \textit{LCD of SQL}\text{–Schema.}\\ &L \text{ heißt } \textit{gültig} \stackrel{def}{\Longleftrightarrow}\\ &\forall o \in O : "o \text{ erfüllt die Bedingungen des Metaklassendiagramms}\\ &\qquad\qquad \text{plus Nebenbedingungen".}\\ &\mathcal{L}_{Valid} \text{ sei die Menge aller gültigen } \textit{LCD of SQL}\text{–Schemata.} \end{aligned} \tag{8.6}$$

Der Zugriff auf Attribute und Referenzen von Objekten erfolgt wieder in der Punktnotation, z. B. „table.name“ zum Zugriff auf den Tabellennamen oder „column.type“ zum Zugriff auf den Datentyp

einer Spalte.
Wie im vorangegangenen Kapitel für die *REMUS*–Objekte sei auch für die *LCD of SQL*–Objekte eine Tupelschreibweise definiert, was in Tabelle 8.7 geschieht.

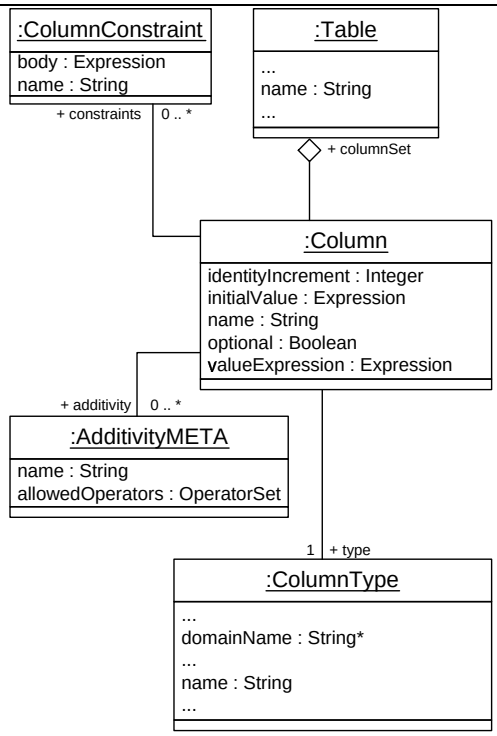
## Tupelschreibweise *LCD of SQL*–Objekte

**AdditivityMETA**



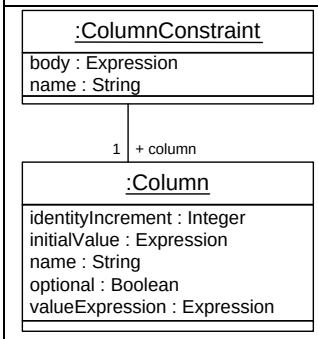

$$(a.name, a.allowedOperators, a.column, a.constraint, \text{AdditivityMETA})$$

**Column**




$$(c.name, c.initialValue, c.valueExpression, c.identityIncrement, c.optional, c.columnSet, c.keys, c.type.domainName, c.constraints, c.additivity, \text{Column})$$

**ColumnConstraint**

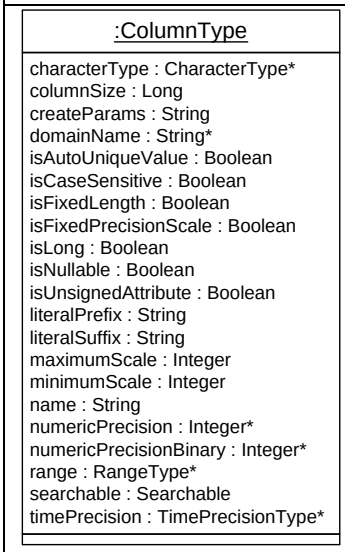



$$(c.name, c.body, c.column, \text{ColumnConstraint})$$

Fortsetzung auf der folgenden Seite

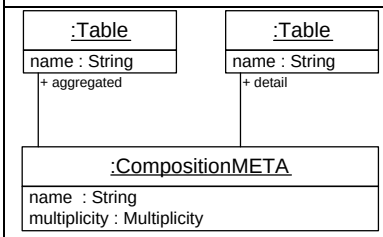
Fortsetzung von der letzten Seite

**ColumnType**



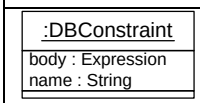

$(c.name, c.domainName,$
$c.characterType, c.columnSize,$
$c.createParams, c.isAutoUniqueValue,$
$c.isCaseSensitive, c.isFixedLength,$
$c.isFixedPrecisionScale, c.isLong,$
$c.isNullable, c.isUnsignedAttribute,$
$c.literalPrefix, c.literalSuffix,$
$c.maximumScale, c.minimumScale,$
$c.numericPrecision,$
$c.numericPrecisionBinary,$
$c.range, c.searchable,$
$c.timePrecision,$
ColumnType)

**CompositionMETA**



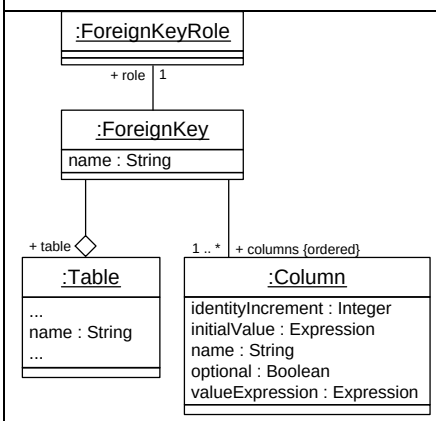

$(c.name, c.multiplicity,$
$c.aggregated, c.detail,$
CompositionMETA)

**DBConstraint**




$(d.name, d.body,$
DBConstraint)

**ForeignKey**

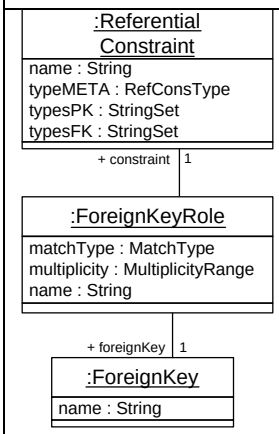



$(f.name, f.table,$
$f.columns, f.role,$
ForeignKey)

Fortsetzung auf der folgenden Seite

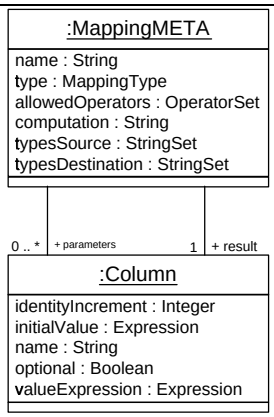
Fortsetzung von der letzten Seite

**ForeignKeyRole**



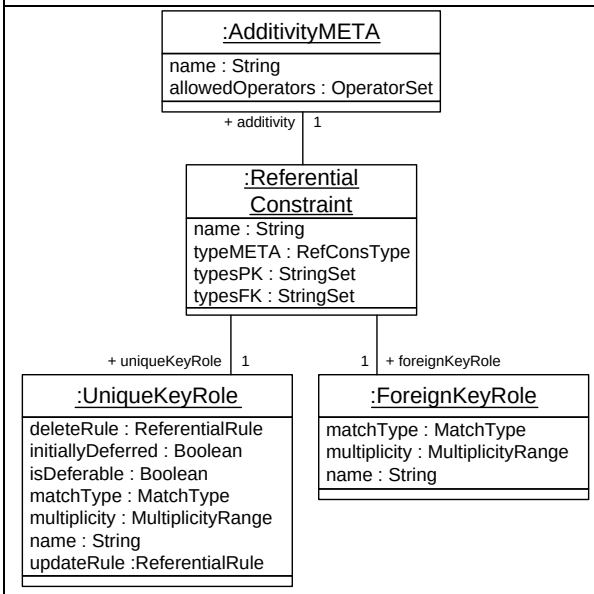

$(f.name, f.multipicity,$
$f.matchType, f.foreignKey,$
$f.constraint,$
ForeignKeyRole)

**MappingMETA**




$(m.name, m.type,$
$m.allowedOperators, m.computation,$
$m.typesSource, m.TypesDestination,$
$m.parameters, m.result,$
MappingMETA)

**ReferentialConstraint**

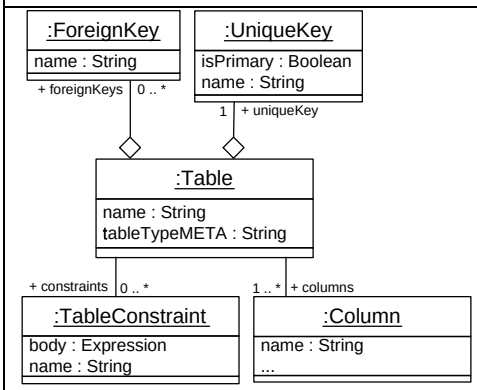



$(r.name, r.typeMETA,$
$r.typesPK, r.typesFK,$
$r.uniqueKeyRole, r.foreignKeyRole,$
$r.additivity,$
ReferentialConstraint)

Fortsetzung auf der folgenden Seite

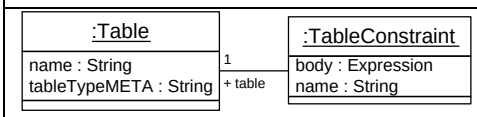
Fortsetzung von der letzten Seite

**Table**



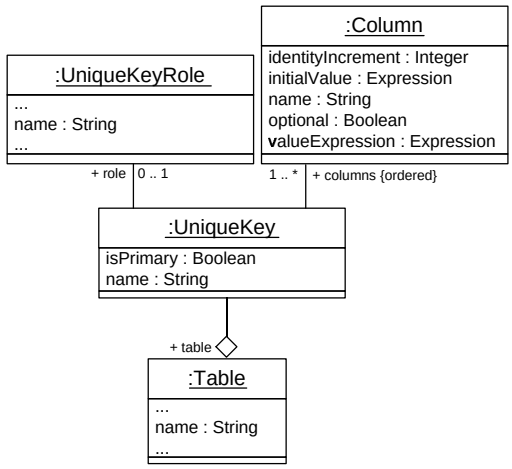

$(t.name, t.tableTypeMETA,$
$t.uniqueKeys, t.foreignKeys,$
$t.columns, t.constraints,$ Table)

**TableConstraint**



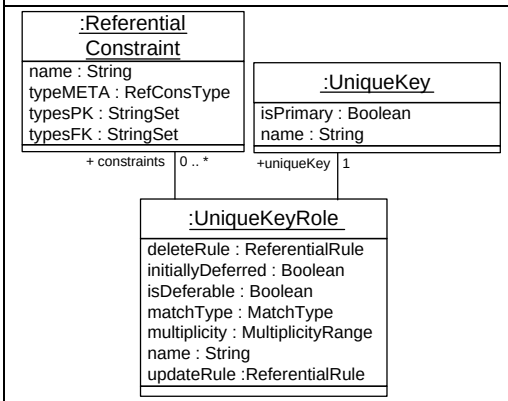

$(t.name, t.body,$
$t.table,$ TableConstraint)

**UniqueKey**




$(u.name, u.isPrimary,$
$u.table, u.columns,$
$u.role,$ UniqueKey)

**UniqueKeyRole**




$(u.name, u.multipicity,$
$u.matchType, u.deleteRule,$
$u.updateRule, u.deferable,$
$u.initiallyDeferred, u.uniqueKey,$
$u.constraints,$ UniqueKeyRole)

Tabelle 8.7: Tupelschreibweise der *LCD of SQL*–Objekte

Im Unterschied zur Transformation in Kapitel 7, bei der in jedem Schritt neue *REMUS*–Objekte erzeugt werden, sind aufgrund der wechselseitigen Referenzen im *LCD of SQL*–Metaschema Manipulationsoperatoren notwendig, um ein korrektes Schema zu erhalten. Aus diesem Grunde sind *Set*– und *Add*–Methoden zur Datenmanipulation definiert. Eine *Set*–Methode ändert eine Komponente eines Tupels, eine *Add*–Methode fügt bei mengenwertigen Einträgen von Referenzen entsprechend einen Eintrag hinzu. Der Name der Methode setzt sich aus „Set“ bzw. „Add“, dem Namen der zu manipulierenden Metaklasse sowie der zu manipulierenden Komponente zusammen. Beispielsweise ändert die Methode $SetTableName(\text{„A“}, \text{„B“})$ den Tabellennamen von „A“ in „B“ und $AddTableConstraint(\text{”A”}, \text{”C”})$ erweitert die Menge der von Tabelle „A“ referenzierten *Constraints* um den Eintrag „C“. Formal:

$$\begin{aligned} &SetTableName : \mathcal{L}_{Table} \times String \rightarrow \mathcal{L}_{Table} \\ &SetTableName(t, s) \stackrel{def}{=\!=} (s, t.tableTypeMETA, \\ &\qquad\qquad t.uniqueKeys, t.constraints, \mathsf{Table}). \end{aligned} \tag{8.7}$$

Hinzufügen bei mengenwertigen Komponenten:

$$\begin{aligned} &AddTableConstraint : \mathcal{L}_{Table} \times \mathcal{L}_{TableConstraint} \rightarrow \mathcal{L}_{Table} \\ &AddTableConstraint(t, c) \stackrel{def}{=\!=} (t.name, t.tableTypeMETA, \\ &\qquad\qquad t.uniqueKeys, t.constraints \cup \{c\}, \mathsf{Table}). \end{aligned} \tag{8.8}$$

### 8.3.3 Schritt 1: Datentypen anlegen

Die Datentypen aus dem REMUS–Schema werden mit Hilfe der deterministischen Funktion $f_{det_{DataType}}$ auf einen der in Abschnitt 8.2.7 vorgestellten Datentypen abgebildet, der ursprüngliche Name in das Attribut *domainName* übernommen. Das Beispiel in Abbildung 8.10 zeigt den Datentyp „Währung“, der auf den Typ „Decimal“ mit der Einschränkung auf positive Werte abgebildet wird.

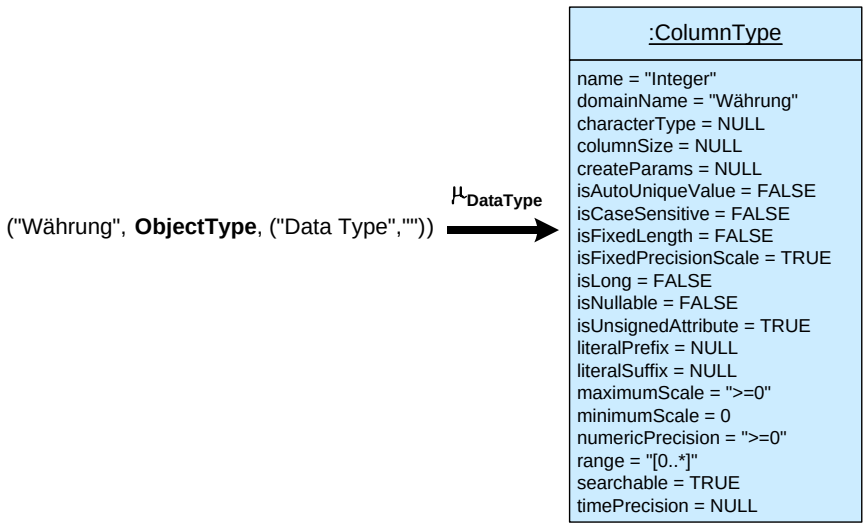



Abbildung 8.10: Abbildung der Datentypen von *REMUS* nach *LCD of SQL*

Die zugehörige Abbildungsvorschrift für einen Datentyp ist in (8.9) definiert, die Menge $UML_{Types}$ bezeichne alle nach Abbildung 8.8 bzw. Tabelle 8.4 möglichen Datentypen.

$$\begin{aligned} &\mu_{DataType_{Create}} : META_{\downarrow(*,\text{ObjectType},(\text{"}DataType\text{"},\text{""}))} \rightarrow UML_{Types} \\ &\mu_{DataType_{Create}}(m) \stackrel{def}{=\!=} f_{det_{DataType}}(m). \end{aligned} \tag{8.9}$$

Die Abbildung aller Datentypen eines Schemas geschieht mittels:

$$\begin{aligned} &\mathcal{M}_{DataType} : \mathcal{R} \times \mathcal{L} \rightarrow \mathcal{L} \\ &\mathcal{M}_{DataType}(R, L) \stackrel{def}{=\!=} \{\mu_{DataType_{Create}}(m) \mid m \in M_{\downarrow(*,\text{ObjectType},(\text{"}DataType\text{"},\text{""}))}\}. \end{aligned} \tag{8.10}$$

### 8.3.4 Schritt 2: Tabellen anlegen

Zu jeder Relation im REMUS–Schema wird ein *Table*–Objekt im *LCD of SQL*–Schema angelegt. Wie exemplarisch für die Relationen „Woche“ und „Verkauf“ in Abbildung 8.11 dargestellt, wird der Tabellenname durch die deterministische Abbildung $f_{det_{TableName}}$ entschieden, die Metainformation des Tabellentyps in das Attribut *tableTypeMETA* übernommen. Durch $f_{det_{TableName}}$ ist es möglich, an dieser Stelle einen Namen für das physische Schema zu vergeben, der es z. B. ermöglicht, organisationsinterne oder projektspezifische Namenskonventionen einzuhalten. Ebenso können an dieser Stelle im physischen Modell nicht erlaubte Zeichen eliminiert werden.

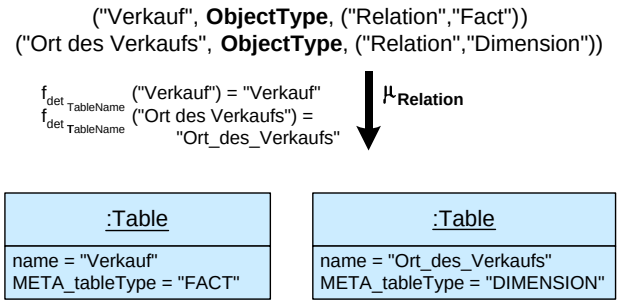



Abbildung 8.11: Abbildung der Relationen von *REMUS* nach *LCD of SQL*

Abbildungsvorschrift für eine Relation:

$$\begin{aligned} &\mu_{Relation_{Create}} : META_{\downarrow(*,\text{ObjectType},(\text{"}Relation\text{"},*))} \rightarrow \mathcal{L}_{Table} \\ &\mu_{Relation_{Create}}(m) \stackrel{def}{=\!=} (f_{det_{TableName}}(m), m.description, \text{Table}). \end{aligned} \tag{8.11}$$

Abbildung aller Relationen:

$$\begin{aligned} &\mathcal{M}_{Relation} : \mathcal{R} \times \mathcal{L} \rightarrow \mathcal{L} \\ &\mathcal{M}_{Relation}(R, L) \stackrel{def}{=\!=} L \cup \{\mu_{Relation_{Create}}(m) \mid m \in M_{\downarrow(*,\text{ObjectType},(\text{"}Relation\text{"},*))}\}. \end{aligned} \tag{8.12}$$

### 8.3.5 Schritt 3: Attribute anlegen

Zu jedem Attribut im REMUS–Schema wird ein *Column*–Objekt im *LCD of SQL*–Schema angelegt, wie in Abbildung 8.12 dargestellt. Der Name wird durch die deterministische Abbildung $f_{det_{AttributeName}}$ entschieden. Somit können, wie schon bei den Tabellen in Schritt 2, wieder Namenskonventionen berücksichtigt oder Sonderzeichen entfernt werden. Ebenso besteht an dieser Stelle die projektindividuelle Möglichkeit, die Präfixschreibweise der Attributnamen beizubehalten. Der Wert des Attributs *identityIncrement* wird für Attribute mit der Endung „ID“ auf „1“ gesetzt, ansonsten auf „0“. Die beiden Attribute *valueExpression* und *initialValue* werden zunächst leer initialisiert und eventuell beim Abarbeiten der Metadaten in Schritt 6 (siehe Seite 190) belegt. Ebenso wird das Attribut *optional* leer initialisiert und in Schritt 8 (siehe Seite 194) bei Abarbeitung der Metadaten manipuliert. Die *type*–Referenz wird auf den für das Attribut referenzierten Datentypen gesetzt, die *table*–Referenz auf die referenzierte Tabelle. Schließlich wird das bereits im letzten Schritt angelegte *Table*–Objekt um Referenzen auf die Spalten aktualisiert.

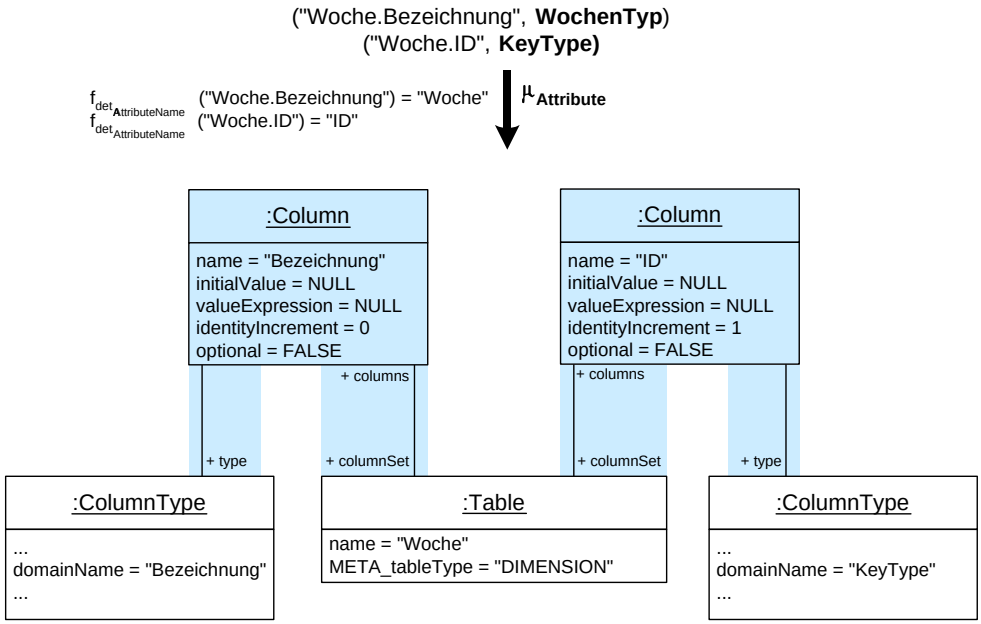



Abbildung 8.12: Abbildung der Attribute von *REMUS* nach *LCD of SQL*

Für die Abbildungsvorschrift seien zunächst zwei Hilfsfunktionen definiert:

- $IncrementValue$ ermittelt in Abhängigkeit von der Endung des Attributnamens den Wert „0“ oder „1“, d. h.

$$IncrementValue : String \rightarrow \{0, 1\}$$
$$IncrementValue(s) \stackrel{def}{=\!=} \begin{cases} 1 & \text{falls } right(s, 2) = "ID" \\ 0 & \text{sonst.} \end{cases} \tag{8.13}$$

- $Prefix$ ermittelt das Präfix des übergebenen Argumentes bis zum ersten Punkt, d. h.

$$Prefix : String \rightarrow String$$
$$String(s) \stackrel{def}{=\!=} \text{"Teilstring von s bis zum ersten Punkt"}.^{4} \tag{8.14}$$

Damit lässt sich das Übertragen eines Attributes formulieren:

$$\begin{aligned}
&\mu_{Attribute_{Create}} : \mathcal{R}_{Attributes} \rightarrow \mathcal{L}_{Column} \\
&\mu_{Attribute_{Create}}(a) \stackrel{def}{=\!=} (f_{det_{AttributeName}}(a.name), \text{NULL}, \text{NULL}, \\
&\qquad\qquad IncrementValue(a.name), \text{NULL}, \\
&\qquad\qquad f_{det_{TableName}}(Prefix(a.name)), NULL, \\
&\qquad\qquad a.type, NULL, NULL, \text{Column}).
\end{aligned} \tag{8.15}$$

Die im zweiten Schritt angelegten Tabellen werden entsprechend aktualisiert:

$$\begin{aligned}
&\mu_{Attribute_{Update\ Table}} : \mathcal{R}_{Attributes} \times \mathcal{L}_{Table} \rightarrow \mathcal{L}_{Table} \\
&\mu_{Attribute_{Update\ Table}}(a, t) \stackrel{def}{=\!=} \\
&\qquad \begin{cases} AddTableColumn(t, f_{det_{AttributeName}}(a.name)) \\ \qquad\qquad\qquad \text{falls } t.name = f_{det_{TableName}}(Prefix(a.name)) \\ t \qquad\qquad\qquad \text{sonst.} \end{cases}
\end{aligned} \tag{8.16}$$

Erweitert auf Mengen von Tabellen ergibt sich:

$$\begin{aligned}
&\mu_{Attribute_{UpdateSet\ Table}} : \mathcal{R}_{Attributes} \times Pot(\mathcal{L}_{Table}) \rightarrow Pot(\mathcal{L}_{Table}) \\
&\mu_{Attribute_{UpdateSet\ Table}}(a, T) \stackrel{def}{=\!=} \bigcup_{t \in T} \mu_{Attribute_{Update\ Table}}(a, t).
\end{aligned} \tag{8.17}$$

Die Abbildung aller Attribute geschieht mittels:

$$\begin{aligned}
&\mathcal{M}_{Attribute} : \mathcal{R} \times \mathcal{L} \rightarrow \mathcal{L} \\
&\mathcal{M}_{Attribute}(R, L) \stackrel{def}{=\!=} (L \setminus O_{Table}) \\
&\qquad \cup \{\mu_{Attribute_{Create}}(a) \mid a \in A\} \\
&\qquad \cup \{\mu_{Attribute_{UpdateSet\ Table}}(a, O_{Table}) \mid a \in A\}.
\end{aligned} \tag{8.18}$$

### 8.3.6 Schritt 4: Primärschlüssel anlegen

In diesem Schritt werden für die in Schritt 2 (siehe Seite 183) angelegten Tabellen die Primärschlüssel eingetragen. Bei den dimensionalen Tabellen sind dies die während der Transformation von *MML* nach *REMUS* angelegten Surrogate, die als *PrimaryKey*–Metadatum dokumentiert sind. Die Fakttabellen hingegen besitzen einen zusammengesetzten Primärschlüssel, der während der Transformation sukzessive durch das Eintragen der Primärschlüssel beteiligter Dimensionen entstand. Die Abbildungen 8.13 und 8.14 zeigen den Vorgang für eine dimensionale Tabelle und eine Fakttabelle. Der Unterschied besteht darin, dass bei der Fakttabelle aufgrund des zusammengesetzten Primärschlüssels das *UniqueKey*–Objekt auf mehrere Spalten verweist. Neben dem Anlegen eines *UniqueKey*–Objektes, dessen Attribut *isPrimary* den Schlüssel als Primärschlüssel definiert, wird ein *UniqueKeyRole*–Objekt angelegt, das als *matchType* eine vollständige Übereinstimmung der beteiligten Attribute festschreibt. Als Verhalten sind sowohl Lösch– wie Änderungsweitergabe sinnvoll, um Aktualisierungen auf den Dimensionen zuzulassen. Ebenso sollte die referentielle Konsistenz an dieser Stelle verzögerbar sein und auch anfangs diesen Status annehmen, um Ladewerkzeugen bei Änderungen möglichst flexibles Zugreifen zu ermöglichen. Weiterhin müssen die beteiligte Tabelle und die beteiligten Spalten um die Primärschlüsseleinträge erweitert werden.

[5] Auf eine formalere Definition sei an dieser Stelle verzichtet.

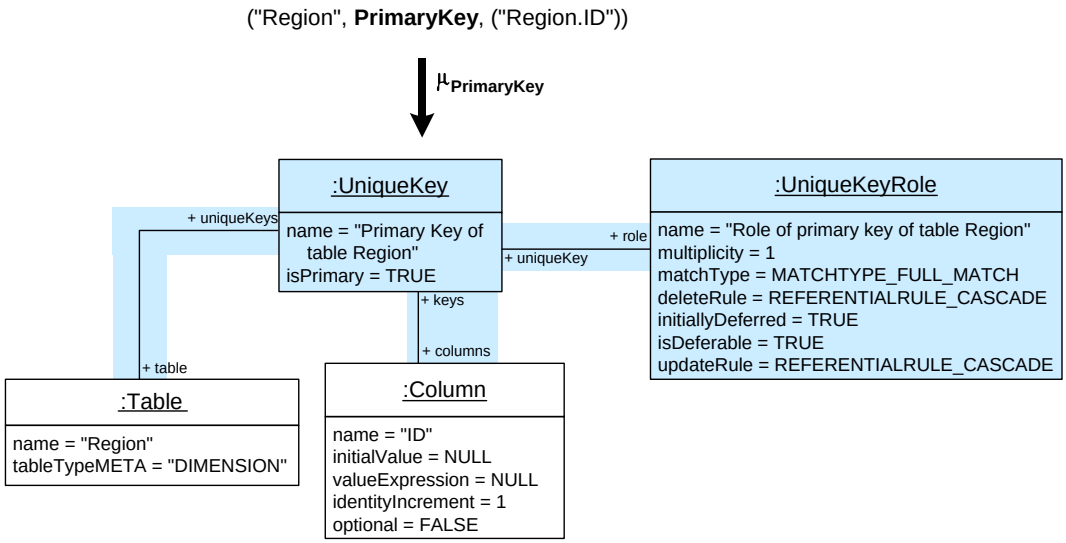



Abbildung 8.13: Abbildung des Primärschlüssels einer dimensionalen Tabelle von *REMUS* nach *LCD of SQL*

Zur Formulierung der Abbildungsvorschrift benötigt man diejenige Zerlegung der *PrimaryKey*–Metadaten, die die einzelnen Primärschlüssel einer Relation zusammenfasst. Diese als $M_{PrimaryKey_{Grouped}}$ bezeichnete Zusammenfassung ist in (8.19) definiert.

$$
\begin{aligned}
&M_{PrimaryKey_{Grouped}} = \{M_1, ..., M_n\} \text{ ist diejenige Zerlegung von } M_{PrimaryKey} \text{ mit} \\
&\quad (i) \forall m', m'' \in M_i \text{ mit } i \in \{1, ..., n\} : m'.a = m''.a \\
&\wedge\ (ii) \forall m' \in M_i \forall m'' \in M_j \text{ mit } i, j \in \{1, ..., n\} \text{ und } i \neq j : m'.a \neq m''.a \\
&\wedge\ (iii) \forall m \in M_{PrimaryKey} : m \in M_i \text{ mit } i \in \{1, ..., n\}.
\end{aligned}
\tag{8.19}
$$

Weil die Anordnung der Attribute innerhalb eines Schlüssels geordnet ist, wird die nachfolgend definierte Funktion eingeführt, die die Elemente einer endlichen Menge in eine Ordnung bringt.

$$
\begin{aligned}
&\text{Sei } S \text{ eine n–elementige Menge. } f_{det_{Order}} \text{ ist eine bijektive Abbildung mit} \\
&f_{det_{Order}} : S \rightarrow \{1, ..., n\} \\
&f_{det_{Order}}(s) \stackrel{def}{=} \text{”Position von s”}.
\end{aligned}
\tag{8.20}
$$

Die *UniqueKey*– und *UniqueKeyRole*–Objekte werden durch die beiden folgenden Abbildungsvorschriften angelegt:

$$
\begin{aligned}
&\mu_{PrimaryKey_{Create\ UniqueKey}} : META_{PrimaryKey_{Grouped}} \rightarrow \mathcal{L}_{UniqueKey} \\
&\mu_{PrimaryKey_{Create\ UniqueKey}}(M) \stackrel{def}{=} \\
&\qquad ((\psi(\text{"Primary key of table "}, f_{det_{TableName}}(M.relation.name)), \\
&\qquad\ \text{TRUE}, f_{det_{TableName}}(M.relation.name), \\
&\qquad\ f_{det_{Order}}(\bigcup_{m \in M} \{f_{det_{AttributeName}}(m.attribute.name)\}), \\
&\qquad\ \text{NULL}, \text{UniqueKey}).
\end{aligned}
\tag{8.21}
$$

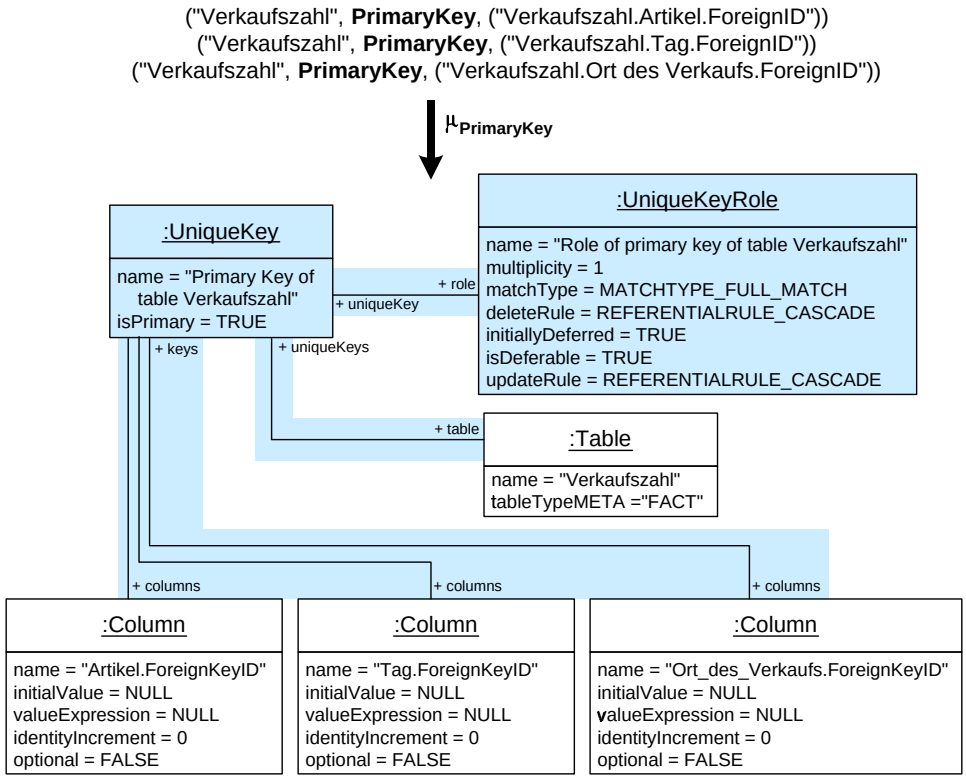



Abbildung 8.14: Abbildung des Primärschlüssels einer Fakttabelle von *REMUS* nach *LCD of SQL*

$$\mu_{PrimaryKey_{Create\ UniqueKeyRole}} : META_{PrimaryKey_{Grouped}} \rightarrow \mathcal{L}_{UniqueKeyRole}$$

$$\mu_{PrimaryKey_{Create\ UniqueKeyRole}}(M) \stackrel{def}{=}$$

$$\begin{aligned}(&\psi(\text{"Role of primary key of table "}, f_{det_{TableName}}(M.relation.name)),\\ &\text{"1"}, \text{FULL}, \text{REFERENTIALRULE\_CASCADE},\\ &\text{REFERENTIALRULE\_CASCADE}, \text{TRUE}, \text{TRUE},\\ &\psi(\text{"Primary key of table "}, f_{det_{TableName}}(M.relation.name)),\\ &NULL, \text{UniqueKeyRole}).\end{aligned} \tag{8.22}$$

Aktualisiert werden muss das in (8.21) erzeugte *UniqueKey*–Objekt um den Verweis auf die in (8.22) definierte Rolle sowie die am Primärschlüssel beteiligten Spalten und die Tabelle.

$$\mu_{PrimaryKey_{Update\ UniqueKey}} : META_{PrimaryKey_{Grouped}} \times \mathcal{L}_{UniqueKey} \rightarrow \mathcal{L}_{UniqueKey}$$

$$\mu_{PrimaryKey_{Update\ UniqueKey}}(M, u) \stackrel{def}{=}$$

$$\begin{cases} AddUniqueKeyRole(u, \psi(\text{"Role of primary key of table "}, & \\ \qquad f_{det_{TableName}}(M.relation.name))), & \\ \qquad \text{falls } u.name = \psi(\text{"Primary key of table "}, & \\ \qquad f_{det_{TableName}}(M.relation.name)) & \\ u \qquad \text{sonst.} & \end{cases} \tag{8.23}$$

$$\mu_{PrimaryKey_{Update\ Table}} : META_{PrimaryKey_{Grouped}} \times \mathcal{L}_{Table} \to \mathcal{L}_{Table}$$
$$\mu_{PrimaryKey_{Update\ Table}}(M,t) \stackrel{def}{=}$$
$$\begin{cases} AddTableUniqueKey(t, \psi(\text{"Primary key of table "}, & \\ \qquad f_{det_{TableName}}(M.relation.name))), & \\ \text{falls } t.name = f_{det_{TableName}}(M.relation.name) & \\ t \qquad \text{sonst.} & \end{cases} \tag{8.24}$$

$$\mu_{PrimaryKey_{Update\ Column}} : META_{PrimaryKey_{Grouped}} \times \mathcal{L}_{Column} \to \mathcal{L}_{Column}$$
$$\mu_{PrimaryKey_{Update\ Column}}(M,c) \stackrel{def}{=}$$
$$\begin{cases} AddColumnUniqueKey(c, \psi(\text{"Primary key of table "}, & \\ \qquad f_{det_{TableName}}(M.relation.name))), & \\ \text{falls } c.name = f_{det_{AttributeName}}(M.attribute.name)) & \\ c \qquad \text{sonst.} & \end{cases} \tag{8.25}$$

Die Erweiterungen dieser Aktualisierungsoperatoren auf Mengen wird in den folgenden drei Abbildungsvorschriften festgehalten.

$$\begin{aligned} &\mu_{PrimaryKey_{UpdateSet\ UniqueKey}} : META_{PrimaryKey_{Grouped}} \times Pot(\mathcal{L}_{UniqueKey}) \\ &\qquad \to Pot(\mathcal{L}_{UniqueKey}) \\ &\mu_{PrimaryKey_{UpdateSet\ UniqueKey}}(M,U) \stackrel{def}{=} \bigcup_{u \in U} \mu_{PrimaryKey_{Update\ UniqueKey}}(M,u). \end{aligned} \tag{8.26}$$

$$\begin{aligned} &\mu_{PrimaryKey_{UpdateSet\ Table}} : META_{PrimaryKey_{Grouped}} \times Pot(\mathcal{L}_{Table}) \to Pot(\mathcal{L}_{Table}) \\ &\mu_{PrimaryKey_{UpdateSet\ Table}}(M,T) \stackrel{def}{=} \bigcup_{t \in T} \mu_{PrimaryKey_{Update\ Table}}(M,t). \end{aligned} \tag{8.27}$$

$$\begin{aligned} &\mu_{PrimaryKey_{UpdateSet\ Column}} : META_{PrimaryKey_{Grouped}} \times Pot(\mathcal{L}_{Column}) \to Pot(\mathcal{L}_{Column}) \\ &\mu_{PrimaryKey_{UpdateSet\ Column}}(M,C) \stackrel{def}{=} \bigcup_{c \in C} \mu_{PrimaryKey_{Update\ Column}}(M,c). \end{aligned} \tag{8.28}$$

Damit ergibt sich für das Anlegen aller Primärschlüssel eines Schemas:

$$\begin{aligned} &\mathcal{M}_{PrimaryKey} : \mathcal{R} \times \mathcal{L} \times META_{PrimaryKey_{Grouped}} \to \mathcal{L} \\ &\mathcal{M}_{PrimaryKey}(R,L,M) \stackrel{def}{=} (L \\ &\quad \cup \{\mu_{PrimaryKey_{Create\ UniqueKey}}(M)\} \\ &\quad \cup \{\mu_{PrimaryKey_{Create\ UniqueKeyRole}}(M)\} \\ &\quad \setminus O_{UniqueKey} \setminus O_{Table} \setminus O_{Column}) \\ &\quad \cup \{\mu_{PrimaryKey_{UpdateSet\ UniqueKey}}(M, O_{UniqueKey})\} \\ &\quad \cup \{\mu_{PrimaryKey_{UpdateSet\ Table}}(M, O_{Table})\} \\ &\quad \cup \{\mu_{PrimaryKey_{UpdateSet\ Column}}(M, O_{Column})\}. \end{aligned} \tag{8.29}$$

### 8.3.7 Schritt 5: Konzeptionelle Schlüssel anlegen

Die Abbildung konzeptioneller Schlüssel erfolgt ähnlich zu der von Primärschlüsseln: Für zusammengehörige *ConceptualKey*–Metadaten, d. h. für eine Teilmenge, bei denen der Wert in der

ersten Komponente gleich ist, wird als eindeutiger Schlüssel ein *UniqueKey*–Objekt angelegt. Im Gegensatz zu einem Primärschlüsel in Schritt 4 wird jedoch das Attribut *isPrimary* auf „FALSE“ gesetzt. Auch wird für konzeptionelle Schlüssel kein *UniqueKeyRole*–Objekt angelegt, weil mit Hilfe konzeptioneller Schlüssel keine Referenzen gebildet werden. Die zugehörige Tabelle und Spalten sind bereits in den Schritten 2 und 3 angelegt worden, es müssen jetzt lediglich die Referenzen aktualisiert werden. Die folgenden Abbildungsvorschriften (8.30) bis (8.36) entsprechen bis auf diesen Unterschied (8.19) bis (8.29) aus Schritt 4. $M_{ConceptualKey_{Grouped}}$ sei eine Partitionierung der *ConceptualKey*–Metadaten, die die einzelnen Komponenten eines konzeptionellen Schlüssels zusammenfasst.

$$\begin{aligned} &M_{ConceptualKey_{Grouped}} = \{M_1, ..., M_n\} \text{ ist diejenige Zerlegung von } M_{ConceptualKey} \text{ mit} \\ &\quad (i) \quad \forall m', m'' \in M_i \text{ mit } i \in \{1, ..., n\} : m'.a = m''.a \\ &\wedge \quad (ii) \quad \forall m' \in M_i \forall m'' \in M_j \text{ mit } i, j \in \{1, ..., n\} \text{ und } i \neq j : m'.a \neq m''.a \\ &\wedge (iii) \quad \forall m \in M_{ConceptualKey} : m \in M_i \text{ mit } i \in \{1, ..., n\}. \end{aligned} \tag{8.30}$$

Die Definition der Abbildung $f_{det_{Order}}$ kann aus (8.20) übernommen werden. Die *UniqueKey*–Objekte werden durch die Abbildungsvorschrift (8.31) angelegt.

$$\begin{aligned} &\mu_{ConceptualKey_{Create\ UniqueKey}} : META_{ConceptualKey_{Grouped}} \rightarrow \mathcal{L}_{UniqueKey} \\ &\mu_{ConceptualKey_{Create\ UniqueKey}}(M) \stackrel{def}{=} \\ &\qquad (\psi(\text{"Conceptual key of table "}, f_{det_{TableName}}(M.relation.name)), \\ &\qquad\ \text{FALSE}, f_{det_{TableName}}(M.relation.name), \\ &\qquad\ f_{det_{Order}}(\bigcup_{m \in M} \{f_{det_{AttributeName}}(m.attribute.name)\}), \\ &\qquad\ \text{NULL}, \text{UniqueKey}). \end{aligned} \tag{8.31}$$

Aktualisiert werden müssen die am Primärschlüssel beteiligten Spalten und die Tabelle, was in den beiden folgenden Definitionen geschieht:

$$\begin{aligned} &\mu_{ConceptualKey_{Update\ Table}} : META_{ConceptualKey_{Grouped}} \times \mathcal{L}_{Table} \rightarrow \mathcal{L}_{Table} \\ &\mu_{ConceptualKey_{Update\ Table}}(M, t) \stackrel{def}{=} \\ &\begin{cases} AddTableUniqueKey(t, \psi(\text{"Conceptual key of table "}, \\ \qquad\qquad f_{det_{TableName}}(M.relation.name))), \\ \qquad \text{falls } t.name = f_{det_{TableName}}(M.relation.name) \\ t \qquad \text{sonst.} \end{cases} \end{aligned} \tag{8.32}$$

$$\begin{aligned} &\mu_{ConceptualKey_{Update\ Column}} : META_{ConceptualKey_{Grouped}} \times \mathcal{L}_{Column} \rightarrow \mathcal{L}_{Column} \\ &\mu_{ConceptualKey_{Update\ Column}}(M, c) \stackrel{def}{=} \\ &\begin{cases} AddColumnUniqueKey(c, \psi(\text{"Conceptual key of table "}, \\ \qquad\qquad f_{det_{TableName}}(M.relation.name))), \\ \qquad \text{falls } c.name = f_{det_{AttributeName}}(M.attribute.name) \\ c \qquad \text{sonst.} \end{cases} \end{aligned} \tag{8.33}$$

Die Erweiterung der Aktualisierungsfunktionen auf Mengen geschieht in den beiden folgenden Abbildungsvorschriften:

$$\begin{aligned} &\mu_{ConceptualKey_{UpdateSet\ Table}} : META_{ConceptualKey_{Grouped}} \times Pot(\mathcal{L}_{Table}) \rightarrow Pot(\mathcal{L}_{Table}) \\ &\mu_{ConceptualKey_{UpdateSet\ Table}}(M, T) \stackrel{def}{=} \bigcup_{t \in T} \mu_{ConceptualKey_{Update\ Table}}(M, t). \end{aligned} \tag{8.34}$$

$$\mu_{ConceptualKey_{UpdateSet\,Column}} : META_{ConceptualKey_{Grouped}} \times Pot(\mathcal{L}_{Column}) \rightarrow Pot(\mathcal{L}_{Column})$$
$$\mu_{ConceptualKey_{UpdateSet\,Column}}(M, C) \stackrel{def}{=\!=} \bigcup_{c \in C} \mu_{ConceptualKey_{Update\,Column}}(M, c). \tag{8.35}$$

Damit ergibt sich für das Anlegen aller konzeptionellen Schlüssel eines Schemas:

$$\begin{aligned}
&\mathcal{M}_{ConceptualKey} : \mathcal{R} \times \mathcal{L} \times META_{ConceptualKey_{Grouped}} \rightarrow \mathcal{L} \\
&\mathcal{M}_{ConceptualKey}(R, L, M) \stackrel{def}{=\!=} (L \\
&\quad \cup \{\mu_{ConceptualKey_{Create\,UniqueKey}}(M)\} \\
&\quad \setminus O_{UniqueKey} \setminus O_{Table} \setminus O_{Column}) \\
&\quad \cup \{\mu_{ConceptualKey_{UpdateSet\,Table}}(M, O_{Table})\} \\
&\quad \cup \{\mu_{ConceptualKey_{UpdateSet\,Column}}(M, O_{Column})\}.
\end{aligned} \tag{8.36}$$

### 8.3.8 Schritt 6: Berechnete Attribute markieren

In diesem Schritt wird die Teilmenge der *Computation*–Metadaten übertragen, die ein berechnetes Attribut ermitteln. Die restlichen *Computation*–Metadaten werden in den Schritten 12 (siehe Seite 202) und 13 (siehe Seite 204) als *Constraints* verarbeitet. Abbildung 8.15 zeigt exemplarisch das Vorgehen: Die deterministische Funktion $f_{det_{Computation}}$ ermittelt aus den Parametern und der Beschreibung aus der konzeptionellen Modellierung einen Ausdruck, der das abgeleitete Attribut berechnet.

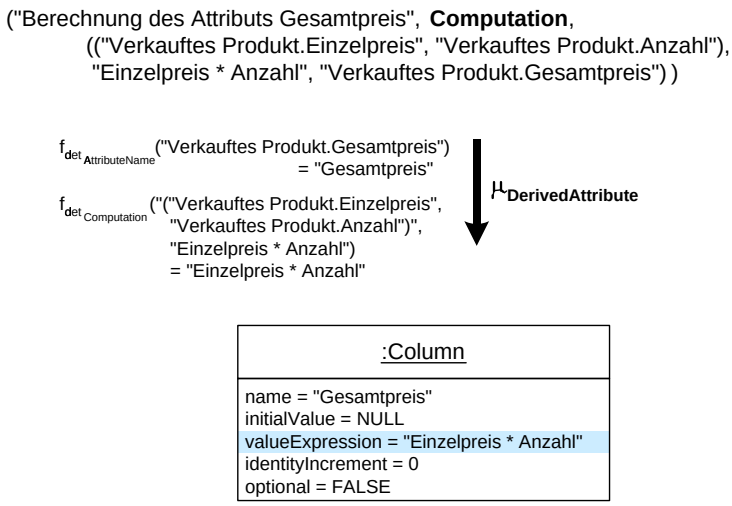



Abbildung 8.15: Abbildung abgeleiteter Attribute von *REMUS* nach *LCD of SQL*

Für die Abbildungsvorschrift sei zunächst in (8.37) die Funktion $DerivedAttribute$ definiert, die einer *Computation*–Instanz in Abhängigkeit davon, ob sie ein abgeleitetes Attribut berechnet oder nicht, „TRUE“ oder „FALSE“ zuweist.

$$DerivedAttribute : META_{Computation} \rightarrow \{\text{TRUE, FALSE}\}$$
$$DerivedAttribute(m) \stackrel{def}{=} \begin{cases} \text{TRUE} & \text{falls "Kein } \textit{SharedRollUp} \text{ und kein} \\ & \qquad \textit{DimensionalMapping} \text{ verweist auf m"} \\ \text{FALSE} & \text{sonst.} \end{cases} \tag{8.37}$$

Desweiteren muss eine deterministische Funktion $f_{det_{Computation}}$ existieren, die als Argumente die Parameterliste und Beschreibung der Berechnungsvorschrift erhält und als Ergebnis einen Ausdruck liefert, d. h. $f_{det_{Computation}} : String \times String \rightarrow ExpressionType$.
Die Aktualisierung der Spalte wird in (8.38) vorgenommen.

$$\mu_{DerivedAttribute_{Update\ Column}} : META_{Computation} \times \mathcal{L}_{Column} \rightarrow \mathcal{L}_{Column}$$
$$\mu_{DerivedAttribute_{Update\ Column}}(m, d) \stackrel{def}{=}$$
$$\begin{cases} SetColumnValueExpression(d, f_{det_{Computation}}(m.parameters.name, m.computation)) \\ \qquad \text{falls } d = m.attribute \wedge DerivedAttribute(m) = TRUE \\ d \qquad \text{sonst.} \end{cases} \tag{8.38}$$

Auf Mengen von Spalten erweitert ergibt sich:

$$\mu_{DerivedAttribute_{UpdateSet\ Column}} : META_{Computation} \times Pot(\mathcal{L}_{Column}) \rightarrow Pot(\mathcal{L}_{Column})$$
$$\mu_{DerivedAttribute_{UpdateSet\ Column}}(m, C) \stackrel{def}{=} \bigcup_{c \in C} \mu_{DerivedAttribute_{Update\ Column}}(m, c). \tag{8.39}$$

Schließlich werden alle abgeleiteten Attribute eines Schemas mittels eingetragen:

$$\mathcal{M}_{DerivedAttribute} : \mathcal{R} \times \mathcal{L} \rightarrow \mathcal{L}$$
$$\mathcal{M}_{DerivedAttribute}(R, L) \stackrel{def}{=} (L \setminus O_{Column}) \cup \{\mu_{DerivedAttribute_{UpdateSet\ Column}}(m, O_{Column})\}. \tag{8.40}$$

### 8.3.9 Schritt 7: *Identifier*, *IdentifierValue* und *Valid*

Die drei Metadatentypen *Identifier, IdentifierValue* und *Valid* fallen während der Transformation von *MML* nach *REMUS* bei der Auflösung von Vererbungen durch Nestung an. Daher wird ihre Abbildung auf ein *LCD of SQL*–Schema an dieser Stelle gemeinsam behandelt. Abbildung 8.16 zeigt die Transformation des einzigen *Identifier*–Objektes des Beispielschemas. Die Metadaten *Identifier* und *IdentifierValue* werden als *ColumnConstraint*–Objekt realisiert: Das als *Identifier* gekennzeichnete Attribut „Type“ darf nur die Werte „Filiale“ oder „Kaufhaus“ annehmen, was durch das *IdentifierValue*–Metadatum spezifiziert wird.

Für die Abbildungsvorschrift sei zunächst die Hilfsabbildung (8.41) definiert, die aus den *Identifier*– und *IdentifierValue*–Metadaten den Ausdruck für das Constraint ermittelt.

$$f_{det_{IdentifierRule}} : META_{IdentifierValue} \rightarrow ExpressionType$$
$$f_{det_{IdentifierRule}}(m) \stackrel{def}{=} \text{"Bilde Ausdruck"}. \tag{8.41}$$

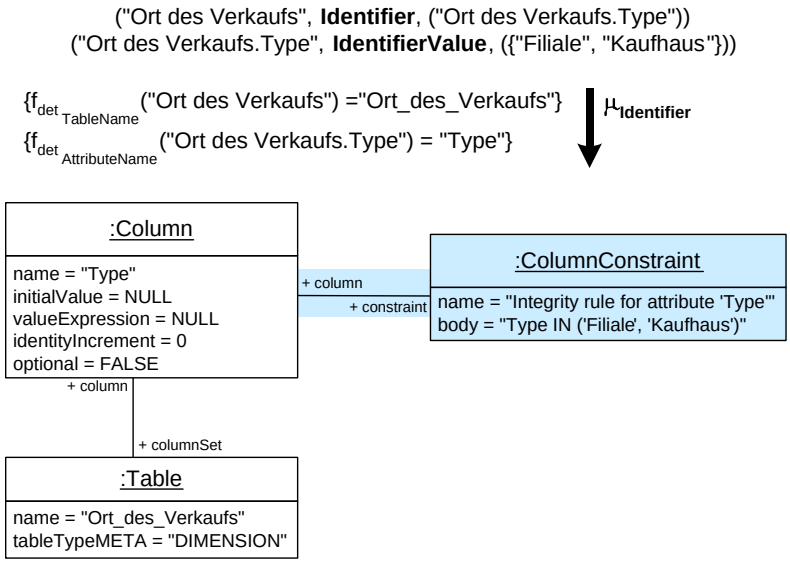



Abbildung 8.16: Abbildung von *Identifier* und *IdentifierValue* von *REMUS* nach *LCD of SQL*

Unter Verwendung dieser Hilfsfunktion lassen sich Funktionen definieren, die ein *ColumnConstraint*–Objekt anlegen (8.42) bzw. ein bestehendes *Column*–Objekt aktualisieren (8.43).

$$\begin{aligned}
&\mu_{Identifier_{Create}} : META_{Identifier} \rightarrow \mathcal{L}_{ColumnConstraint} \\
&\mu_{Identifier_{Create}}(m) \stackrel{def}{=} \\
&\qquad (\psi(\text{"Integrity rule for attribute "}, f_{det_{AttributeName}}(m.attribute.name)), \\
&\qquad f_{det_{Identifier\,Rule}}(m.values), \\
&\qquad f_{det_{AttributeName}}(m.attribute.name), \\
&\qquad \text{ColumnConstraint}).
\end{aligned} \tag{8.42}$$

$$\begin{aligned}
&\mu_{Identifier_{Update}} : META_{Identifier} \times \mathcal{L}_{Column} \rightarrow \mathcal{L}_{Column} \\
&\mu_{Identifier_{Update}}(m, c) \stackrel{def}{=} \\
&\begin{cases}
AddColumnConstraint(c, \psi(\text{"Integrity rule for attribute "}, \\
\qquad\qquad f_{det_{AttributeName}}(m.attribute.name))) \\
\qquad\qquad \text{falls } c.name = f_{det_{AttributeName}}(m.attribute.name) \\
c \qquad\qquad \text{sonst}
\end{cases}
\end{aligned} \tag{8.43}$$

Um eine Menge von *Column*–Objekten zu ergänzen, wird folgende Erweiterung definiert:

$$\begin{aligned}
&\mu_{Identifier_{UpdateSet}} : META_{Identifier} \times Pot(\mathcal{L}_{Column}) \rightarrow Pot(\mathcal{L}_{Column}) \\
&\mu_{Identifier_{UpdateSet}}(m, C) \stackrel{def}{=} \bigcup_{c \in C} \mu_{Identifier_{Update}}(m, c).
\end{aligned} \tag{8.44}$$

Schließlich wird die Abbildung aller *Identifier*–Metadaten eines Schemas realisiert:

$$\begin{aligned}
&\mathcal{M}_{Identifier} : \mathcal{R} \times \mathcal{L} \rightarrow \mathcal{L} \\
&\mathcal{M}_{Identifier}(R, L) \stackrel{def}{=} (L \setminus O_{Column}) \\
&\qquad \cup \{\mu_{Identifier_{Create}}(m) \mid m \in M_{Identifier}\} \\
&\qquad \cup \{\mu_{Identifier_{UpdateSet}}(m, O_{Column}) \mid m \in M_{Identifier}\}.
\end{aligned} \tag{8.45}$$

Durch Generalisierungsauflösung per Nestung entstehen Attribute, die in Abhängigkeit vom Objekttyp keine Werte haben dürfen. So dürfen bei Objekttypen aus einer Oberklasse, die aus Unterklassen hinzugekommenen Attribute keine Werte enthalten. Dieses wird durch das *Valid*–Metadatum ausgedrückt, das nun in ein *TableConstraint*–Objekt verwandelt wird, wie in Abbildung 8.18 dargestellt. Das Attribut „Fläche“ darf nur dann Werte enthalten, wenn das diskriminierende Attribut „Type“ den Wert „Kaufhaus“ enthält.

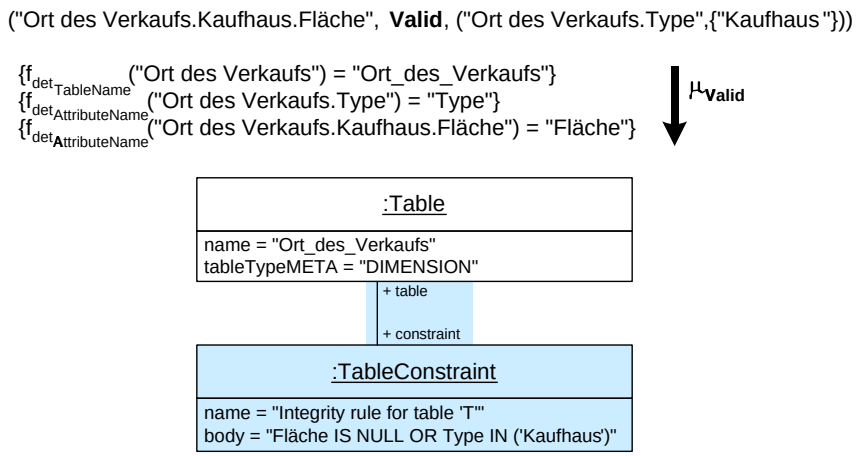



Abbildung 8.17: Abbildung von *Valid* von *REMUS* nach *LCD of SQL*

Ein *Valid*–Metadatum wird mittels folgender Funktion in den Ausdruck für des *TableConstraint* verwandelt.

$$\begin{aligned} &f_{det_{ValidRule}} : META_{Valid} \rightarrow ExpressionType \\ &f_{det_{ValidRule}}(M) \stackrel{def}{=} \text{"Bilde Ausdruck"}. \end{aligned} \tag{8.46}$$

Aufbauend auf Funktion (8.46) lassen sich die Funktionen definieren, die ein *TableConstraint*–Objekt anlegen (8.47) bzw. eine Referenz auf dieses Constraint in eine bestehende Tabelle eintragen (8.48).

$$\begin{aligned} &\mu_{Valid_{Create}} : META_{Valid} \rightarrow \mathcal{L}_{TableConstraint} \\ &\mu_{Valid_{Create}}(m) \stackrel{def}{=} \\ &\quad (\psi(\text{"Integrity rule for table "}, f_{det_{TableName}}(m.attribute.table.name)), \\ &\quad f_{detValidRule}(m), f_{det_{AttributeName}}(m.attribute.name), \\ &\quad \text{TableConstraint}). \end{aligned} \tag{8.47}$$

$$\begin{aligned} &\mu_{Valid_{Update}} : META_{Valid} \times \mathcal{L}_{Table} \rightarrow \mathcal{L}_{Table} \\ &\mu_{Valid_{Update}}(m,t) \stackrel{def}{=} \\ &\begin{cases} AddTableConstraint(t, \psi(\text{"Integrity rule for table "}, \\ \qquad\qquad f_{det_{AttributeName}}(m.a).columnSet.name)) \\ \qquad \text{falls } t.name = f_{det_{AttributeName}}(m.a).columnSet.name \\ t \qquad \text{sonst.} \end{cases} \end{aligned} \tag{8.48}$$

Um eine Menge von *Table*–Objekten zu ergänzen, sei definiert:

$$\begin{aligned} &\mu_{Valid_{UpdateSet}} : META_{Valid} \times Pot(\mathcal{L}_{Table}) \rightarrow Pot(\mathcal{L}_{Table}) \\ &\mu_{Valid_{UpdateSet}}(m,T) \stackrel{def}{=} \bigcup_{t \in T} \mu_{Valid_{Update}}(m,t). \end{aligned} \tag{8.49}$$

Schließlich wird die Abbildung aller *Valid*–Metadaten durchgeführt:

$$\begin{aligned}
&\mathcal{M}_{Valid} : \mathcal{R} \times \mathcal{L} \to \mathcal{L} \\
&\mathcal{M}_{Valid}(R, L, M) \stackrel{def}{=\!=} (L \setminus O_{Table}) \\
&\qquad \cup \{\mu_{Valid_{Create}}(m) \mid m \in M_{Valid}\} \\
&\qquad \cup \{\mu_{Valid_{UpdateSet}}(m, O_{Table}) \mid m \in M_{Valid}\}.
\end{aligned} \tag{8.50}$$

### 8.3.10 Schritt 8: Optionale Attribute markieren

Die durch das *Optional*–Metadatum markierte Optionalitätseigenschaft eines Attributes wird durch Setzen des *optional*–Attributes der Klasse *Column* festgehalten. Abbildung 8.18 zeigt dies für das Attribut „Bezeichnung“ der Tabelle „Artikel“.

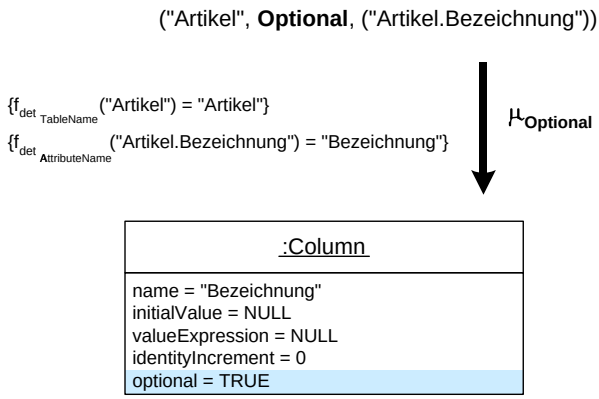



Abbildung 8.18: Abbildung von *Optional* von *REMUS* nach *LCD of SQL*

Eine einzelne Spalte wird mittels Abbildung (8.51) aktualisiert.

$$\begin{aligned}
&\mu_{Optional_{Update}} : META_{Optional} \times \mathcal{L}_{Column} \to \mathcal{L}_{Column} \\
&\mu_{Optional_{Update}}(m, c) \stackrel{def}{=\!=} \\
&\begin{cases}
SetColumnOptional(f_{det_{AttributeName}}(m.attribute.name), \text{TRUE}) \\
\qquad\qquad \text{falls } c.name = f_{det_{AttributeName}}(m.attribute.name) \\
c \qquad\qquad \text{sonst.}
\end{cases}
\end{aligned} \tag{8.51}$$

Eine Menge von *Column*–Objekten wird ergänzt durch:

$$\begin{aligned}
&\mu_{Optional_{UpdateSet}} : META_{Optional} \times Pot(\mathcal{L}_{Column}) \to Pot(\mathcal{L}_{Column}) \\
&\mu_{Optional_{UpdateSet}}(M, C) \stackrel{def}{=\!=} \bigcup_{c \in C} \mu_{Optional_{Update}}(c).
\end{aligned} \tag{8.52}$$

Schließlich erfolgt die Abbildung aller *Optional*–Metadaten eines Schemas:

$$\begin{aligned}
&\mathcal{M}_{Optional} : \mathcal{R} \times \mathcal{L} \to \mathcal{L} \\
&\mathcal{M}_{Optional}(R, L) \stackrel{def}{=\!=} (L \setminus O_{Column}) \\
&\qquad \cup \{\mu_{Optional_{UpdateSet}}(m, O_{Column}) \mid m \in M_{Optional}\}.
\end{aligned} \tag{8.53}$$

### 8.3.11 Schritt 9: Multiplizitäten festlegen

Ist die im Metadatum spezifizierte Multiplizität „0..*“ oder „1..*“, so braucht dieses Metadatum bei der Abbildung ins *LCD of SQL*–Schema nicht berücksichtigt zu werden.
Ansonsten wird für die entsprechende Tabelle ein *TableConstraint*–Objekt angelegt, wobei wie in Abbildung 8.19 dargestellt, die deterministische Funktion $f_{det_{MultiplicityRule}}$ über die konkrete Formulierung dieses Constraint entscheidet.
Der Ausdruck dieser Konsistenzregel kann wie im Beispiel eine SQL–Anweisung sein, die mittels Gruppierung die Anzahl der vorhandenen Datensätze des betroffenen Attributs bzw. der betroffenen Attributgruppe ermittelt und überprüft, ob sie mit der Multiplizitätsangabe im Metadatum übereinstimmt.

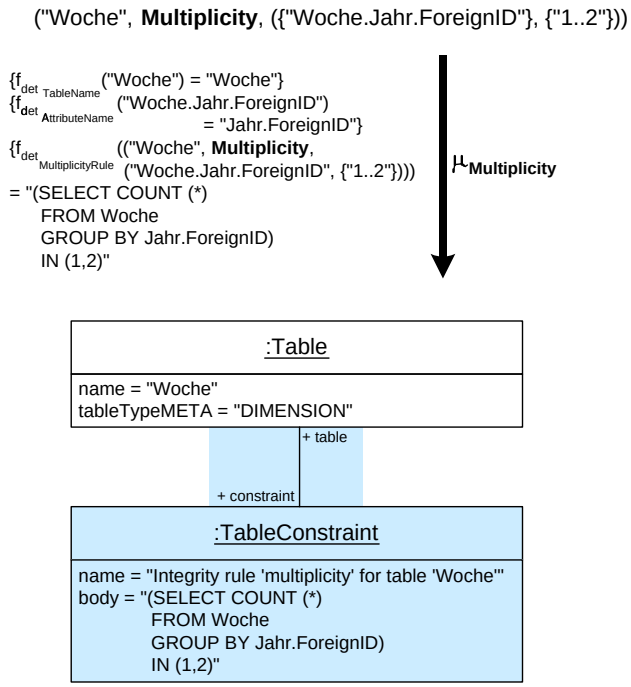



Abbildung 8.19: Abbildung von *Multiplicity* von *REMUS* nach *LCD of SQL*

Für die Abbildungsvorschrift sei zunächst in die Hilfsabbildung $f_{det_{MultiplicityRule}}$ definiert:

$$\begin{aligned} &MultiplicityRule : META_{\downarrow(*,\text{Multiplicity},(*))} \rightarrow ExpressionType \\ &MultiplicityRule(m) \stackrel{def}{=} \text{"Verdichtungsanweisung"}. \end{aligned} \tag{8.54}$$

Unter Ausnutzung dieser Hilfsabbildung lassen sich die Funktionen definieren, die ein *TableConstraint*–Objekt anlegen (8.55) bzw. in einer bestehenden Tabelle die Referenz auf dieses Objekt aktualisieren (8.56).

$$\begin{aligned} &\mu_{Multiplicity_{Create}} : META_{Multiplicity} \rightarrow \mathcal{L}_{TableConstraint} \\ &\mu_{Multiplicity_{Create}}(m) \stackrel{def}{=} (\psi(\text{"Integrity rule 'multiplicity' for table "}, \\ &\qquad f_{det_{TableName}}(m.relation.name)), \\ &\qquad f_{det_{MultiplicityRule}}(m), f_{det_{TableName}}(m.relation.name), \\ &\qquad \text{TableConstraint}). \end{aligned} \tag{8.55}$$

$$\mu_{Multiplicity_{Update}} : META_{Multiplicity} \times \mathcal{L}_{Table} \to \mathcal{L}_{Table}$$

$$\mu_{Multiplicity_{Update}}(m,t) \stackrel{def}{=}$$

$$\begin{cases} AddTableConstraint(t, \psi(\text{"Integrity rule 'multiplicity' for table "}, \\ \qquad\qquad f_{det_{TableName}}(m.relation.name))) \\ \qquad\qquad \text{falls } t.name = f_{det_{TableName}}(m.relation.name) \\ t \qquad\qquad \text{sonst.} \end{cases} \tag{8.56}$$

Eine Menge von *Table*–Objekten wird ergänzt durch:

$$\begin{aligned} &\mu_{Multiplicity_{UpdateSet}} : META_{Multiplicity} \times Pot(\mathcal{L}_{Table}) \to Pot(\mathcal{L}_{Table}) \\ &\mu_{Multiplicity_{UpdateSet}}(m,T) \stackrel{def}{=} \bigcup_{t \in T} \mu_{Multiplicity_{Update}}(m,t). \end{aligned} \tag{8.57}$$

Schließlich wird die Abbildung aller *Multiplicity*–Metadaten realisiert:

$$\begin{aligned} &\mathcal{M}_{Multiplicity} : \mathcal{R} \times \mathcal{L} \to \mathcal{L} \\ &\mathcal{M}_{Multiplicity}(R,L) \stackrel{def}{=} (L \setminus O_{Table}) \\ &\qquad \cup \{\mu_{Multiplicity_{Create}}(m) \mid m \in M_{\downarrow(*,\mathsf{Multiplicity},(*))}\} \\ &\qquad \cup \{\mu_{Multiplicity_{UpdateSet}}(m, O_{Table}) \mid m \in M_{\downarrow(*,\mathsf{Multiplicity},(*))}\}. \end{aligned} \tag{8.58}$$

Die *AggregatedAttribute*–Metadaten werden in analoger Weise behandelt. Die Abbildungsvorschriften (8.59) bis (8.63) sind dementsprechende Modifikationen von (8.54) bis (8.58).

$$\begin{aligned} &AggregatedAttributeRule : META_{\downarrow(*,\mathsf{AggregatedAttribute},(*))} \to ExpressionType \\ &AggregatedAttributeRule(m) \stackrel{def}{=} \text{"Verdichtungsanweisung"}. \end{aligned} \tag{8.59}$$

Unter Zuhilfenahme dieser Abbildung lassen sich die beiden folgenden Funktionen definieren, die ein *TableConstraint*–Objekt anlegen bzw. in einer bestehenden Tabelle die Referenz auf dieses *TableConstraint* aktualisieren:

$$\begin{aligned} &\mu_{AggregatedAttribute_{Create}} : META_{AggregatedAttribute} \to \mathcal{L}_{TableConstraint} \\ &\mu_{AggregatedAttribute_{Create}}(m) \stackrel{def}{=} (\psi(\text{"Integrity rule 'aggregated attribute' for table "}, \\ &\qquad\qquad f_{det_{TableName}}(m.relation.name)), \\ &\qquad f_{det_{AggregatedAttributeRule}}(m), \\ &\qquad f_{det_{TableName}}(m.relation.name), \\ &\qquad \mathsf{TableConstraint}). \end{aligned} \tag{8.60}$$

$$\mu_{AggregatedAttribute_{Update}} : META_{AggregatedAttribute} \times \mathcal{L}_{Table} \to \mathcal{L}_{Table}$$

$$\mu_{AggregatedAttribute_{Update}}(m,t) \stackrel{def}{=}$$

$$\begin{cases} AddTableConstraint(t, \psi(\text{"Integrity rule 'aggregated attribute' for table "}, \\ \qquad\qquad f_{det_{TableName}}(m.relation.name))) \\ \qquad\qquad \text{falls } t.name = f_{det_{TableName}}(m.relation.name) \\ t \qquad\qquad \text{sonst.} \end{cases} \tag{8.61}$$

Die Erweiterung auf eine Menge von *Table*–Objekten:

$$\begin{aligned} &\mu_{AggregatedAttribute_{UpdateSet}} : META_{AggregatedAttribute} \times Pot(\mathcal{L}_{Table}) \to Pot(\mathcal{L}_{Table}) \\ &\mu_{AggregatedAttribute_{UpdateSet}}(m,T) \stackrel{def}{=} \bigcup_{t \in T} \mu_{AggregatedAttribute_{Update}}(m,t). \end{aligned} \tag{8.62}$$

Schließlich wird die Abbildung aller *AggregatedAttribute*–Metadaten eines Schemas durchgeführt:

$$\begin{aligned}
&\mathcal{M}_{AggregatedAttribute} : \mathcal{R} \times \mathcal{L} \to \mathcal{L} \\
&\mathcal{M}_{AggregatedAttribute}(R, L) \stackrel{def}{=\!=} (L \setminus O_{Table}) \\
&\quad \cup \{\mu_{AggregatedAttribute_{Create}}(m) \mid m \in M_{\downarrow(*,\text{AggregatedAttribute},(*))}\} \\
&\quad \cup \{\mu_{AggregatedAttribute_{UpdateSet}}(m, O_{Table}) \mid m \in M_{\downarrow(*,\text{AggregatedAttribute},(*))}\}.
\end{aligned} \tag{8.63}$$

### 8.3.12 Schritt 10: Dimensionspfade anlegen

Für jedes *Dimension–*, *NonCompleteRollUp*– bzw. *RollUp*–Objekt wird die Referenz des entsprechenden *UniqueKeyRole*–Objektes aktualisiert. Weiterhin werden jeweils ein *ForeignKey*– und ein *ForeignKeyRole*–Objekt angelegt und in die entsprechenden *Table*– bzw. *Column*–Objekte eingetragen. Schließlich wird ein *ReferentialConstraint*–Objekt erzeugt.
Ist die *Dimension*– bzw. *(NonComplete)RollUp*–Beziehung an bestimmte Typen geknüpft, so ist zusätzlich ein *TableConstraint*–Objekt notwendig. Im Beispiel aus Abbildung 8.20 ist das *RollUp* nur dann zulässig, wenn ein Datensatz in Tabelle „Ort_des_Verkaufs“ den Typ „Filiale“ besitzt, d. h. der Primärschlüssel der Tabelle „Filialkategorie“ darf nur dann als Fremdschlüssel in Tabelle „Ort_des_Verkaufs“ eingetragen werden, wenn das Attribut „Type“ von „Ort_des_Verkaufs“ den Wert „Filiale“ besitzt. Diese Bedingung lässt sich z. B. durch das in der Abbildung gezeigte *TableConstraint*–Objekt formulieren, i. Allg. sorgt die deterministische Funktion $f_{det_{RollUpTypesRule}}$ für die Formulierung des Constraints. Ergänzend zu diesem *TableConstraint*–Objekt werden die gültigen Typen in den Attributen *typesPKMETA* und *typesFKMETA* des *ReferentialConstraint*–Objektes abgelegt. Dieses doppelte Vorhalten der Informationen ist wie folgt zu erklären: Während mit dem *TableConstraint* die Integrität sichergestellt wird, fungieren die Informationen beim *ReferentialConstraint*–Objekt als Metainformationen, die eventuell später von einem Analysewerkzeug genutzt werden können. Die Transformation von *RollUp*– und *NonCompleteRollUp*–Metadaten unterscheidet sich lediglich in der Multiplizität, die beim neu angelegten *ForeignKeyRole*–Objekt eingetragen wird.

Zur Unterscheidung der beiden *RollUp*–Typen dient die folgende Funktion, die die entsprechende Multiplizität ermittelt.

$$\begin{aligned}
&f_{det_{RollUpMultiplicity}} : META_{\downarrow(*,\text{RollUp},(*))} \to ExpressionType \\
&f_{det_{RollUpMultiplicity}}(m) \stackrel{def}{=\!=} \\
&\qquad \begin{cases} \text{"1..*"} & \text{falls} m.type = \text{COMPLETE} \\ \text{"0..1*"} & \text{sonst} \end{cases}
\end{aligned} \tag{8.64}$$

Die bereits oben erwähnte Hilfsabbildung $f_{det_{RollUpTypesRule}}$ ist folgendermaßen definiert:

$$\begin{aligned}
&f_{det_{RollUpTypesRule}} : META_{\downarrow(*,\text{RollUp},(*))} \to ExpressionType \\
&f_{det_{RollUpTypesRule}}(m) \stackrel{def}{=\!=} \text{"Konsistenzregel"}
\end{aligned} \tag{8.65}$$

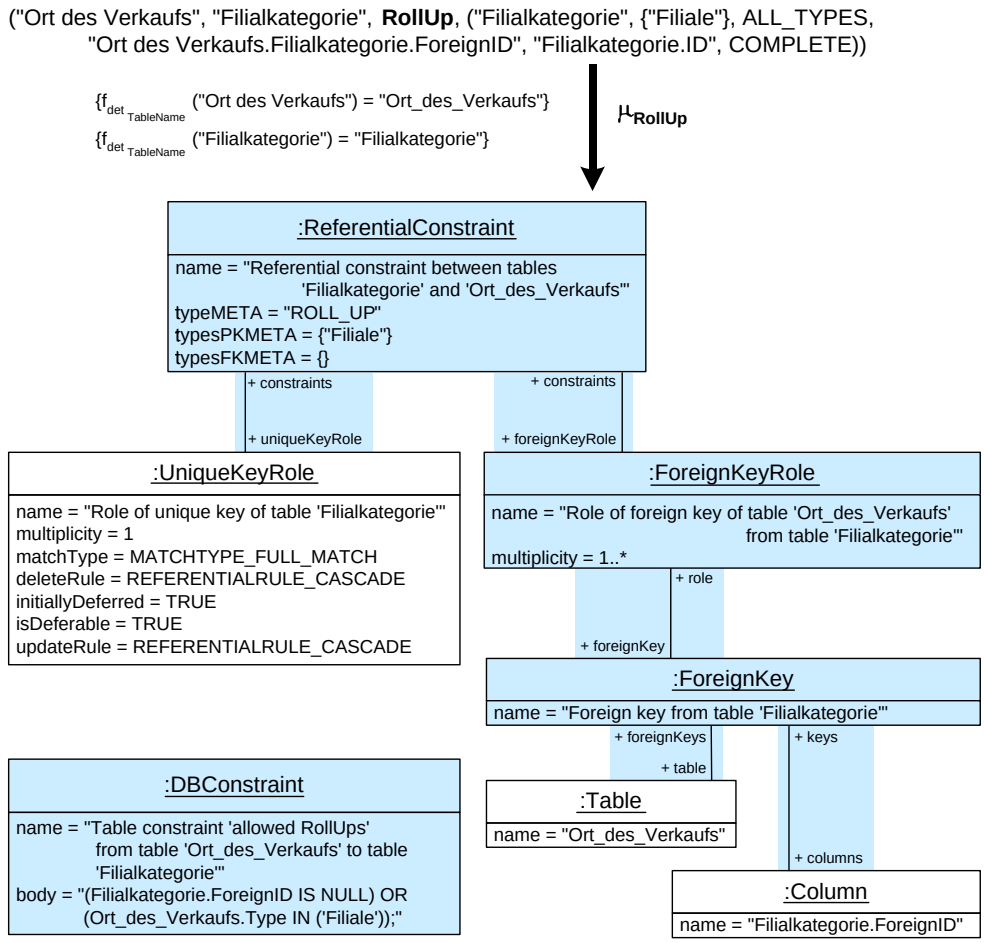



Abbildung 8.20: Abbildung des REMUS–Metadatums *RollUp* auf das *LCD of SQL*–Schema

Die Abbildungsvorschriften zum Anlegen der drei neuen Schemaelemente zeigen (8.66) bis (8.68):

$$
\begin{aligned}
&\mu_{RollUp_{Create\ ForeignKey}} : META_{RollUp} \rightarrow \mathcal{L}_{ForeignKey} \\
&\mu_{RollUp_{Create\ ForeignKey}}(r) \stackrel{def}{=} \\
&\quad (\psi(\text{"Foreign key of table "}, \\
&\qquad f_{det_{TableName}}(r.dimensionalRelationLower.name)), \\
&\qquad f_{det_{TableName}}(r.dimensionalRelationLower.name), \\
&\quad f_{det_{AttributeName}}(r.foreignKey.name), \\
&\quad \psi(\text{"Role of foreign key in referential constraint between tables "}, \\
&\qquad f_{det_{TableName}}(r.dimensionalRelationHigher.name), \\
&\qquad \text{"and"}, f_{det_{TableName}}(r.dimensionalRelationLower.name)), \\
&\quad \text{ForeignKey}).
\end{aligned}
\tag{8.66}
$$

$$\mu_{RollUp_{Create\ ForeignKeyRole}} : META_{RollUp} \rightarrow \mathcal{L}_{ForeignKeyRole}$$

$$\mu_{RollUp_{Create\ ForeignKeyRole}}(r) \stackrel{def}{=}$$

$$\begin{aligned}
&(\psi(\text{"Role of foreign key in referential constraint between tables "},\\
&\quad f_{det_{TableName}}(r.dimensionalRelationHigher.name)\\
&\quad \text{"and"}, f_{det_{TableName}}(r.dimensionalRelationLower.name)),\\
&f_{det_{RollUpMultiplicity}}(r), \text{MATCHTYPE_FULL_MATCH},\\
&\psi(\text{"Foreign key of table "},\\
&\quad f_{det_{TableName}}(r.dimensionalRelationLower.name),\\
&\quad \text{"from table "}, f_{det_{TableName}}(r.dimensionalRelationHigher.name)),\\
&\psi(\text{"Referential constraint between table "},\\
&\qquad f_{det_{TableName}}(r.dimensionalRelationHigher.name),\\
&\qquad \text{"and"}, f_{det_{TableName}}(r.dimensionalRelationLower.name)),\\
&\text{ForeignKeyRole}).
\end{aligned} \tag{8.67}$$

$$\mu_{RollUp_{Create\ ReferentialConstraint}} : META_{RollUp} \rightarrow \mathcal{L}_{ReferentialConstraint}$$

$$\mu_{RollUp_{Create\ ReferentialConstraint}}(r) \stackrel{def}{=}$$

$$\begin{aligned}
&(\psi(\text{"Referential constraint between table "},\\
&\qquad f_{det_{TableName}}(r.dimensionalRelationHigher.name)\\
&\quad \text{"and"}, f_{det_{TableName}}(r.dimensionalRelationLower.name)),\\
&\text{ROLL_UP}, r.typesPK, r.typesFK,\\
&\psi(\text{"Role of primary key of table "},\\
&\quad f_{det_{TableName}}(r.dimensionalRelationHigher.name)),\\
&\psi(\text{"Role of foreign key in referential constraint between tables "},\\
&\quad f_{det_{TableName}}(r.dimensionalRelationHigher.name),\\
&\quad \text{"and "}, f_{det_{TableName}}(r.dimensionalRelationLower.name)),\\
&\text{ReferentialConstraint}).
\end{aligned} \tag{8.68}$$

$$\mu_{RollUp_{Create\ TableConstraint}} : META_{RollUp} \rightarrow \mathcal{L}_{TableConstraint}$$

$$\mu_{RollUp_{Create\ TableConstraint}}(r) \stackrel{def}{=}$$

$$\begin{aligned}
&(\psi(\text{"Table constraint 'allowed RollUps' for table "},\\
&\quad f_{det_{TableName}}(r.dimensionalRelationLower.name),\\
&\quad \text{" to table "}, f_{det_{TableName}}(r.dimensionalRelationHigher.name)),\\
&f_{det_{RollUpTypesRule}}(r),\\
&\text{DBConstraint}).
\end{aligned} \tag{8.69}$$

Aktualisiert werden müssen die entsprechenden *Table*– und *Column*–Objekte, was durch die Funktionen in (8.70) und (8.71) geschieht.

$$\mu_{RollUp_{Update\ Table}} : META_{RollUp} \times \mathcal{L}_{Table} \to \mathcal{L}_{Table}$$

$$\mu_{RollUp_{Update\ Table}}(r,t) \stackrel{def}{=}$$

$$\begin{cases} AddTableForeignKey(t, \psi(\text{"Foreign key of table "}, \\ \qquad\qquad f_{det_{TableName}}(r.dimensionalRelationLower.name), \\ \qquad\qquad \text{" from table "}, f_{det_{TableName}}(r.dimensionalRelationHigher.name))) \\ \qquad \text{falls } t.name = f_{det_{TableName}}(r.dimensionalRelationLower.name) \\ t \qquad\qquad \text{sonst.} \end{cases} \tag{8.70}$$

$$\mu_{RollUp_{Update\ Column}} : META_{RollUp} \times \mathcal{L}_{Column} \to \mathcal{L}_{Column}$$

$$\mu_{RollUp_{Update\ Column}}(r,c) \stackrel{def}{=}$$

$$\begin{cases} AddColumnKey(c, \psi(\text{"Foreign key of table "}, \\ \qquad\qquad f_{det_{TableName}}(r.dimensionalRelationLower.name), \\ \qquad\qquad f_{det_{TableName}}(r.dimensionalRelationLower.name))) \\ \qquad \text{falls } c.foreignKey.name = \\ \qquad\qquad\qquad f_{det_{TableName}}(r.dimensionalRelationLower.name) \\ c \qquad\qquad \text{sonst.} \end{cases} \tag{8.71}$$

Diese Aktualisierungsfunktionen werden in den beiden folgenden Definition auf Mengen erweitert:

$$\begin{aligned} &\mu_{RollUp_{UpdateSet\ Table}} : META_{RollUp} \times Pot(\mathcal{L}_{Table}) \to Pot(\mathcal{L}_{Table}) \\ &\mu_{RollUp_{UpdateSet\ Table}}(m,T) \stackrel{def}{=} \bigcup_{t \in T} \mu_{RollUp_{Update\ Table}}(m,t). \end{aligned} \tag{8.72}$$

$$\begin{aligned} &\mu_{RollUp_{UpdateSet\ Column}} : META_{RollUp} \times Pot(\mathcal{L}_{Column}) \to Pot(\mathcal{L}_{Column}) \\ &\mu_{RollUp_{UpdateSet\ Column}}(m,C) \stackrel{def}{=} \bigcup_{c \in C} \mu_{RollUp_{Update\ Column}}(m,c). \end{aligned} \tag{8.73}$$

Schließlich wird die Abbildung aller *RollUp*–Metadaten eines Schemas durchgeführt:

$$\begin{aligned} &\mathcal{M}_{RollUp} : \mathcal{R} \times \mathcal{L} \to \mathcal{L} \\ &\mathcal{M}_{RollUp}(R,L) \stackrel{def}{=} (L \setminus O_{Table} \setminus O_{Column} \setminus O_{UniqueKeyRole}) \\ &\quad \cup \{\mu_{RollUp_{Create\ ForeignKey}}(m) \mid m \in M_{\downarrow(*,\mathsf{RollUp},(*))}\} \\ &\quad \cup \{\mu_{RollUp_{Create\ ForeignKeyRole}}(m) \mid m \in M_{\downarrow(*,\mathsf{RollUp},(*))}\} \\ &\quad \cup \{\mu_{RollUp_{Create\ ReferentialConstraint}}(m) \mid m \in M_{\downarrow(*,\mathsf{RollUp},(*))}\} \\ &\quad \cup \{\mu_{RollUp_{Create\ TableConstraint}}(m) \mid m \in M_{\downarrow(*,\mathsf{RollUp},(*))}\} \\ &\quad \cup \{\mu_{RollUp_{UpdateSet\ Table}}(m, O_{Table}) \mid m \in M_{\downarrow(*,\mathsf{RollUp},(*))}\} \\ &\quad \cup \{\mu_{RollUp_{UpdateSet\ Column}}(m, O_{Column}) \mid m \in M_{\downarrow(*,\mathsf{RollUp},(*))}\}. \end{aligned} \tag{8.74}$$

Die Abbildungsvorschriften für *Dimension*–Objekte sind analog zu denen der *RollUps*, so dass sie hier nicht nochmals aufgelistet werden. Es gelten entsprechend die Formeln (8.65) bis (8.74), nur mit dem Unterschied, dass jedes Vorkommen von „RollUp“ durch „Dimension“ zu ersetzen ist.

### 8.3.13 Schritt 11: Additivität festlegen

Die in einem *Additivity*–Metadatum festgehaltenen zulässigen Operatoren für eine Faktattribut–Dimension–Kombination werden, wie in Abbildung 8.21 zu sehen, weiterhin als Metainformationen festgehalten.

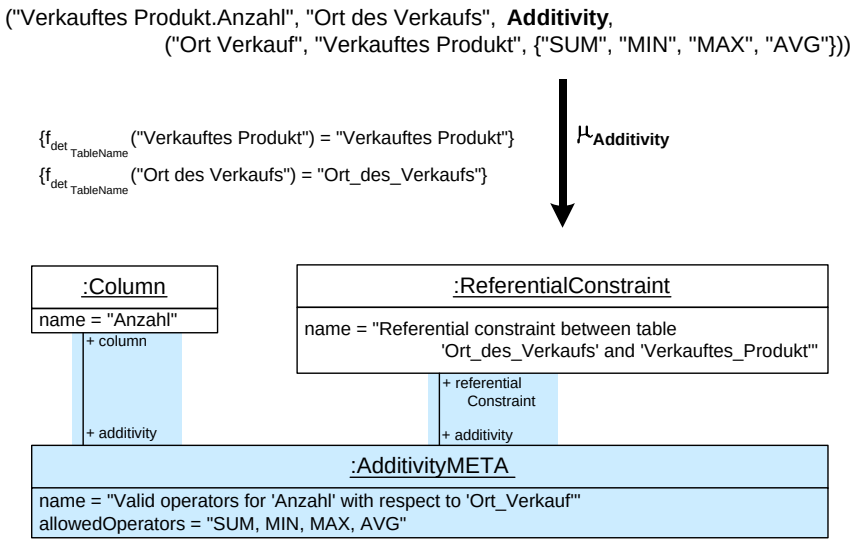



Abbildung 8.21: Abbildung des REMUS–Metadatums *Additivity* auf das *LCD of SQL*–Schema

Das neue Schemaelement wird durch folgende Abbildungsvorschrift angelegt:

$$\begin{aligned}
&\mu_{Additivity_{Create}} : META_{Additivity} \rightarrow \mathcal{L}_{AdditivityMETA}\\
&\mu_{Additivity_{Create}}(m) \stackrel{def}{=} (\psi(\text{"Valid operators for "},\\
&\qquad f_{det_{AttributeName}}(m.factAttribute.name),\\
&\qquad \text{" with respect to "}, m.dimension.name),\\
&\qquad m.allowedOperators,\\
&\qquad f_{det_{AttributeName}}(m.factAttribute.name),\\
&\qquad \psi(\text{"Referential constraint between table "},\\
&\qquad\quad f_{det_{TableName}}(r.factRelation.name)\\
&\qquad\quad \text{" and "}, f_{det_{TableName}}(r.dimensionalRelation.name)),\\
&\qquad \text{AdditivityMETA}).
\end{aligned} \tag{8.75}$$

Aktualisiert werden müssen die entsprechenden *Column*– und *ReferentialConstraint*–Objekte, was durch die beiden folgenden Funktionen geschieht:

$$\begin{aligned}
&\mu_{Additivity_{Update\ Column}} : META_{Additivity} \times \mathcal{L}_{Column} \rightarrow \mathcal{L}_{Column}\\
&\mu_{Additivity_{Update\ Column}}(m, c) \stackrel{def}{=}\\
&\quad \begin{cases}
AddColumnAdditivity(c, \psi(\text{"Valid operators for "},\\
\qquad\qquad f_{det_{AttributeName}}(m.factAttribute.name),\\
\qquad\qquad \text{" with respect to "}, m.dimension.name))\\
\qquad\qquad \text{falls } c.name = f_{det_{AttributeName}}(a.factAttribute.name)\\
c \qquad\qquad \text{sonst.}
\end{cases}
\end{aligned} \tag{8.76}$$

$$\mu_{Additivity_{Update\ ReferentialConstraint}} : META_{Additivity} \times \mathcal{L}_{ReferentialConstraint} \rightarrow \mathcal{L}_{ReferentialConstraint}$$

$$\mu_{Additivity_{Update\ ReferentialConstraint}}(m, r) \stackrel{def}{=}$$

$$\begin{cases} AddReferentialConstraintAdditivity \\ \quad (r, \psi(\text{"Valid operators for "}, \\ \quad f_{det_{AttributeName}}(m.factAttribute.name), \\ \quad \text{" with respect to "}, \\ \quad f_{det_{TableName}}(m.dimensionalRelation.name))) \\ \qquad \text{falls } r.name = \psi(\text{"Referential constraint between table "}, \\ \qquad\quad f_{det_{TableName}}(r.factRelation.name), \\ \qquad\quad \text{" and "}, f_{det_{TableName}}(r.dimensionalRelation.name)) \\ c \qquad\qquad \text{sonst.} \end{cases} \tag{8.77}$$

Die Erweiterung dieser beiden Aktualisierungsfunktionen auf Mengen ergibt:

$$\begin{aligned} &\mu_{Additivity_{UpdateSet\ Column}} : META_{Additivity} \times Pot(\mathcal{L}_{Column}) \rightarrow Pot(\mathcal{L}_{Column}) \\ &\mu_{Additivity_{UpdateSet\ Column}}(m, C) \stackrel{def}{=} \bigcup_{c \in C} \mu_{Additivity_{Update\ Column}}(m, c). \end{aligned} \tag{8.78}$$

$$\begin{aligned} &\mu_{Additivity_{UpdateSet\ ReferentialConstraint}} : META_{Additivity} \times Pot(\mathcal{L}_{ReferentialConstraint}) \rightarrow Pot(\mathcal{L}_{ReferentialConstraint}) \\ &\mu_{Additivity_{UpdateSet\ ReferentialConstraint}}(m, R) \stackrel{def}{=} \bigcup_{r \in R} \mu_{Additivity_{Update\ ReferentialConstraint}}(m, r). \end{aligned} \tag{8.79}$$

Die Übertragung aller *Additivity*–Metadaten:

$$\begin{aligned} &\mathcal{M}_{Additivity} : \mathcal{R} \times \mathcal{L} \rightarrow \mathcal{L} \\ &\mathcal{M}_{Additivity}(R, L) \stackrel{def}{=} (L \setminus O_{Column} \setminus O_{ReferentialConstraint}) \\ &\quad \cup \{\mu_{Additivity_{Create}}(m) \mid m \in M_{\downarrow(*,\text{Additivity},(*))}\} \\ &\quad \cup \{\mu_{Additivity_{UpdateSet\ Column}}(m, O_{Column}) \mid m \in M_{\downarrow(*,\text{Additivity},(*))}\} \\ &\quad \cup \{\mu_{Additivity_{UpdateSet\ ReferentialConstraint}}(m, O_{ReferentialConstraint}) \mid \\ &\qquad\qquad m \in M_{\downarrow(*,\text{Additivity},(*))}\}. \end{aligned} \tag{8.80}$$

### 8.3.14 Schritt 12: *SharedRollUp* markieren

Die *SharedRollUp*–Instanzen lassen sich nicht direkt in ein relationales Modell abbilden. Es handelt sich um *Viele–zu–Viele*–Beziehungen, deren „natürliche Auflösung“ sich in einer Zwischentabelle widerspiegeln würde. Im Hinblick auf spätere Auswertungen ist diese Darstellung aber nicht brauchbar, denn ein Navigieren entlang dieser Pfade wäre nicht mehr möglich. Aus diesem Grunde wird das *SharedRollUp* lediglich als Metainformation festgehalten, evtl. kann später eine auswertende Applikation diese Informationen nutzen. Konkret wird, wie in Abbildung 8.22 dargestellt, ein *Mapping-META*–Objekt angelegt, das die Berechnungsvorschrift und die zulässigen Typen festhält.

("Woche", "Jahr", **SharedRollUp**, ("{Jahr}", ALL_TYPES, ALL_TYPES,
"Berechnung Jahr", {"SUM"}))
("Berechnung Jahr", **Computation**, (("Woche.Bezeichnung"),
"ISO Wochenberechnung", "Jahr.Bezeichnung"))

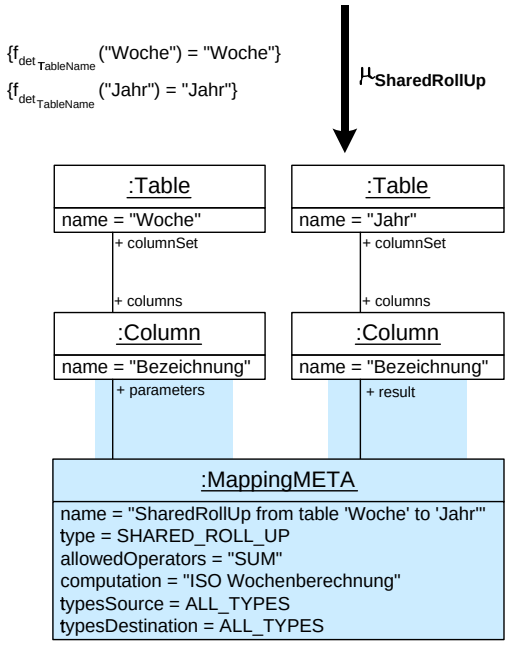



Abbildung 8.22: Abbildung des REMUS–Metadatums *SharedRollUp* auf das *LCD of SQL*–Schema

Die Abbildungsvorschrift zum Anlegen des *MappingMETA*–Objektes ist in (8.81) definiert.

$$
\begin{aligned}
&\mu_{SharedRollUp_{Create}} : META_{SharedRollUp} \rightarrow \mathcal{L}_{MappingMETA} \\
&\mu_{SharedRollUp_{Create}}(m) \stackrel{def}{=\!=} \\
&\quad (\psi(\text{"SharedRollUp from table "}, \\
&\quad\ f_{det_{TableName}}(m.dimensionalRelationLower.name), \text{" to "}, \\
&\quad\ f_{det_{TableName}}(m.dimensionalRelationHigher.name)), \\
&\quad\ \text{SHARED\_ROLL\_UP}, m.allowedOperators, \\
&\quad\ m.computation.computation, \\
&\quad\ \phi(m.validTypesLowerLevel.name), \\
&\quad\ \phi(m.validTypesHigherLevel.name), \\
&\quad\ m.computation.parameters, \\
&\quad\ m.computation.result, \\
&\quad\ \text{MappingMETA}).
\end{aligned}
\tag{8.81}
$$

Eine Aktualisierungsfunktion wird für diesen Schritt nicht definiert, weil die Fragestellung „An welchen *SharedRollUp*–Instanzen ist ein Attribut Parameter ?“ nicht relevant ist. Aufgrund seiner Natur als Metadatum wird der Zugriff immer über das *MappingMETA*–Objekt erfolgen. Im *LCD of SQL*–Metamodell (siehe Seite 165) ist diese Tatsache durch die einseitige Assoziation zwischen den Metaklassen *Column* und *MappingMETA* ausgedrückt. Somit kann das Übertragen aller *SharedRollUp*–Metadaten wie folgt erreicht werden:

$$
\begin{aligned}
&\mathcal{M}_{SharedRollUp} : \mathcal{R} \times \mathcal{L} \rightarrow \mathcal{L} \\
&\mathcal{M}_{SharedRollUp}(R, L) \stackrel{def}{=\!=} L \cup \{\mu_{SharedRollUp_{Create}}(m) \mid m \in M_{\downarrow(*,\text{SharedRollUp},(*))}\}.
\end{aligned}
\tag{8.82}
$$

### 8.3.15 Schritt 13: *DimensionalMapping* markieren

Eine *DimensionalMapping*–Instanz wird analog zum vorigen Schritt wie ein *SharedRollUp*–Objekt transformiert, d. h. es werden Metainformationen angelegt. Abbildung 8.23 zeigt ein Beispiel.

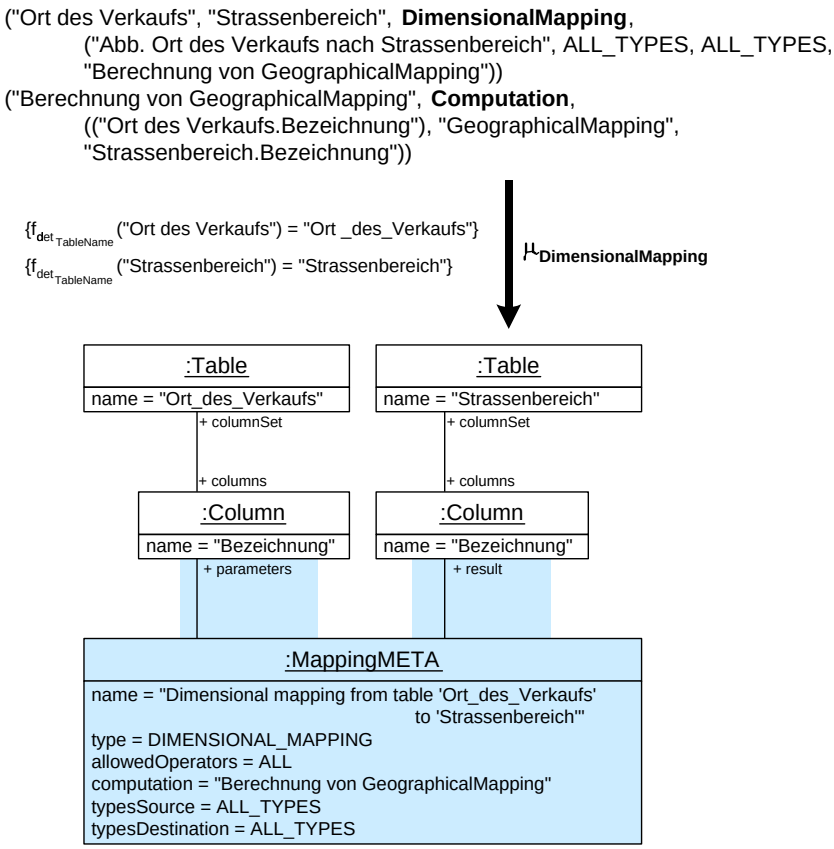



Abbildung 8.23: Abbildung des REMUS–Metadatums *DimensionalMapping* auf das *LCD of SQL*–Schema

Die beiden folgenden Abbildungsvorschriften sind entsprechende Modifikationen von (8.81) und (8.82).

$$
\begin{aligned}
&\mu_{DimensionalMapping_{Create}} : META_{DimensionalMapping} \rightarrow \mathcal{L}_{MappingMETA} \\
&\mu_{DimensionalMapping_{Create}}(m) \stackrel{def}{=} \\
&\quad (\psi(\text{"DimensionalMapping from table "}, \\
&\qquad f_{det_{TableName}}(m.dimensionalRelationSource.name), \text{" to "}, \\
&\qquad f_{det_{TableName}}(m.dimensionalRelationDestination.name)), \\
&\quad \text{DIMENSIONAL\_MAPPING}, m.allowedOperators, \\
&\quad m.computation.computation, \\
&\quad \phi(m.dimensionalRelationSource.name), \\
&\quad \phi(m.dimensionalRelationDestination.name), \\
&\quad m.computation.parameters, \\
&\quad m.computation.result, \\
&\quad \text{MappingMETA}).
\end{aligned}
\tag{8.83}
$$

$$\begin{aligned}&\mathcal{M}_{DimensionalMapping} : \mathcal{R} \times \mathcal{L} \to \mathcal{L}\\&\mathcal{M}_{DimensionalMapping}(R, L) \stackrel{def}{=\!=} L\\&\quad \cup \{\mu_{DimensionalMapping_{Create}}(m) \mid m \in M_{\downarrow(*,\text{DimensionalMapping},(*))}\}.\end{aligned} \tag{8.84}$$

### 8.3.16 Schritt 14: Assoziationen anlegen

Eine Assoziation wird wie in Abbildung 8.24 dargestellt transformiert: Zwischen den beiden über die Tabelle „MTMOrt_des_Verkaufs“ verbundenen Tabellen „Ort_des_Verkaufs“ und „Artikel“ werden jeweils zwei völlig symmetrische Primär–Fremdschlüsselbeziehungen angelegt. Das Änderungs– und Löschverhalten des Primärschlüssels wird auf den Wert „CASCADE“ für Weitergabe gesetzt, d. h. das Ändern oder Löschen von Daten in einer der beiden dimensionalen Tabellen wird an die Zwischentabelle propagiert. Gilt die Assoziation auf der einen wie auf der anderen Seite nur für bestimmte Typen, so wird diese Einschränkung in Form eines Datenbankconstraint festgehalten. Im Beispiel muss ein Datensatz in der Tabelle „Ort_des_Verkaufs“ vom Typ „Filiale“ oder „Kaufhaus“ sein, damit er an der Assoziation teilhaben kann. Dieses wird durch das entsprechende *DBConstraint*–Objekt sichergestellt.

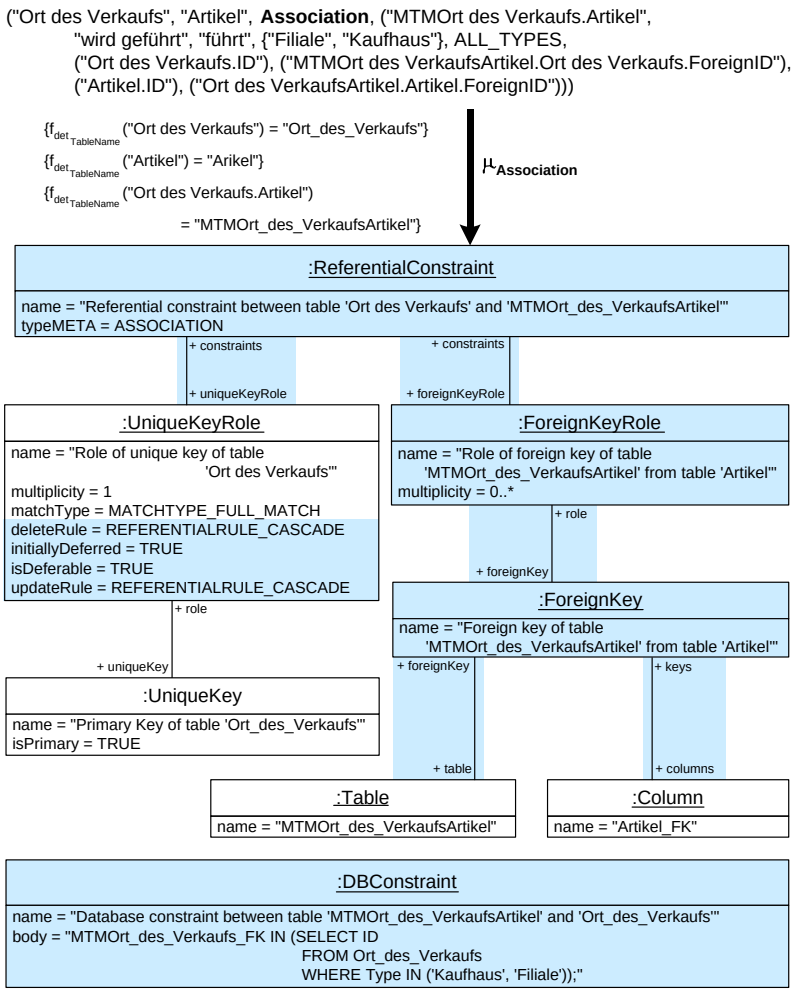



Abbildung 8.24: Abbildung des REMUS–Metadatums *Association* auf das *LCD of SQL*–Schema

Die Abbildungsvorschriften zum Anlegen der neuen Schemaelemente sind in (8.86) bis (8.89) beschrieben, (8.85) stellt eine von (8.89) genutzte Hilfssfunktion zu Verfügung.

$$\begin{aligned} &AsscociationRule : META_{\downarrow(*,\text{Association},*)} \rightarrow ExpressionType \\ &MultiplicityRule(m) \stackrel{def}{=} \text{"Integritätsregel"}. \end{aligned} \tag{8.85}$$

$$\begin{aligned} &\mu_{Association_{Create\ ForeignKey}} : META_{Association} \rightarrow \mathcal{L}_{ForeignKey} \\ &\mu_{Association_{Create\ ForeignKey}}(r) \stackrel{def}{=} \\ &\quad (\psi(\text{"Foreign key of table "}, \\ &\qquad f_{det_{TableName}}(a.dimensionalRelationA.name)), \\ &\qquad f_{det_{TableName}}(a.dimensionalRelationA.name), \\ &\quad f_{det_{AttributeName}}(a.foreignKey.name), \\ &\quad \psi(\text{"Role of foreign key in referential constraint between tables "}, \\ &\qquad f_{det_{TableName}}(a.dimensionalRelationA.name), \\ &\qquad \text{" and "}, f_{det_{TableName}}(a.intermediateRelation.name)), \\ &\quad \text{ForeignKey}). \end{aligned} \tag{8.86}$$

$$\begin{aligned} &\mu_{Association_{Create\ ForeignKeyRole}} : META_{Association} \rightarrow \mathcal{L}_{ForeignKeyRole} \\ &\mu_{Association_{Create\ ForeignKeyRole}}(a) \stackrel{def}{=} \\ &\quad (\psi(\text{"Role of foreign key in referential constraint between tables "}, \\ &\qquad f_{det_{TableName}}(a.dimensionalRelationA.name) \\ &\qquad \text{" and "}, f_{det_{TableName}}(a.intermediateRelation.name)), \\ &\quad \text{"0.. * "}, \text{MATCHTYPE\_FULL\_MATCH}, \\ &\quad \psi(\text{"Foreign key of table "}, \\ &\qquad f_{det_{TableName}}(a.intermediateRelation.name), \\ &\qquad \text{" from table "}, f_{det_{TableName}}(a.dimensionalRelationA.name)), \\ &\quad \psi(\text{"Referential constraint between table "}, \\ &\qquad f_{det_{TableName}}(a.dimensionalRelationA.name), \\ &\qquad \text{" and "}, f_{det_{TableName}}(r.intermediateRelation.name)), \\ &\quad \text{ForeignKeyRole}). \end{aligned} \tag{8.87}$$

$$\begin{aligned} &\mu_{Association_{Create\ ReferentialConstraint}} : META_{Association} \rightarrow \mathcal{L}_{ReferentialConstraint} \\ &\mu_{Association_{Create\ ReferentialConstraint}}(a) \stackrel{def}{=} \\ &\quad (\psi(\text{"Referential constraint between table "}, \\ &\qquad f_{det_{TableName}}(a.dimensionalRelationA.name) \\ &\qquad \text{" and "}, f_{det_{TableName}}(a.intermediateRelation.name)), \\ &\quad \text{ASSOCIATION}, a.typesA, a.typesB, \\ &\quad \psi(\text{"Role of primary key of table "}, \\ &\qquad f_{det_{TableName}}(a.dimensionalRelationA.name)), \\ &\quad \psi(\text{"Role of foreign key in referential constraint between tables "}, \\ &\qquad f_{det_{TableName}}(a.dimensionalRelationA.name), \\ &\qquad \text{" and "}, f_{det_{TableName}}(r.intermediateRelation.name)), \\ &\quad \text{ReferentialConstraint}). \end{aligned} \tag{8.88}$$

$$\mu_{Association_{Create\ DBConstraint}} : META_{Association} \rightarrow \mathcal{L}_{DBConstraint}$$

$$\mu_{Association_{Create\ DBConstraint}}(a) \stackrel{def}{=}$$
$$(\psi(\text{"Database constraint between table "},$$
$$f_{det_{TableName}}(a.intermediateRelation.name),$$
$$\text{" and "}, f_{det_{TableName}}(a.dimensionalRelationA.name)),$$
$$f_{det_{AssociationTypesRule}}(a),$$
$$\text{DBConstraint}). \tag{8.89}$$

Aktualisiert werden müssen die betroffenen *Table–*, *Column–* und *UniqueKeyRole–*Objekte; dies geschieht in den drei folgenden Abbildungsvorschriften.

$$\mu_{Association_{Update\ Table}} : META_{Association} \times \mathcal{L}_{Table} \rightarrow \mathcal{L}_{Table}$$

$$\mu_{Association_{Update\ Table}}(a, t) \stackrel{def}{=}$$
$$\begin{cases} AddTableForeignKey(t, \psi(\text{"Foreign key of table "}, & \\ \qquad f_{det_{TableName}}(a.intermediateRelation.name), & \\ \qquad \text{" from table "}, & \\ \qquad f_{det_{TableName}}(a.dimensionalRelationA.name)) & \\ \qquad \text{falls } t.name = f_{det_{TableName}}(a.intermediateRelation.name) & \\ t \qquad \text{sonst.} & \end{cases} \tag{8.90}$$

$$\mu_{Association_{Update\ Column}} : META_{Association} \times \mathcal{L}_{Column} \rightarrow \mathcal{L}_{Column}$$

$$\mu_{Association_{Update\ Column}}(a, c) \stackrel{def}{=}$$
$$\begin{cases} AddColumnKey(c, \psi(\text{"Foreign key of table "}, & \\ \qquad f_{det_{TableName}}(a.intermediateRelation.name), & \\ \qquad \text{" from table "}, & \\ \qquad f_{det_{TableName}}(a.dimensionalRelationA.name)) & \\ \qquad \text{falls } c.foreignKey.name = f_{det_{TableName}}(a.intermediateRelation.name) & \\ c \qquad \text{sonst.} & \end{cases} \tag{8.91}$$

$$\mu_{Association_{Update\ UniqueKeyRole}} : META_{Association} \times \mathcal{L}_{UniqueKeyRole} \rightarrow \mathcal{L}_{UniqueKeyRole}$$

$$\mu_{Association_{Update\ UniqueKeyRole}}(a, u) \stackrel{def}{=}$$
$$\begin{cases} AddUniqueKeyRoleConstraint(u, \psi(\text{"Referential constraint between table "}, & \\ \qquad f_{det_{TableName}}(r.dimensionalRelationA.name) & \\ \qquad \text{" and "}, f_{det_{TableName}}(r.intermediateRelation.name)) & \\ \qquad \text{falls } u.uniqueKey.table.name = & \\ \qquad\qquad f_{det_{TableName}}(r.dimensionalRelationA.name) & \\ u \qquad \text{sonst.} & \end{cases} \tag{8.92}$$

Diese drei Aktualisierungsfunktionen werden im Folgenden auf Mengen erweitert.

$$\mu_{Association_{UpdateSet\ Table}} : META_{Association} \times Pot(\mathcal{L}_{Table}) \rightarrow Pot(\mathcal{L}_{Table})$$
$$\mu_{Association_{UpdateSet\ Table}}(r, T) \stackrel{def}{=} \bigcup_{t \in T} \mu_{Association_{Update\ Table}}(r, t). \tag{8.93}$$

$$\begin{aligned}&\mu_{Association_{UpdateSet\ Column}} : META_{Association} \times Pot(\mathcal{L}_{Column}) \rightarrow Pot(\mathcal{L}_{Column})\\&\mu_{Association_{UpdateSet\ Column}}(r, C) \stackrel{def}{=} \bigcup_{c \in C} \mu_{Association_{Update\ Column}}(r, c).\end{aligned} \tag{8.94}$$

$$\begin{aligned}&\mu_{Association_{UpdateSet\ UniqueKeyRole}} : META_{Association} \times Pot(\mathcal{L}_{UniqueKeyRole})\\&\qquad\qquad\qquad\qquad\qquad\qquad \rightarrow Pot(\mathcal{L}_{UniqueKeyRole})\\&\mu_{Association_{UpdateSet\ UniqueKeyRole}}(r, U) \stackrel{def}{=} \bigcup_{u \in U} \mu_{Association_{Update\ UniqueKeyRole}}(r, u).\end{aligned} \tag{8.95}$$

Schließlich kann die Abbildung aller *Association*–Metadaten durchgeführt werden:

$$\begin{aligned}&\mathcal{M}_{Association} : \mathcal{R} \times \mathcal{L} \rightarrow \mathcal{L}\\&\mathcal{M}_{Association}(R, L) \stackrel{def}{=} (L \setminus O_{Table} \setminus O_{Column} \setminus O_{UniqueKeyRole})\\&\quad \cup \{\mu_{RollUp_{Create\ ForeignKey}}(m) \mid m \in M_{\downarrow(*,\text{Association},(*))}\}\\&\quad \cup \{\mu_{RollUp_{Create\ ForeignKeyRole}}(m) \mid m \in M_{\downarrow(*,\text{Association},(*))}\}\\&\quad \cup \{\mu_{RollUp_{Create\ ReferentialConstraint}}(m) \mid m \in M_{\downarrow(*,\text{Association},(*))}\}\\&\quad \cup \{\mu_{RollUp_{UpdateSet\ Table}}(m, O_{Table}) \mid m \in M_{\downarrow(*,\text{Association},(*))}\}\\&\quad \cup \{\mu_{RollUp_{UpdateSet\ Column}}(m, O_{Column}) \mid m \in M_{\downarrow(*,\text{Association},(*))}\}\\&\quad \cup \{\mu_{RollUp_{UpdateSet\ UniqueKeyRole}}(m, O_{UniqueKeyRole}) \mid m \in M_{\downarrow(*,\text{Association},(*))}\}.\end{aligned} \tag{8.96}$$

Analoge Abbildungsvorschriften müssen für die „B“–Seite der Assoziation definiert werden.

### 8.3.17 Schritt 15: Komposition markieren

Jedes *Composition*–Metadatum wird als *CompositionMETA*–Objekt festgehalten. Es hat somit im Schema keine unmittelbaren Auswirkungen, aber das Beibehalten dieser Informationen kann eventuell später für Lade– oder Analysewerkzeuge nützlich sein. Abbildung 8.25 zeigt exemplarisch die Abbildung.

Die Abbildungsvorschrift zum Anlegen des *CompositionMETA*–Objektes:

$$\begin{aligned}&\mu_{Composition_{Create}} : META_{Composition} \rightarrow \mathcal{L}_{CompositionMETA}\\&\mu_{Composition_{Create}}(m) \stackrel{def}{=}\\&\qquad (c.name, c.multiplicity, c.detail, c.aggregated,\\&\qquad\ \text{CompositionMETA}).\end{aligned} \tag{8.97}$$

Eine Aktualisierungsfunktion ist in diesem Schritt nicht notwendig, weil die Beziehung von einer Tabelle zum Metadatum nicht relevant ist. Somit kann das Übertragen aller *Composition*–Metadaten durch folgende Abbildung erreicht werden:

$$\begin{aligned}&\mathcal{M}_{Composition} : \mathcal{R} \times \mathcal{L} \rightarrow \mathcal{L}\\&\mathcal{M}_{Composition}(R, L) \stackrel{def}{=} L \cup \{\mu_{Composition_{Create}}(m) \mid m \in M_{\downarrow(*,\text{Composition},(*))}\}.\end{aligned} \tag{8.98}$$

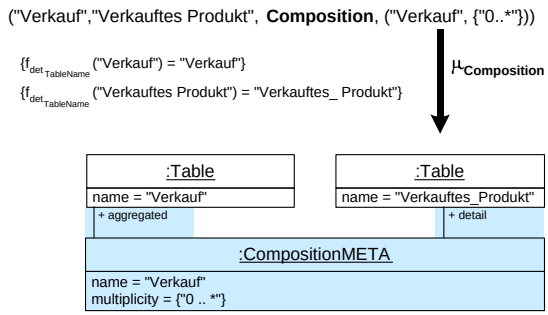



Abbildung 8.25: Abbildung des REMUS–Metadatums *Composition* auf das *LCD of SQL*–Schema

### 8.3.18 Transformation eines Schemas

Unter Anwendung der einzelnen in den Abschnitten 8.3.3 bis 8.3.16 definierten Schritte lässt sich ein ganzes *REMUS*–Schema in ein *LCD of SQL*–Schema abbilden. Dazu werden in (8.99) die einzelnen Transformationsschritte in der vorgestellten Reihenfolge aufgerufen.

Sei $R \in \mathcal{R}$, $L_{Empty}$ ein leeres *LCD of SQL*–Schema,
$M_{PrimaryKey_{Grouped}}$ die in (8.19) definierte Zerlegung der *PrimaryKey*–Metadaten von $R$,
$M_{ConceptualKey_{Grouped}}$ die in (8.30) definierte Zerlegung der *ConceptualKey*–Metadaten von $R$.
Dann ist die Abbildung $\mathcal{M}_{Schema} : \mathcal{R}_{Valid} \rightarrow \mathcal{L}_{Valid}$ wie folgt definiert:

$$
\begin{aligned}
&\mathcal{M}_{Schema}(R) \stackrel{def}{=\!=} \\
&\quad \mathcal{M}_{Composition}(R, \\
&\qquad \mathcal{M}_{Association}(R, \\
&\qquad\quad \mathcal{M}_{DimensionalMapping}(R, \\
&\qquad\qquad \mathcal{M}_{SharedRollUp}(R, \\
&\qquad\qquad\quad \mathcal{M}_{Additivity}(R, \\
&\qquad\qquad\qquad \mathcal{M}_{RollUp}(R, \\
&\qquad\qquad\qquad\quad \mathcal{M}_{Dimension}(R, \\
&\qquad\qquad\qquad\qquad \mathcal{M}_{AggregatedAttribute}(R, \\
&\qquad\qquad\qquad\qquad\quad \mathcal{M}_{Multiplicity}(R, \\
&\qquad\qquad\qquad\qquad\qquad \mathcal{M}_{Optional}(R, \\
&\qquad\qquad\qquad\qquad\qquad\quad \mathcal{M}_{Valid}(R, \\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathcal{M}_{Identifier}(R, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathcal{M}_{DerivedAttribute}(R, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathcal{M}_{ConceptualKey}(R, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathcal{M}_{PrimaryKey}(R, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathcal{M}_{Attribute}(R, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathcal{M}_{Relation}(R, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathcal{M}_{DataType}(R, L_{Empty}))), \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad M_{PrimaryKey_{Grouped}}), \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad M_{ConceptualKey_{Grouped}})), \\
&\qquad\qquad\qquad\qquad\qquad\qquad )))))))))))).
\end{aligned}
\tag{8.99}
$$

## 8.4 Zusammenfassung

In diesem Kapitel wurde die Abbildung vom logischen auf die physische Entwurfsebene beschrieben. Dazu wurde zunächst in Abschnitt 8.2 das physische Metamodell *LCD of SQL* spezifiziert, das einen gemeinsamen Kern aus dem SQL–Standard und aus kommerziellen Systemen umfasst. In Abschnitt 8.3 wurde die eigentliche Abbildung definiert. Diese orientierte sich an den *REMUS*–Schemaelementen, wobei zunächst die Objekte und Attribute übertragen und dann die Kategorie A– und Kategorie B–Metadaten systematisch abgearbeitet wurden. Wie schon im Entwurfsschritt von der konzeptionellen auf die logische Ebene werden gewisse Entwurfsentscheidungen mittels einer deterministischen Funktion bestimmt. Dies betrifft neben der Abbildung der Datentypen und der Formulierung von Constraints und Formeln auch die Benennung von Entwurfsobjekten, womit die Möglichkeit des Einfließens projekt– und organisationsspezifischer Konventionen, z. B. hinsichtlich der Namensgebung, möglich wird.
Um die Funktionsweise der Abbildung zu verdeutlichen, wird in Tabelle 8.8 festgehalten, in welchem Schritt welche REMUS–Schemaelemente abgearbeitet werden. Tabelle 8.9 beschreibt, welche *LCD of SQL*–Objekttypen in welchem Schritt angelegt bzw. manipuliert und welche deterministischen Funktionen an welcher Stelle benutzt werden. Hierbei bedeutet der Eintrag „C“ (*create*) Anlegen des Objektes und der Eintrag „U“ (*update*) Manipulieren eines bereits in einem früheren Schritt angelegten Objektes. Ein Punkt in der Spalte einer deterministischen Funktion bedeutet das Definieren (erstmalige Auftreten) in diesem Schritt. Ein in Klammern gesetzter Punkt besagt, dass die entsprechende Funktion in diesem Schritt auftritt, aber bereits in einem früheren Schritt der Transformation festgelegt worden ist.

| Transformationsalgorithmus $\mathcal{M}_{Schema}$ | | REMUS | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Objekte | | | Kategorie A–Metadaten | | | | | | | | | | | | | Kategorie B–Metadaten | | | | | | | | |
| | | DataTypes | Dimensional Relationen | Faktrelationen | Attribute | AggregatedAttribute | Computation | ConceptualKey | Identifier | IdentifierValue | Multiplicity | ObjectType | Optional | PrimaryKey | Reference | Valid | Additivity | Association | Composition | Dimension | DimensionalMapping | RollUp | SharedRollUp |
| **Objekte anlegen** | Schritt 1: Datentypen anlegen | • | | | | | | | | | | | | | | | | | | | | | | | | |
| | Schritt 2: Tabellen anlegen | | • | • | | | | | | | | • | | | | | | | | | | | | | | |
| | Schritt 3: Attribute anlegen | | | | • | | | | | | | | | | | | | | | | | | | | | |
| **Kategorie A–Metadaten abarbeiten** | Schritt 4: Primärschlüssel | | | | | | | | | | | | | • | | | | | | | | | | | | |
| | Schritt 5: Konzeptionelle Schlüssel | | | | | | | • | | | | | | | | | | | | | | | | | | |
| | Schritt 6: Berechnete Attribute | | | | | | • | | | | | | | | | | | | | | | | | | | |
| | Schritt 7: *Identifier(Value), Valid* | | | | | | | | • | • | | | | | | • | | | | | | | | | | |
| | Schritt 8: Optionale Attribute | | | | | | | | | | | | • | | | | | | | | | | | | | |
| | Schritt 9: Multiplizität | | | | | • | | | | | • | | | | | | | | | | | | | | | |
| **Kategorie B–Metadaten abarbeiten** | Schritt 10: Dimensionspfade | | | | | | | | | | | | | | | | | | | | • | | • | | | |
| | Schritt 11: Additivität | | | | | | | | | | | | | | | | • | | | | | | | | | |
| | Schritt 12: *SharedRollUp* | | | | | | • | | | | | | | | | | | | | | | | | | | • |
| | Schritt 13: *DimensionalMapping* | | | | | | • | | | | | | | | | | | | | | | • | | | | |
| | Schritt 14: Assoziationen | | | | | | | | | | | | | | | | | • | | | | | | | | |
| | Schritt 15: Kompositionen | | | | | | | | | | | | | | | | | | • | | | | | | | |

Tabelle 8.8: Transformationsschritte und genutzte *REMUS*–Objekte

| | | **Transformationsalgorithmus $\mathcal{M}_{Schema}$** | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **Objekte/ Attribute abarbeiten** | | | **Kategorie A– Metadaten abarbeiten** | | | | | | **Kategorie B– Metadaten abarbeiten** | | | | | |
| | | Schritt 1: Datentypen anlegen | Schritt 2: Tabellen anlegen | Schritt 3: Attribute anlegen | Schritt 4: Primärschlüssel | Schritt 5: Konzeptionelle Schlüssel | Schritt 6: Berechnete Attribute | Schritt 7: *Identifier(Value), Valid* | Schritt 8: Optionale Attribute | Schritt 9: Multiplizität | Schritt 10: Dimensionspfade | Schritt 11: Additivität | Schritt 12: *SharedRollUp* | Schritt 13: *DimensionalMapping* | Schritt 14: Assoziationen | Schritt 15: Kompositionen |
| **LCD of SQL** | **Objekte** | | | | | | | | | | | | | | | |
| | AdditivityMETA | | | | | | | | | | | C | | | | |
| | Column | | | C | U | U | U | U | U | | U | U | | | U | |
| | ColumnConstraint | | | | | | | C | | | | | | | | |
| | ColumnType | C | | | | | | | | | | | | | | |
| | CompositionMETA | | | | | | | | | | | | | | | C |
| | DBConstraint | | | | | | | | | | C | | | | C | |
| | ForeignKey | | | | | | | | | | C | | | | C | |
| | ForeignKeyRole | | | | | | | | | | C | | | | C | |
| | MappingMETA | | | | | | | | | | | | C | C | | |
| | Table | | C | U | U | U | | U | | U | U | | C | | U | |
| | TableConstraint | | | | | | | C | | C | | | | | | |
| | ReferentialConstraint | | | | | | | C | | | C | U | | | C | |
| | UniqueKey | | | | C | | | | | | | | | | | |
| | UniqueKeyRole | | | | C | | | | | | | | | | U | |
| | **Funktion $f_{aet}$** | | | | | | | | | | | | | | | |
| | AttributeName | | | • | (•) | (•) | | (•) | (•) | | (•) | (•) | | | | |
| | Computation | | | | | | • | | | | | | | | | |
| | DataType | • | | | | | | | | | | | | | | |
| | IdentifierRule | | | | | | | • | | | | | | | | |
| | MultiplicityRule | | | | | | | | | • | | | | | | |
| | Order | | | | • | • | | | | | | | | | | |
| | RollUpTypesToRule | | | | | | | | | | • | | | | | |
| | TableName | | • | | (•) | (•) | | (•) | | (•) | (•) | (•) | (•) | (•) | (•) | (•) |
| | ValidRule | | | | | | | • | | | | | | | | |

Tabelle 8.9: Transformationsschritte und erzeugte bzw. genutzte *LCD of SQL*–Objekte sowie genutzte Funktionen

# Kapitel 9

# Verfeinerung des Schemas

In diesem Kapitel wird der zweite Teilschritt des physischen Entwurfs vorgenommen (siehe Abbildung 9.1), indem das im letzten Kapitel erstellte *LCD of SQL*–Schema „umstrukturiert“ wird. Ziel dieser Umstrukturierung ist es, in Abhängigkeit vom Zielsystem ein für die Datenanalyse optimales Schema zu erhalten.

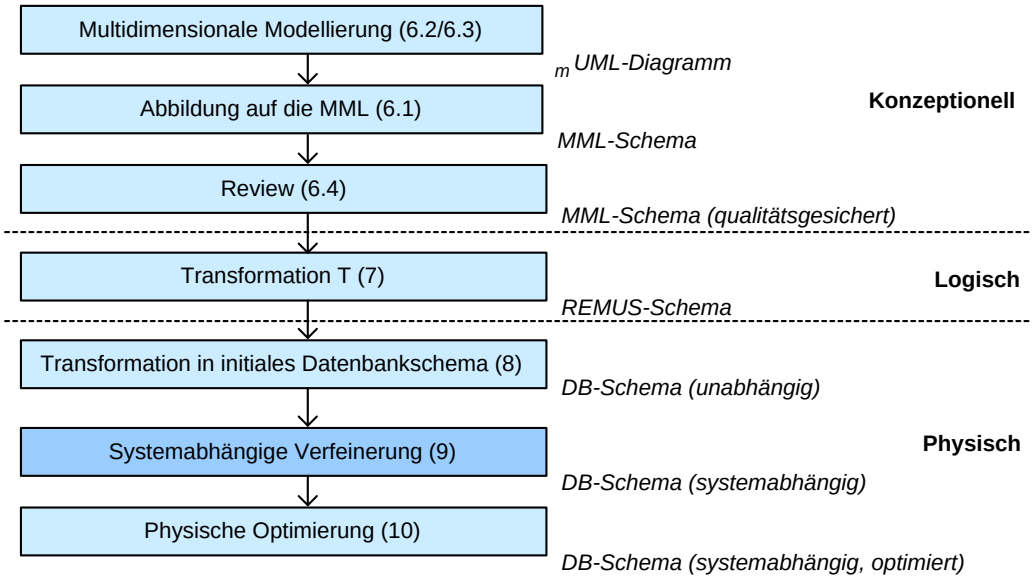



Abbildung 9.1: Einordnung des Schrittes in den Entwurfsprozess

Dazu werden zunächst in Abschnitt 9.1 Operatoren für ein *LCD of SQL*–Schema definiert, mit deren Hilfe Objekte und deren Attribute sowie Referenzen gelesen und verändert werden können. In Abschnitt 9.2 werden algorithmische Aspekte präsentiert, welche in Kombination mit den Elementaroperationen die Möglichkeit zur Bildung komplexer Operatoren bilden. Unter Verwendung von elementaren und komplexen Operatoren sowie den algorithmischen Elementen lassen sich schließlich Verfeinerungsalgorithmen definieren. Das im bisherigen Entwurfsprozess entstandene Schema ist vom Typ Schneeflocke mit Surrogaten (siehe auch Abschnitt 4.2). In Abschnitt 9.3 wird exemplarisch zunächst die Umformung in ein Sternschema mit Surrogaten und dann in eins ohne Surrogate vorgenommen und anhand des Beispiels „Handelswelt“ demonstriert. Die Wahl fiel auf den Typ Sternschema ohne Surrogate, weil das in der Evaluation (siehe Kapitel 12) eingesetzte DBMS bzw. OLAP–Server diesen Typ verlangt. Verfeinerungsalgorithmen für andere „gängige“ Schematypen (siehe hierzu auch Abschnitt 4.2) können in [Her01b] nachgelesen werden.

Zur Darstellung der Schemata soll in diesem Kapitel die in Abschnitt 4.2.1 eingeführte Notation dienen.

## 9.1 Elementare Verfeinerungsoperatoren

Die in diesem Abschnitt definierten Operatoren lassen sich in die Folgenden Kategorien unterteilen, denen im folgenden jeweils ein Unterabschnitt gewidmet ist:

- Operatoren zum Anlegen, Kopieren und Löschen von Objekten (Abschnitt 9.1.1),
- Operatoren zum Lesen und Verändern von Attributwerten (Abschnitt 9.1.2),
- Operatoren zum Hinzufügen und Löschen von Referenzen (Abschnitt 9.1.3).

### 9.1.1 Objekte anlegen und löschen

Zum Anlegen von Objekten sei für jede nicht–abstrakte Metaklasse des *LCD of SQL*–Metaklassendiagramms die in (9.1) beschriebene Methode definiert.

$$new < Classname > (name, < attribute\ list >, < reference\ list >). \tag{9.1}$$

Der Name ist dabei der eindeutige Bezeichner des Objektes, die Attribute und Referenzen werden in der Reihenfolge angegeben, wie in der Tupelschreibweise für *LCD of SQL*–Objekte in Abschnitt 8.3.2 auf den Seiten 178 bis 181 festgelegt.
Als Nebenbedingungen ist das Festlegen von nicht–optionalen Attributen zu definieren, ebenso müssen alle Referenzen eine gemäß der definierten Multiplizität zulässige Anzahl von Einträgen haben. Als Beispiel wird in (9.2) die Syntax zum Anlegen eines neuen *Column*–Objektes gezeigt.

$$\begin{aligned} newColumn(&name, initialValue, valueExpression, identityIncrement,\\ &optional, columnSet, keys, type, constraints, additivity). \end{aligned} \tag{9.2}$$

Die konkrete Anwendung zeigt (9.3): Eine neue Spalte „B“ mit den Eigenschaften Standardwert „0“ und nicht–optional, keine Berechnungsvorschrift und kein sich automatisch erhöhender Wert wird angelegt. Die Spalte wird der Tabelle „A“ zugeordnet, sie ist nicht Bestandteil eines Schlüssels, ihr Datentyp ist „Integer“ und es sind keine Constraints und keine Additivitäten für diese Spalte definiert.

$$newColumn(\text{"}B\text{"}, \text{"0"}, \text{NULL}, \text{"0"}, \text{FALSE}, \text{"}A\text{"}, \text{NULL}, \text{"Integer"}, \text{NULL}, \text{NULL}). \tag{9.3}$$

Neben dem Nennen referenzierter Objekte über ihren Namen existiert für jeden Operator eine analoge Version, in der die Objekte über ihre Referenz angesprochen werden (als Beispiel siehe (9.4)).

Sei $RefToCons$ eine Referenz auf die Spalte "B", $RefToTab$ eine Referenz auf die Tabelle "A".
$newColumn(RefToCons, \text{"0"}, \text{NULL}, \text{"0"}, \text{FALSE}, RefToTab, \text{"Integer"}, \text{NULL}, \text{NULL})$. (9.4)

Eine Spezialform des *new*–Operators ist der in (9.5) definierte *copy*–Operator, der ein neues Objekt durch Kopieren eines bestehenden Objektes anlegt. Dabei werden alle Attributwerte und Referenzen übernommen.

$$copy < Classname > (name_1, name_2). \tag{9.5}$$

Beispielsweise erzeugt der Operatorenaufruf in (9.6) eine Spalte „C“, die genau wie „B“ zur Tabelle „A“ gehört und auch sonst die gleichen Eigenschaften wie Datentyp und Standardwert besitzt.

$$copyColumn(\text{"}C\text{"}, \text{"}B\text{"}). \tag{9.6}$$

Als inverse Operation zum Anlegen gibt es den *delete*–Operator zum Löschen von Objekten, der die folgende Syntax hat:

$$delete < Classname > (name). \tag{9.7}$$

Inwieweit die Ausführung des *delete*–Operators das Löschen aller referenzierten Objekte impliziert, zeigt Tabelle 9.1. Ebenso gibt die Tabelle für jeden Objekttyp an, bei Existenz welcher referenzierten Objekte Instanzen dieses Typs nicht gelöscht werden können.

| **Löschen von Objekten der Metaklasse ...** | **... bewirkt Löschen von referenzierten Objekten der Metaklassen ...** | **... kann nicht gelöscht werden, wenn Referenzen vorhanden sind zur Metaklasse ...** |
|---|---|---|
| AdditivityMETA | — | — |
| Column | AdditivityMETA, ColumnConstraint, MappingMETA | UniqueKey, ForeignKey |
| ColumnConstraint | — | — |
| ColumnType | — | Column |
| DBConstraint | — | — |
| ForeignKey | ForeignKeyRole, Columns | — |
| ForeignKeyRole | ForeignKey | |
| MappingMETA | — | — |
| ReferentialConstraint | ForeignKeyRole | — |
| Schema | DBConstraints, ReferentialConstraints, Schema, Table | — |
| Table | Column, ForeignKey, TableConstraint, UniqueKey | — |
| TableConstraint | — | — |
| UniqueKey | UniqueKeyRole | — |
| UniqueKeyRole | UniqueKey | — |

Tabelle 9.1: „Löschweitergabeverhalten“ der Objekte im *LCD of SQL*–Schema

Die in (9.8) dargestellte Operation löscht die Spalte „B“ und löscht gleichzeitig ihre Referenz auf die zugehörige Tabelle. Implizit mitgelöscht werden (siehe Zeile mit Eintrag *Column* in Tabelle 9.1) möglicherweise vorhandene *AdditivityMETA*–, *ColumnConstraint*– und *MappingMETA*–Objekte. Die Löschoperation würde abgewiesen werden, falls Spalte „B“ an einer Primär– oder Fremdschlüsselbildung beteiligt wäre, d. h. entsprechende Objekte referenziert würden.

$$deleteColumn(\text{"}B\text{"}). \tag{9.8}$$

### 9.1.2 Attribute lesen und verändern

Mit den beiden in (9.9) eingeführten Operatoren lassen sich Attributwerte auslesen bzw. verändern.

$$\begin{aligned} &get < attribute\ name > (< object\ name >) \\ &set < attribute\ name > (< object\ name >, < value >). \end{aligned} \tag{9.9}$$

Während der *get*–Operator den Attributwert des angegebenen Objektes liefert, setzt der *set*–Operator entsprechend den Wert des Attributs für das angegebene Objekt auf den als zweiten Parameter gegebenen Wert. Beispiele:

$$\begin{aligned} &getInitialValue(\text{"}B\text{"}) \\ &getBody(\text{"}Constraint\ for\ column\ B.\text{"}) \\ &setValue(\text{"}B\text{"}, 5) \\ &setBody(\text{"}Constraint\ for\ column\ B.\text{"}, \text{"} \geq 0\text{"}). \end{aligned} \tag{9.10}$$

Solche Methoden sind für alle im *LCD of SQL*–Metamodell festgelegten Objekt–Attribut–Kombinationen definiert.

### 9.1.3 Referenzen lesen und verändern

Analog zu den Operationen auf Attributen im letzten Abschnitt sind auch auf den Referenzen von Objekten Lese– und Manipulationsoperatoren definiert. Der lesende Zugriff wird durch den in (9.11) definierten *get*–Operator festgelegt.

$$get < reference\ name > (< set\ of\ object\ names > [, < condition >]). \tag{9.11}$$

Als Argument dient dabei eine Menge von Objekten. Dadurch ist es möglich, mehrere *get<reference name>*–Operatoren zu verknüpfen und auf diese Weise die Beziehungen im Metaklassendiagramm entlang zu navigieren. Als optionales Argument haben *get<reference name>*–Anweisungen eine Bedingung in Form eines Booleschen Ausdrucks über die Variablen der zurückgelieferten Metaklasse. Resultat ist bei Angabe einer Bedingung nur die Teilmenge, die der Bedingung genügt. *get<reference name>*–Operatoren sind für alle im *LCD of SQL*–Metamodell definierten Referenzen festgelegt. Im Unterschied zu der gleichlautenden Operation auf Attributen ist hier jedoch das Resultat mengenwertig, so liefert das erste Beispiel in (9.12) die Menge der Tabellen des Schemas „S“. Bei Beziehungen im Metamodell mit der Multiplizität „1“ oder „0..1“ wird das Ergebnis als 1–elementige Menge aufgefasst. Das zweite Beispiel in (9.12) liefert den Datentyp der Spalte „B“, der immer eindeutig ist. Das dritte Beispiel zeigt eine Verknüpfung von *get*–Operatoren, die durch das Metaklassendiagramm navigieren und alle im Schema „S“ verwendeten Datentypen ermittelt. Das vierte und letzte Beispiel zeigt einen *get<reference name>*–Operator mit Bedingung, der alle Fakttabellen des Schemas „S“ ermittelt.

$$\begin{aligned} &getTables(\text{"}S\text{"}) \\ &getType(\text{"}B\text{"}) \\ &getType(getColumns(getTables(\text{"}S\text{"}))) \\ &getTables(\text{"}S\text{"}, tableTypeMETA = \text{"}FACT\text{"}). \end{aligned} \tag{9.12}$$

Weiterhin ist für *get<reference name>*–Operatoren ein $^*$–Operator definiert, der den transitiven Abschluss berechnet. Voraussetzung ist hierbei, dass der Typ des Arguments gleich dem des Resultats ist.

$$get < reference\ name > (< set\ of\ object\ names > [, < condition >])^*. \tag{9.13}$$

Um die Referenzen zu manipulieren, werden die in (9.14) gezeigten *add*– und *minus*–Operatoren verwendet. Sie fügen der Referenz eines Objektes ein Element hinzu oder entfernen es.

$$\begin{aligned} &add < reference\ name > (< object\ name >, < value >) \\ &minus < reference\ name > (< object\ name >, < value >). \end{aligned} \tag{9.14}$$

Die Beispiele in (9.15) fügen dem Schema „S“ die Tabelle „A“ hinzu und entfernen die Tabelle „D“.

$$\begin{aligned} &addTables(\text{"}S\text{"}, \text{"}A\text{"}) \\ &minusTables(\text{"}S\text{"}, \text{"}D\text{"}). \end{aligned} \tag{9.15}$$

## 9.2 Komplexe Operatoren

### 9.2.1 Algorithmische Elemente

Aufbauend auf den im letzten Abschnitt definierten elementaren Operatoren lassen sich komplexe Operatoren und Algorithmen formulieren, die ein Schema den Anforderungen des verwendeten DBMS bzw. OLAP–Servers anpassen. Zur Formulierung komplexer Operatoren und Algorithmen im nächsten Abschnitt ist eine Pseudocodesprache definiert, die

- Variablen der Typen *Boolean* und *Integer* kennt.
- *Cursorobjekte* als Variablen verwendet, um mengenwertige Resultate handhaben zu können; auf den Cursorobjekten seien die navigierenden Operationen *MoveFirst, MoveNext, MovePrevious* und *MoveLast* sowie eine die Anzahl der Cursorelemente liefernde Funktion *Count* definiert.
- eine Reihe durch Semikolons getrennte Anweisungen als Sequenz interpretiert.
- als Strukturierungskommandos *while*– und *for*–Schleifen sowie *if–then–else*–Anweisungen kennt.
- ein *forall*–Konstrukt für das Navigieren auf einer Menge benutzt.
- *Prozeduren* mit Variablen als Übergabe– sowie Rückgabeparameter kennt, um Strukturierung und Rekursion zu ermöglichen.

### 9.2.2 Beispiel: Primärschlüssel ändern

In diesem Abschnitt wird unter Verwendung der elementaren Operatoren und der Kontrollstrukturen ein komplexer Operator für das Ändern des Primärschlüssels einer Tabelle definiert. Die Syntax dieses Operators wird definiert durch:

$$changePrimaryKey(< table\ name >, < attribute\ list >). \tag{9.16}$$

Bei Aufruf von *changePrimaryKey* wird ein neuer Eindeutigkeitsschlüssel über die angegebenen Attribute angelegt und als Primärschlüssel der angegebenen Tabelle markiert. Der neue Primärschlüssel erhält die Rolle des alten, alle Fremdschlüsseleinträge werden aktualisiert. Die alten Fremdschlüsselspalten werden gelöscht. Der Effekt des Operators *ChangePrimaryKey* ist in Abbildung 9.2 zu sehen, seine algorithmische Definition in Algorithmus (9.1) angegeben.

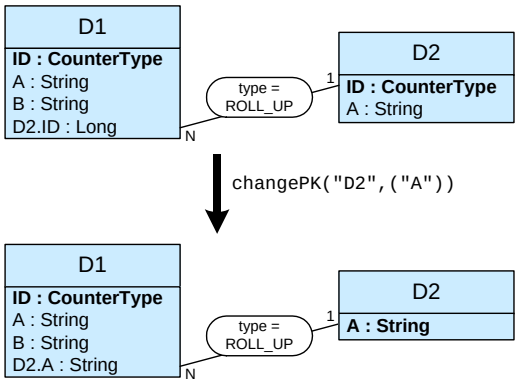



Abbildung 9.2: Funktionsweise des komplexen Operators *changePrimaryKey*

```
( 1)    procedure changePrimaryKey(t,(a1, ..., an))
( 2)        oldPrimaryKey:=GetUniqueKey(t,"isPrimary=TRUE");
( 3)        oldRole:=GetRole(oldPrimaryKey);
( 4)        newPrimaryKey:=createUniqueKey(ψ("Primary key of
( 5)            table ",t.name),TRUE,t,(a1, ..., an),oldRole);
( 6)        for i:=1 to n
( 7)          addKeys(ai,newPrimaryKey);
( 8)        endfor;
( 9)        F:=getForeignKey(getForeignKeyRole
(10)            (getReferentialConstraint
(11)                (getUniqueKeyRole(oldPrimaryKey))));
(12)        forall f in F
(13)          table:=getTable(f);
(14)            C:=getColumns(f);
(15)            forall c in C
(16)              deleteColumn(c);
(17)            endfor;
(18)            for i:=1 to n
(19)              newFKColumn:=createColumn(ψ(t.name, ai),NULL,NULL,
(20)                0,FALSE,table,ai.type,NULL,NULL);
(21)              addForeignKey(newFKColumn,newPrimaryKey,i);
(22)            endfor;
(23)        endfor;
(24)        setName(oldPrimaryKey,ψ("Unique key of table ",t.name));
(25)        setIsPrimary(oldPrimaryKey,FALSE);
(26)    end;
```

Algorithmus 9.1: Algorithmus *changePrimaryKey*

In den Zeilen 2 und 3 werden Verweise auf den alten Primärschlüssel und die alte Rolle vorgenommen. In den Zeilen 4 und 5 wird das neue *UniqueKey*–Objekt angelegt und als Primärschlüssel gekennzeichnet. Die in den Zeilen 6 bis 8 folgende *for*–Schleife setzt die *keys*–Referenz der angegebenen Attribute auf den neuen Schlüssel. In den Zeilen 9 bis 11 werden alle Fremdschlüsseleinträge des alten Primärschlüssels in einen Cursor übertragen, der dann in den Zeilen 12 bis 23 durchlaufen wird, wobei zunächst in jeder Tabelle die alten Fremdschlüsselspalten in den Zeilen 15 bis 17 gelöscht werden. Anschließend wird jede referenzierte Tabelle um die Attribute des neuen Schlüssels erweitert und diese Attributkombination als Fremdschlüssel gekennzeichnet. In den Zeilen 24 und 25

wird der alte Primärschlüssel umbenannt und die Primärschlüsseleigenschaft auf „FALSE“ gesetzt, so dass dieser Schlüssel als „normaler“ Eindeutigkeitsschlüssel erhalten bleibt.

## 9.3 Verfeinerungsalgorithmen

Dieser Abschnitt widmet sich Verfeinerungsalgorithmen, die auf die elementaren und komplexen Operatoren sowie algorithmischen Aspekte aufbauen. Weil die Demonstration der Algorithmen anhand des Beispiels „Handelswelt“ erfolgt, wird im Folgenden Abschnitt 9.3.1 zunächst das Resultat der bisherigen Transformation präsentiert, bevor das Schema in Abschnitt 9.3.2 zunächst in ein Sternschema mit Surrogaten und dann in Abschnitt 9.3.3 in ein Sternschema ohne Surrogate überführt wird.

### 9.3.1 Resultat der bisherigen Transformation

Das durch die bisherige Transformation erzielte Resultat ist das in Abbildung 9.3 dargestellte Schneeflockenschema mit Surrogaten. Die dunkleren Tabellen sind die Fakttabellen, die helleren die dimensionalen Tabellen, von denen jede eine Hierarchieebene repräsentiert. Die weisse Tabelle realisiert die aufgelöste Assoziation zwischen „Ort des Verkaufs“ und „Artikel“. Zur besseren Übersicht sind die eine Dimensionshierarchie bildenden dimensionalen Tabellen jeweils mit einer gestrichelten Linie umrandet.
An verschiedenen Stellen im Schema erkennt man folgende multidimensionale Sachverhalte:

- Von „Artikel“ über „Produktgruppe“ und „Produktfamilie“ zur „Produktkategorie“ ist ein „normaler“ Verdichtungspfad zu sehen.
- Die mit der dimensionalen Tabelle „Tag“ beginnende Zeithierarchie ist eine multiple Hierarchie, denn von „Tag“ aus kann man sowohl zu „Woche“ als auch zu „Monat“ verdichten.
- Weiterhin ist in dieser Hierarchie zu erkennen, dass zwischen Woche und Jahr keine Beziehung besteht, was dem nicht nicht direkten Abbilden des *SharedRollUp* entspricht.
- Ausgehend vom „Ort des Verkaufs“ beginnt die relativ komplexe Ortshierarchie, die multiple Hierarchiepfade besitzt: Einer über „Filialkategorie“ zur „Filialoberkategorie“, der andere von „Stadt“ bis „Staat“. Dieser letztgenannte Hierarchiepfad ist zudem ein alternativer Verdichtungspfad, denn bei einem RollUp von „Stadt“ kann man immer wieder zu „Region“ gelangen.
- Ebenfalls zu erkennen ist die gemeinsame Nutzung von dimensionalen Klassen durch verschiedene Faktklassen. Dieses kann sowohl ganze Hierarchien, z. B. wird die mit „Artikel“ beginnende Hierarchie von „Verkaufszahl“ und „Verkauftes_Produkt“ genutzt, oder aber Teile davon betreffen, z. B. nutzen „Verkauf“ und „Verkaufszahl“ die gesamte Zeithierarchie während „Einkommen“ erst bei „Quartal“ einsteigt.
- Die einzige weiße Tabelle im Schema repräsentiert die Assoziation zwischen „Artikel“ und „Ort des Verkaufs“, hier werden die an einem Verkaufsort geführten Artikel festgehalten.

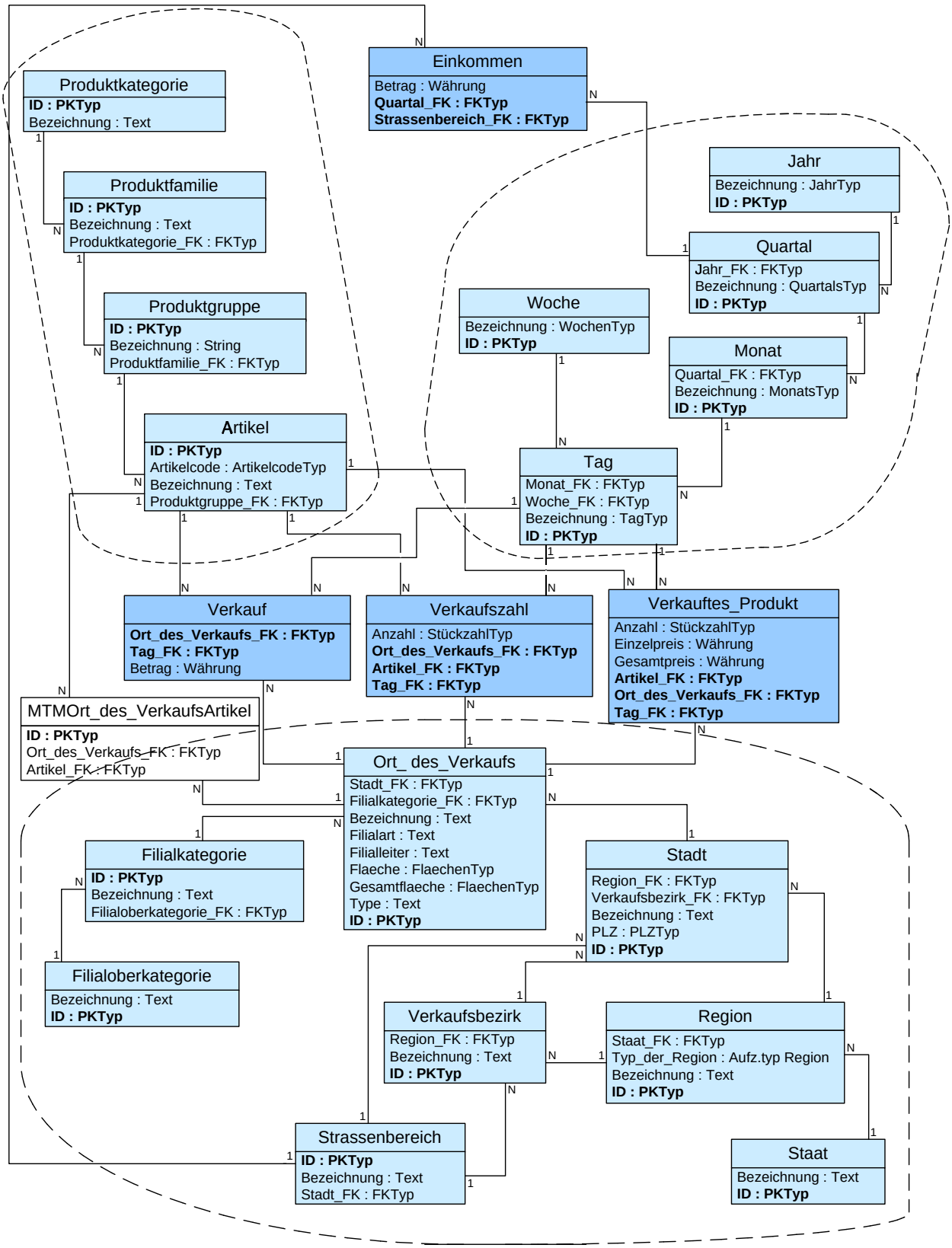



Abbildung 9.3: Handelswelt: Schneeflockenschema mit Surrogaten

Daneben sind zum Verständnis der Funktionsweise der nachfolgenden Algorithmen einige Metadaten relevant, die aus Gründen der Übersichtlichkeit nicht in Abbildung 9.3 zu sehen sind, sondern in den folgenden Abbildungen 9.4 bis 9.7 dargestellt sind. Abbildung 9.4 zeigt die einschränkenden Gruppierungsoperatoren der beiden Attribute „Einzelpreis“ und „Gesamtpreis“ bez. der Ortsdimension.

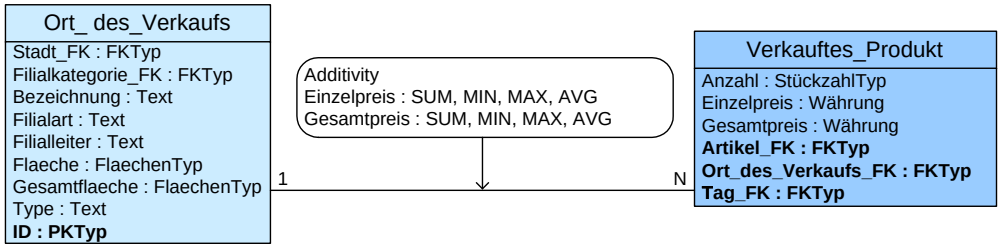



Abbildung 9.4: Handelswelt: Additivität

Die ursprünglich als SharedRollUp modellierte Verdichtung von „Woche“ zu „Jahr“ ist im Schema durch das in Abbildung 9.5 dargestellte Metadatum dokumentiert.

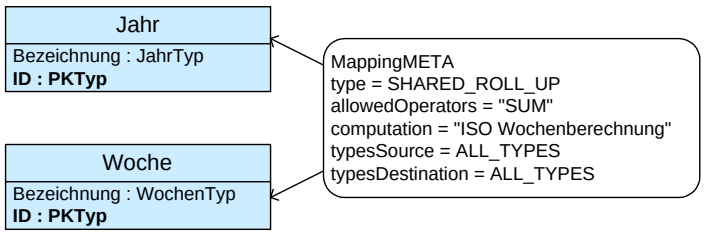



Abbildung 9.5: Handelswelt: SharedRollUp

Die Tabelle „Ort_des_Verkaufs“ hat durch Vererbungsauflösung und die bedingte Hierarchiebildung zur Filialkategorie (nur Datensätze vom Typ „Filiale“ können hieran teilnehmen) eine Reihe von Spalten– und Tabellenconstraints, die in Abbildung 9.6 dargestellt sind. Ebenso ist das *Referential-Constraint* „RC04“ (siehe Anhang A.3), das die bedingte Hierarchiebildung dokumentiert, in dieser Abbildung zu sehen.

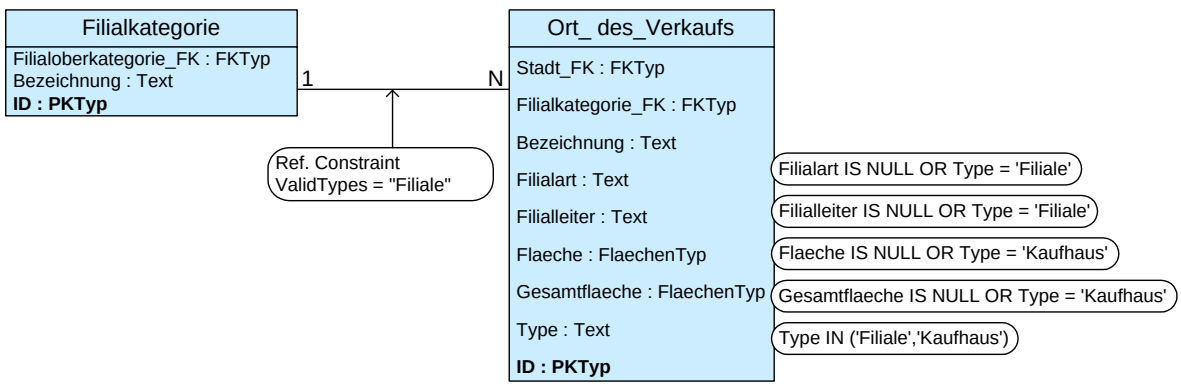



Abbildung 9.6: Handelswelt: Spalten– und Tabellenconstraints

Weiterhin besitzen die dimensionalen Tabellen jeweils einen konzeptionellen Schlüssel: In der Tabelle „Artikel“ ist es das Attribut „Artikelcode“, in „Stadt“ das Attribut „PLZ“ und in allen anderen Tabellen jeweils das Attribut mit dem Namen „Bezeichnung“. Abbildung 9.7 zeigt die entsprechende Notation.

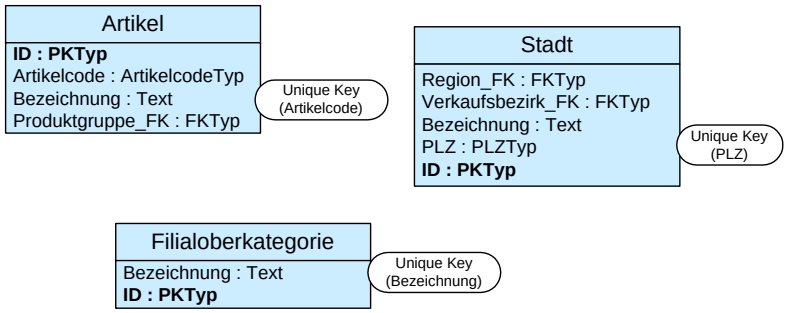



Abbildung 9.7: Handelswelt: Konzeptionelle Schlüssel

### 9.3.2 Sternschema mit Surrogaten

Das in diesem Abschnitt vorgestellte Schema ist ein Sternschema mit Surrogaten, d. h. im Unterschied zum Ausgangsschema aus dem letzten Abschnitt sind alle Tabellen innerhalb einer Dimensionen zu einer Tabelle denormalisiert. Die Verbindung zwischen einer solchen dimensionalen Tabelle und einer Fakttabelle wird über ein Surrogat realisiert. Zur Umformung eines Schneeflockenschemas mit Surrogaten in ein Sternschema mit Surrogaten dient der Algorithmus (9.2):

```
( 1)   procedure createStarWithID(S)
( 2)       T_Root :=getTables(S,tableTypeMETA="DIMENSION" ∧
( 3)            getConstraints(getRole(:=getRole(getUniqueKeys
( 4)            (t,isPrimary="TRUE"))),typeMETA="DIMENSION") ≠ {});
( 5)       forall t in T_Root
( 6)          T:=getTable(getUniqueKey(getUniqueKeyRole(
( 7)           getConstraint(getRole(getForeignKey(t))))))*;
( 8)          forall u in T
( 9)             C:=getColumns(u,name≠"ID");
(10)                forall c in C
(11)                   setName(c,f_detAttributeName(c));
(12)                   setColumnSet(c,t);
(13)                   if "Eingeschraenktes RollUp von t nach u" then
(14)                      setTableConstraint(t,f_detTableConstraint);
(15)                   end;
(16)                   if "Mind. ein nicht vollst. RollUp von t nach u" then
(17)                      setOptional(c, TRUE);
(18)                   end;
(19)                endfor;
(20)          endfor;
(21)          deleteTable(u);
(22)       endfor;
(23)   end;
```

Algorithmus 9.2: Algorithmus *createStarWithID*

Algorithmus (9.2) arbeitet wie folgt: In den Zeilen 2 bis 4 werden alle dimensionalen Tabellen ermittelt, die mit Fakttabellen verbunden sind, d. h. die feingranularste Hierarchieebene darstellen. In der *for*–Schleife von Zeile 5 bis 22 werden diese Tabellen abgearbeitet, indem zunächst in den Zeilen 6 und 7 alle Tabellen der Dimension ermittelt werden. Dann wird zu jeder Tabelle die Attributmenge ermittelt (Zeile 9) und in der innersten *for*–Schleife die Wurzeltabelle der Dimension um alle Attribute bis auf das Surrogat erweitert. Den Attributnamen bestimmt die deterministische Funktion $f_{det_{AttributeName}}$. Eine Möglichkeit ihrer Definition wäre jedem Attribut als Präfix den Tabellennamen zu geben. Hierbei bleiben Datentyp (Referenz der Metaklasse *Column* zur Metaklasse *ColumnType*) und Constraints (Referenz der Metaklasse *Column* zur Metaklasse *ColumnConstraint*) sowie eventuell vorhandene zwischendimensionale Abbildungen (Referenz der Metaklasse *Column* zur Metaklasse *MappingMETA*) erhalten. Neu hinzukommen können in den Zeilen 13 bis 15 zusätzliche Tabbellenconstraints, falls auf dem Pfad von der Wurzeltabelle der Dimension zur aktuellen Tabelle ein *RollUp* existiert, das nur für bestimmte Typen zulässig ist. In diesem Falle dürfen die durch die Denormalisierung hinzukommenden Attribute auch nur für diese Typen gültig sein. In der *if*–Anweisung in den Zeilen 16 bis 18 wird schließlich überprüft, ob auf dem Pfad von der Wurzelklasse der Dimension zur aktuellen Klasse eine nicht vollständige Verdichtung existiert. Wenn dies der Fall ist, dann wird die neu angelegte Spalte als optional markiert. Nicht erhalten bleiben dürfen die Schlüssel, denn durch die Denormalisierung gelten sie nicht mehr. Daher werden sie auch beim Löschen der alten Tabellen in Zeile 21 durch die in Tabelle 9.1 festgelegten Löschweitergaben entfernt. Ebenso mitgelöscht werden evtl. existierende *AdditivityMETA*–Objekte.

Angewendet auf das Ausgangsschema des Beispiels „Handelswelt“ in Abbildung 9.3 erzeugt Algorithmus (9.2) das in Abbildung 9.8 dargestellte Schema: Jede Dimension wird durch eine Tabelle repräsentiert, die einen künstlichen Primärschlüssel besitzt. Beim Denormalisieren der multiplen Hierarchie auf der Ortsdimension entstehen neue *TableConstraint*–Objekte, die ebenfalls in der Abbildung zu sehen sind.
Die Metadaten aus den Abbildungen 9.4 bis 9.7 werden im Zuge des Algorithmus folgendermaßen verändert: Die Additivität bleibt erhalten, ebenso die beiden bei den Attributen definierten *ColumnConstraint*–Objekte. Das in Abbildung 9.6 links abgebildete *ReferentialConstraint*–Objekt wird in das Tabellenconstraint „Filialkategorie.Bezeichnung IS NULL OR Type = ”Filiale”“ verwandelt, denn das Eintragen von Filialkategoriewerten ist nur für Datensätze vom Typ „Filiale“ möglich. Von den konzeptionellen Schlüsseln aus Abbildung 9.8 bleibt nur derjenige der Tabelle „Artikel“ erhalten, die beiden anderen werden in Zeile 21 implizit gelöscht.

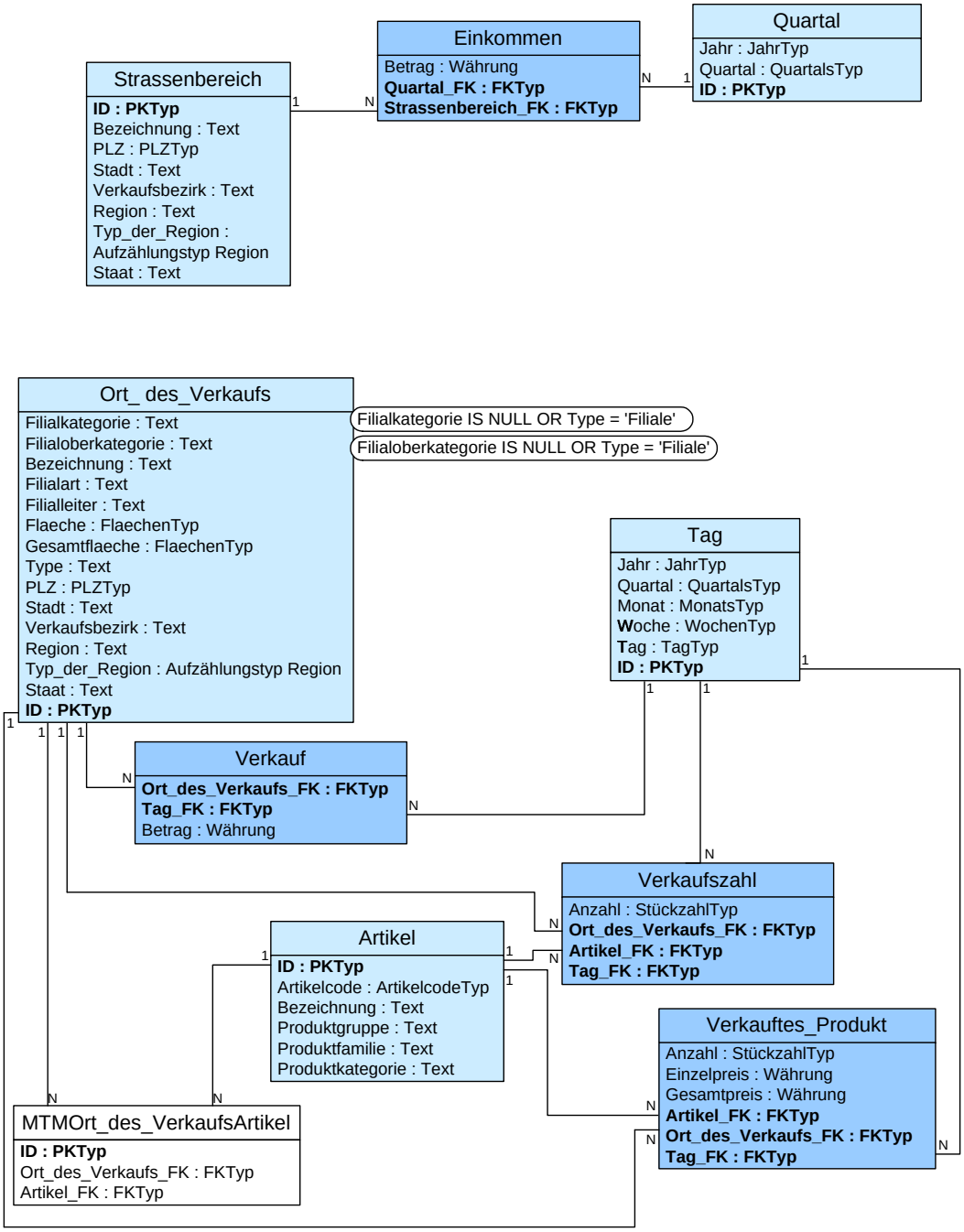



Abbildung 9.8: Handelswelt: Sternschema mit Surrogaten

### 9.3.3 Sternschema ohne Surrogate

Das Sternschema ohne Surrogate ist dadurch gekennzeichnet, dass für jede Dimension eine Tabelle existiert, die referentielle Integrität zwischen Fakttabellen und dimensionalen Tabellen jedoch über den konzeptionellen Schlüssel der dimensionalen Tabellen realisiert ist. Algorithmus (9.3) definiert die Transformation in ein solches Schema.

```
( 1)    procedure createStarWithoutID(S)
( 2)          call createStarWithID(S);
( 3)          T:=getTables(S,tableTypeMETA="DIMENSION");
( 4)          forall t in T
( 5)             call changePrimaryKey(t,f_detAttributeSet(t));
( 6)          endfor;
( 7)    end;
```

Algorithmus 9.3: Algorithmus *createStarWithoutID*

Der Algorithmus nutzt zunächst die Prozedur *createStarWithID* aus dem letzten Abschnitt zur Denormalisierung der Dimensionen, so dass nach Zeile 2 das Resultat des letzten Abschnittes vorliegt. Nun müssen die Primär–Fremdschlüssel–Beziehungen zu den Fakttabellen abgeändert werden, was in der *for*–Schleife in den Zeilen 4 bis 6 unter Benutzung der in (9.16) definierten und in Algorithmus (9.1) spezifizierten Funktion *changePrimaryKey* geschieht. $f_{det_{AttributeSet}}$ ist dabei eine deterministische Funktion, die für eine Tabelle die Attribute des konzeptionellen Schlüssels ermittelt. Aus dem Schema in Abbildung 9.8 wird durch Entfernen der künstlichen Primärschlüssel das Schema in Abbildung 9.9 erreicht.

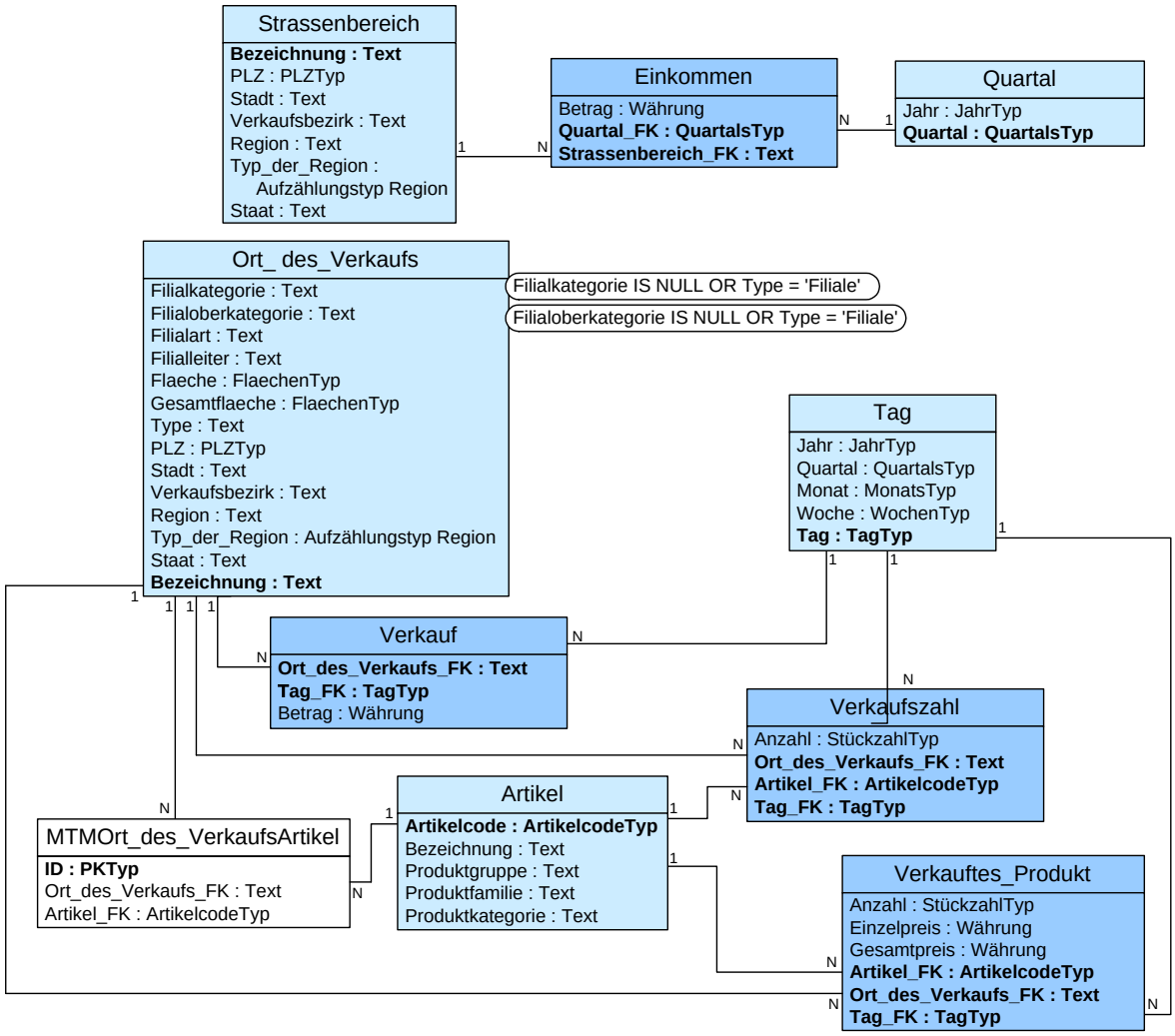



Abbildung 9.9: Handelswelt: Sternschema ohne Surrogate

## 9.4 Zusammenfassung

Kapitel 9 hat sich mit der Verfeinerung von *LCD of SQL*–Schemata befasst, wobei unter Verfeinerung die Anpassung an die speziellen Anforderungen des verwendeten DBMS bzw. OLAP–Servers zu verstehen ist. Um dieses Ziel zu erreichen, wurden in Abschnitt 9.1 einige elementare Operatoren auf einem *LCD of SQL*–Schema definiert. Darauf aufbauend wurden in Kapitel 9.2 algorithmische Elemente und komplexe Verfeinerungsoperatoren definiert. Kapitel 9.3 gibt konkrete Algorithmen an, die (über den Umweg eines Sternschemas mit Surrogaten) den in der Praxis relevanten Schematyp Sternschema ohne Surrogate definieren. Die Anwendung dieser Algorithmen wird anhand des Beispiels „Handelswelt“ demonstriert.
Damit liegt zu diesem Zeitpunkt der Entwicklung ein physisches Datenbankschema vor, das im Hinblick auf ein konkretes Zielsystem angepasst worden ist. Der bisher unbeachtete Aspekt von Optimierungsmöglichkeiten, z. B. in Form von Materialisierungen zur Beschleunigung der Anfrageverarbeitung während der Datenanalyse, wird im kommenden Kapitel behandelt.

# Kapitel 10

# Physische Datenbankoptimierung

In diesem Kapitel wird ein Framework für die physische Datenbankoptimierung beschrieben, dessen Anwendung den letzten Schritt des Entwurfsprozesses bildet (siehe Abbildung 10.1).

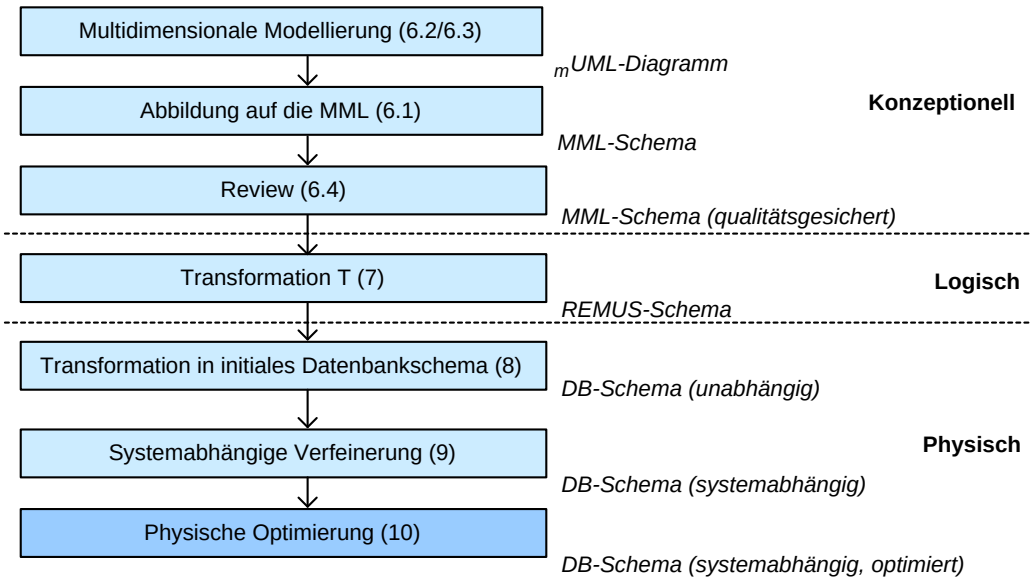



Abbildung 10.1: Einordnung des Schrittes in den Entwurfsprozess

Zunächst werden in Abschnitt 10.1 wesentliche Entwurfskriterien für die physische Datenbankoptimierung genannt und ein Überblick über deren Ablauf gegeben. In Abschnitt 10.2 wird ein Metamodell als formale Beschreibung der statischen Aspekte vorgeschlagen, bevor in Abschnitt 10.3 das Optimierungsproblem formalisiert wird. Darauf aufbauend wird mit einem Ausschnitt des Beispiels „Handelswelt“ in Abschnitt 10.4 fortgesetzt, wobei zunächst in 10.4.1 die Konfiguration vorgenommen wird und dann in 10.4.2 zur Verdeutlichung zwei Beispiele berechnet werden. Abgeschlossen wird das Kapitel mit einer Zusammenfassung in Abschnitt 10.5.

## 10.1 Überblick und Ablauf

In Abschnitt 4.5 wurde das Fazit gezogen, dass eine Vielzahl von Arbeiten zu Optimierungsmöglichkeiten und –verfahren existiert, die meisten jedoch eine Möglichkeit oder ein Verfahren nur isoliert

(evtl. noch auf einen bestimmten Kontext eingeschränkt, aber nicht hinreichend breit und abstrahierend) betrachten. Darauf aufbauend und zusätzlich durch die in Abschnitt 5.3.3 vorgestellten präemptiven Ansätze geprägt, lassen sich folgende Anforderungen an die physische Optimierung im DWH–Entwurfsprozess festhalten, die in die Konzeption des Framework eingeflossen sind:

- Unterschiedliche Optimierungsmaßnahmen sollten gleichzeitig betrachtet werden.
- „Umweltparameter“ (z. B. zeitliche Randbedingungen) sollten individuell (z. B. pro Projekt oder Organisation) konfigurierbar sein.
- Die Auswahl der Optimierungsmaßnahmen sollte einem nachvollziehbaren Prozess unterliegen, der idealerweise mit Werkzeugunterstützung durchgeführt wird.
- Insbesondere sollte auch nachvollzogen werden können, wie konkurrierende Anforderungen oder Zielsetzungen (z. B. schnelle Anfrageverarbeitung vs. Speicherplatzminimierung) behandelt werden.
- Der gesamte Prozess sollte allerdings nicht vollautomatisch durchgeführt werden, sondern an definierten Punkten durch den Entwickler gezielt beeinflusst werden können, um hierdurch Wissen über die Domäne oder das konkrete Projekt einzubringen, die nicht im System modelliert worden sind.
- Der Fortschritt des gesamten Vorgangs des physischen Entwurfs sollte zur Dokumentation und Nachvollziehbarkeit in einem Repository abgespeichert werden.

Dem Framework für den physischen Datenbankentwurf dienen als grobe Basis die in Abschnitt 5.3.3 skizzierten Ansätze, insbesondere [RS91], so dass sich der in Abbildung 10.2 skizzierte Ablauf ergibt. In einer ersten *Konfigurationsphase* legt der Entwickler folgende Eingaben fest, die beim initialen Ausführen dieses Entwurfsschrittes seinem Wissen über Projekt, Domäne und Zielsystem entstammen:

- Als statischen Aspekt des DWH ein *annotiertes Schema*, d. h. ein um Informationen über die Extension wie Volumen oder Zu– und Abnahmeraten angereichertes Schema.
- Als dynamischen Aspekt des DWH einen *Workload*, der aus einer Menge von gewichteten Aufgaben besteht, die auf dem DWH ausgeführt werden. Aufgrund des Einsatzes in der Datenanalyse sind diese Aufgaben vorrangig lesende Anfragen, es kann sich aber auch um Nachladeoperationen von Daten aus dem Back End–Bereich in das DWH oder Löschoperationen im DWH beim Archivieren handeln.
- Eine Menge von *Regeln* legt fest, welche Optimierungsmaßnahme unter welchen Bedingungen „sinnvoll“ ist. Beispiele einfacher Regeln sind „*Wenn* ein Attribut häufig selektiv benutzt wird und eine geringe Kardinalität besitzt, *dann* ist ein Bitmap–Index sinnvoll.“ oder „*Wenn* Daten auf einer bestimmten Verdichtungsstufe häufig angefragt werden, *dann* ist eine Materialisierung sinnvoll.“
  In die Regeln fließt das Wissen der Entwickler ein, das ihrem Erfahrungsschatz entstammt oder Empfehlungen des DBMS–Herstellers entnommen worden sein kann. Das Formulieren der Regeln umfasst auch die Sicherstellung von Vollständigkeit und Widerspruchsfreiheit.

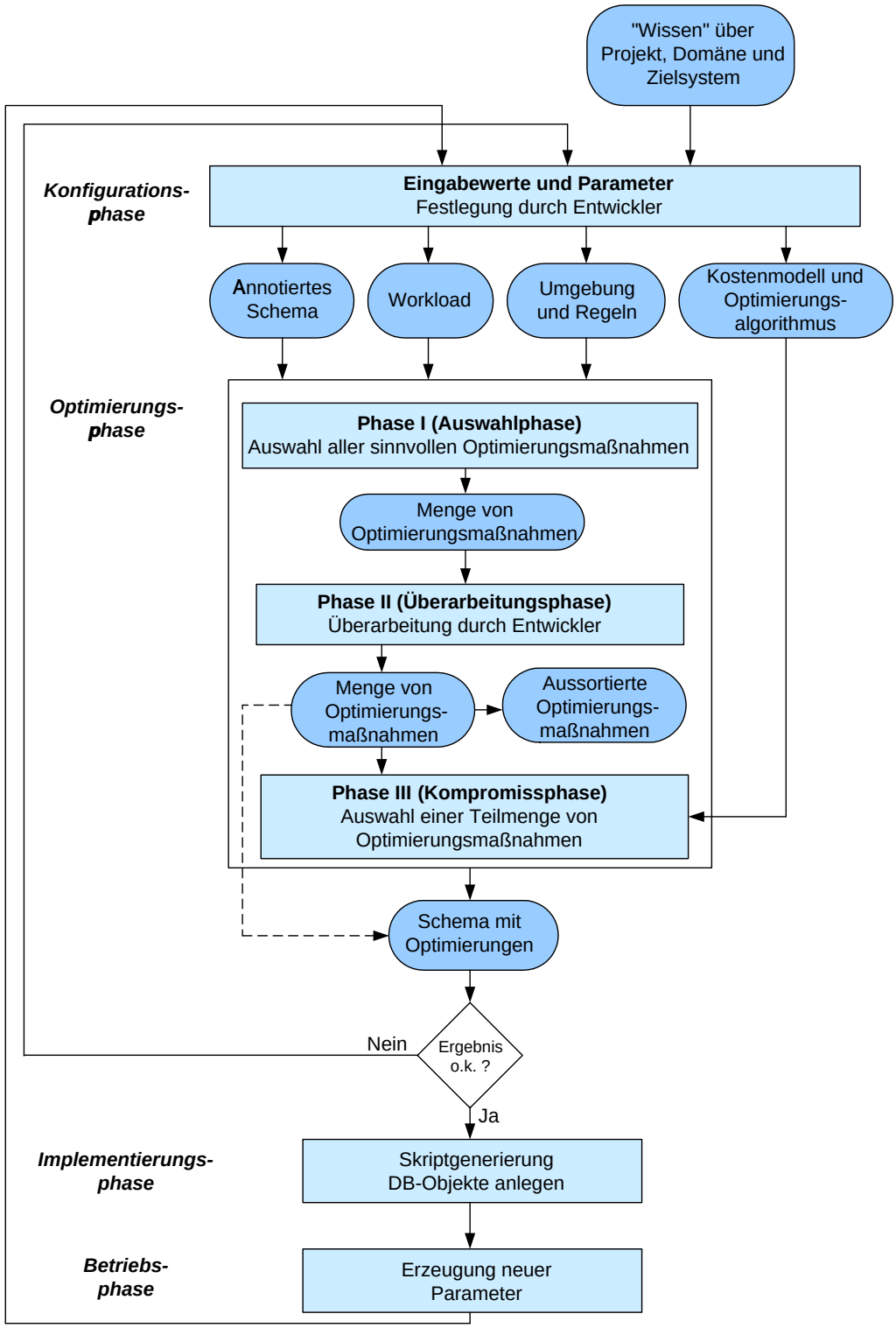



Abbildung 10.2: Ablauf der physischen Datenbankoptimierung

- Angaben über die *Umgebung* charakterisieren das System, z. B. DBMS oder OLAP–Server, auf dem implementiert werden soll. An dieser Stelle muss der Entwickler entscheiden, welche Informationen er als Umgebungsparameter in den Prozess einfließen lassen will und welche Informationen das Resultat des Optimierungsprozesses bilden sollen. Als Hilfe dient hierbei die Klassifikation unterschiedlicher, aufeinander aufbauender Aspekte der Konfiguration und Optimierung von Datenbanken in Abbildung 10.3. Als Basis steht die Auswahl von Hardware (HW), Betriebssystem (BS) und DBMS. Darauf aufbauend erfolgt die Konfiguration dieser Komponenten, wobei Aspekte wie etwa parallele Optionen eine Rolle spielen. Als nächster Schritt erfolgt in der zweithöchsten Ebene die Konfiguration der Datenbank. Hierbei werden etwa Devices, Tablespaces oder Logfile–Optionen eingestellt. Die oberste Ebene schließlich bezieht sich auf die Optimierung der Datenbank, wie z. B. die Bestimmung von Materialisierungen.

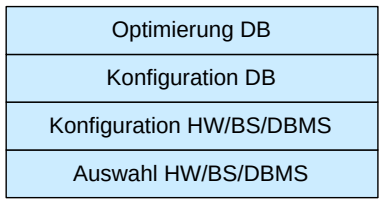



Abbildung 10.3: Aspekte der physischen Datenbankoptimierung

  Bei Bestimmung der Umgebungsparameter muss innerhalb der Abbildung 10.3 eine horizontale Linie gezogen werden, wobei alle Aspekte unterhalb der Linie als vorgegeben angesehen werden und somit als Umgebungsparameter oder Regeln modelliert werden, während die oberhalb der Linie angesiedelten Aspekte das Resultat des Prozesses bilden. Meistens werden die drei unteren Ebenen als gegeben angesehen, während die Maßnahmen der obersten Ebene als Ergebnis der Optimierung ermittelt werden.

- Als letzte Eingaben sind in der Konfigurationsphase ein geeignetes *Kostenmodell* und ein *Optimierungsalgorithmus* für die Kompromissphase festzulegen.

Die Gesamtheit dieser Komponenten dient als Eingabe für die eigentliche Optimierungsphase, die sich aus folgenden Subphasen zusammensetzt: In der als *Auswahlphase* bezeichneten Phase I werden alle aufgrund der eingegebenen Parameter günstigen Optimierungsmaßnahmen bestimmt. Hierbei wird jede Regel unabhängig von den anderen unter den Eingabebedingungen ausgewertet. Im allgemeinen ist die Umsetzung der gesamten Menge ausgewählter Optimierungsmöglichkeiten jedoch aufgrund von Umgebungsparametern, wie z. B. zur Verfügung stehendem Speicherplatz oder zeitlicher Restriktionen, nicht realisierbar. Daher schließt sich als Phase II eine interaktive *Überarbeitungsphase* der Ergebnismenge der ersten Phase durch den Benutzer an. Hierbei besteht die Möglichkeit, für jede in der ersten Phase ausgewählte Optimierungsmaßnahme zu entscheiden, ob sie auf jeden Fall realisiert werden soll, nicht realisiert werden oder als Kandidat für die Kompromissphase Phase III dienen soll[1]. Durch diesen Interaktionsschritt kann weiteres externes Wissen in den Entwurfsprozess einfließen, das nicht in den Regeln abzubilden war. Ebenso ist das Szenario denkbar, in den Regeln nur Herstellerangaben zur Optimierung abzulegen und an dieser Stelle im Entwurf das Wissen des Entwicklers einfließen zu lassen.
Schließlich folgt als dritte und letzte Subphase ein als *Kompromissphase* bezeichneter Abschnitt, in

[1] Die gestrichelte Linie in Abbildung 10.2 steht für die Optimierungsmaßnahmen, die auf jeden Fall realisiert werden sollen. Diese gehören einerseits in die Ergebnismenge, andererseits müssen sie aber auch in der Kompromissphase berücksichtigt werden, z. B. beim Berechnen globaler Nebenbedingungen.

dem folgendes Optimierungsproblem zu lösen ist: Aus der Teilmenge sinnvoller Optimierungsmaßnahmen muss eine Teilmenge bestimmt werden, die gemäß einem Optimierungsziel, einem vorgegebenen Kostenmodell für die Aufgaben und einigen Nebenbedingungen optimal ist. Ein typisches Optimierungsziel ist z. B. die Minimierung der Lesekosten von Aufgaben aus dem Workload, typische Nebenbedingungen sind der zur Verfügung stehende Speicherplatz oder die zur Verfügung stehende Zeit zur Durchführung der Optimierungsmaßnahmen. An das Ende der Optimierungsphase, deren Resultat neben dem anfangs definierten Schema eine Menge von zu realisierenden Optimierungsmaßnahmen ist, schließt sich eine weitere Benutzerinteraktion an. Der Entwickler kann die endgültig ausgewählte Menge an Optimierungsmaßnahmen bewerten und wenn das Resultat in seinen Augen zufriedenstellend ist, kann in die *Implementierungsphase* übergegangen werden, in der vom Zielsystem abhängige Datenbankskripte generiert oder über eine Programmierschnittstelle DB–Objekte angelegt werden. Ist das Resultat jedoch nicht befriedigend, ist ein weiterer Durchlauf des gesamten Optimierungsprozesses mit geänderten Eingabewerten möglich. Abbildung 10.2 zeigt schließlich auch die *Betriebsphase*, in der das DWH mit dem Schema und den Optimierungsmaßnahmen im Einsatz ist. In dieser Zeit ensteht durch Log–Informationen des Systems neues „Wissen“, z. B. ob die im ursprünglichen Workload festgelegten Aufgaben auch tatsächlich auf dem DWH ausgeführt werden. Mit diesen (realistischeren als den à priori vom Entwicklers geschätzten) Eingaben besteht die Möglichkeit, den physischen Optimierungsschritt nach einer gewissen Betriebszeit des DWH zu wiederholen.

## 10.2 Metamodell für den physischen Datenbankentwurf

Die Spezifikation des Metamodells erfolgt analog zu den vorangegangenen Kapiteln mittels eines UML–Klassendiagramms. Das in Abbildung 10.4 dargestellte Metamodell lässt sich in die grau hinterlegten Bereiche unterteilen, die in den folgenden Teilabschnitten detailliert vorgestellt werden:

- Im Bereich *Typen* (Abschnitt 10.2.1) sind Datentypen angesiedelt, die von Attributen anderer Metaklassen genutzt werden.
- Der Bereich *Annotiertes Schema* (Abschnitt 10.2.2) beschreibt ein Schema mit seinen Komponenten und statistischen Informationen über diese.
- Der Bereich *Optimierungsmaßnahmen* (Abschnitt 10.2.3) fasst Metaklassen zusammen, deren Objekte Optimierungsmaßnahmen sowie Mengen von diesen bilden können. Außerdem werden aufgabenunabhängige Kosten festgehalten.
- Die Klassen im Bereich *Aufgaben und Workload* (Abschnitt 10.2.4) definieren auf der Datenbank auszuführende Aufgaben, d. h. lesende oder schreibende Zugriffe. Eine Menge von gewichteten Aufgaben bildet einen Workload.
- Im Bereich *Umgebung und Regeln* (Abschnitte 10.2.5 und 10.2.6) angesiedelte Klassen beschreiben die konkrete Umgebung, bestehend aus dem DBMS und globalen Randbedingungen, sowie die für den physischen Entwurf gültigen Regeln.
- Schließlich beschreibt der Bereich *Prozess* (Abschnitt 10.2.7) Eingaben, Zwischenergebnisse und Resultate der physischen Optimierung einer Datenbank.

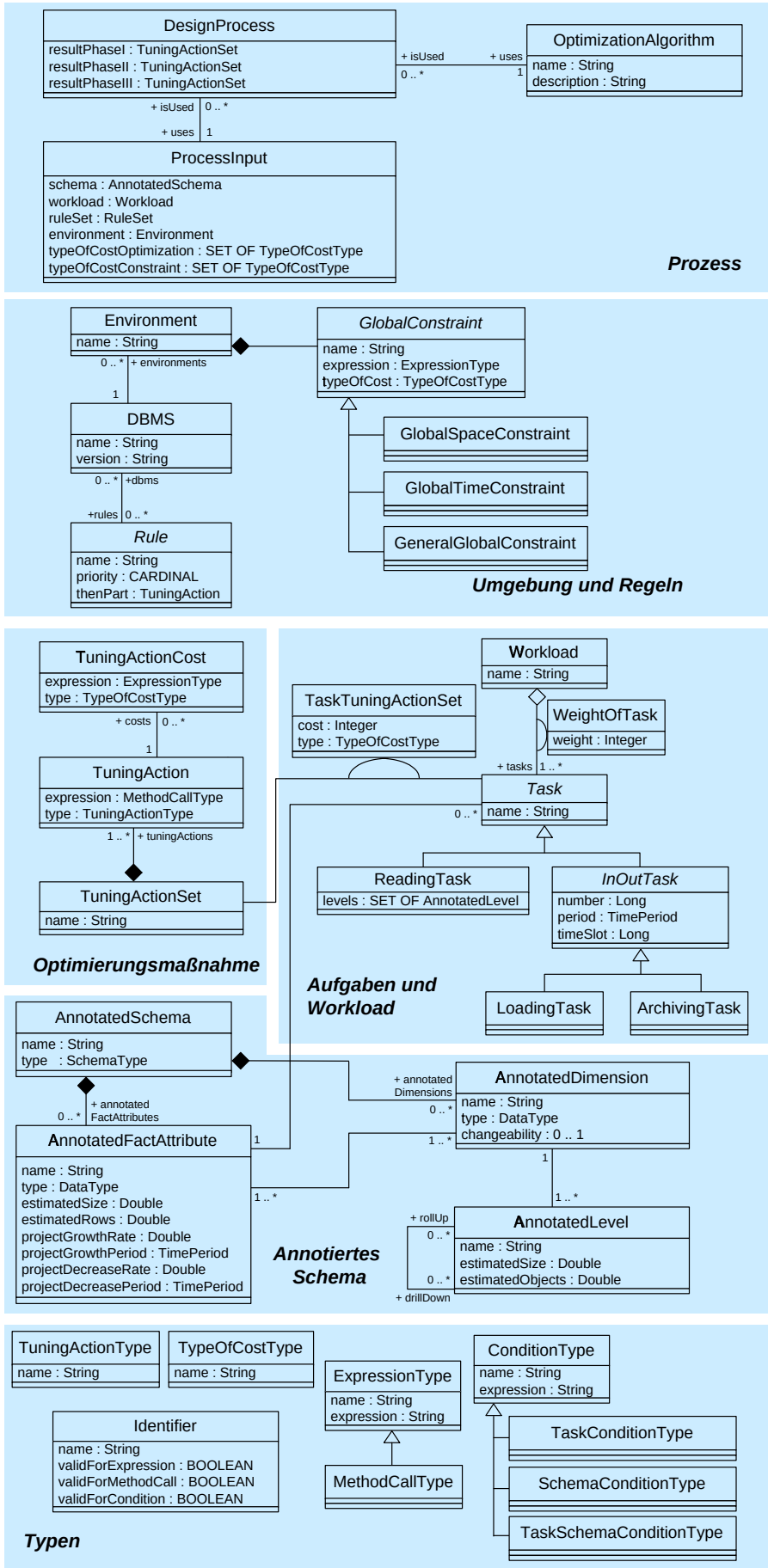



Abbildung 10.4: Metamodell für die physische Datenbankoptimierung

### 10.2.1 Typen

Der Bereich *Typen* stellt Datentypen zur Verfügung, die von Attributen anderer Metaklassen genutzt werden. In Abbildung 10.5 wird mit Hilfe der Klassen *TuningActionType* für verschiedene Optimierungsmöglichkeiten und *TypeOfCostType* für verschiedene Kostenarten die Möglichkeit zur Definition von Aufzählungstypen eingeführt. Die in der Abbildung dargestellten Instanzen zeigen typische Objekte eines „realen Weltausschnitts“.

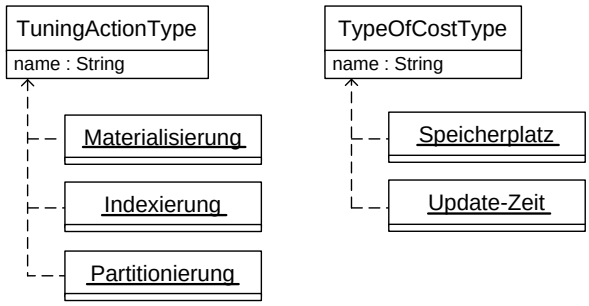



Abbildung 10.5: Datentypen

In Abbildung 10.6 werden Ausdrücke durch die Oberklasse *ExpressionType* mit ihrer Spezialisierung *MethodCallType* definiert. Außerdem können als Instanzen der Klasse *ConditionType* Bedingungen, d. h. boolesche Ausdrücke, festgelegt werden, die je nach Objekten, auf die sie Bezug nehmen, in *TaskConditionType*, *SchemaConditionType* und *TaskSchemaConditionType* spezialisiert werden.

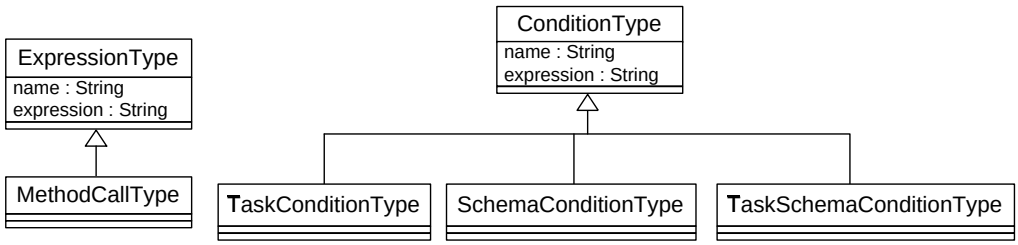



Abbildung 10.6: Ausdrücke und Bedingungen

Die Syntax der Ausdrücke wird nicht im UML–Diagramm festgehalten, sondern durch folgende Grammatik definiert:

**Bedingungen**

$$
\begin{aligned}
Condition ::= {} & Expression\ RelationalOperator\ Expression \\
& \mid \text{"("} Condition \text{")"} \\
& \mid Condition\ BooleanOperator\ Condition \\
& \mid \text{"NOT"}\ Condition.
\end{aligned}
$$

$$BooleanOperator ::= \text{"AND"} \mid \text{"OR"} \mid \text{"XOR"}.$$

**Ausdrücke**

$Expression ::= "(" Expression ")"$
$\quad | ("+" | "-") Expression$
$\quad | Expression\ Operator\ Expression$
$\quad | Integer$
$\quad | Real$
$\quad | String$
$\quad | MethodCall$
$\quad | Identifier.$

**Methodenaufrufe**

$MethodCall ::= MethodName\ ParameterList.$
$ParameterList ::= "(" ParameterName\ ParameterType$
$\quad \{"," ParameterName\ ParameterType\}")".$
$ParameterType ::=$ "INTEGER"|"REAL"|"BOOLEAN"|"STRING".

**Bezeichner und vordefinierte Namen**

$ParameterName ::= Identifier.$
$MethodName ::= Identifier.$
$String ::= "\ "(Letter|Digit)\{Letter|Digit\}"\ ".$
$Identifier ::= Letter\{(Letter|Digit)\}.$
$Integer ::= ["+" | "-"]Digit\{Digit\}.$
$Real ::= Digit\{Digit\}"."Digit\{Digit\}.$
$Letter ::=$ ("a"|"b"|...|"z"|"A"|"B"|...|"Z").
$Digit ::=$ ("0"|"1"|...|"9").
$RelationalOperator ::= ("=" | "\neq" | ">" | "\geq" | "<" | "\leq").$
$Operator ::= ("+" | "-" | "*" | "/" | "\uparrow").$

Schließlich werden in den Ausdrücken zulässige Bezeichner durch die Klasse *Identifier* mit ihren Attributen *ValidFor<X>* festgelegt (siehe Abbildung 10.7). Ist bei einer Instanz dieser Klasse z. B. das Attribut *validForCondition* wahr, so kann der Bezeichner in Bedingungen verwendet werden.

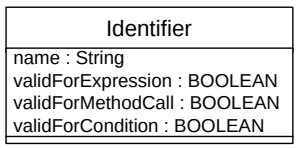



Abbildung 10.7: Gültige Bezeichner

### 10.2.2 Annotiertes Schema

Während bisher ein Schema als Sammlung von Tabellen betrachtet wurde, die sich wiederum aus Spalten zusammensetzen (siehe *LCD of SQL*–Metamodell auf Seite 164 ff.), soll in diesem Kapitel von einem *annotierten Schema* ausgegangen werden. Ein annotiertes Schema (Klasse *AnnotatedSchema*) setzt sich aus mehreren annotierten Faktattributen (Klasse *AnnotatedFactAttribute*) und

Dimensionen (Klasse *AnnotatedDimension*) zusammen, die untereinander entsprechend in Beziehung stehen. Jeder Dimension sind mehrere Hierarchieebenen (Klasse *AnnotatedLevel*) zugeordnet. Die rekursive Assoziation der Klasse *AnnotatedLevel* modelliert die Hierarchien innerhalb einer Dimension. Durch diese Betrachtungsweise lassen sich zum einen unterschiedliche Schematypen wie Stern– und Schneeflockenschemata mit dem Framework einheitlich behandeln und zum anderen ist eine klarere Darstellung möglich, weil komplizierte Typabfragen von Tabellen entfallen.

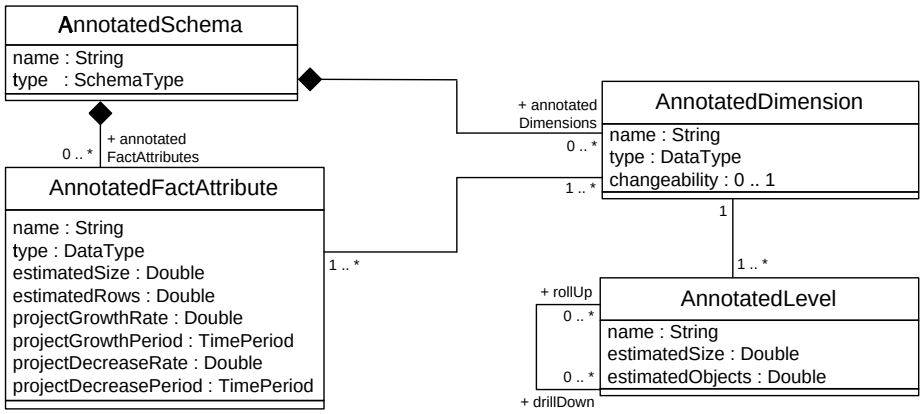



Abbildung 10.8: (Annotierte) Schemata

Ein *AnnotatedSchema*–Objekt wird durch seinen Namen (Attribut *name*) und Schematyp (Attribut *type*) beschrieben. Die Metaklasse *AnnotatedFactAttribute* besitzt die vier bereits aus dem *LCD of SQL*–Metaklassendiagramm (siehe Seite 165) bekannten Attribute *projectGrowthRate, projectGrowthPeriod, projectDecreaseRate* und *projectDecreasePeriod.* Zusätzlich gibt es die Attribute *estimatedSize* und *estimatedRows.* Mit *estimatedSize* wird die geschätzte Größe eines Fakteintrags beschrieben, dieser hängt neben dem Datentyp auch von der Anzahl der Dimensionen ab, denn jede Dimension bedeutet ein Fremdschlüsseleintrag in der Fakttabelle und dieser benötigt auch Platz. Der Wert des Attributes *estimatedRows* beschreibt die geschätzte Anzahl an Zeilen der Tabelle beim initialen Laden des DWH. Neben Namen und Datentyp wird eine Dimension durch das Attribut *changeability* charakterisiert, das einen Wert aus dem Intervall „$[0 .. 1]$“ der reellen Zahlen annehmen kann und die Änderungswahrscheinlichkeit der Dimension angibt. Ein *AnnotatedLevel*–Objekt wird schließlich neben seinem Namen durch die Attribute *estimatedSize* und *estimatedObjects* charakterisiert. Während *estimatedSize* die gleiche Bedeutung wie das gleichnamige Attribut in der Metaklasse *AnnotatedFactAttribute* besitzt, entspricht *estimatedObjects* dem Attribut *estimatedRows* in der Metaklasse *AnnotatedFactAttribute.*

### 10.2.3 Optimierungsmaßnahmen

Wie in Abbildung 10.9 dargestellt, ist eine Optimierungsmaßnahme (Klasse *TuningAction*) durch ihren Typ (z. B. „Materialisierung“) und durch einen Ausdruck in Form eines Methodenaufrufs definiert. Eine Optimierungsmaßnahme kann mehrere aufgabenunabhängige Kosten verursachen (wie z. B. Speicherplatz eines Index), die als Objekte der Klasse *TuningActionCost* jeweils durch eine Kostenart (Attribut *type*) und einen Ausdruck (Attribut *expression*) beschrieben werden. Optimierungsmaßnahmen werden zu Mengen (Klasse *TuningActionSet*) zusammengefasst, die Eingabeparameter für den Entwurfsprozess sein können.

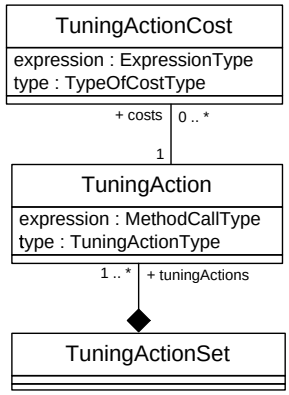



Abbildung 10.9: Optimierungsmaßnahmen

### 10.2.4 Aufgaben und Workload

Der in Abbildung 10.10 abgebildete Bereich *Aufgaben und Workload* stellt mittels der abstrakten Oberklasse *Task* Aufgaben dar, die auf der Datenbank auszuführen sind. Diese werden durch die Vererbung in die speziellen Aufgabentypen *ReadingTask*, *LoadingTask* und *ArchivingTask* unterteilt. Jeder Aufgabentyp referenziert über eine Assoziation ein Faktattribut, das Gegenstand dieser Aufgabe ist. Ein *ReadingTask*–Objekt wird im Attribut *levels* zusätzlich durch die Menge von Hierarchieebenen charakterisiert, auf die sich die Anfrage bezieht. Die Lade– und Archivierungsaufgaben hingegen besitzen die beschreibenden Attribute *number* (Anzahl der zu ladenden bzw. zu archivierenden Datensätze), *period* (Zeitintervall, in dem die Aufgabe auftritt) und *timeSlot* (Zeitschranke, innerhalb der die Aufgabe zu bewältigen ist). Die Kosten, die eine Aufgabe bei Vorliegen eines bestimmten *TuningActionSet*–Objektes verursacht, werden durch die an der Beziehung zur Klasse *TuningActionSet* modellierten Assoziationsklasse *TaskTuningActionSet* angegeben. Eine Menge von Aufgaben wird zu einem *Workload* zusammengefasst, wobei den einzelnen Aufgaben innerhalb des Workloads unterschiedliche Gewichte zugeordnet werden können, was im Attribut *weight* der Assoziationsklasse *WeightOfTask* festgehalten wird.

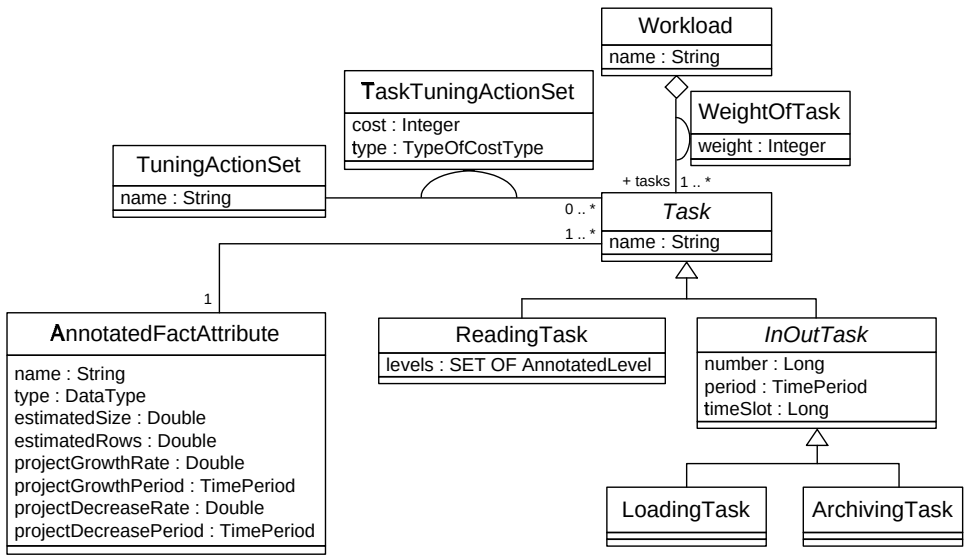



Abbildung 10.10: Aufgaben und Workload

### 10.2.5 Umgebung

In Abbildung 10.11 wird die Umgebung (Klasse *Environment*) definiert, die sich aus globalen Randbedingungen (Klasse *GlobalConstraint*) und dem verwendeten System (Klasse *DBMS*) zusammensetzt. Globale Randbedingungen werden durch ihren Namen (Attribut *name*), einen Ausdruck (Attribut *expression*) und die durch sie verursachte Kostenart (Attribut *typeOfCost*) beschrieben. Weiterhin können sie sich auf den Speicherplatz (Klasse *GlobalSpaceConstraint*) oder die Zeit (Klasse *GlobalTimeConstraint*) beziehen oder allgemeiner Natur sein (Klasse *GeneralGlobalConstraint*). Ein *DBMS* ist durch seine Bezeichnung und eine Versionsnummer gekennzeichnet und referenziert eine Menge von Regeln, die für diese Version des Systems gültig sind.

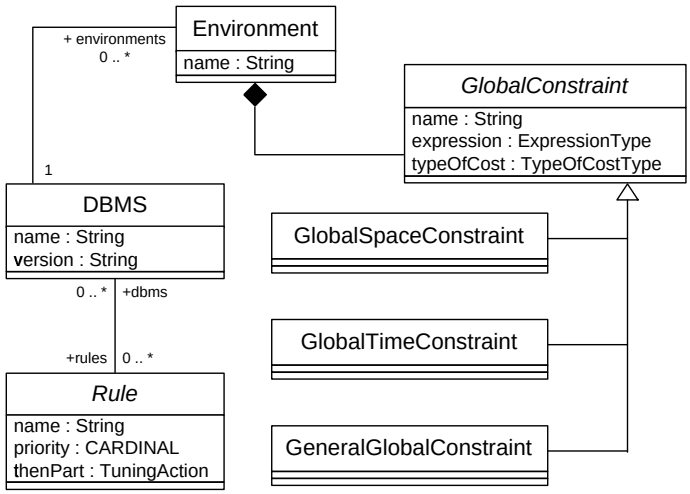



Abbildung 10.11: Umgebung und Regeln

### 10.2.6 Regeln

Abbildung 10.12 zeigt den Bereich *Regeln*: Für ein DBMS können ein oder mehrere *Wenn–Dann–Regeln* (Klasse *Rule*) festgelegt. Die drei Typen von Regeln (Klassen *TaskRule*, *SchemaRule* und *TaskSchemaRule*) unterscheiden sich in ihrem *Wenn*–Teil. In diesem dürfen nur Bezeichner verwendet werden, die gemäß der Klasse *Identifier* in Abbildung 10.7 für den entsprechenden Ausdruckstyp zulässig sind. Regeln werden zu Regelmengen (Klasse *RuleSet*) zusammengefasst, wobei eine Regel auch mehreren Mengen zugeordnet sein kann. Die Bildung von Regelmengen dient der Strukturierung, eine Regelmenge dient als Eingabe für den physischen Entwurfsprozess (siehe Abschnitt 10.2.7).

Die durch den *Dann*–Teil von zwei Regeln $R_1$ und $R_2$ hervorgerufenen Optimierungsmaßnahmen $O_1$ und $O_2$ können Beziehungen aufweisen, wobei folgende Szenarien auftreten können, für deren Behandlung der Entwickler in der Überarbeitungsphase zuständig ist:

- $O_1$ kann $O_2$ *überdecken*, z. B. $O_1$ = „Materialisiere alle Filialen.“ und $O_2$ = „Materialisiere alle Filialen im Gebiet Nord.“ Eine Realisierung von $O_1$ würde beide Fälle abdecken, andererseits ist $O_2$ natürlich weniger kostenintensiv. Deckt $O_2$ aber schon einen Großteil der Anfragen an das System ab, so kann ihre Realisierung durchaus sinnvoll sein.

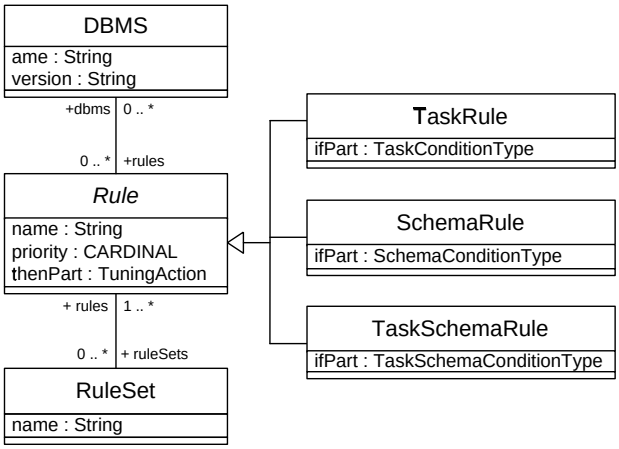



Abbildung 10.12: Umgebung und Regeln

- $O_1$ und $O_2$ können *überlappen,* z. B. $O_1$ = „Materialisiere alle Filialen der Gebiet Ost und West.“ und $O_2$ =„Materialisiere alle Filialen der Gebiete Nord und West.“. Eine Realisierung beider Optimierungsmaßnahmen wäre speicherplatzintensiver als das Realisieren der *Vereinigung* „Materialisiere alle Filialen der Gebiete Nord, West und Ost.“
- $O_1$ und $O_2$ können sich *widersprechen,* z. B. $O_1$ = „Partitioniere die Fakttabelle nach Monaten.“ und $O_2$ = „Partitioniere die Fakttabelle nach Filialen.“ Ein solcher Konflikt muss aufgelöst werden, indem der Entwickler sich in der Überarbeitungsphase für eine der beiden Maßnahmen entscheidet.

### 10.2.7 Prozess

Abbildung 10.13 zeigt die Bestandteile des in Abbildung 10.2 dargestellten Optimierungsablaufs: Ein Optimierungsprozess (Klasse *DesignProcess*) bekommt eine Eingabe (Klasse *ProcessInput*), die sich aus einem annotierten Schema, einem Workload, einer Menge von Regeln und einer Umgebung zusammensetzt. Weiterhin werden als Eingabe des Prozesses diejenigen Kostenarten genannt, nach denen optimiert werden soll und diejenigen, die als Constraint in die Optimierung einfließen sollen. Die Ergebnisse der einzelnen Phasen werden durch die Attribute *result<X>* der Klasse *DesignProcess* repräsentiert, die Beziehung zur Klasse *OptimizationAlgorithm* dokumentiert den in Phase III verwendeten Optimierungsalgorithmus.

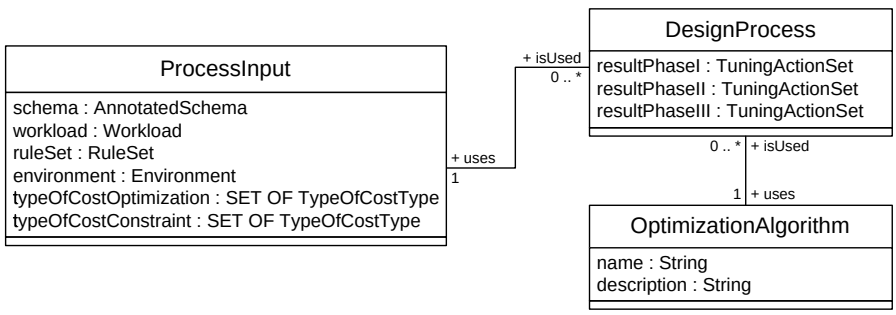



Abbildung 10.13: Prozess des physischen Datenbank–Entwurfs

## 10.3 Formalisierung des Entwurfsprozesses

Nachdem durch Beschreibung an der physischen Optimierung beteiligter Objekte der Rahmen geschaffen worden ist, wird in diesem Abschnitt der eigentliche Prozess detailliert dargestellt. Dabei werden zunächst die im vorigen Abschnitt genannten Begriffe formaler eingeführt und das Optimierungsproblem spezifiziert.

### 10.3.1 Eingabe

In diesem Abschnitt werden zunächst die Begriffe *annotiertes Schema* (10.1), *Workload* (10.2), *Regelmengen* (10.3), *Umgebung* (10.4) und *Kostenarten* (10.5) sowie *Optimierungsmaßnahmen* (10.6) formal definiert.

Ein Tripel $S = (A, D, Type)$ heißt *annotiertes Schema*.
Dabei ist $A$ die Menge der Faktattribute und $D$ die Menge der Dimensionen.
$Type$ gibt den Schematy an.
$\mathcal{S}$ ist die Menge aller *annotierten Schemata*. (10.1)

Ein endliche Menge von Paaren $W = \{(t_i, \phi_i) \mid i \in \{1, \ldots, m\}\})$ heißt *Workload*,
wobei jedes $t_i$ eine Aufgabe und jedes $\phi_i$ die Gewichtung der i–ten Aufgabe
im Workload ist.
$\mathcal{W}$ ist die Menge aller *Workloads*. (10.2)

$R$ ist eine *Regelmenge* (*RuleSet*), wobei jedes $r \in R$ durch seinen
Namen, einen *Wenn*–Teil und einen *Dann*–Teil beschrieben wird.
$\mathcal{R}$ ist die Menge aller *Regelmengen*. (10.3)

$E$ ist eine *Umgebung* (*Environment*), wobei
jedes $e \in E$ ein DBMS und eine Menge $GC$ globaler Nebenbedingungen umfasst.
Dabei gelte $\forall gc \in GC : gc \Theta Expression$ mit $\Theta \in \{=, >, \leq, >, \geq\}$.
$\mathcal{E}$ ist die Menge aller *Umgebungen*. (10.4)

$T$ ist eine endliche Menge von *Kostenarten*.
$\mathcal{T}$ ist die Menge aller Mengen von *Kostenarten*. (10.5)

$A$ ist eine Menge von *Optimierungsmaßnahmen*, wobei jedes $a \in A$ durch einen Typ und
einen Ausdruck beschrieben wird. Weiterhin kann jedes $a \in A$ eine Menge von
Kosten (bestehend aus einer Kostenart und einem Ausdruck) besitzen.
$\mathcal{A}$ ist die Menge aller Mengen von *Optimierungsmaßnahmen*. (10.6)

### 10.3.2 Die drei Subphasen der Optimierungsphase

Unter Verwendung der im letzten Abschnitt eingeführten Definitionen der Konfigurationsphase werden in diesem Abschnitt die drei Subphasen der Optimierungsphase spezifiziert.

**Phase I: Selektion**

Aufbauend auf den Definitionen (10.1) bis (10.6) wird mit der Selektion in (10.7) die erste Subphase der Optimierungsphase definiert. Sie bildet ein annotiertes Schema, einen Workload, eine Menge von Regeln und eine Umgebung auf eine Menge von Optimierungsmaßnahmen ab.

$$\begin{aligned} &Phys_{PhaseI} : \mathcal{S} \times \mathcal{W} \times \mathcal{R} \times \mathcal{E} \rightarrow \mathcal{A} \\ &Phys_{PhaseI}(S, W, R, E) \stackrel{def}{=\!=} \{a \mid \exists r \in R : r.thenPart = a\}. \end{aligned} \tag{10.7}$$

**Phase II: Überarbeitung**

DieUeberarbeitungsphase@Überarbeitungsphase Überarbeitungsphase nutzt das Resultat der Selektion aus (10.7). Unter der Annahme, dass $f_{det_{Modification}}$ die Modifikation einer Menge von Optimierungsmaßnahmen vornimmt, beschreibt (10.8) diesen Schritt.

$$\begin{aligned} &Phys_{PhaseII} : \mathcal{A} \rightarrow \mathcal{A} \\ &Phys_{PhaseII}(A) \stackrel{def}{=\!=} f_{det_{Modification}}(A). \end{aligned} \tag{10.8}$$

**Phase III: Kompromiss**

Die Menge der Kostenarten $T \in \mathcal{T}$ zerfällt in zwei Teilmengen:

- Die Menge der zur Optimierung zu berücksichtigenden Kostenarten $T_{Opt} = \{toc_{Opt_1}, \ldots, toc_{Opt_k}\}$[2] und
- die Menge der als Nebenbedingungen zu berücksichtigenden Kostenarten $T_{Cons} = \{toc_{Cons_1}, \ldots, toc_{C_k}\}$.

Für jede zu optimierende Kostenart $toc \in T_{Opt}$ sei eine (partielle) Kostenfunktion $cot_{toc} : T \times \mathcal{A} \rightarrow Integer$[3] definiert, die einem Paar $(toc, A)$ die Kosten der Aufgabe $toc$ bei Realisierung der Optimierungsmaßnahmen der Menge $A$ zuordnet. Wünschenwertes Ziel der Kompromiss– und somit der gesamten Optimierungsphase wäre die Minimierung der Kosten für den Workload für jede Kostenart, d. h. die Ausdrücke

$$cost_{toc_j}(W, A) \stackrel{def}{=\!=} \sum_{i=1}^{m} \phi_i \cdot cot_{toc_j}(t_i) \tag{10.9}$$

müssten für jedes $j$ unter Einhaltung der globalen Nebenbedingungen (d. h. jedes $gc \in GC$ ist wahr) minimiert werden. Da jedoch nicht immer bez. aller Kostenarten optimiert werden kann (z. B. Vorliegen eines Time–Space–Tradeoff), muss ein alternativer Weg gewählt werden, wobei verschiedene Strategien denkbar sind:

- *Hierarchisierung der Kostenarten*
  Die Optimierung nach einer Kostenart wird als Ziel festgelegt, für die anderen werden Obergrenzen gesetzt, d. h. sie werden ebenfalls zu Nebenbedingungen.
  Sei $K \in \{1, \ldots, k\}$ der Index der zu optimierenden Kostenart, dann gibt es nun

[2] toc = type of cost.
[3] cot = cost of task.

$|GC| + |TOC| - 1$ globale Nebenbedingungen, nämlich $GC$ und zusätzlich
$cost_{toc_j}(W, A) \leq C_j$ für $j \in \{1, \ldots, k\} \setminus \{K\}$,
wobei die $C_j$ die vom Benutzer geeignet zu wählenden Obergrenzen sind. Die Optimierungsaufgabe lautet nun

$$cost_{toc_K}(W, A) = \sum_{i=1}^{m} \phi_i \cdot cot_{toc_K}(t_i) \rightarrow min. \tag{10.10}$$

- *Verrechnung der Kostenarten*
  Die $|TOC|$ Optimierungsziele werden zu einem zu optimierenden Ausdruck verschmolzen. Zunächst wird für jede Kostenart eine Normierungsfunktion $\| \ . \ \|_i$ definiert, die die Kosten auf das Intervall [0 .. 1] abbildet. Die Optimierungsaufgabe lautet in diesem Falle

$$\sum_{i=1}^{k} \| \ cost_{toc_i}(W, A) \ \|_i \rightarrow min. \tag{10.11}$$

- *Alternative Optimierungen zur Benutzerentscheidung*
  Die Optimierung wird für jede Kostenart unter Vernachlässigung der anderen Kostenarten einmal durchgeführt, am Ende wählt der Benutzer eine Lösung aus.
  Es wird also für jedes $K \in \{1, \ldots, k\}$ die Optimierung

$$cost_{toc_K}(W, A) = \sum_{i=1}^{m} \phi_i \cdot cot_{toc_K}(t_i) \rightarrow min. \tag{10.12}$$

  durchgeführt, danach entscheidet der Benutzer.

Hat man sich für eine dieser Strategien entschieden, so ist die Optimierungsaufgabe eindeutig spezifiziert und der Algorithmus zur Kompromissfindung kann bestimmt werden. Ein Brute Force–Ansatz kommt nur für (sehr) kleine Mengen von Optimierungsmaßnahmen in Frage, weil seine Komplexität $O(n!)$ beträgt, wobei $n$ die Anzahl an Optimierungsmaßnahmen ist. Als alternative Ansätze für die Kompromissphase kommen Greedy–Strategien, Hoch–Tief–Verfahren oder Verfahren wie Genetische Algorithmen oder Simulated Annealing in Betracht. Diese liefern zwar nicht die optimale Lösung, weisen aber praktikable Laufzeiten auf. Phase III des physischen Entwurfs wird in (10.13) definiert, wobei $f_{det_{Algorithm}}$ eine Funktion ist, die den gewünschten Algorithmus in Abhängigkeit von der gewählten Strategie beschreibt.

$$\begin{aligned} &Phys_{PhaseIII} : \mathcal{S} \times \mathcal{W} \times \mathcal{R} \times \mathcal{E} \times \mathcal{T} \rightarrow \mathcal{A} \\ &Phys_{PhaseIII}(S, W, R, E, T_{Opt}, T_{Cons}) \stackrel{def}{=} f_{det_{Algorithm}}(S, W, R, E, T_{Opt}, T_{Cons}). \end{aligned} \tag{10.13}$$

## 10.4 Beispiel

Um mit dem Framework zu arbeiten, sind Konfigurationen und die Angabe aktueller Daten notwendig. Dementsprechend unterteilt sich das Beispiel in zwei Phasen: Zunächst wird in Abschnitt 10.4.1 die Konfiguration vorgenommen, die den Algorithmus der Kompromissphase und die Kostenfunktion sowie ihre Berechnung festlegt. Anschließend werden in Abschnitt 10.4.2 konkrete Daten für einen Ausschnitt des Beispiels „Handelswelt“ festgelegt. Abschnitt 10.4.2 zeigt die Materialisierung eines Datenwürfels, in Abschnitt 10.4.2 wird dieses Beispiel um einen konkurrierenden Schreibzugriff erweitert.

### 10.4.1 Konfiguration

Zur Konfiguration wird als Algorithmus ein elementares Greedy–Verfahren vorgeschlagen, ebenso wird eine einfache, vorwiegend Lesekosten berücksichtigende Kostenfunktion vorgestellt, bevor dem Ansatz in [HRU96] folgend die Darstellung des annotierten Schemas als Gitter erfolgt.

#### Algorithmus

Als Algorithmus soll für das Beispiel ein Greedy–Ansatz dienen, der die Menge gewählter Optimierungsmaßnahmen sukzessiv erweitert, indem die Lösungsmenge immer um die im Moment „am meisten einbringende“ Lösung erweitert wird. Hierzu wird der Begriff des *(größten) Nutzens* (=*(maximal) benefit*) eingeführt. Sei die Teilmenge $T' \subset T$ bereits als zu realisierende Optimierungsmaßnahmen ausgewählt, dann ist der *Nutzen* der Optimierungsmaßnahme $t \in T \setminus T'$ relativ zu $T'$, bezeichnet als $B(t, T')$, wie folgt definiert:

$$B(t,T') := \begin{cases} cost_{toc_K}(W,T') - cost_{toc_K}(W,T' \cup \{t\}) \\ \qquad\qquad \text{falls } cost_{toc_K}(W,T') - cost_{toc_K}(W,T' \cup \{t\}) > 0 \\ 0 \qquad\qquad \text{sonst.} \end{cases} \tag{10.14}$$

Den größten (relativen) Nutzen weist dasjenige $s \in T \setminus T'$ auf, für das $B(s,T') > B(t,T')$ mit $t \in T \setminus T'$ und $s \neq t$ ist. *Relativ* bedeutet in diesem Falle relativ zur bisher gefundenen Lösungsmenge $T'$. Damit lässt sich der folgende Algorithmus formulieren:

```
( 1)    T' = ∅
( 2)    while ∀i ∈ GC : gc_i = true
( 3)       ermittle Optimierungsmassnahme t ∈ T \ T' mit groesstem Nutzen
( 4)       T' = T' ∪ {t}
( 5)    end
```

Algorithmus 10.1: Greedy–Algorithmus

Resultat ist die Menge T der zu realisierenden Optimierungsmaßnahmen. Macht der Entwickler von der interaktiven Eingriffsmöglichkeit in Phase II Gebrauch, so würde man nicht mit einer leeren Menge starten, sondern $T'$ enthielte zu Beginn die auf jeden Fall zu realisierenden Optimierungsmaßnahmen.

#### Die Kostenfunktion

Als Leseaufgaben sollen Anfragen an ein Schema als Kombination aus Hierarchieebenen modelliert werden, wobei für jede Dimension eine Hierarchieebene bestimmt wird (vergleiche Attribut *levels* in Abschnitt 10.2.4). Mit den für ein DWH charakteristischen Annahmen, dass in einem Datenwürfel

- die Kardinalität eines Fakts wesentlich größer als die Kardinalität einer jeden Dimension ist und
- der Lesezugriff auf die Festplatte erheblich teurer ist als Rechen– und Selektionsoperationen im Hauptspeicher,

können die Kosten eines *ReadingTask* als die Lesekosten vom Festspeichermedium festgelegt werden. Die Einheit soll die Anzahl der Tupel sein. Im schlimmsten Falle (wenn keine Materialisierungen

vorliegen) müssen alle Datensätze aus der Fakttabelle ausgelesen werden. Ansonsten ist es die Anzahl Datensätze der *nächst niedrigeren materialisierten Sicht*. Um diesen Begriff zu formalisieren, wird das in [HRU96] vorgestellte *Gittermodell* eingeführt.

**Das Gittermodell**

Anfragen an ein annotiertes Schema können in einer Beziehung stehen: Liegen zwei Anfragen $A_1$ und $A_2$ vor, dann gelte $A_1 \preceq A_2$ genau dann, wenn $A_1$ ausschließlich aus dem Ergebnis von $A_2$ berechnet werden kann. Wir sagen in diesem Falle auch $A_1$ ist *abhängig* von $A_2$. Der Operator $\preceq$ führt auf der Menge aller Anfragen eine partielle Ordnung ein, die Anfragen an ein multidimensionales Datenobjekt bilden ein Gitter [TM75, Lei97]. Als Notation für ein Gitter gelte $\langle L, \preceq \rangle$, wobei L die Menge der Anfragen und $\preceq$ die Abhängigkeitsrelation ist. Weiterhin werden folgende Begriffe zu Beziehungen von Elementen im Gitter definiert:

- Zwei Elemente a,b ∈ L seien *stark voneinander abhängig* (Notation: $a \prec b$), genau dann, wenn $a \preceq b$ und $a \neq b$.
- Für ein Element $a \in L$ sei die *Menge seiner Vorgänger* definiert als $ancestor(a) \stackrel{def}{=\!=} \{b | a \preceq b\}$.
- Für ein Element $a \in L$ sei die *Menge seiner Nachfolger* definiert als $descendant(a) \stackrel{def}{=\!=} \{b | b \preceq a\}$.

Der Zusammenhang zwischen Gittern und multidimensionalen Datenobjekten ist wie folgt: Die Hierarchieebenen einer jeden Dimension bilden ein Gitter, z. B. lässt sich die Zeitdimension mit Hilfe der Abhängigkeitsrelation $(Jahr) \preceq (Monat) \preceq (Tag)$ darstellen. Anschaulich gesprochen heißt dies, die Ergebnisse für ein Jahr können auf die Resultate der Monatsebene und diese wiederum auf die Resultate der Tagesebene zurückgeführt werden. Um auch Anfragen, die sich auf mehr als eine Dimension beziehen, als Gitter modellieren zu können, seien solche Anfragen als Vektor dargestellt, wobei jede Komponente die Hierarchieebene einer Dimension darstellt, z. B. $(Filiale, Produktgruppe, Quartal)$. Das Gitter des gesamten Datenwürfels wird durch Berechnung des *direkten Produkts* der Gitter der einzelnen Dimensionen gebildet. Bildlich gesprochen berechnet sich das direkte Produkt, indem „jeder mit jedem“ kombiniert wird, formaler gilt:

$$\begin{aligned} &\prod_{i=1}^{n} \langle L_i, \preceq_i \rangle \stackrel{def}{=\!=} \langle L, \preceq \rangle \\ &\text{mit } L \stackrel{def}{=\!=} \{(a_1, \ldots, a_n) \mid a_i \in L_i\} \\ &\text{und } \forall a, b \in L : a \preceq b \stackrel{def}{\Longleftrightarrow} a_i \preceq_i b_i. \end{aligned} \tag{10.15}$$

Die $\preceq$–Relation lässt sich folgendermaßen kanonisch auf Vektoren übertragen:

$$(a_1, a_2, ..., a_n) \preceq (b_1, b_2, ..., b_n) \stackrel{def}{\Longleftrightarrow} \forall i : a_i \preceq b_i. \tag{10.16}$$

Damit sind Anfragen die Knoten eines solchen Gitters und die *nächst niedrigere materialisierte Sicht* ist dasjenige Element der Menge $ancestor$, das die geringsten Kosten hat. Der Begriff der Kosten wird mit Hilfe einer Knotenmarkierung erreicht, im Falle der Dimensionen sind das für jede Hierarchieebene die Anzahl der Instanzen (Attribut *estimatedSize* in der Klasse *AnnotatedFactAttribute*), bei der Bildung des direkten Produkts werden die Gewichte der beteiligten Knoten multipliziert.

### 10.4.2 Beispielrechnungen

Für die Beispielrechnungen soll der in Abbildung 10.14 dargestellte Schemaausschnitt betrachtet werden.

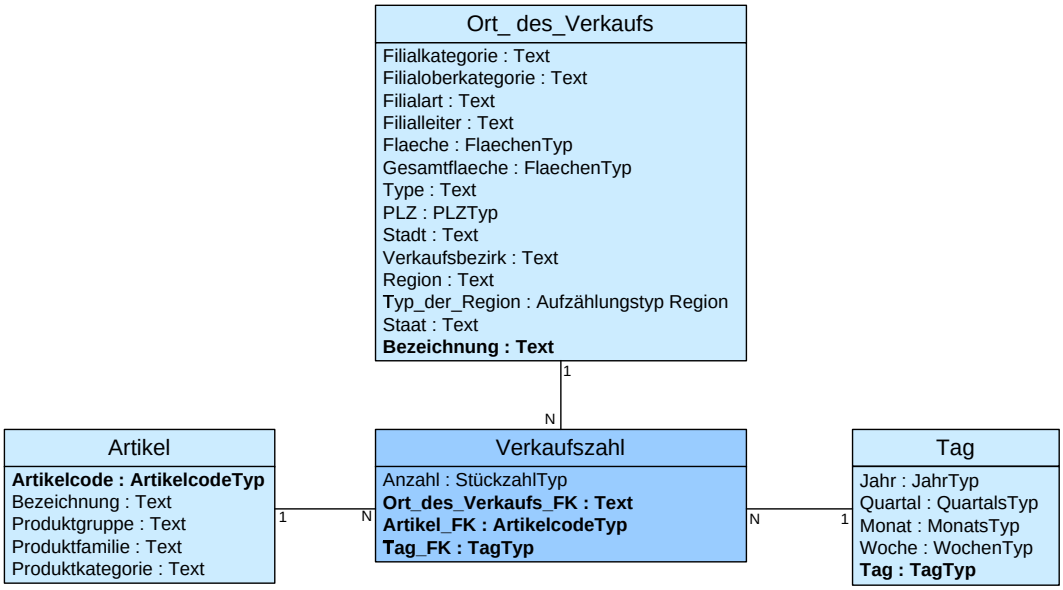



Abbildung 10.14: Aufgaben und Workload

#### Schema, Workload, Regeln und Kosten

Als Regel für das Framework soll die (relativ unspezifische) Aussage gelten, dass alle Kombinationen von Hierarchieebenen in einem Würfel materialisiert werden sollen. Resultat der Auswahlphase mit dieser Regel ist immer eine Menge von Optimierungsmaßnahmen, die die Materialisierung aller Kombinationen von Hierarchieebenen, d. h. aller Knoten des Gitters, vorschlägt. Die Hierarchien des Schemas aus Abbildung 10.14 in Gitternotation sind in (10.17) dargestellt.

$$\begin{aligned}&(Jahr) \preceq (Quartal) \preceq (Monat) \preceq (Tag) \text{ und}\\ &(Gebiet) \preceq (Staat) \preceq (Region) \preceq (Stadt) \preceq (Filiale) \text{ sowie}\\ &(Produktbereich) \preceq (Produktgruppe) \preceq (Produkt).\end{aligned} \tag{10.17}$$

Eine Funktion $card$, die jeder Hierarchieebene ihre Kardinalität zuordnet, wird in (10.18) für Beispielwerte definiert.

$$\begin{aligned}&card : \text{Hierarchieebenen} \rightarrow Integer\\ &Tag \mapsto 1460, Monat \mapsto 48, Quartal \mapsto 16, Jahr \mapsto 4,\\ &Filiale \mapsto 4500, Stadt \mapsto 1112, Region \mapsto 116, Staat \mapsto 9, Gebiet \mapsto 2\\ &Produkt \mapsto 10000, Produktgruppe \mapsto 100 \text{ und } Produktbereich \mapsto 10.\end{aligned} \tag{10.18}$$

Als globale Randbedingung soll Festplattenplatz im Umfang von 1,2 GByte für die zu materialisierenden Sichten zur Verfügung stehen. Damit können die Formeln aus Abschnitt 10.3 in (10.19) initialisiert werden.

$$\begin{aligned} &TOC_{Opt} = \{Anfragezeit\} \\ &TOC_{Cons} = \{Speicherplatz\} \\ &GC = \{Speicherplatz \leq 1,2\,GByte\} \\ &A = \{(z,o,p) | z \in \{Tag, Monat, Quartal, Jahr\} \\ &\qquad\qquad \wedge\, o \in \{Filiale, Stadt, Region, Staat, Gebiet\} \\ &\qquad\qquad \wedge\, p \in \{Produkt, Produktgruppe, Produktbereich\}\}. \end{aligned} \tag{10.19}$$

Die aufgabenunabhängigen Kosten sind durch

$$cost_{Speicherplatz}(a) \stackrel{def}{=\!=} card(z) \cdot card(o) \cdot card(p)$$

festgelegt. Die Kosten für die Nebenbedingung errechnen sich mittels

$$Speicherplatz \stackrel{def}{=\!=} \sum_{a \in A'} cost_{Speicherplatz}(a), \text{ wobei } A' \subset A \text{ gilt.}$$

Der Workload lässt sich durch

$W = \langle T, \Phi \rangle$ mit $T = \{t_1, \ldots, t_m\}$ und $\Phi = \{\phi_1, \ldots, \phi_m\}$ beschreiben,
wobei $t_1$ = Tag–Stadt–Produktgruppe ($\phi_1 = 3$),
$t_2$ = Monat–Stadt–Produkt ($\phi_2 = 1$),
$t_3$ = Jahr–Stadt–Produktbereich ($\phi_3 = 7$),
$t_4$ = Tag–Filiale–Produktgruppe ($\phi_4 = 9$)
$t_5$ = Monat–Stadt–Produktbereich ($\phi_5 = 5$)gelten soll.

Die aufgabenabhängigen Kosten berechnen sich nach

$$cost_{Anfragezeit}(t, A') \stackrel{def}{=\!=} \min\{cost(a') \mid a' \in A' \wedge a \preceq t\}.$$

Die Optimierungsaufgabe lautet damit schließlich

$$cost_{Anfragezeit}(W, A) = \sum_{i=1}^{5} \phi_i \cdot cost_{Anfragezeit}(t_i, A) \rightarrow min.$$

unter Berücksichtigung der Nebenbedingung

$$\sum_{i=1}^{5} cost_{Speicherplatz}(a_i) \leq 1,2\,GByte.$$

**Beispiel 1: Materialisierung**

Mit Hilfe des Greedy–Algorithmus erhält man das Resultat aus Tabelle 10.1. Hinter den Materialisierungen ist jeweils der relative Nutzen im betreffenden Durchlauf des Algorithmus angegeben, Einheit ist die Anzahl der Tupel. Der Wert gibt die Anzahl der eingesparten, zu lesenden Tupel an, wenn die enstprechende Materialisierung realisiert wird.

| **Materialisierung** | **1. Durchlauf** | **2. Durchlauf** | **3. Durchlauf** | **4. Durchlauf** |
|---|---|---|---|---|
| Tag–Filiale–Produktgruppe | **1561032** | | | |
| Tag–Stadt–Produktgruppe | 983064 | 7419 | 5940 | 1483 |
| Monat–Stadt–Produkt | 847161 | **66645** | | |
| Monat–Filiale–Produkt | 826020 | 63540 | 0 | 0 |
| Tag–Stadt–Produkt | 791436 | 49464 | 0 | 0 |
| Monat–Stadt–Produktbereich | 788393 | 7877 | **6398** | |
| Monat–Filiale–Produktbereich | 788374 | 7858 | 6379 | 0 |
| Monat–Stadt–Produktgruppe | 788335 | 7819 | 6341 | 0 |
| Tag–Stadt–Produktbereich | 788205 | 7689 | 6210 | 0 |
| Monat–Filiale–Produktgruppe | 788140 | 7624 | 6145 | 0 |
| ... | ... | ... | ... | ... |
| Kosten Nebenbedingung in MByte | 657 | 1190 | 1191 | 1354 |
| Kosten Workload in Tupeln | 81468 | 14823 | 8424 | 6940 |

Tabelle 10.1: Ablauf Beispiel 1

Man erkennt das kontinuierliche Anwachsen der Nebenbedingungskosten bei gleichzeitiger Abnahme der Workloadkosten. Der Algorithmus terminiert mit dem vierten Schritt, weil der Speicherplatz überschritten ist[4].

**Beispiel 2: Materialisierung und Schreibzugriff**

Das Szenario aus dem letzten Abschnitt soll erweitert werden, indem der Workload um einen konkurrierenden Schreibzugriff ergänzt werden soll. Dazu nehmen wir an, dass in jedem Monat neue Werte en bloc in die Faktentabelle eingetragen werden. Dabei müssen natürlich auch die (redundanten) Materialisierungen gepflegt werden. Angenommen, es würden bei jeder dieser Anfügeaktionen 1,35 Milliarden Tupel in das DWH übertragen[5]. Die Update–Kosten für ein neues Tupel hängen von der Anzahl und Kardinalität der materialisierten Sichten ab. Weil aber nicht jede Instanz jeder materialisierten Sicht betroffen ist, errechnen sich die Kosten in Schreibzugriffen auf das Festspeichermedium wie folgt: Sei $R$ die Menge einzufügender Tupel, $MV$ die Menge materialisierter Sichten, $Dim$ die Menge der Dimensionen. Dann ist für jedes $v \in MV$ eine Funktion $card_v : Dim \rightarrow Integer$ definiert, die die Kardinalität der angegebenen Dimension $d$ in der materialisierten Sicht $v$ ermittelt:

$$cost_{Update}(r) = \sum_{v \in MV} \frac{1}{card_v(Zeit)} \cdot card(v).$$

Damit kann die Beschreibung aus (10.19) folgendermaßen ergänzen:

$$TOC_{Cons} = \{Speicherplatz, Updatekosten\}$$
$$GC = \{Speicherplatz \leq 1.2\,GByte, Updatekosten \leq 56.58\,Mio.Schreibzugriffe\}.$$

Die Menge $A$ selektierter Optimierungsmaßnahmen bleibt gleich. Die Funktion für die Speicherkosten und die Nebenbedingung bleiben bestehen. Neu hinzu kommt für die Berechnung der Updatekosten

$$Updatekosten \stackrel{def}{=} \sum_{r \in R} cost_{Update}(r).$$

[4] An dieser Stelle könnte eine Optimierung des Algorithmus vorgenommen werden: es wird im letzten Schritt die Optimierungsmaßnahme mit größtem Nutzen realisiert, die noch die Nebenbedingung erfüllt.

[5] Dieser Schätzwert kommt folgendermaßen zustande: 10000 Produkte · 4500 Filialen · 30 Tage

Der Workload wird um die Schreibaufgabe erweitert, d. h.

$$t_6 = \text{„Nachladen von neuen, monatlichen Verkaufszahlen“ mit } \phi_6 = 1.$$

Auch die Optimierungsaufgabe bleibt so bestehen, allerdings kommt als zusätzliche Nebenbedingung neben dem Speicherplatz die Bedingung

$$\sum_{r \in R} cost_{Update}(r) \leq 56,58 Mio.$$

hinzu. Unter Verwendung des Greedy–Algorithmus erhält man das in Tabelle 10.2 wiedergegebene Resultat.

| **Materialisierung** | **1.Durchlauf** | **2.Durchlauf** | **3.Durchlauf** |
|---|---|---|---|
| Tag–Filiale–Produktgruppe | **1561032** | | |
| Tag–Stadt–Produktgruppe | 983064 | 7419 | 5940 |
| Monat–Stadt–Produkt | 847161 | **66645** | |
| Monat–Filiale–Produkt | 826020 | 63540 | 0 |
| Tag–Stadt–Produkt | 791436 | 49464 | 0 |
| Monat–Stadt–Produktbereich | 788393 | 7877 | 6398 |
| Monat–Filiale–Produktbereich | 788374 | 7858 | 6379 |
| Monat–Stadt–Produktgruppe | 788335 | 7819 | 6341 |
| Tag–Stadt–Produktbereich | 788205 | 7689 | 6210 |
| Monat–Filiale–Produktgruppe | 788140 | 7624 | 6145 |
| ... | ... | ... | ... |
| Kosten Nebenbedingung “Speicherplatz“ in MByte | 657 | 1190 | 1191 |
| Kosten Nebenbedingung “Updatekosten“ in Mio. Schreibzugriffen | 45,45 | 56,57 | 56,581 |
| Kosten Workload in Tupeln | 81468 | 14823 | 8424 |

Tabelle 10.2: Ablauf Beispiel 2

Man sieht, dass schon im dritten Durchlauf keine zu materialisierende Sicht mehr ausgewählt wird, weil die neue Nebenbedingung verletzt ist. Auf analoge Weise lässt sich das Framework um Archivierungsaufgaben erweitern, denn auch in diesem Falle müssen die materialisierten Sichten aktualisiert werden. Ebenso ist es möglich, neben der Materialisierung auch Indexierungen oder Partitionierungen als Optimierungsmaßnahmen zu berücksichtigen.

## 10.5 Zusammenfassung

Dieses Kapitel hat den abschließenden Schritt der physischen Datenbankoptimierung behandelt. Dabei wurde zunächst ein Framework zur parallelen Betrachtung unterschiedlicher Optimierungsmaßnahmen eingeführt und dessen Ablauf erläutert. Anschließend erfolgte eine Formalisierung, zunächst in Form eines Metamodells für die statischen Aspekte in Abschnitt 10.2, danach in Abschnitt 10.3 eine formalisierte Formulierung des Optimierungsproblems. Abschließend wurde die Konfiguration des Framework durchgeführt und zur Demonstration der Funktionsweise wurden für einen Ausschnitt des Beispiels „Handelswelt“ zwei Beispielrechnungen vorgenommen.

Tabelle 10.3 wiederholt die zu Beginn des Kapitels aufgestellten Anforderungen an den Optimierungsschritt und erläutert, wie diesen Anforderungen bei Konzeption des Framework begegnet wurde.

**Anforderungen an die physische Optimierung**

| Anforderung | Umsetzung im Framework |
|---|---|
| Unterschiedliche Optimierungsmaßnahmen sollten gleichzeitig betrachtet werden. | In den *Dann*–Teilen der Regeln können unterschiedliche Optimierungsmaßnahmen, wie z. B. Materialisierungen und Partitionierungen festgelegt werden. |
| „Umweltparameter“ (z. B. zeitliche Randbedingungen) sollten individuell (z. B. pro Projekt oder Organisation) konfigurierbar sein. | Im Bereich *Umgebung* lassen sich diese Anforderungen definieren. |
| Die Auswahl der Optimierungsmaßnahmen sollte einem nachvollziehbaren Prozess unterliegen, der idealerweise mit Werkzeugunterstützung durchgeführt wird. | Durch Anwenden des Framework wird ein definiertes und nachvollziehbares Vorgehen gewährleistet. Durch die Spezifikation des Metamodells und die formale Formulierung der Kompromissphase als Optimierungsproblem ist die Basis für eine Implementierung zur Werkzeugunterstützung gegeben. |
| Insbesondere sollte auch nachvollzogen werden können, wie konkurrierende Anforderungen oder Zielsetzungen (z. B. schnelle Anfrageverarbeitung vs. Speicherplatzminimierung) behandelt werden. | Konkurrierenden Anforderungen wird durch die Möglichkeit der Festlegung unterschiedlicher Kostenarten begegnet. Durch die Bestimmung zu optimierender und als Nebenbedingung zu berücksichtigender Kostenarten fließen alle Aspekte in die Optimierung ein. |
| Der gesamte Prozess sollte allerdings nicht vollautomatisch durchgeführt werden, sondern an definierten Punkten sollte der Entwickler gezielt eingreifen können, um hierdurch Wissen über die Domäne oder das konkrete Projekt einzubringen, die nicht im System modelliert worden sind. | Der Entwickler kann (neben der Konfiguration) an zwei Stellen in den Ablauf eingreifen: Zum einen kann er am Ende der Auswahlphase auf jeden Fall zu realisierende Maßnahmen als „gesetzt“ markieren, zum anderen kann er vor der endgültigen Implementierung den gesamtem Entwurfsschritt unter neuen Nebenbedingungen, wie z. B. mehr Speicherplatz, wiederholen. |
| Der Fortschritt des gesamten Vorgangs des physischen Entwurfs sollte zur Dokumentation und Nachvollziehbarkeit in einem Repository abgespeichert werden. | Die Basis zur Erfüllung dieser Anforderung ist durch das Framework gegeben; werden alle Eingaben und die Resultate der einzelnen Schritte mitprotokolliert, so können diese Angaben zu Dokumentationszwecken und zur Nachvollziehbarkeit bei einer vorliegenden Implementierung im Repository gespeichert werden. |

Tabelle 10.3: Anforderungen an die physische Optimierung und ihre Umsetzung im Framework

# Teil III

# Implementierung und Evaluation

# Überblick

Die in Teil II vorgestellten Entwurfsmethodik wurde im Rahmen des OFFIS–Projektes ODAWA (OFFIS Tools for Data Warehousing) in Form eines Prototypen softwaretechnisch umgesetzt.
Das dabei entstandene Entwurfswerkzeug trägt den Projektnamen, seine Konzeption und einige Architektur– und Implementierungsdetails werden in Kapitel 11 beschrieben.
Um die Anwendbarkeit der Methodik und des Werkzeugs ODAWA zu zeigen, werden in Kapitel 12 beide anhand eines realen Beispiels evaluiert. Als Anwendungsdomäne wurde dabei das EKN (Epidemiologisches Krebsregister Niedersachsen) gewählt. Das im Zuge dieser Evaluation entstandene DWH sowie die exemplarisch entwickelten, auf dem DWH aufsetzenden Applikationen werden unter dem Begriff ODAWA@EKN zusammengefasst.

# Kapitel 11

# ODAWA: Eine Implementierung der Entwurfsmethodik

In diesem Kapitel wird ODAWA, die Implementierung der in Teil II vorgestellten Entwurfsmethodik, beschrieben. Zunächst werden in Abschnitt 11.1 einige Aspekte zur Konzeption des Werkzeugs vorgenommen. Die aus diesen Überlegungen resultierende Architektur wird in Abschnitt 11.2 vorgestellt. Abschnitt 11.3 nennt eingesetzte Sprachen sowie Werkzeuge und skizziert die Realisierung. Abschnitt 11.4 vermittelt anhand einiger Bildschirmfotografien einen Eindruck von der Benutzungsschnittstelle. Das Kapitel endet mit einer Zusammenfassung in Abschnitt 11.5.

## 11.1 Konzeption

Abbildung 11.1 skizziert die Konzeption des Werkzeugs ODAWA, das die in den Kapiteln 6 bis 10 vorgestellte Methodik implementiert.

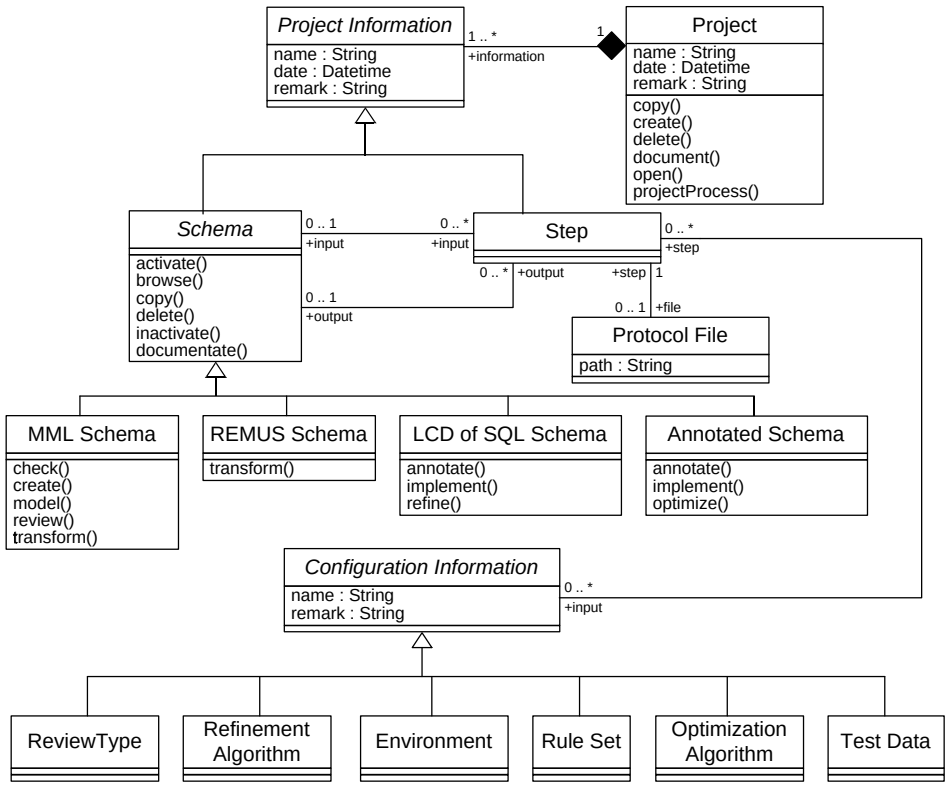



Abbildung 11.1: Konzeption des Werkzeugs ODAWA

Zentraler Einstiegspunkt ist hierbei ein Projekt (Klasse *Project*), das eine Menge von Projektinformationen (Metaklasse *Project Informationen*) umfasst. Diese setzen sich aus Schemata (Klasse *Schema*) und Prozessschritten (Klasse *Step*) zusammen, wobei jede Projektinformation durch eine Bezeichnung, ein Datum und eine Bemerkung gekennzeichnet ist.
Neben den durch ihren Namen selbsterklärenden Methoden *copy*, *create*, *delete*, *document* und *open* ist für ein Projekt auch die Methode *projectProcess* definiert, die als Resultat den Ablauf des Projektes durch Darstellen der einzelner Schemata und der Prozessschritte zwischen diesen dokumentiert. Ein Beispiel zeigt Abbildung 11.2: Nach Projektstart wurde zunächst das konzeptionelle Schema „$K_1$“ erzeugt, auf diesem zwei Reviews durchgeführt, welche über das Schema „$K_2$“ zum Schema „$K_3$“ geführt haben. Dieses ist dann zum Schema „$K_3'$“ dupliziert worden. Dadurch wurden zwei alternative Entwicklungspfade verfolgt, wobei der obere nach der Transformation in das logische (REMUS–)Schema „$L_1$“ und das physische (*LCD of SQL*–)Schema „$P_1$“ beendet wurde. Der alternative Entwicklungsweg wurde bis zur Verfeinerung in das physische Schema „$P_3$“ verfolgt und dann ebenfalls beendet. Das hierbei zwischenzeitlich entstandene Schema „$P_2$“ wurde nach „$P_2'$“ kopiert.

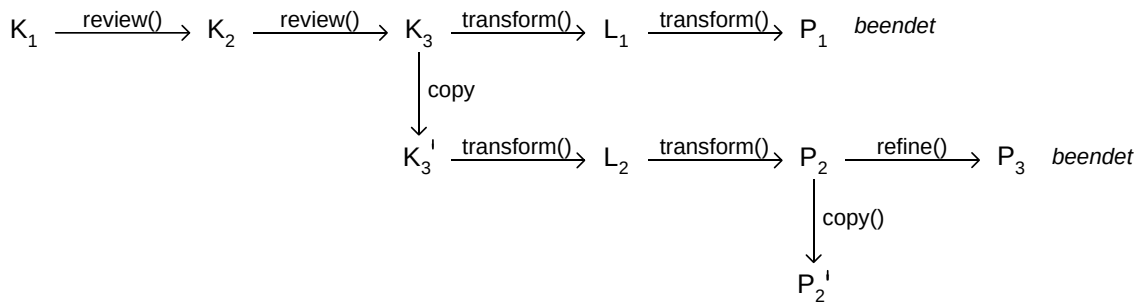



Abbildung 11.2: Resultat der Methode *projectProcess()* der Klasse *Process*

Jeder Prozessschritt kann jeweils ein Schema als Ein– und Ausgabe referenzieren (Referenzen *input* und *output* auf die Metaklasse *Schema*) sowie optional auf eine Protokolldatei verweisen (Referenz *file* auf die Metaklasse *Protocol File*). In der Protokolldatei werden Detailinformationen des konkreten Prozessschrittes festgehalten.
Schließlich kann ein Prozessschritt mehrere Konfigurationen (Objekte der Metaklasse *Configuration Information*) als Eingabe erhalten. Dabei bestehen die in Tabelle 11.1 dargelegten Abhängigkeiten zwischen den Schematypen der Ein– und Ausgabe und zulässigen Verweisen auf Konfigurationen.

| **Referenzierter Schematyp: Eingabe** | **Referenzierter Schematyp: Ausgabe** | **Zulässige Referenzen auf die Klasse** *Configuration Information* |
|---|---|---|
| MML–Schema | MML–Schema | ReviewType |
| *LCD of SQL*–Schema | *LCD of SQL*–Schema | Refinement Algorithm |
| Annotated Schema | Annotated Schema | Environment, Optimization Algorithm, Rule Set |
| Annotated Schema | — | Test Data |

Tabelle 11.1: Metaklasse *Step*: Abhängigkeiten zwischen referenzierten Schemata und Konfigurationen

Jedes *Schema*–Objekt wird genau wie ein Projekt durch die Attribute *name*, *date* und *remark* beschrieben und besitzt die Methoden *browse* zum Navigieren der Elemente des Schemas, *copy* zum

Kopieren, *delete* zum Löschen, *document* zum Dokumentieren und *(in)activate* zum (De–)aktivieren eines Schemas. Diese letzten beiden Methoden bewirken das in Abbildung 11.2 dargestellte Beenden von Entwicklungspfaden bzw. die Wiederaufnahme eines solchen. Die abstrakte Metaklasse *Schema* besitzt die vier Spezialisierungen *MML Schema* (siehe Abschnitt 6.1), *REMUS Schema* (siehe Abschnitt 8.3), *LCD of SQL Schema* (siehe Abschnitt 8.2.1) und *Annotated Schema* (siehe Abschnitt 10.2). Neben den jeweiligen Objekten, die in den entsprechenden Abschnitten von Teil II definiert sind, besitzt jede dieser Klassen spezielle Methoden, deren Bedeutung in Tabelle 11.1 erläutert werden.

| **Klasse** | **Methode** | **Bedeutung** |
|---|---|---|
| MML–Schema | check() | Führt eine Überprüfung auf Korrektheit des Schemas bez. des in Abschnitt 6.1 definierten Metaklassendiagramms samt Nebenbedingungen sowie der in Abschnitt 6.1.8 definierten Wohlgeformtheitseigenschaften durch. |
| | create() | Legt ein neues Schema an. |
| | model() | Ermöglicht das Bearbeiten eines Schemas im graphischen Editor. |
| | review() | Ermöglicht das Erfassen der Ergebnisse eines Schemareviews. |
| | transform() | Stößt die in Abschnitt 7.2 definierte Transformation zur Erzeugung eines REMUS–Schemas an. |
| REMUS–Schema | transform() | Führt die in Abschnitt 7.2 definierte Transformation zur Erzeugung eines *LCD of SQL*–Schemas aus. |
| *LCD of SQL*–Schema | annotate() | Überführt ein *LCD of SQL*–Schema in ein annotiertes Schema. |
| | implement() | Realisiert das Schema in Form eines DB–Skriptes oder über eine Programmierschnittstelle. |
| | refine() | Transformiert ein *LCD of SQL*–Schema in ein anderes *LCD of SQL*–Schema mit Hilfe eines Verfeinerungsalgorithmus. |
| Annotated Schema | annotate() | Ermöglicht das Manipulieren von Annotationen des Schemas. |
| | implement() | Realisiert das Schema in Form eines DB–Skriptes oder über eine Programmierschnittstelle. |
| | optimize() | Führt den in Abschnitt 10.2.7 beschriebenen Optimierungsprozess durch. |

Tabelle 11.2: Methoden der verschiedenen Schematypen

## 11.2 Architektur

Die Überlegungen aus Abschnitt 11.1 haben zu der in Abbildung 11.3 dargestellten Architektur geführt.

Im Back End–Bereich ist das Repository angesiedelt, in dem alle Projektinformationen persistent abgelegt werden. Darauf aufbauend basiert die eigentliche Applikation, die sich ihrerseits in vier Schichten unterteilen lässt: Die mittlere Schicht ist die Objektschicht, in der die in Teil II vorgestellten Metamodelle realisiert sind. Auf dieser Schicht ist eine Exportschnittstelle angesiedelt, die es ermöglicht, Schemata eines jeden Metamodells im XML–Format (Extensible Markup Language) zu exportieren. Somit wird einerseits die Erzeugung einer Dokumentation der Schemata ermöglicht, andererseits dient diese Schnittstelle zur Kommunikation mit anderen Werkzeugen eines DWS und

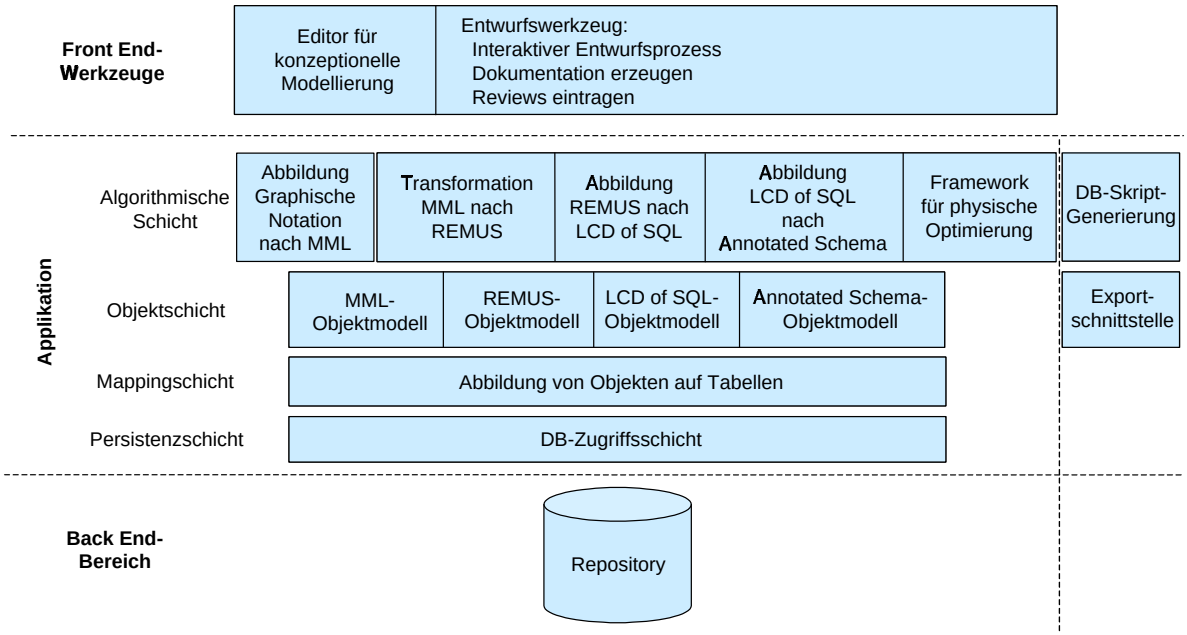



Abbildung 11.3: Architekturübersicht

erleichtert die Integration des Entwurfswerkzeugs in ein DWS. Im Sinne der in Abschnitt 4.4.2 vorgenommenen Klassifikation der Metadatenhaltung in einem DWS unterstützt das Werkzeug damit den *Shared Repository*–Ansatz: Das Werkzeug hält alle seine Metadaten im lokalen Repository, durch die Exportschnittstelle sind die Anbindung und der Austausch an ein zentrales Repository im DWS möglich.
Die unterhalb der Objektschicht angesiedelte Mappingschicht sorgt für die Abbildung dieser objektorientierten Modelle auf die Speicherstrukturen des Repository. Diese Mappingschicht greift jedoch nicht direkt auf das Repository zu, vielmehr wird dieses durch eine weitere DB–Zugriffsschicht gekapselt. Die oberhalb der Objektschicht liegende algorithmische Schicht realisiert die Transformationen und Abbildungen zwischen den verschiedenen Metamodellen und stellt das Framework für die physische Optimierung zur Verfügung. Ebenso ist in dieser Schicht eine Komponente zu finden, die die Abbildung zwischen der graphischen Repräsentation und der MML als Sprache der konzeptionellen Modellierungsebene vornimmt.
Die Front End–Komponenten setzen sich aus einem graphischen Editor zur Modellierung der konzeptionellen Schemata und dem Entwurfswerkzeug im engeren Sinne zusammen. Dieses bietet eine Reihe von Masken an, die den Dialog mit dem Benutzer ermöglichen, um z. B. Interaktionen im Entwurfsprozess vorzunehmen, Dokumentationen zu erzeugen oder die Resultate von Reviews zu verwalten.

## 11.3 Konkrete Umsetzung

Als Repository kam der Microsoft SQL Server 2000 zum Einsatz. Diese Wahl ist durch die Anlehnung der Metamodelle an das OIM und die ursprünglich mit dem SQL Server 2000 ausgelieferte Implementierung des OIM begründet. Als graphischer Editor zur konzeptionellen Modellierung wird die Professional Edition von Rational Rose98 in der Version 4.5 eingesetzt, weil dieses Werkzeug in der UML–Modellierung etabliert ist, inklusive der für die $_m$UML benötigten Erweiterungsmechanismen. Diese Funktionalität sollte im Projekt ausgenutzt werden, anstatt die Eigenentwicklung eines Modellierungseditors „auf der grünen Wiese“ beginnend, durchzuführen. Als Sprache zur Implemen-

tierung bot sich aufgrund der objektorientierten Spezifikation der Metamodelle und aus Gründen eines sauberen Software Engineerings eine objektorientierte Sprache an, so dass C++ und Java in Betracht kamen. Aufgrund der zu realisierenden Schnittstellen zum Repository und zu Rational Rose wurde schließlich Microsoft Visual C++ in Version 6.0 mit der Microsoft Visual Studio–Umgebung gewählt. Entwicklungsplattform war dementsprechend Microsoft Windows NT. Grundlage für die Anbindung des SQL Server als Repository ist die Standardschnittstelle ODBC, durch die die Unabhängigkeit der Implementierung von einem spezifischen DBMS gewährleistet wird.
Bei der Implementierung ergeben sich aufgrund der Architektur aus Abbildung 11.3 verschiedene Themenbereiche, die jeweils eine separate Bibliothek bilden:

- *GUI* enthält die Masken der Front End–Komponente.
- *MML, Remus, LCDofSQL* und *AnnotatedSchema* implementieren die entsprechenden Metamodelle sowie die Abbildung auf Tabellen[1]. Die Bibliothek *LCDofSQL* enthält zusätzlich eine Methode *refinement*, die die Umstrukturierung eines *LCD of SQL*–Schemas gemäß den Anforderungen des Zielsystems ermöglicht.
- *MML2REMUS* bietet die für die Transformation eines MML–Schemas in ein REMUS–Schema notwendigen Abbildungen.
- *REMUS2LCD* ermöglicht die für die Transformation eines REMUS–Schemas in ein *LCD of SQL*–Schema notwendigen Abbildungen.
- *LCD2Annotated* beinhaltet die für die Umspeicherung eines *LCD of SQL*–Schemas in ein annotiertes Schema notwendigen Abbildungen.
- *PhysFramework* realisiert den Prozess der physischen Optimierung[2].
- *MMLmUML* bietet Methoden für die Überprüfung von $_m$UML–Diagrammen und deren Konvertierung in MML–Diagramme (Schnittstelle zwischen graphischer Präsentation und Sprache auf der konzeptionellen Ebene).
- *Rose* dient der Herstellung einer OLE (Object Linking and Embedding)–Verbindung zu Rational Rose und enthält durch Rose zur Verfügung gestellte Kapselungsklassen für den Zugriff auf Rose–Modelle und –Modellelemente.
- *Database* dient der Kapselung des Datenbankzugriffs.
- *Utility* enthält Hilfsklassen, die der Unterstützung der übrigen Komponenten dienen. Beispielsweise ist hier eine Smart Pointer–Implementierung realisiert.

Ergänzend werden in der Implementierung folgende Fremdbibliotheken genutzt:

- Fast durchgehend werden von allen Komponenten Klassen der STL (*Standard Template Library*) [Jos96] verwendet.
- Zur persistenten Speicherung verwenden die Bibliotheken MML, Remus, LCDofSQL und AnnotatedSchema sowohl *ODBC* (Open Database Connectivity) als auch *OLE DB (ADO)* (Object Linking and Embedding Database (Access Data Objects))[3] [Woo99].

[1] Das Zusammenfassen der Objekt– und Mappingschicht ist mit dem geringeren Implementierungsaufwand zu erklären.
[2] Im aktuellen Prototypen werden lediglich die in Abschnitt 10.4 vorgestellten Elemente bez. Regel, Umgebung, Optimierungsalgorithmus zur Verfügung gestellt.
[3] Der parallele Einsatz beider Techniken erklärt sich durch den im Laufe des Projektes inkrementell entstandenen Prototypen.

- Die Bibliothek *GUI* nutzt zur Implementierung der Dialoge die *MFC (Microsoft Foundation Classes)* [Jon99].

Die Abhängigkeiten der einzelnen Bibliotheken sind in Abbildung 11.4 skizziert. Ein gestrichelter Pfeil von „A“ nach „B“ bedeutet „A nutzt B“, weiße Elemente bezeichnen selbsterstellte Teile, grau hinterlegte verwendete Fremdbibliotheken, –techniken und –produkte.

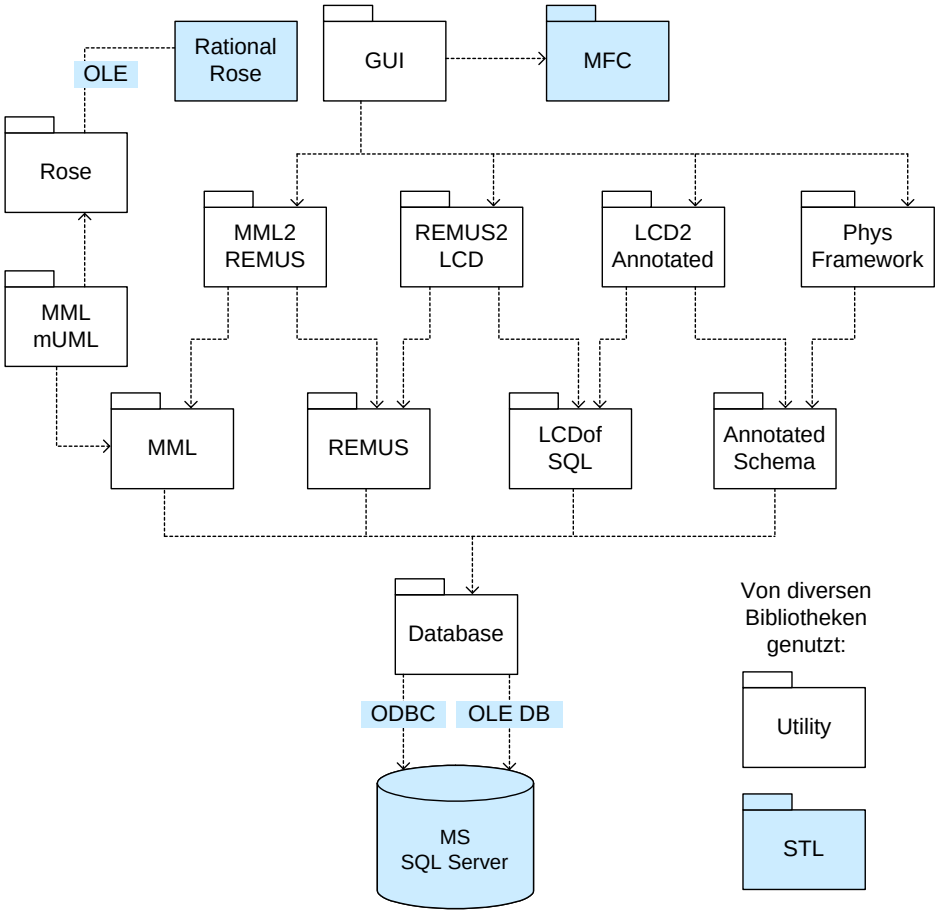



Abbildung 11.4: Abhängigkeiten einzelner Bibliotheken

Bei der relationalen Gestaltung des Repository wurde für die einzelnen Klassendiagramme jede Metaklasse auf eine eigene Tabelle abgebildet. Grundgedanke dieser Vorgehensweise ist die einfachere Erweiterbarkeit: Sowohl Attribut– als auch Beziehungsergänzungen der Metamodelle bleiben damit auf die Änderung einer Tabelle beschränkt.
Bei der Einbindung von Rational Rose als graphischen Modellierungseditor wurde hauptsächlich der Erweiterungsmechanismus zur Bereitstellung von Stereotypen und Elementeigenschaften genutzt, so dass die in Abschnitt 6.2 vorgestellten Modellierungskonstrukte der $_m$UML zur Verfügung stehen. Darüber hinaus kann Rational Rose basierend auf einem *AddIn*–Mechanismus mit zusätzlicher Programmfunktionalität ausgestattet werden. Dies wurde beispielsweise ausgenutzt, um auf einem konzeptionellen Schema die Methode *check* (siehe Abbildung 11.1 und Tabelle 11.2) zu realisieren, die die Einhaltung der $_m$UML–Notation überprüft. Abschließend werden in Tabelle 11.3 die Anzahl der Klassen und die Summe der Quellcodezeilen aufgelistet, um einen Eindruck von der Komplexität der Software zu vermitteln.
Bei einigen Bibliotheken wird keine Angabe gemacht, weil diese im Prototypen nur rudimentär realisiert worden sind, d. h. auf eine saubere Klassenstruktur und Implementierung der Methoden

wurde weniger geachtet als auf einen Nachweis der prinzipiellen Umsetzbarkeit der Konzepte. Dadurch sind die Quantitätsangaben mit denen anderer Bibliotheken nicht vergleichbar und würden zu einem „schiefen“ Bild führen.

| **Bibliothek** | **Anzahl Klassen** | **Anzahl Methoden** | **Anzahl Quellcodezeilen** | |
|---|---|---|---|---|
| | | | **Generiert** | **Hand–codiert** |
| GUI | Im Prototyp nur rudimentär realisiert. | | | |
| MML | 37 | 717 | 13497 | 13497 |
| Remus | 32 | 428 | 8246 | 8246 |
| LCDofSQL | 33 | 410 | 9321 | 9321 |
| AnnotatedSchema | Im Prototyp nur rudimentär realisiert. | | | |
| MML2REMUS | 24 | 184 | 4705 | 4705 |
| REMUS2LCD | 31 | 271 | 2297 | 2297 |
| LCD2Annotated | Im Prototyp nur rudimentär realisiert. | | | |
| PhysFramework | Im Prototyp nur rudimentär realisiert. | | | |
| MMLmUML | 3 | 105 | 3332 | 3332 |
| Rose | 5 | 22 | 32321 | 125 |
| Database | 4 | 46 | 1091 | 1091 |
| Utility | 9 | 46 | 970 | 970 |
| **Summe** | **158** | **2229** | **75780** | **43584** |

Tabelle 11.3: Umfang der Implementierung

## 11.4 Graphische Benutzungsoberfläche

Die in Abschnitt 6.2 vorgestellten Erweiterungen der UML zur $_m$UML wurden in Rational Rose realisiert, die graphische Modellierung hiermit ist in Abbildung 11.5 dargestellt. Beispielsweise stehen die neuen Stereotypen *FactClass* und *DimensionalClass* zur Modellierung zur Verfügung.

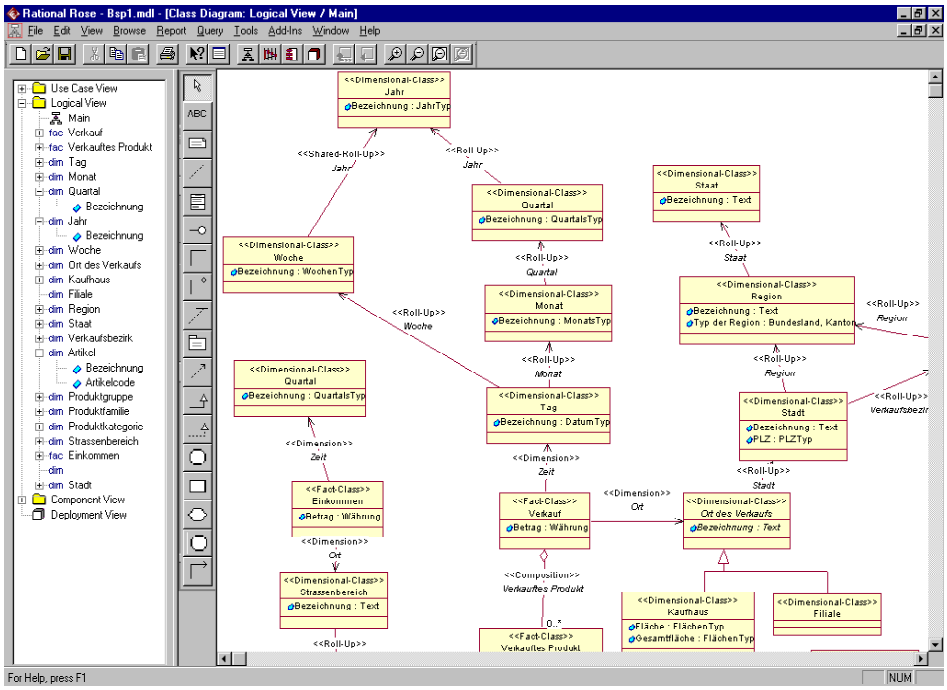



Abbildung 11.5: Graphische konzeptionelle Modellierung mit erweitertem Rational Rose

Die zusätzliche Programmfunktionalität, die über den am Ende des letzten Abschnitts erwähnten *Add In*–Mechanismus zur Verfügung gestellt wird, wird über erweiterte Menüeinträge aktiviert, wie in Abbildung 11.6 dargestellt.

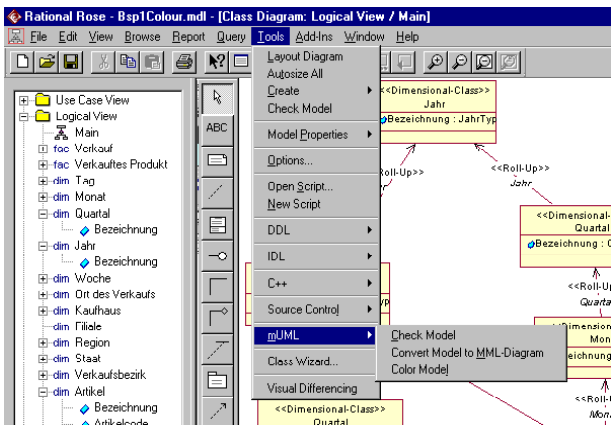



Abbildung 11.6: Um $_m$UML –Unterstützung ergänztes Rational Rose–Menü

Das GUI an sich besitzt als zentralen Einstiegspunkt die sog. Projektstartseite (siehe Abbildung 11.7), die alle im bisherigen Projektverlauf angefallenen Informationen anzeigt. Von hier aus sind die diversen im Metamodell aus Abbildung 11.1 definierten Methoden aufrufbar.

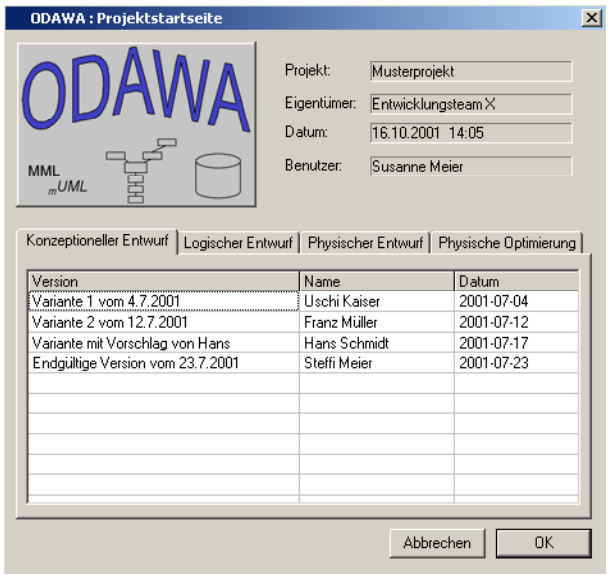



Abbildung 11.7: Projekt–Startseite

Bei den einzelnen Dialogen wurde darauf geachtet, dass der Benutzer durch Visualisierung stets über den aktuellen Entwurfsschritt informiert ist, wie die in Abbildung 11.8 dargestellte Maske verdeutlicht. Im linken Teil des Formulars wird der Benutzer durch die Hinterlegung darauf hingewiesen, in welchem Entwurfsschritt der Methodik er sich gerade befindet, im rechten Teil wird durch das Fortschreiten der Hinterlegung der Fortschritt des momentan ablaufenden Transformationsalgorithmus verdeutlicht.

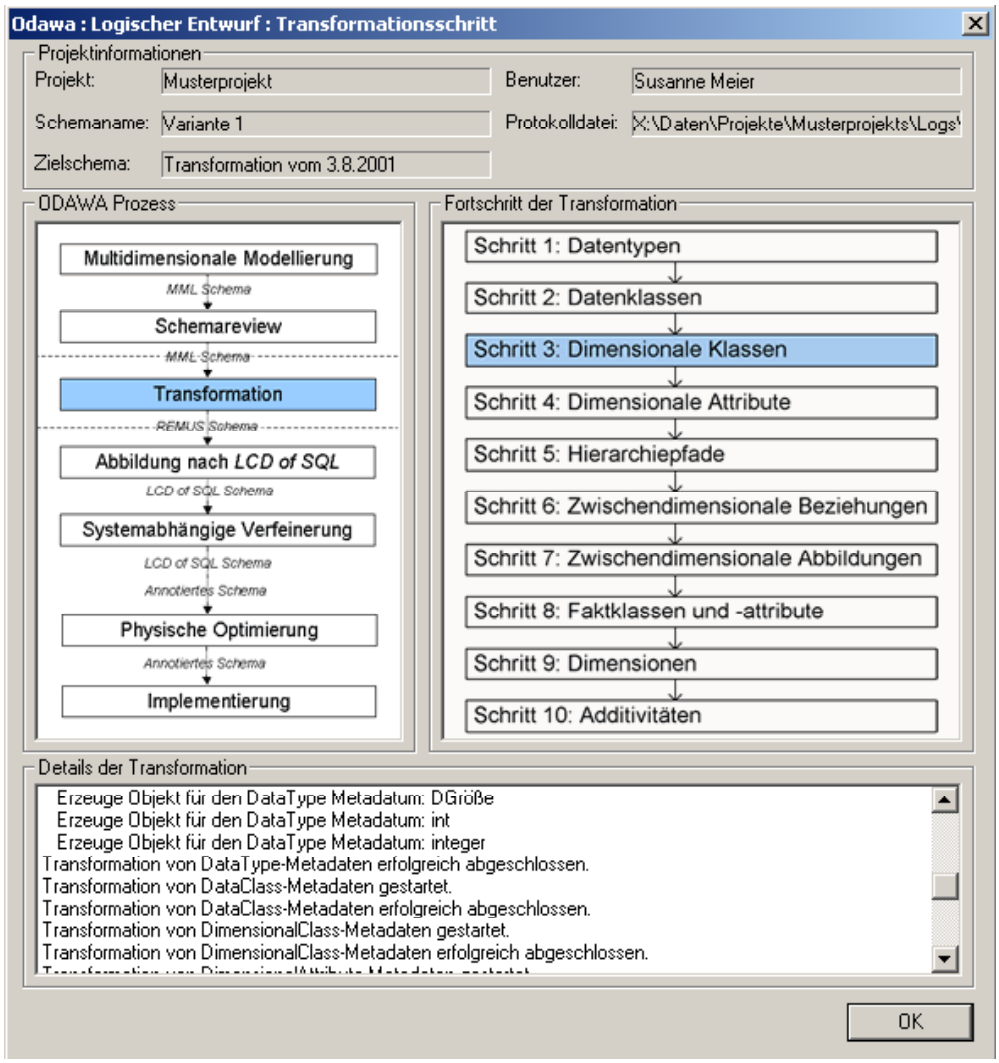



Abbildung 11.8: Masken der Applikation

Außerdem existieren Dialogformulare, die Benutzerinteraktion zulassen. Dies tritt an den Stellen auf, an denen während der Transformationen in Teil II eine deterministische Funktion $f_{det_{<X>}}$ definiert worden ist. Beispielhaft ist in Abbildung 11.9 die Auswahl des Namens dimensionaler Relationen bei der Transformation eines MML–Schemas in ein REMUS–Schema zu sehen.

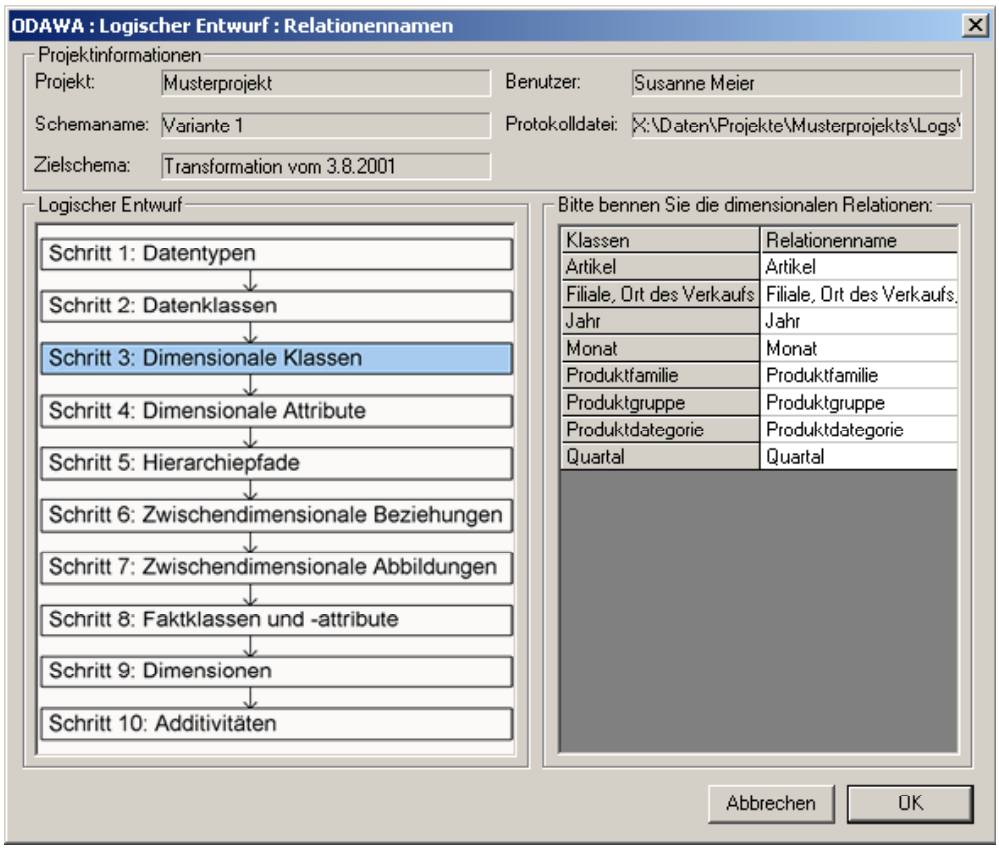



Abbildung 11.9: Beispieldialog für Interaktion

## 11.5 Zusammenfassung

Dieses Kapitel hat die prototypische Implementierung der Entwurfsmethodik beschrieben. Die resultierende Software trägt mit ODAWA den Namen des OFFIS–Projektes, in das die Forschungsarbeiten eingebettet waren.
Zunächst wurden in Abschnitt 11.1 die Metamodelle und Transformationen unter dem Begriff des Projektes zusammengefasst sowie Methoden für die einzelnen Komponenten festgelegt. Diese Konzeption wurde in 11.2 in eine Architektur umgesetzt, deren konkrete Realisierung in Abschnitt 11.3 skizziert wurde. Weiterführende Implementierungsdetails können in [Har99b] und vertiefende Programmdokumentation in [MHH01] nachgelesen werden. Abschnitt 11.4 vermittelt dem Leser anhand einiger Bildschirmfotografien einen Eindruck der graphischen Benutzungsoberfläche des Prototypen. Die komplette Oberfläche ist im Benutzerhandbuch [HHM01] beschrieben.

# Kapitel 12

# Evaluation

Dieses Kapitel fasst eine in [HK01] ausführlicher beschriebene Evaluation der in Teil II dieser Arbeit konzipierten Methodik zusammen. Als Domäne für die Evaluation wurde das im OFFIS und damit in einer unmittelbaren Arbeitsumgebung betriebene EKN (Epidemiologische Krebsregister Niedersachsen) gewählt. Die Vorstellung von Zielen, Aufbau und eingesetzter Software im EKN erfolgt in Abschnitt 12.1. Die eigentliche Evaluation, d. h. das Durchlaufen des Entwurfsprozesses, wird in Abschnitt 12.2 beschrieben. Ein Überblick über das entstandene System, das mit ODAWA@EKN bezeichnet wird, wird in Abschnitt 12.3 gegeben. In Abschnitt 12.4 werden einige weitere Aspekte mit Einfluss auf die Methodik diskutiert. Abschnitt 12.5 fasst die wichtigsten Erkenntnisse der Evaluation zusammen.

## 12.1 Epidemiologisches Krebsregister Niedersachsen

Dieser Abschnitt beschreibt das EKN [AFH+97, ABH+98, EKN01], wobei in 12.1.1 zunächst die grundlegenden Ziele widergegeben werden. Abschnitt 12.1.2 schildert die Konzeption des EKN, bevor in 12.1.3 Informationen über Mengengerüste und erfasste Daten gegeben werden. Abschnitt 12.1.4 stellt mit den CARTools (Cancer Registry Tools) die im EKN eingesetzte Software vor.

### 12.1.1 Ziele

Hauptaufgabe des EKN ist die epidemiologische Auswertung möglichst vollständig erfasster Krebsneuerkrankungen und –sterbefälle in Niedersachsen. Dabei verfolgte Ziele sind u. a. die Schätzung von Inzidenz– und Mortalitätsraten, die Beobachtung von zeitlichen Trends bei diesen Raten, die Identifikation von Teilregionen oder Zeitperioden mit auffälligen Inzidenzraten sowie die Bereitstellung einer Datengrundlage für epidemiologische Studien.

### 12.1.2 Struktur und Meldewege

Das EKN gliedert sich entsprechend dem 1994 im Gesetz über Krebsregister (Krebsregistergesetz – KRG) festgeschriebenen Konzept in zwei räumlich und organisatorisch voneinander getrennte Stellen: Eine *Vertrauensstelle*, bei der die Krebsmeldungen des Landes eingehen und für die Bearbeitung nur temporär gespeichert werden, sowie eine *Registerstelle*, die von der Vertrauensstelle verschlüsselte Krebsmeldungen erhält und zu einem bevölkerungsbezogenen, epidemiologischen Krebsregister — unter Einbeziehung weiterer Datensätze aus Totenscheinen, Pathologenmeldungen u.a. — verdichtet und auswertet.

Im EKN wird im Regelfall nach der *Einwilligungslösung* erfasst, d. h. der Melder holt von seinem Patienten dessen Einwilligung ein und meldet anschließend den Krebsfall an die Vertrauensstelle des EKN. Diese erfasst und codiert die eingehende Meldung, prüft diese auf Vollständigkeit und führt weitere qualitätssichernde Maßnahmen durch. Bei Unklarheiten kann sich die Vertrauensstelle direkt an den Melder wenden. Anschließend werden die Meldungen anonymisiert. Dazu werden die personenidentifizierenden Angaben verschlüsselt und sog. Kontrollnummern gebildet, die eine eindeutige Zuordnung der Meldung zu eventuell bereits vorhandenen Meldungen im Krebsregister ermöglichen, jedoch keine Zuordnung zum Patienten gestatten. Anhand dieser Kontrollnummern wird in der Registerstelle die Meldung mit dem bereits vorhandenen Datenbestand abgeglichen. Handelt es sich um eine Neumeldung, so wird sie in den Datenbestand eingetragen. Gibt es bereits Meldungen zu dem Patienten, so findet eine Aktualisierung der bereits vorhandenen Meldungen statt. Die Meldungen werden nach Abschluss der Bearbeitung in der Registerstelle, jedoch spätestens drei Monate nach Eingang in der Vertrauensstelle, gelöscht. Anhand der verschlüsselten personenidentifizierenden Angaben kann in speziell genehmigten Fällen (z. B. für Forschungsvorhaben) unter Verwendung des bei einer dritten Stelle aufbewahrten geheimen Schlüssels eine Entschlüsselung von Meldungen in der Vertrauensstelle vorgenommen werden.
Neben dem Meldeweg mit Einwilligung besteht im EKN weiterhin die Möglichkeit, ausschließlich diagnostizierende oder begutachtende Ärzte ohne direkten Patientenkontakt (z. B. Pathologen) direkt in das EKN einzubeziehen. Dazu wird die Krebsmeldung beim Melder in einen personenidentifizierenden und einen epidemiologischen Anteil zerlegt, wobei beide Teile mit einer eindeutigen Kennung versehen werden. Der personenidentifizierende Teil wird an die Vertrauensstelle gesendet, die daraus die Kontrollnummern generiert und diese an die Registerstelle weiterleitet. Diese bekommt vom Melder direkt die epidemiologischen Angaben, kann die Kontrollnummern anhand der eindeutigen Kennung hinzufügen und somit die Meldung in den Datenbestand integrieren. Ein entscheidender Unterschied zum Meldeweg mit Einwilligung liegt darin, dass Meldungen von ausschließlich diagnostizierenden oder begutachtenden Ärzten für z. B. Forschungsvorhaben nicht rückverschlüsselt werden können.

### 12.1.3 Basiszahlen

Die folgenden Zahlenangaben sollen die behandelten Mengengerüste verdeutlichen:

- Einwohner im Meldebereich Niedersachsen: ca. 7.300.000
- Erwartete Neuerkrankungen jährlich: ca. 40.000 (etwa 500 Fälle pro 100.000 Einwohner)
- Todesfälle an Krebs jährlich: ca. 21.000 (25 bis 30% aller Todesfälle)
- Erwartete Meldungen jährlich: mit sog. „Mehrfachmeldungen“ über verschiedene Meldewege ca. 60.000 klinische Meldungen, 80.000 sog. Pathologenmeldungen und 87.000 Todesbescheinigungen.

Historie und Zukunft des EKN lassen sich grob anhand von drei Phasen beschreiben. In einer *Pilotphase* in den Jahren 1993/94 wurde der Nachweis der prinzipiellen Funktionalität eines Landeskrebsregisters nach dem im KRG beschriebenen Konzept der Krebsregistrierung gezeigt. Ebenso wurde die Entwicklung eines Standards für epidemiologische Krebsregister zur technischen Umsetzung konkreter Verfahren vorgenommen, die zur Realisierung des im KRG enthaltenen Meldemodells sowie für die Durchführung eines bundesweiten Abgleichs der Krebsmeldungen (Kontrollnummern, Standardisierung der Erfassungsrichtlinien, Einwegverschlüsselungsverfahren) beitragen.
Es folgte eine *Erprobungsphase* in den Jahren 1995 bis 1999, deren Ziel die Umsetzung der in der Pilotphase entwickelten Verfahrensweisen in Niedersachsen war, um anschließend mit der *Dauerphase*

des EKN beginnen zu können. In diese Phase fallen neben der Erprobung der Datensammlung seit 1995 auch die dauerhafte Datenspeicherung seit Anfang 1997 sowie vor allem die Konzeption und Entwicklung der zur Unterstützung der Arbeit notwendigen Softwarewerkzeuge, der in Abschnitt 12.1.4 skizzierten sog. *CARTools*.
Mit Inkrafttreten des Gesetzes über das Epidemiologische Krebsregister Niedersachsen zum 1.1.2000 begann die *Aufbauphase*. Das Gesetz sieht vor, dass das EKN in der ersten Aufbauphase zunächst systematisch Meldungen aus dem Regierungsbezirk Weser–Ems erfassen soll. Die vorgesehene landesweit flächendeckende Erfassung wird in drei weiteren Ausbaustufen bis zum 1.1.2003 umgesetzt: 2001 Regierungsbezirk Lüneburg, 2002 Regierungsbezirk Braunschweig und 2003 Regierungsbezirk Hannover.

### 12.1.4 CARTools: Die Softwarewerkzeuge im EKN

Zur Unterstützung der zuvor genannten Aufgaben ist beim Aufbau des EKN ein von OFFIS entwickelter, CARTools genannter Werkzeugkasten entstanden, der diese Anforderungen abdeckt. Der Werkzeugkasten enthält insgesamt vier Komponenten:

- CARAMEL (CARLOS – *A*ttaching *M*ultiple *E*xisting *L*ocal Registration Units)
- CARTRUST (CARLOS – *T*rusted *R*egistration *U*nit *S*oftware *T*ool)
- CARELIS (CARLOS – *Re*cord *Li*nkage *S*ystem)
- CARESS (CARLOS – *E*pidemiological and *S*tatistical Data Exploration *S*ystem)

Abbildung 12.1 zeigt das Zusammenspiel der Werkzeugkomponenten.

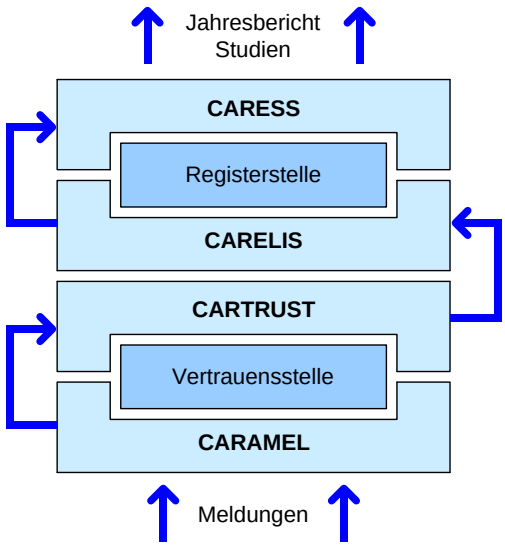



Abbildung 12.1: CARTools: Software im EKN

Während die Werkzeuge CARAMEL und CARTRUST in der Vertrauensstelle eingesetzt werden, um die dortigen Aufgaben zu unterstützen, bieten die Werkzeuge CARELIS und CARESS Software–Unterstützung für die Registerstelle.
Die Komponente CARAMEL ermöglicht die effiziente Anbindung heterogener EDV–Melder an das EKN. Die beiden Komponenten CARTRUST und CARELIS müssen aufeinander abgestimmt sein,

da zwischen ihnen permanent Transfers von z. B. Meldungen, Nachfragen oder Auskunfts– und Widerspruchangaben durchgeführt werden. Sie realisieren die durch das KRG geforderten datenschutzrechtlichen Aspekte, wie z. B. Anonymisierung der Meldungen, Kontrollnummerngenerierung sowie anonymisierten Meldungsabgleich mit Ergebnisaufbereitung. CARESS bildet abschließend das Auswertungssystem des EKN, mit dem die dauerhaft erfassten Meldungen epidemiologisch ausgewertet werden können.

## 12.2 Anwenden der Entwurfsmethodik

In diesem Abschnitt wird der Ablauf der Evaluation im engeren Sinne beschrieben, d. h. der in Teil II konzipierte Entwurfsprozess wird auf das EKN bezogen ausgeführt.

### 12.2.1 Anforderungsanalyse

Zum Feststellen der Anforderungen dienten Interviews mit Fachexperten (dies waren drei der in der Registerstelle tätigen Mitarbeiter) und die Analyse des aktuellen Datenschemas [RW99] sowie folgender Publikationen:

- „Krebs in Deutschland – Häufigkeiten und Trends“ [Arb99],
- „Krebs in Niedersachsen – Jahresbericht mit den Daten von 1996“ [EKN96] und
- „Morbidität und Mortalität an bösartigen Neubildungen im Saarland“ [Saa96].

Dabei wurde ein Anforderungskatalog in Form von natürlichsprachlichen Aussagen entwickelt. Ein Beispiel für die Ableitung solcher Aussagen aus den Dokumenten ist in Abbildung 12.2 zu sehen. Die korrespondierenden Aussagen sind in Tabelle 12.1 dargestellt.

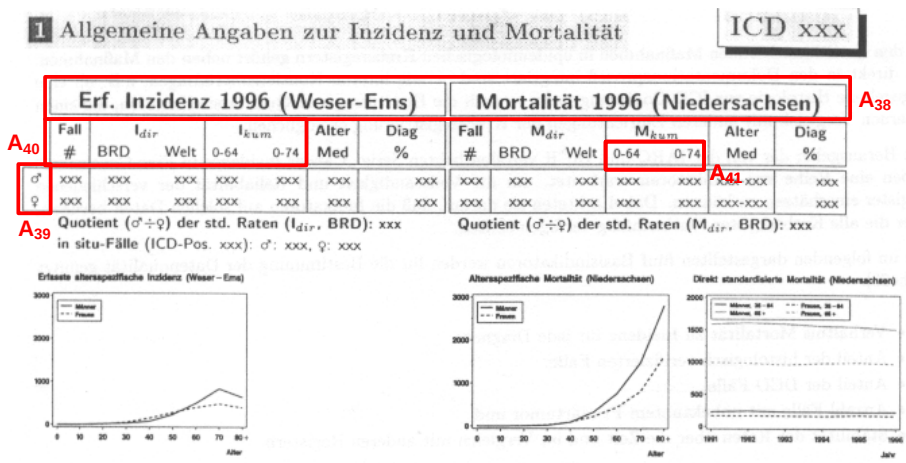



Abbildung 12.2: Beispiel für die Ermittlung von Aussagen

Die erste Version des Anforderungskataloges enthielt 117 Aussagen und wurde auf Vollständigkeit und Richtigkeit überprüft. Die überarbeitete Endversion des Anforderungskataloges enthält schließlich 125 natürlichsprachliche Aussagen.

| Lfd. Nr. | Aussage | Quelle |
|---|---|---|
| $A_{38}$ | Die Basisberichterstattung des EKN umfasst für einen Berichtszeitraum die erfassten Inzidenzen und Mortalitäten. | [EKN96], S.14/15 |
| $A_{39}$ | Sowohl erfasste Inzidenzen als auch Mortalitäten werden nach Geschlecht unterschieden. | [EKN96], S.15 |
| $A_{40}$ | Neben der reinen Fallzahl wird jeweils auch die direkt altersstandardisierte und eine kumulative Inzidenzrate gebildet. | [EKN96], S.15 |
| $A_{41}$ | Bei der kumulativen Inzidenrate werden die beiden Altersgruppen „0–64“, „0–74“ gebildet. | [EKN96], S.15 |

Tabelle 12.1: Aus dem EKN–Bericht abgeleitete Aussagen

### 12.2.2 Konzeptionelle Modellierung

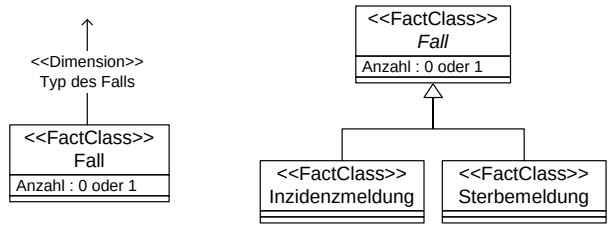
Basierend auf dem in der Anforderungsanalyse des letzten Abschnitts entwickelten Aussagenkataloges und dem 18–Schritt–Leitfaden zur Findung eines MML–Schemas aus Abschnitt 6.3 wurde das konzeptionelle Datenschema erstellt, das in Anhang B zu finden ist.

Das Schema weist folgende Charakteristika auf:

- Die zentrale Faktklasse „Fall“ besitzt mit 36 eine relativ hohe Dimensionalität.
- Es gibt viele Dimensionen mit nur einer Hierarchieebene.
- Von der Orts–Dimension der Vergleichsbevölkerung existieren dimensionale Abbildungen auf verschiedene Hierarchieebenen der Ortsdimension einer Meldung.
- Die für die Analyse bedeutendste Faktklasse „Fall“ hat nur ein numerisches Attribut, welches auch nur die Werte 0 und 1 annehmen kann.
- Zwischen der Faktklasse „Fall“ und der dimensionalen Klasse „Gebiet“ existieren vier Verbindungen, die unterschiedliche räumliche Aspekte des Faktes beschreiben.

Wesentliche Entwurfsentscheidungen während der konzeptionellen Modellierung waren:

- In der Ortshierarchie ergibt sich durch die kreisfreien Städte ein Problem, da diese sich nicht in die strenge Hierarchie einordnen lassen. Es liegt die in Abbildung 3.3 (Seite 21) beschriebene Struktur einer unbalancierten Hierarchie vor.
- Auf der feingranularsten Ebene der Ortshierarchie sind nicht nur Gemeinden zu verwalten, sondern allgemeiner „Gebiete“, was durch den Klassennamen und ein unterscheidendes Typ–Attribut zum Ausdruck kommt. Hintergrund ist die aus Datenschutzgründen in Datenanalysen notwendige Zusammenfassung mehrerer kleinerer Gemeinden, sowie umgekehrt der Wunsch, größere Gemeinden u. U. aufzusplitten.
- Zur Modellierung der Faktklasse „Fall“ boten sich die beiden in Abbildung 12.3 dargestellten Alternativen an: Eine Klasse und zusätzlich ein unterscheidendes Attribut bzw. eine Dimension zur Unterscheidung (Variante (a)) oder Darstellung der beiden unterschiedlichen Typen durch Vererbung (Variante (b)). Die Entscheidung fiel schließlich zugunsten von Variante (a), weil die zu erwartenden Datenanalysen doch relativ ähnlich sind.




(a) Modellierung als eine Klasse (b) Modellierung als Generalisierung

Abbildung 12.3: Alternative Darstellungsmöglichkeiten der Faktklasse Fall

- Das Schema besitzt zwar analyseorientierten Charakter, aber einige Aspekte aus dem alten Schema sind nicht in der gleichen Form realisiert, z. B. die „Darstellungsebene“, die Auswertungen wie „Niedersachsen auf Kreisebene“ zulässt.
  Dies ist eine Vermischung von Schema und Daten und kann wesentlich effizienter von einer Applikation oder der Datenbank mittels einer Abfrage realisiert werden, wie in Abbildung 12.4 skizziert.

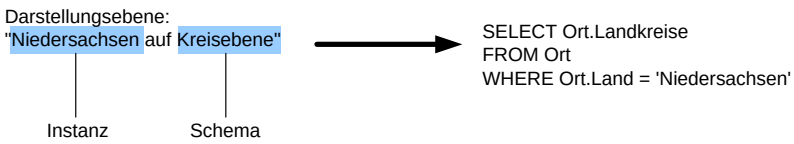



Abbildung 12.4: Darstellungsebene

Aufgrund der hohen Dimensionalität existieren viele Abhängigkeiten zwischen den Dimensionen. Sechs Abhängigkeiten ergeben sich durch die Modellierung des Falles als eine Faktklasse (z. B. dürfen Angaben zur Todeszeit nur bei Mortalitätsmeldungen auftreten), 25 bestehen zwischen Paaren von Dimensionen (z. B. können bestimmte Diagnosen nur bei einem bestimmten Geschlecht auftreten). Diese Abhängigkeiten wurden in der in Tabelle 12.2 dargestellten Form dokumentiert: Neben einer laufenden Nummer als eindeutiger Identifikator der Abhängigkeit wurden die betroffenen Dimensionen genannt, die medizinisch–epidemiologische Aussage verbal formuliert und schließlich eine Konsistenzregel in Aussagenlogik formuliert. Auf eine Abbildung dieser Aspekte in das konzeptionelle Schema wurde jedoch verzichtet, weil diese Aspekte bereits in der bestehenden DB berücksichtigt werden bzw. beim Transformations– und Ladeprozess behandelt werden.

| **Laufende Nummer**<br>P17 |
|---|
| **Betroffene Dimensionen**<br>Diagnose und Fernmetastasen |
| **Medizinisch–epidemiologische Aussage**<br>Metastasen dürfen nur bei malignen Erkrankungen auftreten. |
| **Konsistenzregel**<br>$Fernmetastasen.Bezeichnung = 1 \Rightarrow$<br>$Diagnose.Diagnose \geq C00 \wedge Diagnose.Diagnose \leq C76.$ |

Tabelle 12.2: Darstellung von Abhängigkeiten zwischen Dimensionen

### 12.2.3 Review des konzeptionellen Schemas

Im Laufe der konzeptionellen Modellierung wurde ein Review zur Qualitätssicherung nach dem in Abschnitt 6.4 beschriebenen Vorgehen durchgeführt. Als zu messende Kriterien wurden dabei fachliche Korrektheit, fachliche Konsistenz und Dokumentation ausgewählt. Die anderen in Tabelle 6.3 aufgeführten Kriterien sind im Kontext dieser Evaluation als nicht so relevant einzustufen, denn einerseits lag mit dem aktuellen EKN–Schema eine recht präzise Spezifikation als Ausgangsbasis vor, was das Messen von z. B. Schematiefe oder Vollständigkeit überflüssig macht. Andererseits soll die Evaluation „nur“ dem Nachweis der Anwendbarkeit der Methodik dienen, so dass z. B. Integrationsfähigkeit kein relevantes Qualitätskriterium für das Review darstellt.
Als Messverfahren und Metriken fanden für die drei ausgewählten Kriterien die in Tabelle 6.3 angegeben Vorschläge Verwendung. Bei der Dokumentation wurde auf Vollständigkeit und Qualität im Sinne von Aussagekraft der Kommentare geachtet. Zur Messung der fachlichen Korrektheit und Konsistenz wurde das konzeptionelle Schema in natürlichsprachliche Aussagen umgewandelt, wie in Abbildung 12.5 exemplarisch gezeigt, die dann überprüft wurden.

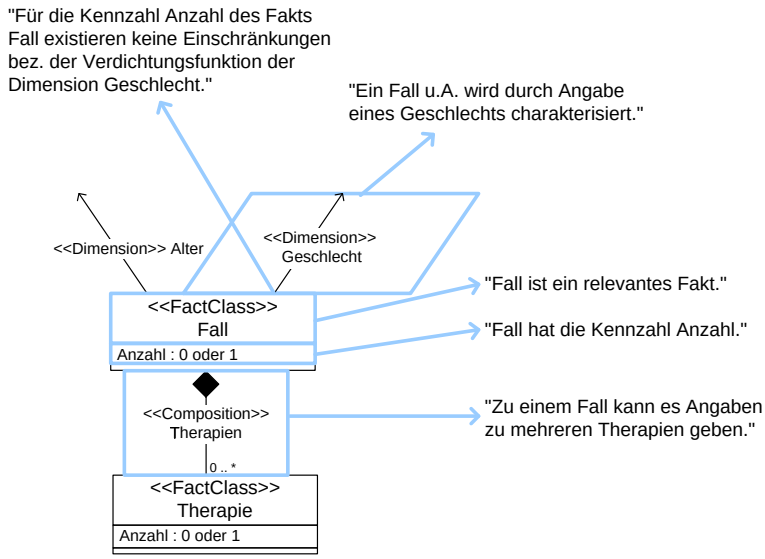



Abbildung 12.5: Review des konzeptionellen Schemas

Zu den gemessenen Werten lässt sich festhalten, dass schon aus der ersten Messung sehr gute Werte resultierten. Grund hierfür ist das Vorhandensein der in Abschnitt 12.2.1 erwähnten Quellen, die eine gute Spezifikation darstellen. Von einer solchen, nahezu optimalen Lage ist in anderen Projekten und Kontexten sicherlich nicht immer auszugehen. Möglicherweise werden die Anforderungen der zukünftigen Benutzer weniger klar sein oder es werden bei Beteiligung mehrerer Organisationseinheiten gar widersprüchliche Anforderungen aufgestellt. In solchen Fällen kann ein Review zur Findung des „richtigen“ Schemas beitragen.

### 12.2.4 Logischer Entwurf

Der in Abschnitt 7.2 vorgestellte Transformationsalgorithmus wurde ausgeführt, wobei an den interaktiven Stellen die sprechenden Namen der konzeptionellen Entwurfsebene beibehalten wurden. Eine wichtige Entwurfsentscheidung ist im achten Schritt bei der Transformation der Faktklassen zu treffen. Hier ist über die Auflösung der von der Faktklasse „Fall“ ausgehenden Kompositionen zu entscheiden, wobei Möglichkeit II (Übertragung der Dimensionen an die Detailklasse, siehe Seite 148) angewendet wurde. Diese Entscheidung begründet sich wie folgt:

- Möglichkeit I (Vernachlässigung der Komposition) würde zu einem Schema führen, das für die Datenanalyse wichtige Zusammenhänge, wie z. B. zwischen Tätigkeiten und Diagnosen, nicht mehr bieten würde.
- Möglichkeit III (Übertragung der Attribute) würde neben dem Defizit von Möglichkeit I lediglich zur Anzeige führen, dass jede Tätigkeit zu einem Fall gehört.
- Möglichkeit IV scheidet aus, da die gleichen Nachteile wie bei Möglichkeit III auftreten würden.

Bei der durch die Transformation erzielten Konstellation ist zu berücksichtigen, dass Auswertungen auf Tätigkeiten, Familienanamnesen und Therapien immer die entsprechenden Einträge berücksichtigen müssen. Wertet man beispielsweise auf dem Teilschema Tätigkeit aus und untersucht z. B. lediglich den Zusammenhang zwischen Orten, Diagnosen und Geschlecht, so erhält man falsche Ergebnisse, denn jeder Fall wird jetzt so oft mitgezählt wie es zugehörige Tätigkeiten gibt. Einerseits wird der die Datenanalyse vornehmenden Person soviel Domänenwissen unterstellt, dass die Analyse korrekt vorgenommen wird. Andererseits dient zur Vermeidung solcher Fehler das *Composition*–Metadatum. Dieses kann von den Analysewerkzeugen des Front End–Bereichs genutzt werden und den Benutzer auf z. B. falsche Verdichtungen hinweisen.
Zur Vermittlung eines Eindrucks über den Umfang des durch die Transformation entstandenen logischen Schemas gibt Tabelle 12.3 die Anzahl der erzeugten REMUS–Schemaelemente nach Typen geordnet an.

| **Schemaelementtyp** | **Anzahl** |
|---|---|
| **Objekte** | 74 |
| **Attribute** | 300 |
| **Kategorie A–Metadaten** | |
| AggregatedAttribute | 0 |
| Computation | 4 |
| ConceptualKey | 67 |
| Identifier | 0 |
| IdentifierValue | 0 |
| Multiplicty | 166 |
| ObjectType | 74 |
| Optional | 0 |
| PrimaryKey | 200 |
| Reference | 163 |
| Valid | 0 |
| **Kategorie B–Metadaten** | |
| Additivity | 268 |
| Association | 0 |
| Composition | 3 |
| Dimension | 163 |
| DimensionalMapping | 4 |
| RollUp | 10 |
| SharedRollUp | 0 |

Tabelle 12.3: Anzahl der erzeugten REMUS–Schemaelemente

### 12.2.5 Physischer Entwurf

Das im letzten Abschnitt erzielte logische Schema wird nun mittels der in Abschnitt 8.3 definierten Transformation in ein *LCD of SQL*–Schema überführt. Durch Anwendung der deterministischen Funktion $f_{det}$ wurden dabei Umlaute, Leerzeichen etc. eliminiert. Im ersten Schritt der Transformation, der die Abbildung der Datentypen realisiert, wurden die in Tabelle 12.5 aufgeführten Zuordnungen vorgenommen. Der erste Eintrag in der rechten Spalte jeder Zeile gibt dabei den Datentyp aus dem *Common Data Types*–Modell des OIM an (siehe Abschnitt 8.2.7), die weiteren Einträge in der rechten Spalte die entsprechenden Wertzuweisungen an die Variablen des *ColumnType*–Objektes im *LCD of SQL*–Schema.

**Abbildung der Datentypen**

| **REMUS** | ***LCD of SQL*** |
|---|---|
| „0 oder 1“ | ShortInt |
| „AbkürzungsTyp“ | String |
| „Wahr oder falsch“ | Boolean |
| „ForeignKeyType“ | LongInt |
| „Ganze Zahl zwischen 0 und 9999“ | ShortInt<br>range = ”[0 .. 9999]”<br>numericPrecision=5<br>isUnsignedAttribute = TRUE |
| „GebietTyp“ | String |
| „ICD–9, ICD–10“ | String |
| „IdentifierValueType“ | String |
| „Inzidenzfall, Sterbefall“ | String<br>range = ”Inzidenzfall, Sterbefall” |
| „KeyType“ | LongInt |
| „LändercodeTyp“ | String<br>columnSize=4 |
| „LandkreisTyp“ | String |
| „Positive, ganze Zahl“ | LongInt<br>numericPrecision=10<br>isUnsignedAttribute = TRUE |
| „Text“ | String |
| „Text, 2–stellig“ | String<br>isFixedLength=TRUE<br>ColumnSize = 2 |
| „Text, 3–stellig“ | String<br>isFixedLength=TRUE<br>ColumnSize = 3 |
| „Text, 4–stellig“ | String<br>isFixedLength=TRUE<br>ColumnSize = 4 |
| „Text, 5–stellig“ | String<br>isFixedLength=TRUE<br>ColumnSize = 5 |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | |
|---|---|
| „Text, 6–stellig“ | String<br>isFixedLength=TRUE<br>ColumnSize = 6 |
| „Text, 8–stellig“ | String<br>isFixedLength=TRUE<br>ColumnSize = 8 |
| „UrbanisierungdsgradTyp“ | String<br>range = ”Städtisch, Ländlich, Gemischt” |
| „Zahl 4–stellig“ | ShortInt<br>range = ”[0 .. 9999]”<br>numericPrecision=5<br>isUnsignedAttribute = TRUE |

Tabelle 12.5: Abbildung der Datentypen von REMUS nach *LCD of SQL*

Um einen Eindruck des Schemaumfangs zu geben, sind in Tabelle 12.6 die verschiedenen *LCD of SQL*–Schemaelementtypen und die von ihnen erzeugte Anzahl aufgelistet.

| **Schemaelementtyp** | **Anzahl** |
|---|---:|
| AdditivityMETA | 268 |
| Column | 300 |
| ColumnConstraint | 0 |
| ColumnType | 22 |
| DatabaseConstraint | 0 |
| ForeignKey | 117 |
| MappingMETA | 4 |
| ReferentialRole | 117 |
| Table | 46 |
| TableConstraint | 0 |
| UniqueKey | 46 |

Tabelle 12.6: Anzahl der erzeugten *LCD of SQL*–Schemaelemente

Das resultierende Schema ist nach der Klassifikation in Abschnitt 4.2.2 ein Schneeflockenschema mit Surrogaten. In der in Abschnitt 4.2.1 eingeführten Notation ist in Abbildung 12.6 der Ausschnitt mit der Fakttabelle „Fall“ zu sehen.

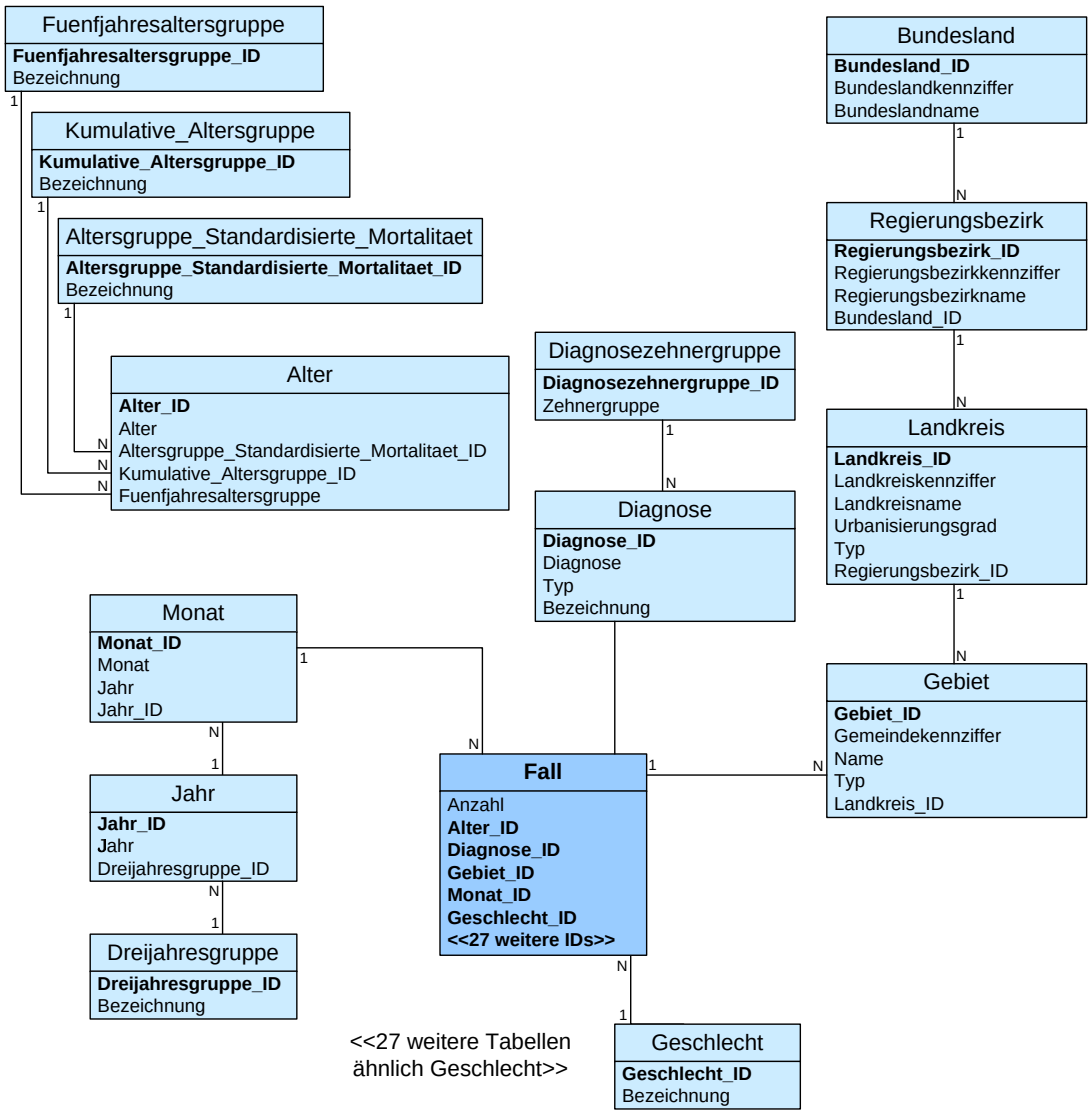



Abbildung 12.6: Resultat des physischen Entwurfs (Schneeflockenschema für „Fall“)

### 12.2.6 Schemaverfeinerung

Zur Verfeinerung des Schemas wurde während der Evaluation der in Abschnitt 9.3.3 angegebene Algorithmus angewendet, der das Schneeflockenschema mit Surrogaten über die Zwischenstufe eines Schneeflockenschemas ohne Surrogate in ein Sternschema überführt. Das Resultat ist in Abbildung 12.7 dargestellt.

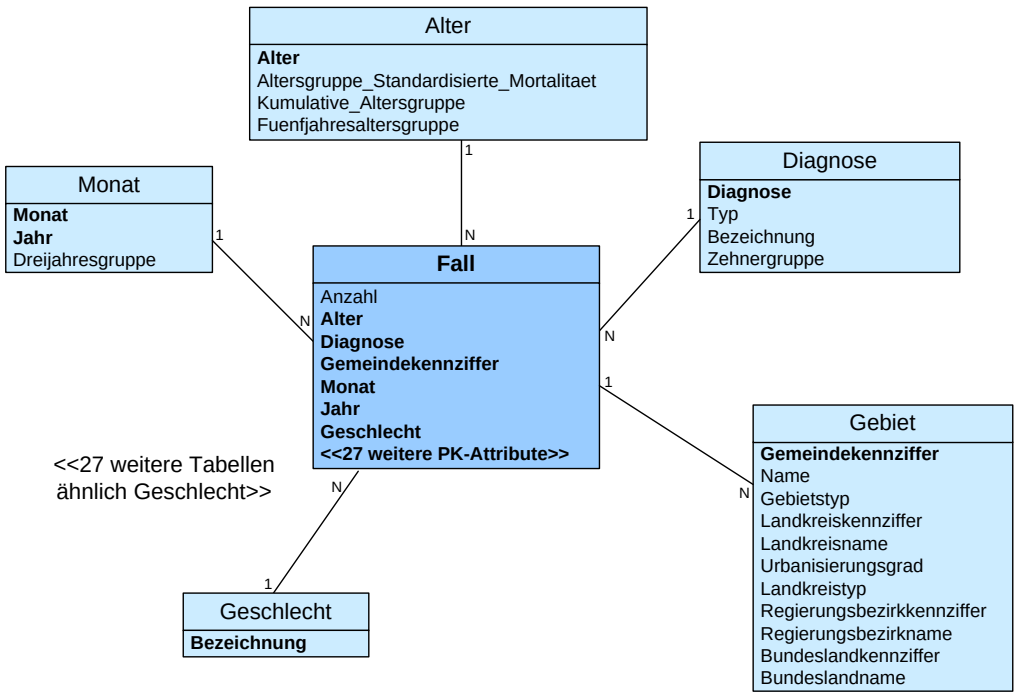



Abbildung 12.7: Resultat der Schemaverfeinerung (Sternschema für „Fall“)

## 12.3 Das realisierte System: ODAWA@EKN

Dieser Unterabschnitt skizziert das realisierte System ODAWA@EKN. Dabei wird zunächst in Abschnitt 12.3.1 die Implementierung des DWH als Produkt der Entwurfsmethodik beschrieben sowie die Befüllung des DWH mit Daten. Abschnitt 12.3.2 nennt einige auf dem DWH prototypisch realisierte Applikationen.

### 12.3.1 Data Warehouse

Das als Resultat von Abschnitt 12.2 erzielte Schema wurde sowohl auf einer MS SQL Server 2000 wie auch Oracle 8–Datenbank implementiert, um die Zielsystemunabhängigkeit zu zeigen. Die Dimensionen wurde mit Daten aus der bestehenden EKN–Datenbank gefüllt, wobei sich die in Tabelle 12.7 angegebenen Mengengerüste ergeben haben.

| **Dimension** | **Anzahl Tupel** |
|---|---:|
| Alter | 150 |
| Ausbreitung | 6 |
| Autopsie | 3 |
| Beruf | 2193 |
| C–Faktor | 6 |
| Diagnose | 20696 |
| Diagnoseanlass | 8 |
| Diagnosesicherung | 7 |
| Differenzierungsgrad | 13 |
| Dignität | 6 |
| Fernmetastasen | 3 |
| Geschlecht | 3 |
| Grundleiden | (siehe Diagnose) |
| Histologie | 3189 |
| Längste Tätigkeit | (siehe Beruf) |
| Letzte Tätigkeit | (siehe Beruf) |
| Lokalisation | 1759 |
| Lymphknoten | 4 |
| Mehrling | 5 |
| Ort | 1314 |
| Ort des Aufwachsens | (siehe Ort) |
| Ort der Geburt | (siehe Ort) |
| Ort des längsten Aufenthalts | (siehe Ort) |
| Qualität | 3 |
| Rauchen beendet | 103 |
| Raucherstatus | 4 |
| Seite | 5 |
| Staatsangehörigkeit | 223 |
| Therapieart | 8 |
| Therapiestatus | 4 |
| Therapieziel | 4 |
| Todesursache | (siehe Diagnose) |
| Todeszeit | (siehe Zeit) |
| Tumorausbreitung | 8 |
| Tumorbedingter Tod | 4 |
| Tumorfolge | 3 |
| Typ des Falles | 2 |
| Validität | 4 |
| Verstorben | 3 |
| Verwandschaft | 10 |
| Zeit | 120 |

Tabelle 12.7: Mengengerüst der einzelnen Dimensionen

Zur Erzeugung der Faktdaten wurde ein Algorithmus zum Testdatengenerieren gewählt, der für die unterschiedlichen Ausprägungen der Dimensionen unterschiedliche Verteilungen vornimmt. Die Annahmen über diese Verteilungen wurden den aktuell vorliegenden Daten der aktuellen EKN–Datenbank entnommen.

### 12.3.2 Applikationen

Aufbauend auf dem DWH wurden exemplarisch Applikationen realisiert, die in diesem Abschnitt durch Bildschirmfotografien dokumentiert werden. Zum einen wurde mit der Berichtskomponente von Microsoft Access die in [EKN96] definierten Berichte nachgebildet (siehe Abbildung 12.8) und zum anderen wurden Microsoft Excel und der Microsoft Cube Browser als OLAP–Client verwendet (siehe Abbildung 12.9).

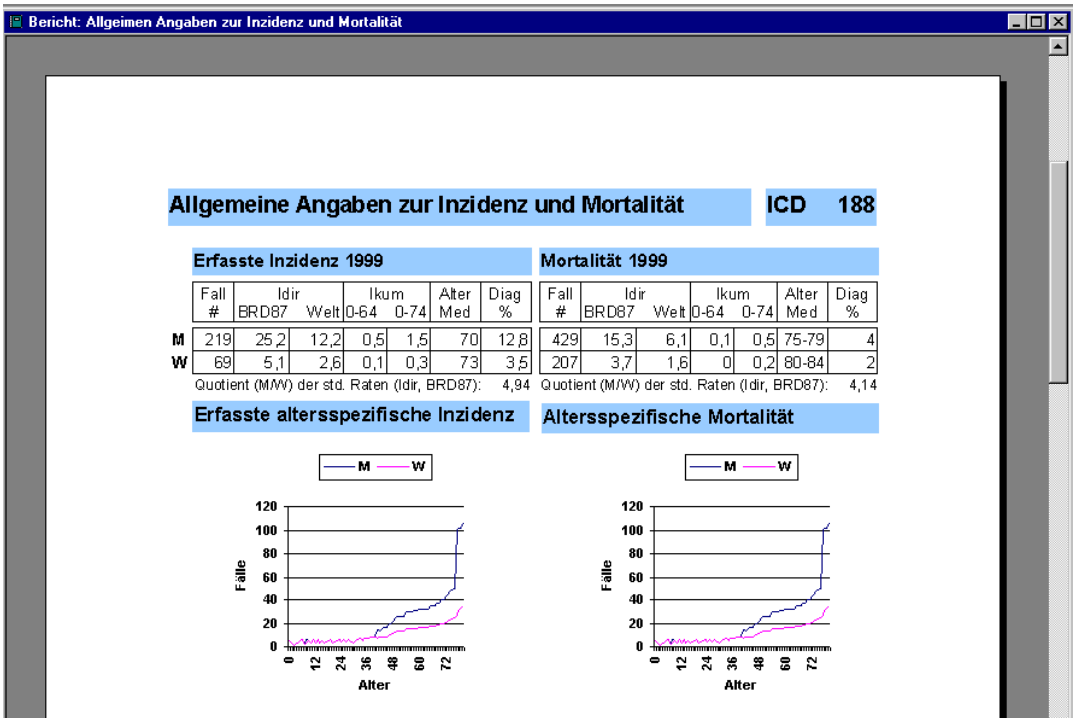
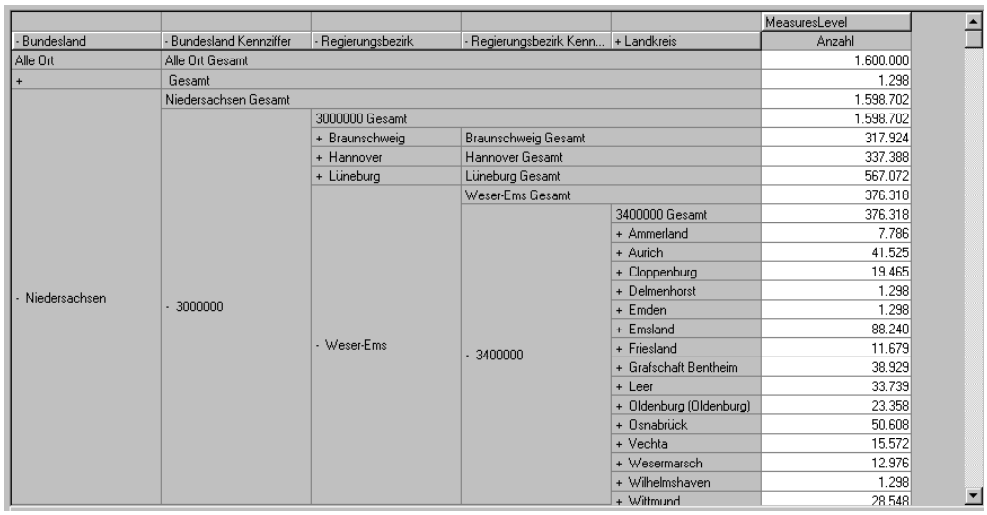



Abbildung 12.8: Bericht

## 12.4 Weitere Aspekte

In diesem Abschnitt werden einige weitere Aspekte diskutiert, die auf die Methodik Einfluss haben. Zum einen sind *Änderungen* von äußeren Parametern zu nennen, wie z. B. neue Hardwarevoraussetzungen, eine neue Version des DBMS oder ein anderes Zielsystem evtl. mit einem anderen Datenmodell. Für jede dieser Änderungen müsste die „geeignete“ Stelle im Prozess identifiziert werden, an der wiedereingesetzt werden soll. Beispielsweise müsste bei neuen Hardwarevoraussetzungen lediglich der Schritt der physischen Optimierung erneut durchgeführt werden, bei einem DBMS mit einem anderen Datenmodell hingegen müsste bei der Transformation von der konzeptionellen zur logischen Ebene wieder eingesetzt werden.
Ein anderes Kriterium sind *Änderungen und Erweiterungen* des Schemas. In diesem Falle ist das konzeptionelle Schema entsprechend zu modifizieren und die weiteren Schritte der Methodik sind erneut durchzuführen.
Schließlich kann noch die *Skalierbarkeit* genannt werden. Diese kann sich auf Schema– oder Extensionsgröße beziehen. Für die Schemagröße empfiehlt die Methodik im Rahmen des Leitfadens (siehe Abschnitt 6.3) eine Strukturierung in Subschemata, bei wachsender Extensionsgröße gestaltet sich die physische Optimierung eventuell schwieriger, aber in diesem Falle erlaubt die in Abschnitt 10.1 skizzierte Konfiguration die Möglichkeit entsprechende Anpassungen vorzunehmen.



| | | | | | MeasuresLevel |
|---|---|---|---|---|---|
| - Bundesland | - Bundesland Kennziffer | - Regierungsbezirk | - Regierungsbezirk Kenn... | + Landkreis | Anzahl |
| Alle Ort | Alle Ort Gesamt | | | | 1.600.000 |
| + | Gesamt | | | | 1.298 |
| - Niedersachsen | Niedersachsen Gesamt | | | | 1.598.702 |
| | - 3000000 | 3000000 Gesamt | | | 1.598.702 |
| | | + Braunschweig | Braunschweig Gesamt | | 317.924 |
| | | + Hannover | Hannover Gesamt | | 337.388 |
| | | + Lüneburg | Lüneburg Gesamt | | 567.072 |
| | | - Weser-Ems | Weser-Ems Gesamt | | 376.310 |
| | | | - 3400000 | 3400000 Gesamt | 376.318 |
| | | | | + Ammerland | 7.786 |
| | | | | + Aurich | 41.525 |
| | | | | + Cloppenburg | 19.465 |
| | | | | + Delmenhorst | 1.298 |
| | | | | + Emden | 1.298 |
| | | | | + Emsland | 88.240 |
| | | | | + Friesland | 11.679 |
| | | | | + Grafschaft Bentheim | 38.929 |
| | | | | + Leer | 33.739 |
| | | | | + Oldenburg (Oldenburg) | 23.358 |
| | | | | + Osnabrück | 50.608 |
| | | | | + Vechta | 15.572 |
| | | | | + Wesermarsch | 12.976 |
| | | | | + Wilhelmshaven | 1.298 |
| | | | | + Wittmund | 28.548 |

Abbildung 12.9: OLAP mit dem Microsoft Cube Browser

## 12.5 Zusammenfassung

In Kapitel 12 wurde die Evaluation der in Teil II dieser Arbeit entwickelten Entwurfsmethodik beschrieben. Zur Werkzeugunterstützung für die Evaluation wurde das in Kapitel 11 entwickelte Werkzeug ODAWA verwendet. Domäne für die Evaluation war das in Abschnitt 12.1 vorgestellte EKN. Weitere Hintergrundinformationen hierzu können z. B. in [AFH+97, ABH+98, EKN01] nachgelesen werden. In Abschnitt 12.2 wurde die Evaluation im engeren Sinne, d. h. das Anwenden der in Teil II beschriebenen Entwurfsschritte skizziert, wobei eine Beschränkung auf die Darstellung grundlegender Aspekte stattfand. Weiterführende Details können im Evaluationsbericht [HK01] nachgelesen werden. Das Resultat dieses Prozesses ist ein physisches DB–Schema. Neben seiner Implementierung wurde dieses Schema mit Testdaten befüllt und auf diesem Schema arbeitende Applikationen realisiert. Diese als ODAWA@EKN bezeichnete Software war Thema von Abschnitt 12.3. Um das Verhalten der Methodik aufzuzeigen, wurden in Abschnitt 12.4 ein paar Szenarien vorgestellt, die eine Änderung bzw. Erweiterung des Schemas nach sich ziehen bzw. ein starkes Anwachsen der verwalteten Extension zur Folge haben.
Als wesentliche Resultate der in diesem Kapitel beschriebenen Evaluation lassen sich die folgenden Punkte festhalten:

- Die *Anwendbarkeit* der Methodik konnte nachgewiesen werden.
- Zusammen mit dem durchgängigen Beispiel „Handelswelt“ aus Teil II der Arbeit konnte die *Domänenunabhängigkeit* der Methodik nachgewiesen werden.
- Durch Anwendung des Leitfadens aus Abschnitt 6.3 zur systematischen Gewinnung eines konzeptionellen Schemas lässt sich eine Schemaversion gewinnen, die eine gute Vorgabe für die weitere Diskussion mit den Fachvertretern ist. Während dieser Diskussion bekommt das Schema „den letzten Schliff“. Der Leitfaden kann somit als *nützliches Hilfsmittel* gewertet werden.
- Das *explizite Review* des konzeptionellen Schemas hat sich als praktikabel erwiesen. Insbesondere können hierdurch Bezeichnungsungenauigkeiten frühzeitig entdeckt und Abhängigkeiten zwischen einzelnen Dimensionen gut dokumentiert werden. In Diskurswelten, in denen mehr und/oder kompliziertere Verdichtungspfade vorliegen, ist auch deren frühzeitiges Erkennen durch ein explizites Schemareview zu erwarten.

- Durch die *interaktiven Schritte* während der Transformationen von der konzeptionellen auf die logische Entwurfsebene und von dieser auf die physische Ebene können *domänenspezifische Aspekte* in Kombination mit Benutzerwissen gewinnbringend in den Entwurfsprozess einfließen. So konnten z. B. durch das in Abschnitt 7.2.9 beschriebene Herunterreichen der Dimensionen für die Datenanalyse geeignete Schemata erzeugt werden.
- Weiterhin können durch die interaktiven Entwurfsschritte in der Organisation oder dem Projekt vereinbarte *Richtlinien*, wie z. B. Namenskonventionen, angewendet werden.
- Durch die Diskussion der Aspekte in Abschnitt 12.4 konnte die *Erweiterbarkeit* und *Änderbarkeit* von Schemata bei Anwendung der Methodik verdeutlicht werden.
- Dadurch ist auch die Eignung der Methodik für *Prototyping* als Vorgehensweise belegt.
- Die Realisierung auf zwei unterschiedlichen DB–Plattformen zeigt die *Zielsystemunabhängigkeit* der Methodik.
- Gleiches gilt auch für *Versionswechsel*, auch wenn dieses im Rahmen der Evaluation nicht explizit gezeigt wurde, denn bei einem Versionswechsel eines bestimmten DBMS können die beiden Versionen als unterschiedliche Zielsysteme aufgefasst werden.
- Ebenso ist durch Nachhalten aller Entwurfsinformationen im Repository *Nachvollziehbarkeit* gewährleistet, indem jederzeit festgestellt werden kann, wer welche Entwurfsentscheidung vorgenommen hat.
- Durch das Repository und die Schnittstelle des ODAWA–Werkzeugs kann jederzeit *Dokumentation* erzeugt werden, die einzelne Schemata oder auch den Fortgang des gesamten Prozesses beschreibt.

# Teil IV

# Zusammenfassung und Ausblick

# Kapitel 13

# Zusammenfassung und Ausblick

Dieses abschließende Kapitel gibt eine Zusammenfassung und einen Ausblick, wobei in Abschnitt 13.1 zunächst die im Rahmen dieser Arbeit erreichten Ziele aufgeführt werden. Abschnitt 13.2 nennt mögliche Erweiterungen im unmittelbaren Umfeld der Entwurfsmethodik, bevor in Abschnitt 13.3 etwas weitergehend zukünftige Entwicklungen und Tendenzen im gesamten DWS–Umfeld diskutiert werden.

## 13.1 Erreichte Ziele

Schon seit langer Zeit besteht in Organisationen der Wunsch, neben der Unterstützung operativer Aufgaben, Informationssysteme auch zur Entscheidungsunterstützung einzusetzen. Diese Anforderung wurde in den letzten Jahren beispielsweise durch Marktsättigung im klassischen Handel oder durch Marktliberalisierung im Telekommunikationssektor verstärkt. Dabei haben sich in den 90er Jahren DWHs als Basis entscheidungsunterstützender Informationssysteme etabliert. Aufgrund der stark unterschiedlichen Eigenschaften von DWHs und operativen DBen sind herkömmliche Entwurfsmethodiken jedoch nur eingeschränkt anwendbar.
Ziel dieser Arbeit war daher die Konzeption einer durchgängigen Entwurfsmethodik für DWH. Bei dem gewählten Ansatz bildet der im Entwurf operativer DBen etablierte Drei–Ebenen–Entwurf die Basis und wurde unter Berücksichtigung DWH–spezifischer Aspekte erweitert.
Im Einzelnen wurden dabei die folgende Ziele erreicht:

- Für die konzeptionelle Modellierung wurde die Sprache MML entworfen, deren wesentliches Charakteristikum die Existenz sowohl multidimensionaler als auch objektorientierter Konstrukte ist.

- Aufbauend auf der MML können unterschiedliche graphische Notationen verwendet werden, wobei mit der $_m$UML eine Sprache vorgestellt wurde, die die UML unter Ausnutzung der Mechanismen Stereotype und standardisierte Annotationen erweitert.

- Als Anleitung zur Schemagewinnung wurde ergänzend zu MML bzw. $_m$UML ein Leitfaden vorgeschlagen, mit dessen Hilfe mittels eines definierten Vorgehens ein Schema erzielt wird.

- Weil der konzeptionelle Entwurf als Ausgangspunkt der weiteren Entwicklung zentraler Bestandteil der Methodik ist, bildet ein explizites Review durch einen Domänenexperten den Abschluss dieser Phase. Dieses Review dient vor allem dem Nachweis von inhaltlichen Qualitätskriterien des modellierten Schemas.

- Als Beschreibungsmittel der logischen Entwurfsebene wurde der verallgemeinerte Relationenschematyp REMUS definiert, der ein herkömmliches Relationenschema um verschiedene Metadatentypen ergänzt. Diese Metadaten halten Informationen über die objektorientierten und multidimensionalen Konstrukte des konzeptionellen Modells fest, die sich nicht direkt mit den Beschreibungsmitteln des Relationenmodells ausdrücken lassen.
- Für die Transformation eines MML– in ein REMUS–Schema wurde ein Algorithmus entworfen, der größtenteils automatisch abläuft, an verschiedenen Stellen jedoch dem Benutzer Einflussnahme auf einige Entwurfsentscheidungen erlaubt. Diese Kombination von maschineller Transformation und menschlichem Kontextwissen gewährleistet einen optimalen Transformationsprozess.
- Als Beschreibungsmittel der physischen Entwurfsebene wurde das Metamodell *LCD of SQL* eingeführt. Dieses Modell bietet elementaren Umfang, wie er vom SQL–Standard und kommerziellen Systemen mit großer Marktverbreitung unterstützt wird.
- Auch die Abbildung für Schemata von REMUS nach *LCD of SQL* besitzt einige interaktive Schritte, in denen der Benutzer projekt– bzw. organisationsspezifische Aspekte einfließen lassen kann.
- Als nächster Schritt erfolgt eine Verfeinerung des *LCD of SQL*–Schemas, um spezielle Anforderungen des verwendeten DBMS oder OLAP–Servers zu erfüllen. Dazu wurden auf *LCD of SQL*–Schemata Verfeinerungsoperatoren definiert, die sich mit Hilfe algorithmischer Elemente zu Transformationsvorschriften kombinieren lassen.
- Als letzter Schritt des Entwurfsprozesses erfolgt die physische Optimierung des bis zu diesem Zeitpunkt erreichten Schemas. Hierfür wurde ein Framework vorgestellt, das die gleichzeitige Behandlung verschiedener Optimierungsmaßnahmen ermöglicht.
- Zum Nachweis der Umsetz– und Anwendbarkeit der vorgestellten Konzepte erfolgte die softwaretechnische Realisierung in Form eines Prototypen, der in einer anschließenden Evaluation anhand des „Realweltbeispiels“ Epidemiologisches Krebsregister Niedersachsen eingesetzt wurde.

Der Notation aus Abschnitt 5.1 folgend ergibt sich zusammenfassend der in Abbildung 13.1 dargestellte Entwurfsprozess. Die Zahlen in Klammern verweisen auf den jeweiligen Abschnitt der Arbeit, in dem der entsprechende Schritt bzw. die entsprechende Notation ausführlich behandelt werden.

## 13.2 Erweiterungen der Methodik

Neben den erreichten Zielen wurden im Rahmen der Arbeit an einigen Stellen Einschränkungen vorgenommen bzw. wurde auf bestehende Probleme hingewiesen. Einige dieser Einschränkungen oder Probleme bilden den Ausgangspunkt für mögliche zukünftige Arbeiten, z. B.:

- Zunächst ist die „Abrundung“ des Werkzeuges zu nennen, indem noch fehlende oder nur sehr rudimentär gestaltete Teile implementiert werden. Ebenso könnte das Werkzeug um Konzepte wie Zugriffsrechte, ein Hilfesystem oder Versions– bzw. Variantenmanagement erweitert werden.
- Die konzeptionelle Entwurfsebene als zentraler Bestandteil der Methodik und Ausgangspunkt aller weiteren Transformationen ließe sich beispielsweise um Kataloge von Analysemustern erweitern. In diesen Mustern könnte das in vergangenen Projekten erlangte multidimensionale

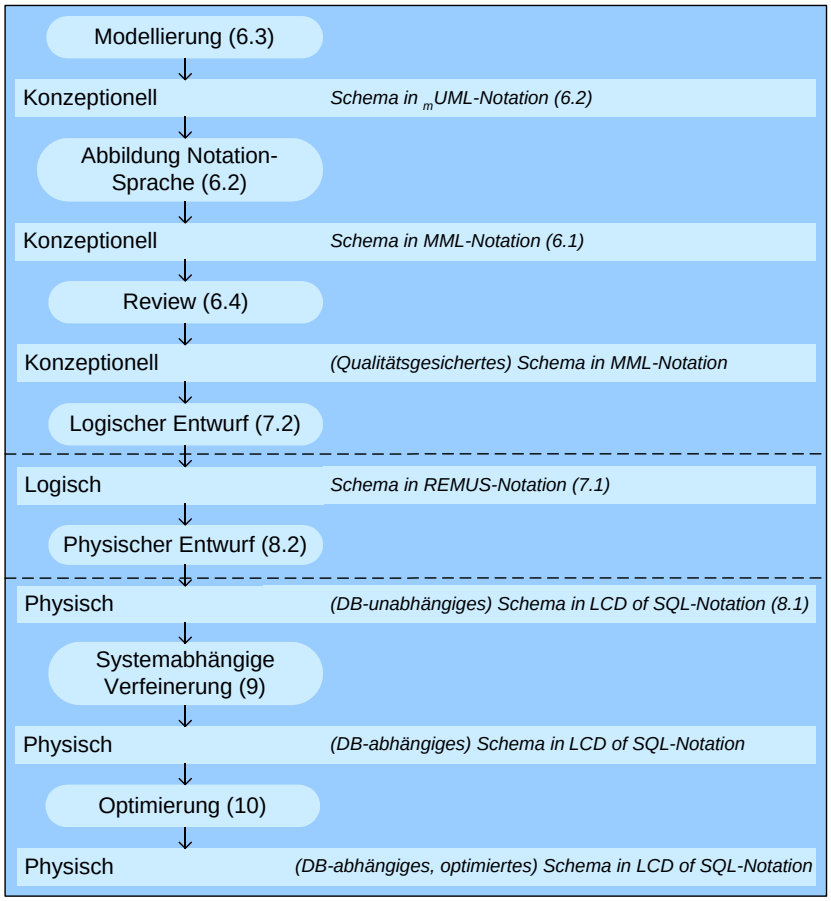



Abbildung 13.1: Ablauf des Entwurfsprozesses

Modellierungswissen festgehalten werden. Neben „guten“ Lösungen können in diesen Musterkatalogen auch Negativbeispiele festgehalten werden, die zukünftige Benutzer vor Wiederholung dieser gleichen Modellierungsfehler bewahren.

- Die bisher nur grob skizzierten Transformationen in nicht–relationale Datenmodelle (siehe Abschnitt 7.3) im Zuge des logischen Entwurfs könnten detailliert ausgearbeitet werden. Dazu sind für die jeweilige Zielwelt Metamodelle zu spezifizieren und die Abbildung von MML–Schemaobjekten auf Elemente dieser Modelle ist festzulegen.

- Die konzeptionelle Entwurfsebene als zentraler Punkt der Methodik könnte ausgebaut werden, wobei der Charakter einer Workbench von Werkzeugen für die konzeptionelle multidimensionale Modellierung verfolgt werden kann. Denkbar sind beispielsweise Werkzeuge, die Konzepte und Verfahren des Requirements Engineering in die Phase der konzeptionellen Modellierung einbinden.

- Der die physische Optimierung behandelnde letzte Entwurfsschritt der Methodik könnte ausgebaut und verfeinert werden. Insbesondere kann hier ein Konzept entwickelt werden, wie während des laufenden Betriebs des DWH gewonnene Log–Informationen wieder als Eingabe in den Optimierungsprozess genutzt werden können.

- Bisher berücksichtigt der Entwurfsprozess keine Verteilung. Dementsprechend könnten Konzepte zur Verteilung auf mehrere DBen konzipiert und in die Methodik integriert werden. Dabei können aus dem Bereich verteilter und paralleler DBen bekannte Verteilungstechniken

[Rah94, OV99] unter Berücksichtigung des multidimensionalen Datenmodells und Analyse–spezifischen Aspekten untersucht werden.

- Die Methodik könnte dahingehend ausgebaut werden, dass als Resultat neben der Erzeugung eines DWH–Schemas auch z. B. Berichte automatisch generiert werden oder mit einem Ladewerkzeug kommuniziert wird.
- Die Methodik unterstützt bisher ein reines Forward Engineering. Insbesondere unter dem in der Einleitung genannten Kritikpunkt gescheiterter DWH–Projekte ist es wünschenswert, auch die Umkehrungen der Entwurfsschritte zu spezifizieren und somit zunächst ein Reverse und anschließend auch ein komplettes Re–Engineering zu unterstützen.
- Eine Änderung auf der konzeptionellen Entwurfsebene verlangt bisher die Transformation des gesamten Schemas. Sind die Änderungen oder Erweiterungen aber nur gering, so wäre es denkbar, nur die von der Änderung betroffenen Schemateile „quasi inkrementell“ zu transformieren.

## 13.3 Visionen im DWS–Umfeld

Abschließend werden einige zukünftige Tendenzen im Kontext von DWS aufgeführt, mit denen sich die Scientific Community und die kommerzielle Praxis in den nächsten Jahren beschäftigen werden, die jedoch nicht unmittelbar im Bereich der in dieser Arbeit vorgestellten Methodik liegen. Zu diesen Aufgaben sind zu zählen:

- Weiterentwicklung des DWH–basierten Data Mining bzw. die Kombination von OLAP und Data Mining als Analysetechnik. Durch Zusammenführung dieser beiden Techniken und ihre Auswertung und Interpretation durch einen menschlichen Benutzer entsteht ein entscheidender Mehrwert bei der Analyse.
- Ebenso ist eine Kopplung dieser beiden Techniken mit den sich zunehmend etablierenden Wissensmanagementsystemen (WMS) zu erwarten. Ein denkbares Szenario besteht darin, dass in einem WMS abgelegte Dokumente während des Analyseprozesses abgerufen werden können und dabei weitere Hintergrundinformationen liefern.
- Heutige Front End–Komponenten in einem DWS sind größtenteils als Desktop–Applikationen realisiert. Hier ist zukünftig in der Praxis eine stärkere Browser–Orientierung zu erwarten. Weil die Front End–Komponenten umfangreiche Funktionalität bieten (z. B. die in Abschnitt 3.1.2 beschriebenen multidimensionalen Operationen), stellt diese Migration eine besondere Herausforderung dar. Ebenso ist die Handhabung sehr großer Datenmengen in heutigen OLAP Front End–Komponenten nur unzureichend gelöst.
- Durch das Entstehen einzelner Data Marts innerhalb einer Organisation oder das Fusionieren zweier Organisationen wird häufiger die Aufgabe zu lösen sein, mehrere DWHs und somit mehrere multidimensionale Schemata zu integrieren. Dies bedeutet, bestehende Konzepte und Verfahren der Schemaintegration auf multidimensionale Datenmodelle zu übertragen.
- Um die gesamten Informationsbedürfnisse einer Organisation zu befriedigen, wird die Beschränkung auf ein innerhalb der Organisation angesiedeltes DWH in Zukunft nicht mehr ausreichend sein. Aus diesem Grunde werden einerseits innerhalb des Semantic Web [Wor01] DWHs als Informationsquelle dienen, andererseits können aber auch Webdaten als DWH angeboten werden. Hierbei ist die Darstellung multidimensionaler Sachverhalte mit den Ausdrucksmitteln des Semantic Web zu erforschen. Aber auch die Integration von Webdaten mit denen aus herkömmlichen Datenbanken stellt eine neue Herausforderung dar.

- Veränderungen in den Datenquellen durch neue operative Systeme oder Stilllegung von Altsystemen stellen einen großen Anspruch an den Back End–Prozess. Somit sind hier fortgeschrittene Konzepte für den Datenbewirtschaftungsprozess notwendig.
- Es ist wünschenswert, dass Änderungen in den Datenquellen — sowohl auf Instanz– als auch auf Schemaebene — an das DWH propagiert werden, damit temporale Abfragen während der Datenanalyse angeboten werden können. Somit sind Konzepte für temporale DWHs und temporales OLAP zu entwickeln und deren effiziente Realisierung notwendig.
- Durch stark anwachsende Datenvolumina und steigende Komplexität der Applikationen werden zukünftig Benutzer nicht mehr alle für die Entscheidungsunterstützung notwendigen Informationen durch eigene Analyse ermitteln können. Vielmehr ist schon im Vorfeld eine koordinierte „Zuteilung“ der Daten notwendig. Dieses kann beispielsweise durch sog. Informationsfilter unterstützt werden, die die Benutzer durch einschränkende Konfigurationen vor einem Informationsüberfluss bewahren.

# Anhänge und Verzeichnisse

# Anhang A

# Das Beispiel *Handelswelt*

In diesem Anhang werden die Schemata des durchgängigen Beispiels „Handelswelt“ aus Teil II der Arbeit zusammengefasst. Zunächst wird in Abschnitt A.1 das Resultat des konzeptionellen Entwurfsschrittes in Form eines dokumentierten $_m$UML–Schemas wiedergegeben. Das Ergebnis der Transformation nach REMUS ist Gegenstand von Abschnitt A.2, bevor in Abschnitt A.3 die Schemaelemente des hieraus resultierenden *LCD of SQL*–Schemas aufgeführt werden.

## A.1 MML–Schema

Aus Gründen der Übersichtlichkeit wurde das Schema in Schritt 16 des Leitfadens (siehe Seite 115 in Abschnitt 6.5.2) in mehrere Subschemata unterteilt. Die Faktklassen und ihre gegenseitigen Beziehungen sowie die einer Faktklasse zugeordneten Dimensionen inkl. der die Ebene der feinsten Granularität bildenden dimensionalen Klasse bilden ein Subschema (siehe Abschnitt A.1.1), jede Hierarchiestruktur bildet ebenso ein Subschema (Dimension „Zeit“ siehe Abschnitt A.1.2, Dimension „Produkt“ siehe Abschnitt A.1.3 und Dimension „Ort“ siehe Abschnitt A.1.4). Die Faktklasse „Einkommen“ mit zugehörigen dimensionalen Strukturen ist in Abschnitt A.1.5 dokumentiert. In diesem Falle wurde auf eine separate Darstellung der Dimensionen verzichtet, weil diese Dimensionen nur sehr klein sind.

### A.1.1 Fakten

„Verkäufe“ als Ausgangspunkt der Datenanalysen bilden eine Faktklasse, diese setzen sich aus mehreren verkauften Produkten zusammen, so dass diese ebenfalls eine Faktklasse bilden und zwischen beiden eine Komposition gebildet wird. Die Ebenen der feinsten Granularität sind gemäß den Anforderungen „Artikel“, „Tag“ und „Ort des Verkaufs“.

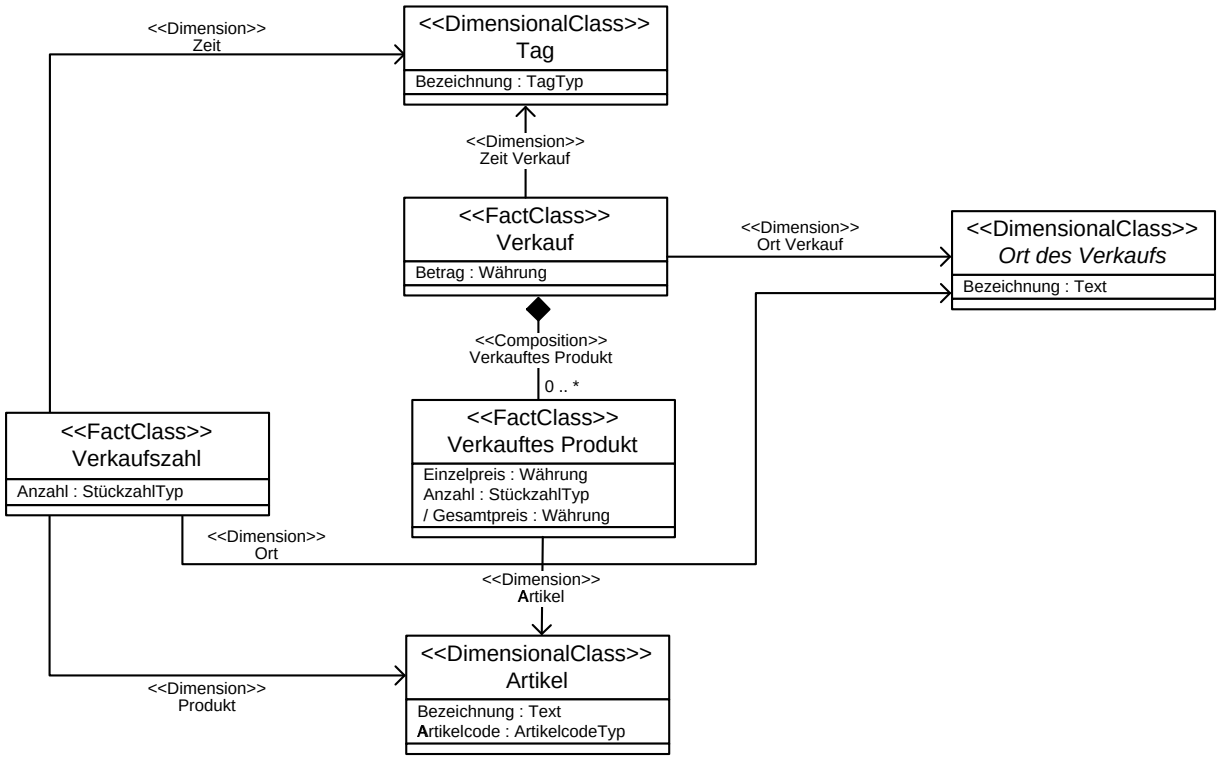



Abbildung A.1: Ergebnis Konzeptionelle Modellierung: Subschema Fakten *Verkauf, Verkauftes Produkt* und *Verkaufszahl*

### A.1.2 Dimension Zeit

Die „Zeit“–Dimension beschreibt die Verdichtungsstufen der für Auswertungen und Analysen der Verkaufszahlen relevanten Zeitaspekte. Die detaillierteste Zeiteinheit ist dabei der Tag. Der Anforderung „unterschiedliche Abteilungen benötigen Verkaufszahlen nach unterscheidlichen zeitlichen Perioden“ wird durch dadurch Rechnung getragen, dass mit „Woche“, „Monat“, „Quartal“ und „Jahr“ diverse zeitliche Aspekte abgedeckt werden. Weil die Woche „schief“ zu den anderen Zeiteinheiten liegt, mussten parallele Hierarchiepfade gebildet werden und die Verdichtung von der Woche zum Jahr ist über das *SharedRollUp*–Konstrukt mit einer Berechnungsvorschrift verbunden, weil sich eine Woche u. U. auf zwei Jahre aufteilen kann.

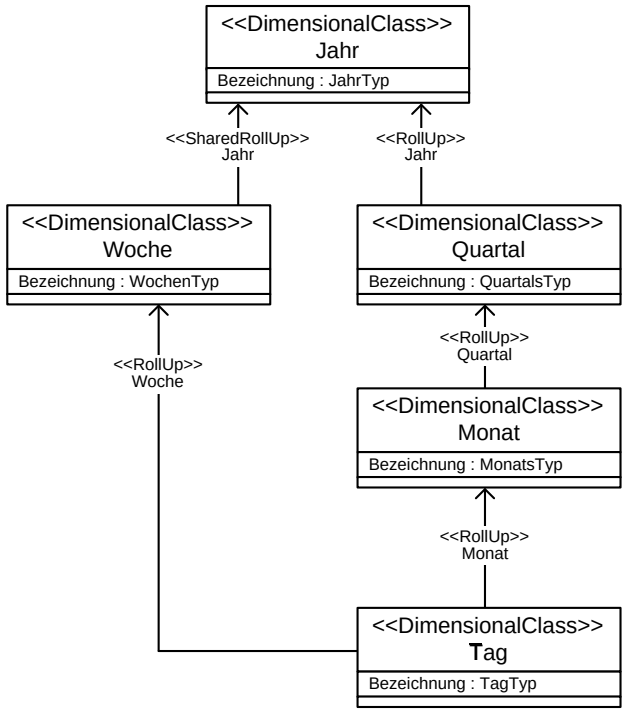



Abbildung A.2: Ergebnis Konzeptionelle Modellierung: Subschema Dimension *Zeit*

### A.1.3 Dimension Produkt

Die „Produkt“–Dimension beschreibt die Verdichtungsstufen der Artikel. Die Zusammenfassung von Artikeln zu Produktgruppen, –familien und –kategorien entspricht dem branchenüblichen Sprachgebrauch und hält die Anforderungen der Beschreibung des Szenarios aus Abschnitt 6.5.1 fest.

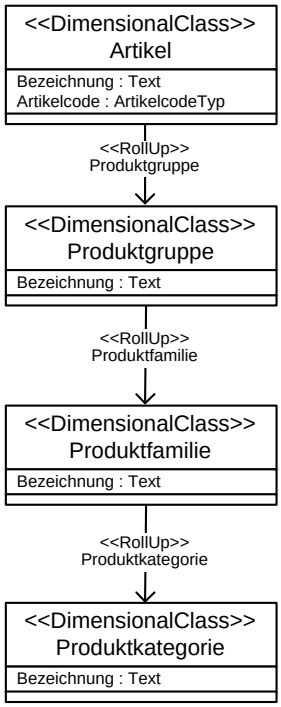



Abbildung A.3: $_m$UML Beispiel – Ergebnis Konzeptionelle Modellierung: Subschema Dimension *Produkt*

### A.1.4 Dimension Ort

Die Dimension „Ort“ beschreibt die Verdichtungsstufen der Orte des Verkaufs. Dabei wird auf der feingranularsten Ebene zwischen Verkaufseinheiten in Kaufhäuser und rechtlich bzw. wirtschaftlich selbständige Filialen unterschieden. Haupthierarchiepfad ist die Verdichtung zu Städten, Regionen und Staaten, daneben existiert die insbesondere für die deutschen Filialen bedeutende Zusammenfassung zu Verkaufsbezirken. Beim Verfolgen des Verdichtungspfades von den Verkaufsbezirken zu den Regionen ist zu beachten, dass hierbei nur Einträge deutscher Verkaufsorte berücksichtigt werden. Um den Anforderungen für die internen Auswertungen zu genügen, existiert von Filialen ausgehend ein Verdichtungspfad über Filialkategorien zu Filialoberkategorien.

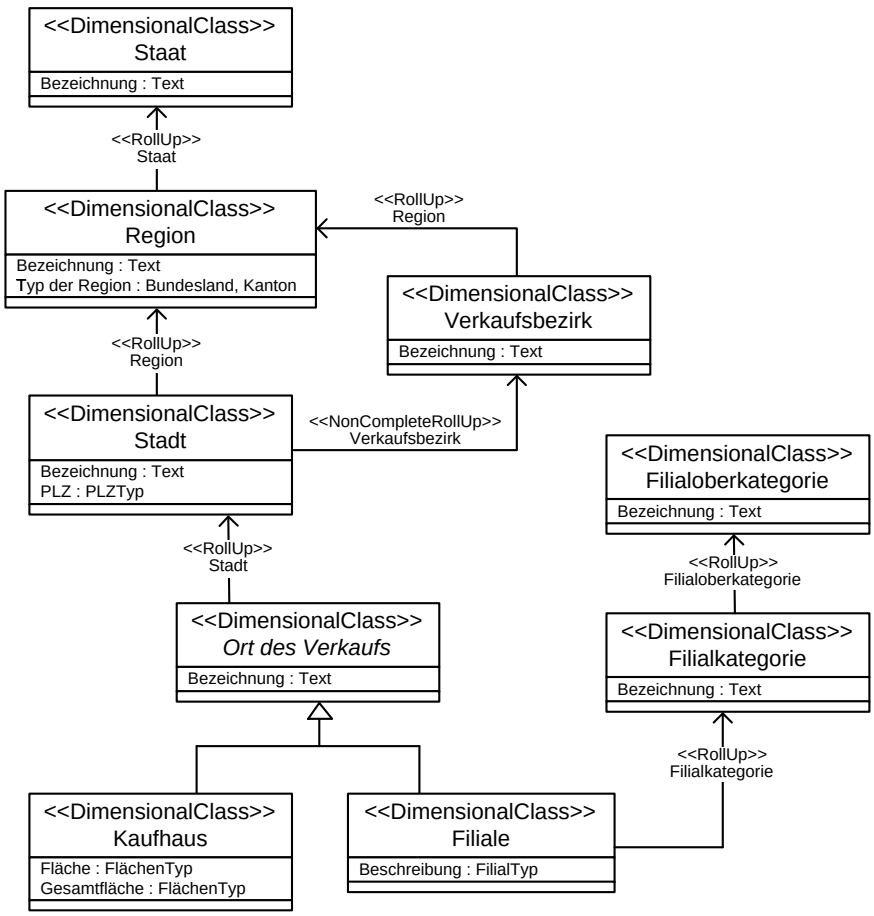



Abbildung A.4: Ergebnis Konzeptionelle Modellierung: Subschema Dimension *Ort*

### A.1.5 Subschema Einkommenszahlen

Die Faktklasse „Einkommen“ beschreibt die extern bezogenen Daten über das Einkommen von Personen im Einzugsgebiet. Über das Dimensionskonstrukt „Zeit“ wird die zeitliche Komponente dieser extern bezogenen Einkommenszahlen beschrieben. Weil diese vierteljährlich geliefert werden, bilden Quartale die Ebene der feinsten Granularität. Ansonsten können die in Abschnitt A.1.2 beschriebenen Hierarchieebenen genutzt werden. Die Dimension „Ort“ beschreibt die Verdichtungsstufen der Orte für die extern bezogenen Einkommenszahlen. Feingranularste Ebene sind die Strassenbereiche, welche Städten zugeordnet werden können, womit auf der Ebene der Städte eine Zusammenführung mit der in Abschnitt A.1.4 beschriebenen Dimension möglich ist. Weil die vergleichenden Analysen der Marketingsabteilung zwischen dem Einkommen der Personen und den abgesetzten Produkten aber auch auf feingranularerer Ebene als Städten notwendig sind, sind die Strassenbereiche Ziel eines *DimensionalMapping*–Konstrukts, dessen Ausgangspunkt die Verkaufsorte sind.

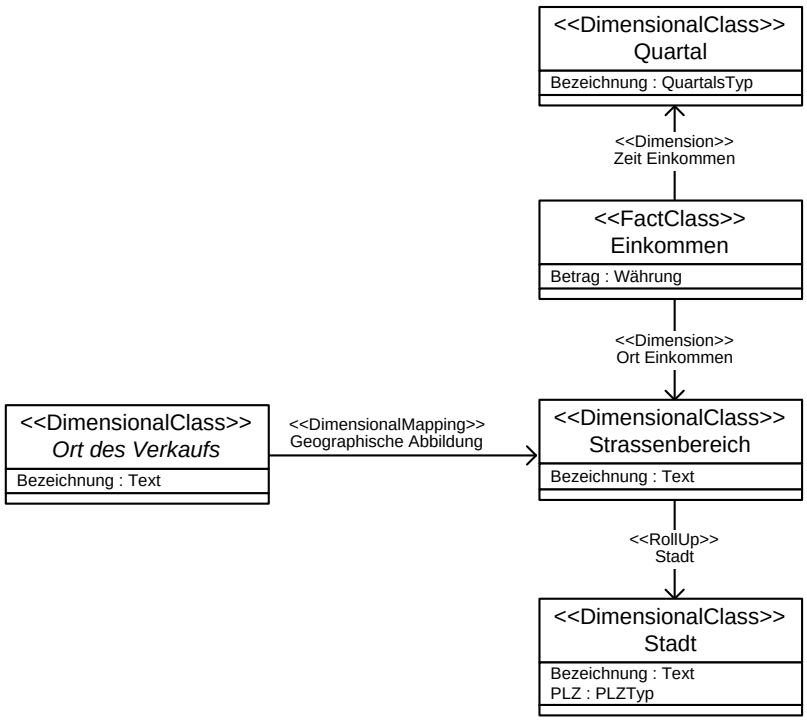



Abbildung A.5: Ergebnis Konzeptionelle Modellierung: Subschema Einkommenszahlen mit Hierarchien und *DimensionalMapping*

## A.2 REMUS–Schema

Dieser Abschnitt gibt in Tabellenform eine Auflistung aller während der Transformation des Schemas aus dem letzten Abschnitt nach dem in Abschnitt 7.2 beschriebenen Transformationsalgorithmus anfallenden REMUS–Objekte. Die Objekte und Attribute sind alphabetisch sortiert, die Metadaten sind nach Metadatentyp und innerhalb dieser Gruppen nach den betreffenden Objekten bzw. Attributen alphabetisch sortiert. Hinter jedem REMUS–Objekt steht in Klammern der Transformationsschritt, in dem dieses Objekt angelegt worden ist.
Zur Erreichung des Resultats seien für die deterministische Funktion $f_{det}$ folgende Annahmen getroffen:

- In Schritt 3 übernimmt $f_{det_{ClassName}}$ für jedes einelementige Zerlegungselement den Namen der dimensionalen Klasse. Für die dreielementige Teilmenge mit der Vererbungshierarchie soll $f_{det_{ClassName}}(\{\text{"Filiale"}, \text{"Kaufhaus"}, \text{"Ort des Verkaufs"}\}) \stackrel{def}{=} \text{"Ort des Verkaufs"}$ gelten.
- In Schritt 8 bestimmt $f_{det_{Composition}}$ die Auflösung der Komposition zwischen "Verkauf" und "Verkauftes Produkt". Dabei soll das in Abschnitt 7.2.9 als Möglichkeit II dargestellte Übertragen der Dimensionen gewählt werden.
- Die Funktion $f_{det_{Additivity}}$ übernimmt schließlich in Schritt 10 die bereits modellierten Additivitäten und gibt für die neu entstandenen Faktattribut–Dimension–Kombinationen alle Verdichtungsoperatoren frei.

### REMUS–Objekte des Beipiels „Handelswelt“

| Objekte | |
|---|---|
| "Artikel" | (3) |
| "ArtikelcodeTyp" | (1) |
| "Bundesland, Kanton" | (1) |
| "Einkommen" | (8) |
| "Filialkategorie" | (3) |
| "Filialoberkategorie" | (3) |
| "FlächenTyp" | (1) |
| "ForeignKeyType" | (1) |
| "IdentifierValueType" | (1) |
| "Jahr" | (3) |
| "JahrTyp" | (1) |
| "KeyType" | (1) |
| "Monat" | (3) |
| "MonatsTyp" | (1) |
| "PLZTyp" | (1) |
| "Ort des Verkaufs" | (3) |
| "Ort des VerkaufsArtikel" | (6) |
| "Produktfamilie" | (3) |
| "Produktgruppe" | (3) |
| "Produktkategorie" | (3) |
| "Quartal" | (3) |
| "QuartalsTyp" | (1) |
| "Region" | (3) |
| "Staat" | (3) |
| "Stadt" | (3) |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | |
|---|---|
| "Strassenbereich" | (3) |
| "StückzahlTyp" | (1) |
| "Tag" | (3) |
| "TagTyp" | (1) |
| "Text" | (1) |
| "Verkauf" | (8) |
| "Verkaufsbezirk" | (3) |
| "Verkaufszahl" | (8) |
| "Verkauftes Produkt" | (8) |
| "Währung" | (1) |
| "Woche" | (3) |
| "WochenTyp" | (1) |

| **Attribute** | |
|---|---|
| ("Artikel.Artikelcode", "**ArtikelcodeTyp**") | (4) |
| ("Artikel.Bezeichnung", "**Text**") | (4) |
| ("Artikel.ID","**KeyType**") | (3) |
| ("Artikel.Produktgruppe.ForeignID", "**ForeignKeyType**") 5 | |
| ("Einkommen.Betrag","**Währung**") | (8) |
| ("Einkommen.Quartal.ForeignID", "**ForeignKeyType**") | (9) |
| ("Einkommen.Strassenbereich.ForeignID", "**ForeignKeyType**") | (9) |
| ("Filialkategorie.Bezeichnung", "**Text**") | (4) |
| ("Filialkategorie.ID", "**KeyType**") | (3) |
| ("Filialkategorie.Filialoberkategorie.ForeignID", "**ForeignKeyType**") 5 | |
| ("Filialoberkategorie.Bezeichnung", "**Text**") | (4) |
| ("Filialoberkategorie.ID", "**KeyType**") | (3) |
| ("Jahr.Bezeichnung", "**JahrTyp**") | (4) |
| ("Jahr.ID","**KeyType**") | (3) |
| ("Monat.Bezeichnung", "**MonatsTyp**") | (4) |
| ("Monat.ID","**KeyType**") | (3) |
| ("Monat.Quartal.ForeignID", "**ForeignKeyType**") 5 | |
| ("Ort des VerkaufsArtikel.ID" ,"**PrimaryKeyType**") | (6) |
| ("Ort des VerkaufsArtikel.Artikel.ForeignID" ,"**ForeignKeyType**") | (6) |
| ("Ort des VerkaufsArtikel.Ort des Verkaufs.ForeignID" ,"**ForeignKeyType**")) | (6) |
| ("Ort des Verkaufs.Bezeichnung", "**Text**") | (4) |
| ("Ort des Verkaufs.ID","**KeyType**") | (3) |
| ("Ort des Verkaufs.Filialkategorie.ForeignID", "**ForeignKeyType**") 5 | |
| ("Ort des Verkaufs.Kaufhaus.Fläche", "**FlächenTyp**") | (4) |
| ("Ort des Verkaufs.Kaufhaus.Gesamtfläche", "**FlächenTyp**") | (4) |
| ("Ort des Verkaufs.Filiale.Filialleiter", "**Text**") | (4) |
| ("Ort des Verkaufs.Filiale.Filialart", "**Text**") | (4) |
| ("Ort des Verkaufs.Stadt.ForeignID", "**ForeignKeyType**") 5 | |
| ("Ort des Verkaufs.Type", "**IdentifierValueType**") | (3) |
| ("Produktfamilie.Bezeichung", "**Text**") | (4) |
| ("Produktfamilie.ID","**KeyType**") | (3) |
| ("Produktfamilie.Produktkategorie.ForeignID", "**ForeignKeyType**") 5 | |
| ("Produktgruppe.Bezeichnung", "**Text**") | (4) |
| ("Produktgruppe.ID","**KeyType**") | (3) |
| ("Produktgruppe.Produktfamilie.ForeignID", "**ForeignKeyType**") 5 | |
| ("Produktkategorie.Bezeichnung", "**Text**") | (4) |
| ("Produktkategorie.ID","**KeyType**") | (3) |
| ("Quartal.Bezeichnung", "**QuartalsTyp**") | (4) |
| ("Quartal.ID","**KeyType**") | (3) |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | |
|---|---|
| (”Quartal.Jahr.ForeignID”, ”**ForeignKeyType**”) 5 | |
| (”Region.Bezeichnung”, ”**Text**”) | (4) |
| (”Region.ID”,”**KeyType**”) | (3) |
| (”Region.Staat.ForeignID”, ”**ForeignKeyType**”) 5 | |
| (”Region.Typ der Region”, ”**Bundesland, Kanton**”) | (4) |
| (”Staat.Bezeichnung”, ”**Text**”) | (4) |
| (”Staat.ID”,”**KeyType**”) | (3) |
| (”Stadt.Bezeichnung”, ”**Text**”) | (4) |
| (”Stadt.ID”,”**KeyType**”) | (3) |
| (”Stadt.PLZ”, ”**PLZTyp**”) | (4) |
| (”Stadt.Region.ForeignID”, ”**ForeignKeyType**”) 5 | |
| (”Stadt.Verkaufsbezirk.ForeignID”, ”**ForeignKeyType**”) 5 | |
| (”Strassenbereich.Bezeichnung”, ”**Text**”) | (4) |
| (”Strassenbereich.ID”,”**KeyType**”) | (3) |
| (”Strassenbereich.Stadt.ForeignID”, ”**ForeignKeyType**”) 5 | |
| (”Tag.Bezeichnung”, ”**DatumTyp**”) | (4) |
| (”Tag.ID”,”**KeyType**”) | (3) |
| (”Tag.Monat.ForeignID”, ”**ForeignKeyType**”) 5 | |
| (”Tag.Woche.ForeignID”, ”**ForeignKeyType**”) 5 | |
| (”Verkauf.Betrag”,”**Währung**”) | (8) |
| (”Verkauf.Ort des Verkaufs.ForeignID”, ”**ForeignKeyType**”) | (9) |
| (”Verkauf.Tag.ForeignID”, ”**ForeignKeyType**”) | (9) |
| (”Verkaufsbezirk.Bezeichnung”, ”**Text**”) | (4) |
| (”Verkaufsbezirk.ID”,”**KeyType**”) | (3) |
| (”Verkaufsbezirk.Region.ForeignID”, ”**ForeignKeyType**”) 5 | |
| (”Verkaufszahl.Anzahl”,”**StückzahlTyp**”) | (8) |
| (”Verkaufszahl.Artikel.ForeignID”, ”**ForeignKeyType**”) | (9) |
| (”Verkaufszahl.Ort des Verkaufs.ForeignID”, ”**ForeignKeyType**”) | (9) |
| (”Verkaufszahl.Tag.ForeignID”, ”**ForeignKeyType**”) | (9) |
| (”Verkauftes Produkt.Anzahl”,”**StückzahlTyp**”) | (8) |
| (”Verkauftes Produkt.Artikel.ForeignID”, ”**ForeignKeyType**”) | (9) |
| (”Verkauftes Produkt.Einzelpreis”,(”**Währung**”) | (8) |
| (”Verkauftes Produkt.Gesamtpreis”,”**Währung**”) | (8) |
| (”Verkauftes Produkt.Ort des Verkaufs.ForeignID”, ”**ForeignKeyType**”) | (9) |
| (”Verkauftes Produkt.Tag.ForeignID”, ”**ForeignKeyType**”) | (9) |
| (”Woche.Bezeichnung”, ”**WochenTyp**”) | (4) |
| (”Woche.ID”” **KeyType**”) | (3) |

## Metadaten

| **Additivity** | |
|---|---|
| (”Einkommen.Betrag”,”Strassenbereich”,Additivity,(”Ort Einkommen”,”Einkommen”, ”ALL”)) | (10) |
| (”Einkommen.Betrag”,”Quartal”,Additivity, (”Zeit Einkommen”,”Einkommen”, ”ALL”)) | (10) |
| (”Verkauf.Betrag”,”Ort des Verkaufs”,Additivity,(”Ort Verkauf”,”Verkauf”,”ALL”)) | (10) |
| (”Verkauf.Betrag”,”Tag”,Additivity,(”Zeit Verkauf”,”Verkauf”,”ALL”)) | (10) |
| (”Verkaufszahl.Anzahl”,”Artikel”,Additivity,(”Produkt”,”Verkaufszahl”,”ALL”)) | (10) |
| (”Verkaufszahl.Anzahl”,”Ort des Verkaufs”,Additivity,(”Ort”,”Verkaufszahl”,”ALL”)) | (10) |
| (”Verkaufszahl.Anzahl”,”Tag”,Additivity,(”Zeit”,”Verkaufszahl”,”ALL”)) | (10) |
| (”Verkauftes Produkt.Anzahl”,”Artikel”,Additivity, (”Artikel”,”Verkauftes Produkt”, ”ALL”)) | (10) |
| (”Verkauftes Produkt.Anzahl”,”Ort des Verkaufs”,Additivity, (”Ort Verkauf”,”Verkauftes Produkt”,{”SUM, MIN, MAX, AVG”})) | (10) |
| (”Verkauftes Produkt.Anzahl”,”Tag”,Additivity, (”Zeit Verkauf”, ”Verkauftes Produkt”,”ALL”)) | (10) |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | |
|---|---|
| (”Verkauftes Produkt.Einzelpreis”,”Artikel”,Additivity, (”Artikel”,”Verkauftes Produkt”,”ALL”)) | (10) |
| (”Verkauftes Produkt.Einzelpreis”,”Ort des Verkaufs”,Additivity, (”Ort Verkauf”,”Verkauftes Produkt”,{”SUM, MIN, MAX, AVG”})) | (10) |
| (”Verkauftes Produkt.Einzelpreis”,”Tag”,Additivity, (”Zeit Verkauf”,”Verkauftes Produkt”,”ALL”)) | (10) |
| (”Verkauftes Produkt.Gesamtpreis”,”Artikel”,Additivity, (”Artikel”,”Verkauftes Produkt”,”ALL”)) | (10) |
| (”Verkauftes Produkt.Gesamtpreis”,”Ort des Verkaufs”,Additivity, (”Ort Verkauf”,”Verkauftes Produkt”,{”SUM, MIN, MAX, AVG”})) | (10) |
| (”Verkauftes Produkt.Gesamtpreis”,”Tag”,Additivity, (”Zeit Verkauf”,”Verkauftes Produkt”,”ALL”)) | (10) |
| **Association** | |
| (”Ort des Verkaufs”,”Artikel”,Association,(”Geführter Artikel”,”wird geführt”, ”führt”,ALL_TYPES,ALL_TYPES,(”Ort des Verkaufs.ID”), (”Ort des VerkaufsArtikel.Ort des Verkaufs.ForeignID”),(”Artikel.ID”), (”Ort des VerkaufsArtikel.Artikel.ForeignID”))) | (6) |
| **Composition** | |
| (”Verkauf”,”Verkauftes Produkt”,Composition, (”Verkauftes Produkt”,”{0..*}”)) | (8) |
| **Computation** | |
| (”Berechnung Jahr”,Computation,((”Woche.Bezeichnung”),”ISO Wochenberechnung”, ”Jahr.Bezeichnung”)) | (5b) |
| (”Berechnung von Geographische Abbildung”,Computation, ((”Ort des Verkaufs.Bezeichnung”),”Geographische Abbildung”,”Strassenbereich.Bezeichnung”)) | (7) |
| (”Berechnung von Attribut Gesamtpreis”,Computation,((”Verkauftes Produkt.Einzelpreis”), ”Verkauftes Produkt.Anzahl”),”Einzelpreis * Anzahl”,”Verkauftes Produkt.Gesamtpreis”)) | (8) |
| **ConceptualKey** | |
| (”Artikel”,ConceptualKey,(”Artikel.Artikelcode”)) | (4) |
| (”Filialkategorie”,ConceptualKey,(”Filialkategorie.Bezeichnung”)) | (4) |
| (”Filialoberkategorie”,ConceptualKey,(”Filialoberkategorie.Bezeichnung”)) | (4) |
| (”Jahr”,ConceptualKey,(”Jahr.Bezeichnung”)) | (4) |
| (”Monat”,ConceptualKey,(”Monat.Bezeichnung”)) | (4) |
| (”Ort des VerkaufsArtikel”,ConceptualKey, (”Ort des VerkaufsArtikel.Ort des Verkaufs.ForeignID”)) | (6) |
| (”Ort des VerkaufsArtikel”,ConceptualKey, (”Ort des VerkaufsArtikel.Artikel.ForeignID”)) | (6) |
| (”Produktfamilie”,ConceptualKey,(”Produktfamilie.Bezeichnung”)) | (4) |
| (”Produktgruppe”,ConceptualKey,(”Produktgruppe.Bezeichnung”)) | (4) |
| (”Produktkategorie”,ConceptualKey,(”Produktkategorie.Bezeichnung”)) | (4) |
| (”Quartal”,ConceptualKey,(”Quartal.Bezeichnung”)) | (4) |
| (”Region”,ConceptualKey,(”Region.Bezeichnung”)) | (4) |
| (”Staat”,ConceptualKey,(”Staat.Bezeichnung”)) | (4) |
| (”Stadt”,ConceptualKey,(”Stadt.PLZ”)) | (4) |
| (”Strassenbereich”,ConceptualKey,(”Strassenbereich.Bezeichnung”)) | (4) |
| (”Tag”,ConceptualKey,(”Tag.Bezeichnung”)) | (4) |
| (”Verkaufsbezirk”,ConceptualKey,(”Verkaufsbezirk.Bezeichnung”)) | (4) |
| (”Woche”,ConceptualKey,(”Woche.Bezeichnung”)) | (4) |
| **Dimension** | |
| (”Einkommen”,”Quartal”, Dimension, (”Zeit Einkommen”, ALL_TYPES, ALL_TYPES, (”Einkommen.Quartal.ForeignID”),(”Quartal.ID”)) | (9) |
| (”Einkommen”, ”Strassenbereich”, Dimension, (”Ort Einkommen”, ALL_TYPES, ALL_TYPES, (”Einkommen.Strassenbereich.ForeignID”),(”Strassenbereich.ID”)) | (9) |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | |
|---|---|
| (”Verkauf”, ”Ort des Verkaufs”, Dimension, (”Ort Verkauf”, ALL_TYPES, ALL_TYPES, (”Verkauf.Ort des Verkaufs.ForeignID”),(”Ort des Verkaufs.ID”)) | (9) |
| (”Verkauf”, ”Tag”, Dimension, (”Zeit Verkauf”, ALL_TYPES, ALL_TYPES, (”Verkauf.Tag.ForeignID”),(”Tag.ID”)) | (9) |
| (”Verkaufszahl”, ”Artikel”, Dimension, (”Produkt”, ALL_TYPES, ALL_TYPES, (”Verkaufszahl.Artikel.ForeignID”),(”Artikel.ID”)) | (9) |
| (”Verkaufszahl”, ”Tag”, Dimension, (”Zeit”, ALL_TYPES, ALL_TYPES, (”Verkaufszahl.Tag.ForeignID”),(”Tag.ID”)) | (9) |
| (”Verkaufszahl”, ”Ort des Verkaufs”, Dimension, (”Ort”, ALL_TYPES, ALL_TYPES, (”Verkaufszahl.Ort des Verkaufs.ForeignID”),(”Ort des Verkaufs.ID”)) | (9) |
| (”Verkauftes Produkt”, ”Artikel”, Dimension, (”Artikel”, ALL_TYPES, ALL_TYPES, (”Verkauftes Produkt.Artikel.ForeignID”),(”Artikel.ID”)) | (9) |
| (”Verkauftes Produkt”, ”Ort des Verkaufs”, Dimension, (”Ort Verkauf”, ALL_TYPES, ALL_TYPES, (”Verkauftes Produkt.Ort des Verkaufs.ForeignID”),(”Ort des Verkaufs.ID”)) | (9) |
| (”Verkauftes Produkt”, ”Tag”, Dimension, (”Zeit Verkauf”, ALL_TYPES, ALL_TYPES, (”Verkauftes Produkt.Tag.ForeignID”),(”Tag.ID”)) | (9) |

| **DimensionalMapping** | |
|---|---|
| (”Ort des Verkaufs” ,”Strassenbereich”, DimensionalMapping, (”Abbildung Ort des Verkaufs nach Strassenbereich”, ALL_TYPES, ALL_TYPES, ”Berechnung von Geographische Abbildung”)) | (7) |

| **Identifier** | |
|---|---|
| (”Ort des Verkaufs”,Identifier, (”Ort des Verkaufs.Type”)) | (3) |

| **IdentifierValue** | |
|---|---|
| (”Ort des Verkaufs.Type”,IdentifierValue, ({”Filiale”, ”Kaufhaus”})) | (3) |

| **Multiplicity** | |
|---|---|
| (”Artikel”,Multiplicity, ({”Artikel.Produktgruppe.ForeignID”},”{0..*}”)) 5 | |
| (”Einkommen” ,Multiplicity, (”Einkommen.Quartal.ForeignID”, ”{0..*}”)) | (9) |
| (”Einkommen” ,Multiplicity, (”Einkommen.Strassenbereich.ForeignID”, ”{0..*}”)) | (9) |
| (”Filialkategorie”,Multiplicity, ({”Filialkategorie.Filialoberkategorie.ForeignID”},(”{0..*}”)) 5 | |
| (”Monat”,Multiplicity, ({”Monat.Quartal.ForeignID”},”{0..*}”)) 5 | |
| (”Ort des Verkaufs”,Multiplicity, ({”Ort des Verkaufs.Filialkategorie.ForeignID”},”{0..*}”)) 5 | |
| (”Ort des Verkaufs”,Multiplicity, ({”Ort des Verkaufs.Stadt.ForeignID”},”{0..*}”)) 5 | |
| (”Ort des VerkaufsArtikel” ,Multiplicity, (”Ort des VerkaufsArtikel.Artikel.ForeignID”, ”{0..*}”)) | (6) |
| (”Ort des VerkaufsArtikel” ,Multiplicity, (”Ort des VerkaufsArtikel.Ort des Verkaufs.ForeignID”, ”{0..*}”)) | (6) |
| (”Produktfamilie”,Multiplicity, ({”Produktfamilie.Produktkategorie.ForeignID”},”{0..*}”)) 5 | |
| (”Produktgruppe”,Multiplicity, ({”Produktgruppe.Produktfamilie.ForeignID”},”{0..*}”)) 5 | |
| (”Quartal”,Multiplicity, ({”Quartal.Jahr.ForeignID”},”{0..*}”)) 5 | |
| (”Region”,Multiplicity, ({”Region.Staat.ForeignID”},”{0..*}”)) 5 | |
| (”Stadt”,Multiplicity, ({”Stadt.Region.ForeignID”},”{0..*}”)) 5 | |
| (”Stadt”,Multiplicity, ({”Stadt.Verkaufsbezirk.ForeignID”},”{0..*}”)) 5 | |
| (”Strassenbereich”,Multiplicity, ({”Strassenbereich.Stadt.ForeignID”},”{0..*}”)) 5 | |
| (”Tag”,Multiplicity, ({”Tag.Monat.ForeignID”},”{0..*}”)) 5 | |
| (”Tag”,Multiplicity, ({”Tag.Woche.ForeignID”},”{0..*}”)) 5 | |
| (”Verkauf” ,Multiplicity, (”Verkauf.Ort des Verkaufs.ForeignID”, ”{0..*}”)) | (9) |
| (”Verkauf” ,Multiplicity, (”Verkauf.Tag.ForeignID”, ”{0..*}”)) | (9) |
| (”Verkaufsbezirk”,Multiplicity, ({”Verkaufsbezirk.Region.ForeignID”},”{0..*}”)) 5 | |
| (”Verkaufszahl” ,Multiplicity, (”Verkaufszahl.Artikel.ForeignID”, ”{0..*}”)) | (9) |
| (”Verkaufszahl” ,Multiplicity, (”Verkaufszahl.Ort des Verkaufs.ForeignID”, ”{0..*}”)) | (9) |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | |
|---|---|
| ("Verkaufszahl" ,Multiplicity, ("Verkaufszahl.Tag.ForeignID", "{0..*}")) | (9) |
| ("Verkauftes Produkt" ,Multiplicity,<br>("Verkauftes Produkt.Artikel.ForeignID", "{0..*}")) | (9) |
| ("Verkauftes Produkt" ,Multiplicity,<br>("Verkauftes Produkt.Ort des Verkaufs.ForeignID", "{0..*}")) | (9) |
| ("Verkauftes Produkt" ,Multiplicity, ("Verkauftes Produkt.Tag.ForeignID", "{0..*}")) | (9) |

| **ObjectType** | |
|---|---|
| ("Artikel",ObjectType, ("Relation", "Dimension")) | (3) |
| ("ArtikelcodeTyp",ObjectType, ("DataType","")) | (1) |
| ("Bundesland, Kanton",ObjectType, ("DataType","")) | (1) |
| ("Einkommen", ObjectType,("Relation", "Fact")) | (8) |
| ("Filialkategorie",ObjectType, ("Relation", "Dimension")) | (3) |
| ("Filialoberkategorie",ObjectType, ("Relation", "Dimension")) | (3) |
| ("FlächenTyp",ObjectType, ("DataType","")) | (1) |
| ("ForeignKeyType",ObjectType, ("DataType","")) | (1) |
| ("IdentifierValueType",ObjectType, ("DataType","")) | (1) |
| ("Jahr",ObjectType, ("Relation", "Dimension")) | (3) |
| ("JahrTyp",ObjectType, ("DataType","")) | (1) |
| ("KeyType",ObjectType, ("DataType","")) | (1) |
| ("Monat",ObjectType, ("Relation", "Dimension")) | (3) |
| ("MonatsTyp",ObjectType, ("DataType","")) | (1) |
| ("Ort des Verkaufs",ObjectType, ("Relation", "Dimension")) | (3) |
| ("PLZTyp",ObjectType, ("DataType","")) | (1) |
| ("Produktfamilie",ObjectType, ("Relation", "Dimension")) | (3) |
| ("Produktgruppe",ObjectType, ("Relation", "Dimension")) | (3) |
| ("Produktkategorie",ObjectType, ("Relation", "Dimension")) | (3) |
| ("Quartal",ObjectType, ("Relation", "Dimension")) | (3) |
| ("QuartalsTyp",ObjectType, ("DataType","")) | (1) |
| ("Region",ObjectType, ("Relation", "Dimension")) | (3) |
| ("Staat",ObjectType, ("Relation", "Dimension")) | (3) |
| ("Stadt",ObjectType, ("Relation", "Dimension")) | (3) |
| ("Strassenbereich",ObjectType, ("Relation", "Dimension")) | (3) |
| ("StückzahlTyp",ObjectType, ("DataType","")) | (1) |
| ("Tag",ObjectType, ("Relation", "Dimension")) | (3) |
| ("TagTyp",ObjectType, ("DataType","")) | (1) |
| ("Text",ObjectType, ("DataType","")) | (1) |
| ("Verkauf", ObjectType,("Relation", "Fact")) | (8) |
| ("Verkaufsbezirk",ObjectType, ("Relation", "Dimension")) | (3) |
| ("Verkaufszahl", ObjectType,("Relation", "Fact")) | (8) |
| ("Verkauftes Produkt", ObjectType,("Relation", "Fact")) | (8) |
| ("Währung",ObjectType, ("DataType","")) | (1) |
| ("Woche",ObjectType, ("Relation", "Dimension")) | (3) |
| ("WochenTyp",ObjectType, ("DataType","")) | (1) |

| **Optional** | |
|---|---|
| ("Artikel",Optional,("Artikel.Bezeichnung")) | (4) |

| **PrimaryKey** | |
|---|---|
| ("Artikel",PrimaryKey, ("Artikel.ID")) | (3) |
| ("Einkommen", PrimaryKey, ("Einkommen.Quartal.ForeignID")) | (9) |
| ("Einkommen", PrimaryKey, ("Einkommen.Strassenbereich.ForeignID")) | (9) |
| ("Filialkategorie",PrimaryKey, ("Filialkategorie.ID")) | (3) |
| ("Filialoberkategorie",PrimaryKey, ("Filialoberkategorie.ID")) | (3) |
| ("Jahr",PrimaryKey, ("Jahr.ID")) | (3) |
| ("Monat",PrimaryKey, ("Monat.ID")) | (3) |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | |
|---|---|
| (”Ort des Verkaufs”,PrimaryKey, (”Ort des Verkaufs.ID”)) | (3) |
| (”Ort des VerkaufsArtikel”, PrimaryKey, (”Ort des VerkaufsArtikel.ID”)) | (6) |
| (”Produktfamilie”,PrimaryKey, (”Produktfamilie.ID”)) | (3) |
| (”Produktgruppe”,PrimaryKey, (”Produktgruppe.ID”)) | (3) |
| (”Produktkategorie”,PrimaryKey, (”Produktkategorie.ID”)) | (3) |
| (”Quartal”,PrimaryKey, (”Quartal.ID”)) | (3) |
| (”Region”,PrimaryKey, (”Region.ID”)) | (3) |
| (”Staat”,PrimaryKey, (”Staat.ID”)) | (3) |
| (”Stadt”,PrimaryKey, (”Stadt.ID”)) | (3) |
| (”Strassenbereich”,PrimaryKey, (”Strassenbereich.ID”)) | (3) |
| (”Tag”,PrimaryKey, (”Tag.ID”)) | (3) |
| (”Verkauf”, PrimaryKey, (”Verkauf.Ort des Verkaufs.ForeignID”)) | (9) |
| (”Verkauf”, PrimaryKey, (”Verkauf.Tag.ForeignID”)) | (9) |
| (”Verkaufsbezirk”,PrimaryKey, (”Verkaufsbezirk.ID”)) | (3) |
| (”Verkaufszahl”, PrimaryKey, (”Verkaufszahl.Artikel.ForeignID”)) | (9) |
| (”Verkaufszahl”, PrimaryKey, (”Verkaufszahl.Tag.ForeignID”)) | (9) |
| (”Verkaufszahl”, PrimaryKey, (”Verkaufszahl.Ort des Verkaufs.ForeignID”)) | (9) |
| (”Verkauftes Produkt”, PrimaryKey, (”Verkauftes Produkt.Artikel.ForeignID”)) | (9) |
| (”Verkauftes Produkt”, PrimaryKey, (”Verkauftes Produkt.Ort des Verkaufs.ForeignID”)) | (9) |
| (”Verkauftes Produkt”, PrimaryKey, (”Verkauftes Produkt.Tag.ForeignID”)) | (9) |
| (”Woche”,PrimaryKey, (”Woche.ID”)) | (3) |

| **Reference** | |
|---|---|
| (”Artikel.Produktgruppe.ForeignID”, Reference,(”Produktgruppe.ID”)) 5 | |
| (”Einkommen.Quartal.ForeignID”, Reference, (”Quartal.ID”)) | (9) |
| (”Einkommen.Strassenbereich.ForeignID”, Reference, (”Strassenbereich.ID”)) | (9) |
| (”Filialkategorie.Filialoberkategorie.ForeignID”, Reference,(”Filialoberkategorie.ID”)) 5 | |
| (”Monat.Quartal.ForeignID”, Reference,(”Quartal.ID”)) 5 | |
| (”Ort des Verkaufs.Filialkategorie.ForeignID”, Reference,(”Filialkategorie.ID”)) 5 | |
| (”Ort des Verkaufs.Stadt.ForeignID”, Reference,(”Stadt.ID”)) 5 | |
| (”Ort des VerkaufsArtikel.Ort des Verkaufs.ForeignID” ,Reference, (”Ort des Verkaufs.ID”)) | (6) |
| (”Ort des VerkaufsArtikel.Artikel.ForeignID” ,Reference, (”Artikel.ID”)) | (6) |
| (”Produktfamilie.Produktkategorie.ForeignID”, Reference,(”Produktkategorie.ID”)) 5 | |
| (”Produktgruppe.Produktfamilie.ForeignID”, Reference,(”Produktfamilie.ID”)) 5 | |
| (”Quartal.Jahr.ForeignID”, Reference,(”Jahr.ID”)) 5 | |
| (”Region.Staat.ForeignID”, Reference,(”Staat.ID”)) 5 | |
| (”Stadt.Region.ForeignID”, Reference,(”Region.ID”)) 5 | |
| (”Stadt.Verkaufsbezirk.ForeignID”, Reference,(”Verkaufsbezirk.ID”)) 5 | |
| (”Strassenbereich.Stadt.ForeignID”, Reference,(”Stadt.ID”)) 5 | |
| (”Tag.Monat.ForeignID”, Reference,(”Monat.ID”)) 5 | |
| (”Tag.Woche.ForeignID”, Reference,(”Woche.ID”)) 5 | |
| (”Verkauf.Ort des Verkaufs.ForeignID”, Reference, (”Ort des Verkaufs.ID”)) | (9) |
| (”Verkauf.Tag.ForeignID”, Reference, (”Tag.ID”)) | (9) |
| (”Verkaufsbezirk.Region.ForeignID”, Reference,(”Region.ID”)) 5 | |
| (”Verkaufszahl.Artikel.ForeignID”, Reference, (”Artikel.ID”)) | (9) |
| (”Verkaufszahl.Ort des Verkaufs.ForeignID”, Reference, (”Ort des Verkaufs.ID”)) | (9) |
| (”Verkaufszahl.Tag.ForeignID”, Reference, (”Tag.ID”)) | (9) |
| (”Verkauftes Produkt.Artikel.ForeignID”, Reference, (”Artikel.ID”)) | (9) |
| (”Verkauftes Produkt.Ort des Verkaufs.ForeignID”, Reference, (”Ort des Verkaufs.ID”)) | (9) |
| (”Verkauftes Produkt.Tag.ForeignID”, Reference, (”Tag.ID”)) | (9) |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| **RollUp** | |
|---|---|
| (”Artikel”,”Produktgruppe”, RollUp,(”Produktgruppe”,ALL_TYPES,ALL_TYPES, 5 (”Artikel.Produktgruppe.ForeignID”),(”Produktgruppe.ID”), COMPLETE)) | |
| (”Filialkategorie”,”Filialoberkategorie”, RollUp,(”Filialoberkategorie”,{”Filiale”}, 5 ALL_TYPES, (”Filialkategorie.Filialoberkategorie.ForeignID”),(”Filialoberkategorie.ID”), COMPLETE)) | |
| (”Monat”,”Quartal”, RollUp,(”Quartal”,ALL_TYPES,ALL_TYPES, 5 ”Monat.Quartal.ForeignID”,”Quartal.ID”, COMPLETE)) | |
| (”Ort des Verkaufs”,”Filialkategorie”, RollUp,(”Filialkategorie”,{”Filiale”}, 5 ALL_TYPES, (”Ort des Verkaufs.Filialkategorie.ForeignID”),(”Filialkategorie.ID”), NON-COMPLETE)) | |
| (”Ort des Verkaufs”,”Stadt”,RollUp,(”Stadt”,ALL_TYPES,ALL_TYPES, (”Ort des Verkaufs.Stadt.ForeignID”),(”Stadt.ID”), COMPLETE)) 5 | |
| (”Produktfamilie”,”Produktkategorie”, RollUp, (”Produktkategorie”,ALL_TYPES, 5 ALL_TYPES, (”Produktfamilie.Produktkategorie.ForeignID”),(”Produktkategorie.ID”), COMPLETE)) | |
| (”Produktgruppe”,”Produktfamilie”, RollUp,(”Produktfamilie”,ALL_TYPES,ALL_TYPES, 5 (”Produktgruppe.Produktfamilie.ForeignID”),(”Produktfamilie.ID”), COMPLETE)) | |
| (”Quartal”,”Jahr”, RollUp,(”Jahr”,ALL_TYPES,ALL_TYPES, 5 (”Quartal.Jahr.ForeignID”),(”Jahr.ID”), COMPLETE)) | |
| (”Region”,”Staat”, RollUp,(”Staat”,ALL_TYPES,ALL_TYPES, 5 (”Region.Staat.ForeignID”),(”Staat.ID”), COMPLETE)) | |
| (”Stadt”,”Region”, RollUp,(”Region”,ALL_TYPES,ALL_TYPES, 5 (”Stadt.Region.ForeignID”),(”Region.ID”), COMPLETE)) | |
| (”Stadt”,”Verkaufsbezirk”, RollUp,(”Verkaufsbezirk”,ALL_TYPES,ALL_TYPES, 5 (”Stadt.Verkaufsbezirk.ForeignID”),(”Verkaufsbezirk.ID”), NONCOMPLETE)) | |
| (”Strassenbereich”,”Stadt”, RollUp,(”Stadt”,ALL_TYPES,ALL_TYPES, 5 (”Strassenbereich.Stadt.ForeignID”),(”Stadt.ID”), COMPLETE)) | |
| (”Tag”,”Monat”, RollUp,(”Monat”,ALL_TYPES,ALL_TYPES, 5 (”Tag.Monat.ForeignID”),(”Monat.ID”), COMPLETE)) | |
| (”Tag”,”Woche”, RollUp,(”Woche”,ALL_TYPES,ALL_TYPES, 5 (”Tag.Woche.ForeignID”),(”Woche.ID”), COMPLETE)) | |
| (”Verkaufsbezirk”,”Region”, RollUp,(”Region”,ALL_TYPES,ALL_TYPES, 5 (”Verkaufsbezirk.Region.ForeignID”),(”Region.ID”), COMPLETE)) | |

| **SharedRollUp** | |
|---|---|
| (”Woche”,”Jahr”, SharedRollUp,(”Jahr”, ALL_TYPES, ALL_TYPES, ”Berechnung Jahr”, {”SUM”})) | (5b) |

| **Valid** | |
|---|---|
| (”Ort des Verkaufs.Kaufhaus.Fläche”, Valid, (”Ort des Verkaufs.Type”,{”Kaufhaus”})) | (4) |
| (”Ort des Verkaufs.Kaufhaus.Gesamtfläche”, Valid, (”Ort des Verkaufs.Type”, {”Kaufhaus”})) | (4) |
| (”Ort des Verkaufs.Filiale.Filialleiter”, Valid, (”Ort des Verkaufs.Type”,{”Filiale”})) | (4) |
| (”Ort des Verkaufs.Filiale.Filialart”, Valid, (”Ort des Verkaufs.Type”,{”Filiale”})) | (4) |

Tabelle A.1: Alle REMUS–Objekte des Beispiels Handelswelt

## A.3 *LCD of SQL*–Schema

In diesem Abschnitt wird die Abbildung des Beispielschemas von REMUS nach *LCD of SQL* dokumentiert, indem zunächst die deterministischen Funktionen definiert und anschließend alle erzeugten Schemaobjekte aufgelistet werden.

### A.3.1 Die deterministischen Funktionen

**Schritt 1:** $f_{det_{DataType}}$

Tabelle A.2 zeigt die in Schritt 1 von $f_{det_{DataType}}$ gelieferten Werte.

| **REMUS–Objekt** | **Datentyp in** *LCD of SQL* |
|---|---|
| ArtikelcodeTyp | String |
| Bundesland, Kanton | String<br>range=”Bundesland, Kanton” |
| FlächenTyp | LongInt |
| JahrTyp | Date<br>timePrecision=YEARS |
| MonatsTyp | Date<br>timePrecision=MONTHS |
| PLZTyp | String<br>columnSize=5 |
| QuartalsTyp | Date<br>timePrecision=QUARTERS |
| StückzahlTyp | LongInt |
| TagTyp | Date<br>timePrecision=DAYS |
| Text | String |
| Währung | Decimal |
| WochenTyp | Date<br>timePrecision=WEEKS |
| KeyType | AutoIncrement |
| ForeignKeyType | QuadInt |
| IdentifierValueType | String |

Tabelle A.2: Abbildung der Datentypen im Beispielschema

**Schritt 2:** $f_{det_{TableName}}$

Die Funktion $f_{det_{TableName}}$ liefert den Namen des Objektes im REMUS–Schemas, sofern dieser keine Sonderzeichen enthält, Leerzeichen sollen eliminiert und durch Unterstriche ersetzt werden, so dass es folgende Ausnahmen gibt:
$f_{det_{DataType}}(\text{”Ort des Verkaufs”}) = \text{”Ort\_des\_Verkaufs”}$
$f_{det_{DataType}}(\text{”Verkauftes Produkt”}) = \text{”Verkauftes\_Produkt”}$

Ebenso sollen Verbindungstabellen aufgelöster Assoziationen aufgrund einer organisationsweiten Richtlinie das Präfix "MTM"[1] tragen müssen, so dass
$f_{det_{DataType}}$("Ort des VerkaufsArtikel") = "MTMOrt_des_VerkaufsArtikel"
gilt.

**Schritt 3:** $f_{det_{AttributeName}}$

Beim Anlegen der Attribute soll $f_{det_{AttributeName}}$ jeweils den Tabellennamen als Präfix entfernen, und Umlaute sowie Sonderzeichen umwandeln. Außerdem sollen Fremdschlüsseleinträge das Suffix "FK" tragen.
Einige Beispiele:
$f_{det_{AttributeName}}$("Artikel.ID") = "ID".
$f_{det_{AttributeName}}$("Artikel.Produktgruppe.ForeignID") = "Produktgruppe_FK".
$f_{det_{AttributeName}}$("Ort des Verkaufs.Kaufhaus.Fläche") = "Flaeche".

**Schritt 4:** $f_{det_{Order}}$

Die Funktion $f_{det_{Order}}$ spielt bei den Fakttabellen eine Rolle, weil sich deren Primärschlüssel aus der Menge der Fremdschlüssel der Dimensionen zusammensetzen und diese Menge in der Regel mehrere Elemente enthält, ist eine Sortierung notwendig. Es sollen hierbei die Attribute alphabetisch sortiert werden.

**Schritt 6:** $f_{det_{Computation}}$

Das einzige berechnete Attribut im Schema ist "Gesamtpreis" in der Tabelle "Verkauftes_Produkt". Dieser ergibt sich aus dem Produkt von "Einzelpreis" und "Anzahl", so daß $f_{det_{Computation}}$ als Ergebnis "Einzelpreis * Anzahl" liefert.

**Schritt 7:** $f_{det_{IdentifierRule}}$ **und** $f_{det_{ValidRule}}$

Für die Auflüsung der Metadaten *Identifier*, *IdentifierValue* und *Valid* seien die beiden folgenden Funktionen definiert:

$f_{det_{IdentifierRule}}$
("Ort des Verkaufs", Identifier, ("Ort des Verkaufs.Type")))
$\stackrel{def}{=}$ "Type IN ('Filiale','Kaufhaus')"

und

$f_{det_{ValidRule}}$
("Ort des Verkaufs.Kaufhaus.Fläche", Valid, ("Ort des Verkaufs.Type",{"Kaufhaus"}))
$\stackrel{def}{=}$ "Flaeche IS NULL OR TYPE IN ('Kaufhaus')"

Für die anderen drei *Valid*–Metadaten sind analoge Regeln definiert.

[1] MTM steht als Akronym für *Many–To–Many*.

**Schritt 9:** $f_{det_{MultiplicityRange}}$

Alle im Schema definierten *Multiplicity*–Metadaten haben eine Multiplizität von "0..*" bzw. "1..*", so dass die Funktion $f_{det_{MultiplicityRange}}$ nicht näher spezifiziert zu werden braucht.

**Schritt 10:** $f_{det_{RollUpTypesToRule}}$

Die Funktion $f_{det_{RollUpTypesToRule}}$ soll für diejenigen *RollUp*– und *Dimension*–Metadaten, die nicht alle Typen zulassen, Regeln nach dem folgenden Muster bilden:

$f_{det_{RollUpTypesToRule}}$
((”Ort des Verkaufs”,”Filialkategorie”, RollUp,(”Filialkategorie”,{”Filiale”},
ALL_TYPES, (”Ort des Verkaufs.Filialkategorie.ForeignID”),(”Filialkategorie.ID”),
NONCOMPLETE)))
$\stackrel{def}{=}$ "(Filialkategorie.ForeignID IS NULL) OR
(Ort_des_Verkaufs.Type IN (”Filiale”))".

### A.3.2 Liste aller *LCD of SQL*–Schemaelemente

In der folgenden Tabelle sind die Objekte nach Typ sortiert. Die erste Spalte gibt für jedes Objekt einen eindeutigen Identifikator an, der von anderen Objekten referenziert werden kann. In der zweiten Spalte folgen die Werte des Objektes in der in Tabelle 8.7 festgelegten Tupelnotation. Die letzten beiden Spalten geben die Schritte der Abbildung an, in denen dieses Objekt erzeugt bzw. manipuiert worden ist.

*LCD of SQL*–**Objekte des Beipiels „Handelswelt“**

| **AdditivityMETA** | | | |
|---|---|---|---|
| AD01 | (”Valid operators for ’Betrag’ with respect to ’Ort_Verkauf’”, ALL, CO08, RC18, AdditivityMETA) | 11 | |
| AD02 | (”Valid operators for ’Betrag’ with respect to ’Zeit_Verkauf’”, ALL, CO08, RC19, AdditivityMETA) | 11 | |
| AD03 | (”Valid operators for ’Anzahl’ with respect to ’Produkt’”, ALL, CO01, RC20, AdditivityMETA) | 11 | |
| AD04 | (”Valid operators for ’Anzahl’ with respect to ’Ort’”, ALL, CO01, RC22, AdditivityMETA) | 11 | |
| AD05 | (”Valid operators for ’Anzahl’ with respect to ’Zeit’”, ALL, CO01, RC21, AdditivityMETA) | 11 | |
| AD06 | (”Valid operators for ’Anzahl’ with respect to ’Artikel’”, ALL, CO02, RC23, AdditivityMETA) | 11 | |
| AD07 | (”Valid operators for ’Anzahl’ with respect to ’Ort_Verkauf’”, {”SUM”, ”MIN”, ”MAX”, ”AVG”}, CO02, RC24, AdditivityMETA) | 11 | |
| AD08 | (”Valid operators for ’Anzahl’ with respect to ’Zeit_Verkauf’”, ALL, CO02, RC25, AdditivityMETA) | 11 | |
| AD09 | (”Valid operators for ’Einzelpreis’ with respect to ’Artikel’”, ALL, CO02, RC23, AdditivityMETA) | 11 | |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | | | |
|---|---|---|---|
| AD10 | (”Valid operators for ’Einzelpreis’ with respect to ’Ort_Verkauf’”, {”SUM”, ”MIN”, ”MAX”, ”AVG”}, CO02, RC24, AdditivityMETA) | 11 | |
| AD11 | (”Valid operators for ’Einzelpreis’ with respect to ’Zeit_Verkauf’”, ALL, CO02, RC25, AdditivityMETA) | 11 | |
| AD12 | (”Valid operators for ’Gesamtpreis’ with respect to ’Artikel’”, ALL, CO02, RC23, AdditivityMETA) | 11 | |
| AD13 | (”Valid operators for ’Gesamtpreis’ with respect to ’Ort_Verkauf’”, {”SUM”, ”MIN”, ”MAX”, ”AVG”}, CO02, RC24, AdditivityMETA) | 11 | |
| AD14 | (”Valid operators for ’Gesamtpreis’ with respect to ’Zeit_Verkauf’”, ALL, CO02, RC25, AdditivityMETA) | 11 | |
| AD15 | (”Valid operators for ’Betrag’ with respect to ’Ort_Einkommen’”, ALL, CO07, RC17, AdditivityMETA) | 11 | |
| AD16 | (”Valid operators for ’Betrag’ with respect to ’Zeit_Einkommen’”, ALL, CO07, RC16, AdditivityMETA) | 11 | |
| **Column** | | | |
| CO01 | (”Anzahl”, NULL, NULL, FALSE, FALSE, TA18, {}, ”StückzahlTyp”, NULL, {AD03, AD04, AD05}, Column) | 3 | 11 |
| CO02 | (”Anzahl”, NULL, NULL, FALSE, FALSE, TA19, {}, ”StückzahlTyp”, NULL, {AD06, AD07, AD08, AD09, AD10, AD11, AD12, AD13, AD14}, Column) | 3 | 11 |
| CO03 | (”Artikelcode”, NULL, NULL, FALSE, FALSE, TA01, {UK21}, ”ArtikelcodeTyp”, NULL, NULL, Column) | 3 | 5 |
| CO04 | (”Artikel_FK”, NULL, NULL, FALSE, FALSE, TA06, {UK24, FK26}, ”QuadInt”, NULL, NULL, Column) | 3 | 5,14 |
| CO05 | (”Artikel_FK”, NULL, NULL, FALSE, FALSE, TA18, {UK18, FK20}, ”QuadInt”, NULL, NULL, Column) | 3 | 4,10 |
| CO06 | (”Artikel_FK”, NULL, NULL, FALSE, FALSE, TA19, {UK19, FK23}, ”QuadInt”, NULL, NULL, Column) | 3 | 4,10 |
| CO07 | (”Betrag”, NULL, NULL, FALSE, FALSE, TA03, {}, ”Währung”, NULL, {AD15, AD16}, Column) | 3 | 11 |
| CO08 | (”Betrag”, NULL, NULL, FALSE, FALSE, TA16, {}, ”Währung”, NULL, {AD01, AD02}, Column) | 3 | 11 |
| CO09 | (”Bezeichnung”, NULL, NULL, FALSE, TRUE, TA01, {}, ”String”, NULL, NULL, Column) | 3 | 8 |
| CO10 | (”Bezeichnung”, NULL, NULL, FALSE, FALSE, TA02, {UK22}, ”JahrTyp”, NULL, NULL, Column) | 3 | 5 |
| CO11 | (”Bezeichnung”, NULL, NULL, FALSE, FALSE, TA04, {UK23}, ”MonatsTyp”, NULL, NULL, Column) | 3 | 5 |
| CO12 | (”Bezeichnung”, NULL, NULL, FALSE, FALSE, TA05, {}, ”String”, NULL, NULL, Column) | 3 | |
| CO13 | (”Bezeichnung”, NULL, NULL, FALSE, FALSE, TA07, {UK25}, ”String”, NULL, NULL, Column) | 3 | 5 |
| CO14 | (”Bezeichnung”, NULL, NULL, FALSE, FALSE, TA08, {UK26}, ”String”, NULL, NULL, Column) | 3 | 5 |
| CO15 | (”Bezeichnung”, NULL, NULL, FALSE, FALSE, TA09, {UK27}, ”String”, NULL, NULL, Column) | 3 | 5 |
| CO16 | (”Bezeichnung”, NULL, NULL, FALSE, FALSE, TA10, {UK28}, ”QuartalsTyp”, NULL, NULL, Column) | 3 | 5 |
| CO17 | (”Bezeichnung”, NULL, NULL, FALSE, FALSE, TA11, {UK40}, ”String”, NULL, NULL, Column) | 3 | 5 |
| CO19 | (”Bezeichnung”, NULL, NULL, FALSE, FALSE, TA12, {UK30}, ”Text”, NULL, NULL, Column) | 3 | 5 |
| CO20 | (”Bezeichnung”, NULL, NULL, FALSE, FALSE, TA13, {}, ”Text”, NULL, NULL, Column) | 3 | |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | | | |
|---|---|---|---|
| CO21 | (”Bezeichnung”, NULL, NULL, FALSE, FALSE, TA14, {UK32}, ”Text”, NULL, NULL, Column) | 3 | 5 |
| CO22 | (”Bezeichnung”, NULL, NULL, FALSE, FALSE, TA15, {UK33}, ”DatumTyp”, NULL, NULL, Column) | 3 | 5 |
| CO23 | (”Bezeichnung”, NULL, NULL, FALSE, FALSE, TA17, {UK34}, ”Text”, NULL, NULL, Column) | 3 | 5 |
| CO24 | (”Bezeichnung”, NULL, NULL, FALSE, FALSE, TA20, {UK35}, ”WochenTyp”, NULL, NULL, Column) | 3 | 5 |
| CO25 | (”Einzelpreis”, NULL, NULL, FALSE, FALSE, TA19, {}, ”Währung”, NULL, NULL, Column) | 3 | 11 |
| CO26 | (”Filialart”, NULL, NULL, FALSE, FALSE, TA05, {}, ”String”, NULL, NULL, Column) | 3 | |
| CO27 | (”Filialleiter”, NULL, NULL, FALSE, FALSE, TA05, {}, ”String”, NULL, NULL, Column) | 3 | |
| CO28 | (”Flaeche”, NULL, NULL, FALSE, FALSE,TA05, {}, ”LongInt”, NULL, NULL, Column) | 3 | |
| CO29 | (”Gesamtflaeche”, NULL, NULL, FALSE, FALSE, TA05, {}, ”LongInt”, NULL, NULL, Column) | 3 | |
| CO30 | (”Gesamtpreis”, NULL, ”Einzelpreis * Anzahl”, FALSE, FALSE, TA19, {}, ”Währung”, NULL, NULL, Column) | 3 | 6,11 |
| CO31 | (”ID”, NULL, NULL,TRUE, FALSE, TA01, {UK01}, ”AutoIncrement”, NULL, NULL, Column) | 3 | 4 |
| CO32 | (”ID”, NULL, NULL,TRUE, FALSE, TA02, {UK02}, ”AutoIncrement”, NULL, NULL, Column) | 3 | 4 |
| CO33 | (”ID”, NULL, NULL,TRUE, FALSE, TA04, {UK04}, ”AutoIncrement”, NULL, NULL, Column) | 3 | 4 |
| CO34 | (”ID”, NULL, NULL,TRUE, FALSE, TA05, {UK05}, ”AutoIncrement”, NULL, NULL, Column) | 3 | 4 |
| CO35 | (”ID”, NULL, NULL,TRUE, FALSE, TA06, {UK06}, ”AutoIncrement”, NULL, NULL, Column) | 3 | 4 |
| CO36 | (”ID”, NULL, NULL,TRUE, FALSE, TA07, {UK07}, ”AutoIncrement”, NULL, NULL, Column) | 3 | 4 |
| CO37 | (”ID”, NULL, NULL,TRUE, FALSE, TA08, {UK08}, ”AutoIncrement”, NULL, NULL, Column) | 3 | 4 |
| CO38 | (”ID”, NULL, NULL,TRUE, FALSE, TA09, {UK09}, ”AutoIncrement”, NULL, NULL, Column) | 3 | 4 |
| CO39 | (”ID”, NULL, NULL,TRUE, FALSE, TA10, {UK10}, ”AutoIncrement”, NULL, NULL, Column) | 3 | 4 |
| CO40 | (”ID”, NULL, NULL,TRUE, FALSE, TA11, {UK11}, ”AutoIncrement”, NULL, NULL, Column) | 3 | 4 |
| CO41 | (”ID”, NULL, NULL,TRUE, FALSE, TA12, {UK12}, ”AutoIncrement”, NULL, NULL, Column) | 3 | 4 |
| CO42 | (”ID”, NULL, NULL,TRUE, FALSE, TA13, {UK13}, ”AutoIncrement”, NULL, NULL, Column) | 3 | 4 |
| CO43 | (”ID”, NULL, NULL,TRUE, FALSE, TA14, {UK14}, ”AutoIncrement”, NULL, NULL, Column) | 3 | 4 |
| CO44 | (”ID”, NULL, NULL,TRUE, FALSE, TA15, {UK15}, ”AutoIncrement”, NULL, NULL, Column) | 3 | 4 |
| CO45 | (”ID”, NULL, NULL,TRUE, FALSE, TA17, {UK17}, ”AutoIncrement”, NULL, NULL, Column) | 3 | 4 |
| CO46 | (”ID”, NULL, NULL,TRUE, FALSE, TA20, {UK20}, ”AutoIncrement”, NULL, NULL, Column) | 3 | 4 |
| CO47 | (”Jahr_FK”, NULL, NULL, FALSE, FALSE, TA10, {FK08}, ”QuadInt”, NULL, NULL, Column) | 3 | 10 |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | | | |
|---|---|---|---|
| CO49 | (”Monat_FK”, NULL, NULL, FALSE, FALSE, TA15, {FK13}, ”QuadInt”, NULL, NULL, Column) | 3 | 10 |
| CO50 | (”Ort_des_Verkaufs_FK”, NULL, NULL, FALSE, FALSE, TA06, {UK24, FK27}, ”QuadInt”, NULL, NULL, Column) | 3 | 5,14 |
| CO51 | (”Ort_des_Verkaufs_FK”, NULL, NULL, FALSE, FALSE, TA16, {UK16, FK18}, ”QuadInt”, NULL, NULL, Column) | 3 | 4,10 |
| CO52 | (”Ort_des_Verkaufs_FK”, NULL, NULL, FALSE, FALSE, TA18, {UK18, FK22}, ”QuadInt”, NULL, NULL, Column) | 3 | 4,10 |
| CO53 | (”Ort_des_Verkaufs_FK”, NULL, NULL, FALSE, FALSE, TA19, {UK19, FK24}, ”QuadInt”, NULL, NULL, Column) | 3 | 4,10 |
| CO54 | (”PLZ”, NULL, NULL, FALSE, FALSE, TA13, {UK31}, ”PLZTyp”, NULL, NULL, Column) | 3 | 5 |
| CO55 | (”Produktfamilie_FK”, NULL, NULL, FALSE, FALSE, TA08, {FK07}, ”QuadInt”, NULL, NULL, Column) | 3 | 10 |
| CO56 | (”Produktgruppe_FK”, NULL, NULL, FALSE, FALSE, TA01, {FK01}, ”QuadInt”, NULL, NULL, Column) | 3 | 10 |
| CO57 | (”Produktkategorie_FK”, NULL, NULL, FALSE, FALSE, TA07, {FK06}, ”QuadInt”, NULL, NULL, Column) | 3 | 10 |
| CO58 | (”Quartal_FK”, NULL, NULL, FALSE, FALSE, TA03, {UK03, FK16}, ”QuadInt”, NULL, NULL, Column) | 3 | 4,10 |
| CO59 | (”Quartal_FK”, NULL, NULL, FALSE, FALSE, TA04, {FK03}, ”QuadInt”, NULL, NULL, Column) | 3 | 10 |
| CO60 | (”Region_FK”, NULL, NULL, FALSE, FALSE, TA13, {FK10}, ”QuadInt”, NULL, NULL, Column) | 3 | 10 |
| CO61 | (”Region_FK”, NULL, NULL, FALSE, FALSE, TA17, {FK15}, ”QuadInt”, NULL, NULL, Column) | 3 | 10 |
| CO62 | (”Staat_FK”, NULL, NULL, FALSE, FALSE, TA11, {FK09}, ”QuadInt”, NULL, NULL, Column) | 3 | 10 |
| CO63 | (”Stadt_FK”, NULL, NULL, FALSE, FALSE, TA05, {FK05}, ”QuadInt”, NULL, NULL, Column) | 3 | 10 |
| CO64 | (”Stadt_FK”, NULL, NULL, FALSE, FALSE, TA14, {FK12}, ”QuadInt”, NULL, NULL, Column) | 3 | 10 |
| CO65 | (”Strassenbereich_FK”, NULL, NULL, FALSE, FALSE, TA03, {UK03, FK17}, ”QuadInt”, NULL, NULL, Column) | 3 | 4,10 |
| CO66 | (”Tag_FK”, NULL, NULL, FALSE, FALSE, TA16, {UK16, FK19}, ”QuadInt”, NULL, NULL, Column) | 3 | 4,10 |
| CO67 | (”Tag_FK”, NULL, NULL, FALSE, FALSE, TA18, {UK18, FK21}, ”QuadInt”, NULL, NULL, Column) | 3 | 4,10 |
| CO68 | (”Tag_FK”, NULL, NULL, FALSE, FALSE, TA19, {UK19, FK25}, ”QuadInt”, NULL, NULL, Column) | 3 | 4,10 |
| CO69 | (”Type”, NULL, NULL, FALSE, FALSE, TA05, {}, ”String”, {CC1}, NULL, Column) | 3 | 7 |
| CO70 | (”Typ_der_Region”, NULL, NULL, FALSE, FALSE, TA11, {}, ”Bundesland, Kanton”, NULL, NULL, Column) | 3 | |
| CO71 | (”Verkaufsbezirk_FK”, NULL, NULL, FALSE, FALSE, TA13, {FK11}, ”QuadInt”, NULL, NULL, Column) | 3 | 10 |
| CO72 | (”Woche_FK”, NULL, NULL, FALSE, FALSE, TA15, {FK14}, ”QuadInt”, NULL, NULL, Column) | 3 | 10 |
| CO73 | (”Bezeichnung”, NULL, NULL, FALSE, FALSE, TA21, {UK38}, ”Text”, NULL, NULL, Column) | 3 | 5 |
| CO74 | (”Bezeichnung”, NULL, NULL, FALSE, FALSE, TA22, {UK39}, ”Text”, NULL, NULL, Column) | 3 | 5 |
| CO75 | (”ID”, NULL, NULL,TRUE, FALSE, TA21, {UK36}, ”AutoIncrement”, NULL, NULL, Column) | 3 | 4 |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | | | |
|---|---|---|---|
| CO76 | (”ID”, NULL, NULL,TRUE, FALSE, TA22, {UK37}, ”AutoIncrement”, NULL, NULL, Column) | 3 | 4 |
| CO77 | (”Filialkategorie_FK”, NULL, NULL, FALSE, FALSE, TA05, {FK04}, ”QuadInt”, NULL, NULL, Column) | 3 | 10 |
| CO78 | (”Filialoberkategorie_FK”, NULL, NULL, FALSE, FALSE, TA21, {FK02}, ”QuadInt”, NULL, NULL, Column) | 3 | 10 |
| **ColumnConstraint** | | | |
| CC01 | (”Integrity rule for attribute ’Type’”, ”Type IN (’Filiale’,’Kaufhaus’)”, CO69, ColumnConstraint) | (7) | |
| **ColumnType** | | | |
| CT01 | (”String”, ”ArtikelcodeTyp”, DOUBLE_BYTE, NULL, NULL, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, NULL,NULL, NULL, NULL, NULL, NULL, NULL, NULL, TRUE, NULL, ColumnType) | 1 | |
| CT02 | (”String”, ”Aufzählungstyp Region”, DOUBLE_BYTE, NULL, NULL, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, NULL,NULL, NULL, NULL, NULL, NULL, NULL, ”Bundesland, Kanton”, TRUE, NULL, ColumnType) | 1 | |
| CT03 | (”LongInt”, ”FlächenTyp”, NULL, NULL, NULL, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, TRUE, NULL, NULL, 0, NULL, 10, NULL, ”>=0”, TRUE, NULL, ColumnType) | 1 | |
| CT04 | (”Date”, ”JahrTyp”, NULL, NULL, NULL, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, NULL,NULL, NULL, NULL, NULL, NULL, NULL, NULL, TRUE, TIMEPRECISION_YEARS, ColumnType) | 1 | |
| CT05 | (”Date”, ”MonatsTyp”, NULL, NULL, NULL, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, NULL,NULL, NULL, NULL, NULL, NULL, NULL, NULL, TRUE, TIMEPRECISION_MONTHS, ColumnType) | 1 | |
| CT06 | (”String”, ”PLZTyp”, NULL, NULL, NULL, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, NULL,NULL, NULL, NULL, NULL, NULL, NULL, ”>=’00000’ AND <=’99999’”, TRUE, NULL, ColumnType) | 1 | |
| CT07 | (”Date”, ”QuartalsTyp”, NULL, NULL, NULL, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, NULL,NULL, NULL, NULL, NULL, NULL, NULL, NULL, TRUE, TIMEPRECISION_QUARTERS, ColumnType) | 1 | |
| CT08 | (”LongInt”, ”StückzahlTyp”, NULL, NULL, NULL, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, TRUE, NULL, NULL, 0, NULL, 10, NULL, ”>=0”, TRUE, NULL, ColumnType) | 1 | |
| CT09 | (”Date”, ”TagTyp”, NULL, NULL, NULL, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, NULL,NULL, NULL, NULL, NULL, NULL, NULL, NULL, TRUE, TIMEPRECISION_DAYS, ColumnType) | 1 | |
| CT10 | (”String”, ”Text”, DOUBLE_BYTE, NULL, NULL, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, NULL,NULL, NULL, NULL, NULL, NULL, NULL, NULL, TRUE, NULL, ColumnType) | 1 | |
| CT11 | (”Double”, ”Währung”, NULL, NULL, NULL, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, TRUE, NULL, NULL, 0, NULL, 10, NULL, ”>=0”, TRUE, NULL, ColumnType) | 1 | |
| CT12 | (”Date”, ”WochenTyp”, NULL, NULL, NULL, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, NULL,NULL, NULL, NULL, NULL, NULL, NULL, NULL, TRUE, TIMEPRECISION_WEEKS, ColumnType) | 1 | |
| CT13 | (”AutoIncrement”, ”PKTyp”, NULL, NULL, NULL, TRUE, TRUE, FALSE, FALSE, FALSE, FALSE, NULL,NULL, NULL, NULL, NULL, NULL, NULL, NULL, TRUE, NULL, ColumnType) | 1 | |
| CT14 | (”QuadInt”, ”FKTyp”, NULL, NULL, NULL, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, NULL,NULL, NULL, NULL, NULL, NULL, NULL, NULL, TRUE, NULL, ColumnType) | 1 | |
| CT15 | (”String”, ”IdentifierValueType”, DOUBLE_BYTE, NULL, NULL, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, NULL,NULL, NULL, NULL, NULL, NULL, NULL, NULL, TRUE, NULL, ColumnType) | 1 | |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| **CompositionMETA** | | | |
|---|---|---|---|
| CM01 | (”Verkauftes_Produkt”,{”0..*”},TA16,TA19,CompositionMETA) | 15 | |

| **ForeignKey** | | | |
|---|---|---|---|
| FK01 | (”Foreign key of table ’Artikel’ from table ’Produktgruppe’”, TA01,(CO56), FR01,ForeignKey) | 10 | |
| FK02 | (”Foreign key of table ’Filialkategorie’ from table ’Filialoberkategorie’”, TA21,(CO78), FR02,ForeignKey) | 10 | |
| FK03 | (”Foreign key of table ’Monat’ from table ’Quartal’”, TA04,(CO59), FR03,ForeignKey) | 10 | |
| FK04 | (”Foreign key of table ’Ort_des_Verkaufs’ from table ’Filialkategorie’”, TA05,(CO77), FR04,ForeignKey) | 10 | |
| FK05 | (”Foreign key of table ’Ort_des_Verkaufs’ from table ’Stadt’”, TA05,(CO63), FR05,ForeignKey) | 10 | |
| FK06 | (”Foreign key of table ’Produktfamilie’ from table ’Produktkategorie’”, TA07,(CO57), FR06,ForeignKey) | 10 | |
| FK07 | (”Foreign key of table ’Produktgruppe’ from table ’Produktfamilie’”, TA08,(CO55), FR07,ForeignKey) | 10 | |
| FK08 | (”Foreign key of table ’Quartal’ from table ’Jahr’”, TA10,(CO47), FR08,ForeignKey) | 10 | |
| FK09 | (”Foreign key of table ’Region’ from table ’Staat’”, TA11,(CO62), FR09,ForeignKey) | 10 | |
| FK10 | (”Foreign key of table ’Stadt’ from table ’Region’”, TA13,(CO60), FR10,ForeignKey) | 10 | |
| FK11 | (”Foreign key of table ’Stadt’ from table ’Verkaufsbezirk’”, TA13,(CO11), FR11,ForeignKey) | 10 | |
| FK12 | (”Foreign key of table ’Strassenbereich’ from table ’Stadt’”, TA14,(CO64), FR12,ForeignKey) | 10 | |
| FK13 | (”Foreign key of table ’Tag’ from table ’Monat’”, TA15,(CO49), FR13,ForeignKey) | 10 | |
| FK14 | (”Foreign key of table ’Tag’ from table ’Woche’”, TA15,(CO72), FR14,ForeignKey) | 10 | |
| FK15 | (”Foreign key of table ’Verkaufsbezirk’ from table ’Region’”, TA17,(CO61), FR15,ForeignKey) | 10 | |
| FK16 | (”Foreign key of table ’Einkommen’ from table ’Quartal’”, TA03,(CO58), FR16,ForeignKey) | 10 | |
| FK17 | (”Foreign key of table ’Einkommen’ from table ’Strassenbereich’”, TA03,(CO65), FR17,ForeignKey) | 10 | |
| FK18 | (”Foreign key of table ’Verkauf’ from table ’Ort_des_Verkaufs’”, TA16,(CO51), FR18,ForeignKey) | 10 | |
| FK19 | (”Foreign key of table ’Verkauf’ from table ’Tag’”, TA16,(CO66), FR19,ForeignKey) | 10 | |
| FK20 | (”Foreign key of table ’Verkaufszahl’ from table ’Artikel’”, TA18,(CO05), FR20,ForeignKey) | 10 | |
| FK21 | (”Foreign key of table ’Verkaufszahl’ from table ’Tag’”, TA18,(CO67), FR21,ForeignKey) | 10 | |
| FK22 | (”Foreign key of table ’Verkaufszahl’ from table ’Ort_des_Verkaufs’”, TA18,(CO52), FR22,ForeignKey) | 10 | |
| FK23 | (”Foreign key of table ’Verkauftes_Produkt’ from table ’Artikel’”, TA19,(CO06), FR23,ForeignKey) | 10 | |
| FK24 | (”Foreign key of table ’Verkauftes_Produkt’ from table ’Ort_des_Verkaufs’”, TA19,(CO53), FR24,ForeignKey) | 10 | |
| FK25 | (”Foreign key of table ’Verkauftes_Produkt’ from table ’Tag’”, TA19,(CO68), FR25,ForeignKey) | 10 | |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | | | |
|---|---|---|---|
| FK26 | (”Foreign key of table ’MTMOrt_des_VerkaufsArtikel’ from table ’Artikel’”, TA06,(CO04), FR26,ForeignKey) | 14 | |
| FK27 | ("Foreign key of table ’MTMOrt_des_VerkaufsArtikel’ from table ’Ort_des_Verkaufs’", TA06,(CO50), FR26,ForeignKey) | 14 | |
| **ForeignKeyRole** | | | |
| FR01 | (”Role of foreign key between tables ’Artikel’ and ’Produktgruppe’”,”0..*”, MATCHTYPE_FULL_MATCH, FK01, RC01, ForeignKeyRole) | 10 | |
| FR02 | (”Role of foreign key between tables ’Filialkategorie’ and ’Filialoberkategorie’”,10 ”0..*”,MATCHTYPE_FULL_MATCH, FK02, RC02, ForeignKeyRole) | | |
| FR03 | (”Role of foreign key between tables ’Monat’ and ’Quartal’”,”0..*”, MATCHTYPE_FULL_MATCH, FK03, RC03, ForeignKeyRole) | 10 | |
| FR04 | (”Role of foreign key between tables ’Ort_des_Verkaufs’ and ’Filialkategorie’”, 10 ”0..*”,MATCHTYPE_FULL_MATCH, FK04, RC04, ForeignKeyRole) | | |
| FR05 | (”Role of foreign key between tables ’Ort_des_Verkaufs’ and ’Stadt’”,”0..*”, MATCHTYPE_FULL_MATCH, FK05, RC05, ForeignKeyRole) | 10 | |
| FR06 | (”Role of foreign key between tables ’Produktfamilie’ and ’Produktkategorie’”, 10 ”0..*”,MATCHTYPE_FULL_MATCH, FK06, RC06, ForeignKeyRole) | | |
| FR07 | (”Role of foreign key between tables ’Produktgruppe’ and ’Produktfamilie’”, 10 ”0..*”,MATCHTYPE_FULL_MATCH, FK07, RC07, ForeignKeyRole) | | |
| FR08 | (”Role of foreign key between tables ’Quartal’ and ’Jahr’”,”0..*”, MATCHTYPE_FULL_MATCH, FK08, RC08, ForeignKeyRole) | 10 | |
| FR09 | (”Role of foreign key between tables ’Region’ and ’Staat’”,”0..*”, MATCHTYPE_FULL_MATCH, FK09, RC09, ForeignKeyRole) | 10 | |
| FR10 | (”Role of foreign key between tables ’Stadt’ and ’Region’”,”0..*”, MATCHTYPE_FULL_MATCH, FK10, RC10, ForeignKeyRole) | 10 | |
| FR11 | (”Role of foreign key between tables ’Stadt’ and ’Verkaufsbezirk’”,”0..*”, MATCHTYPE_FULL_MATCH, FK11, RC11, ForeignKeyRole) | 10 | |
| FR12 | (”Role of foreign key between tables ’Strassenbereich’ and ’Stadt’”,”0..*”, MATCHTYPE_FULL_MATCH, FK12, RC12, ForeignKeyRole) | 10 | |
| FR13 | (”Role of foreign key between tables ’Tag’ and ’Monat’”,”0..*”, MATCHTYPE_FULL_MATCH, FK13, RC13, ForeignKeyRole) | 10 | |
| FR14 | (”Role of foreign key between tables ’Tag’ and ’Woche’”,”0..*”, MATCHTYPE_FULL_MATCH, FK14, RC14, ForeignKeyRole) | 10 | |
| FR15 | (”Role of foreign key between tables ’Verkaufsbezirk’ and ’Region’”,”0..*”, MATCHTYPE_FULL_MATCH, FK15, RC15, ForeignKeyRole) | 10 | |
| FR16 | (”Role of foreign key between tables ’Einkommen’ and ’Quartal’”,”0..*”, MATCHTYPE_FULL_MATCH, FK16, R106, ForeignKeyRole) | 10 | |
| FR17 | (”Role of foreign key between tables ’Einkommen’ and ’Strassenbereich’”, ”0..*”,MATCHTYPE_FULL_MATCH, FK17, RC17, ForeignKeyRole) | 10 | |
| FR18 | (”Role of foreign key between tables ’Verkauf’ and ’Ort_des_Verkaufs’”, ”0..*”,MATCHTYPE_FULL_MATCH, FK18, RC18, ForeignKeyRole) | 10 | |
| FR19 | (”Role of foreign key between tables ’Verkauf’ and ’Tag’”,”0..*”, MATCHTYPE_FULL_MATCH, FK19, RC19, ForeignKeyRole) | 10 | |
| FR20 | (”Role of foreign key between tables ’Verkaufszahl’ and ’Artikel’”,”0..*”, MATCHTYPE_FULL_MATCH, FK20, RC20, ForeignKeyRole) | 10 | |
| FR21 | (”Role of foreign key between tables ’Verkaufszahl’ and ’Tag’”,”0..*”, MATCHTYPE_FULL_MATCH, FK21, RC21, ForeignKeyRole) | 10 | |
| FR22 | (”Role of foreign key between tables ’Verkaufszahl’ and ’Ort_des_Verkaufs’”, ”0..*”,MATCHTYPE_FULL_MATCH, FK22, RC22, ForeignKeyRole) | 10 | |
| FR23 | (”Role of foreign key between tables ’Verkauftes_Produkt’ and ’Artikel’”, ”0..*”,MATCHTYPE_FULL_MATCH, FK23, RC23, ForeignKeyRole) | 10 | |
| FR24 | ("Role of foreign key between tables ’Verkauftes_Produkt’ and ’Ort_des_Verkaufs’",”0..*”,MATCHTYPE_FULL_MATCH, FK24, RC24, ForeignKeyRole) | 10 | |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | | | |
|---|---|---|---|
| FR25 | (”Role of foreign key between tables ’Verkauftes_Produkt’ and ’Tag’”,”0..*”, MATCHTYPE_FULL_MATCH, FK25, RC25, ForeignKeyRole) | 10 | |
| FR26 | ("Role of foreign key between tables ’MTMOrt_des_VerkaufsArtikel’ and ’Artikel’", ”0..*”,MATCHTYPE_FULL_MATCH, FK26, RC26, ForeignKeyRole) | 14 | |
| FR27 | ("Role of foreign key between tables ’MTMOrt_des_VerkaufsArtikel’ and ’Ort_des_Verkaufs’", ”0..*”,MATCHTYPE_FULL_MATCH, FK27, RC27, ForeignKeyRole) | 14 | |
| **MappingMETA** | | | |
| MM01 | (”SharedRollUp from table ’Woche’ to ’Jahr’”, SHARED_ROLL_UP, {”SUM”}, ”ISO Wochenberechnung”, ALL_TYPES, ALL_TYPES, (CO24), CO10, MappingMETA) | 12 | |
| MM02 | (”DimensionalMapping from table ’Ort des Verkaufs’ to ’Strassenbereich’”, DIMENSIONAL_MAPPING, ALL_TYPES, ”GeographicalMapping”, ALL_TYPES, ALL_TYPES, (CO12), CO21, MappingMETA) | 13 | |
| **ReferentialConstraint** | | | |
| RC01 | (”Referential constraint between tables ’Artikel’ and ’Produktgruppe’”, ROLL_UP, ALL_TYPES, ALL_TYPES, UR08, FR01, NULL, ReferentialConstraint) | 10 | |
| RC02 | ("Referential constraint between tables ’Filialkategorie’ and ’Filialoberkategorie’", ROLL_UP, ALL_TYPES, ALL_TYPES, UR22, FR02, NULL, ReferentialConstraint) | 10 | |
| RC03 | (”Referential constraint between tables ’Monat’ and ’Quartal’”, ROLL_UP, ALL_TYPES, ALL_TYPES, UR10, FR03, NULL, ReferentialConstraint) | 10 | |
| RC04 | ("Referential constraint between tables ’Ort_des_Verkaufs’ and ’Filialkategorie’", ROLL_UP, {”Filiale”}, ALL_TYPES, UR21, FR04, NULL, ReferentialConstraint) | 10 | |
| RC05 | (”Referential constraint between tables ’Ort_des_Verkaufs’ and ’Stadt’”, ROLL_UP, ALL_TYPES, ALL_TYPES, UR13, FR05, NULL, ReferentialConstraint) | 10 | |
| RC06 | ("Referential constraint between tables ’Produktfamilie’ and ’Produktkategorie’", ROLL_UP, ALL_TYPES, ALL_TYPES, UR09, FR06, NULL, ReferentialConstraint) | 10 | |
| RC07 | (”Referential constraint between tables ’Produktgruppe’ and ’Produktfamilie’”, ROLL_UP, ALL_TYPES, ALL_TYPES, UR01, FR07, NULL, ReferentialConstraint) | 10 | |
| RC08 | (”Referential constraint between tables ’Quartal’ and ’Jahr’”, ROLL_UP, ALL_TYPES, ALL_TYPES, UR08, FR08, NULL, ReferentialConstraint) | 10 | |
| RC09 | (”Referential constraint between tables ’Region’ and ’Staat’”, ROLL_UP, ALL_TYPES, ALL_TYPES, UR12, FR09, NULL, ReferentialConstraint) | 10 | |
| RC10 | (”Referential constraint between tables ’Stadt’ and ’Region’”, ROLL_UP, ALL_TYPES, ALL_TYPES, UR11, FR10, NULL, ReferentialConstraint) | 10 | |
| RC11 | (”Referential constraint between tables ’Stadt’ and ’Verkaufsbezirk’”, ROLL_UP, ALL_TYPES, ALL_TYPES, UR17, FR11, NULL, ReferentialConstraint) | 10 | |
| RC12 | (”Referential constraint between tables ’Strassenbereich’ and ’Stadt’”, ROLL_UP, ALL_TYPES, ALL_TYPES, UR13, FR12, NULL, ReferentialConstraint) | 10 | |
| RC13 | (”Referential constraint between tables ’Tag’ and ’Monat’”, ROLL_UP, ALL_TYPES, ALL_TYPES, UR04, FR13, NULL, ReferentialConstraint) | 10 | |
| RC14 | (”Referential constraint between tables ’Tag’ and ’Woche’”, ROLL_UP, ALL_TYPES, ALL_TYPES, UR20, FR14, NULL, ReferentialConstraint) | 10 | |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | | | |
|---|---|---|---|
| RC15 | (”Referential constraint between tables ’Verkaufsbezirk’ and ’Region’”, ROLL_UP, ALL_TYPES, ALL_TYPES, UR11, FR15, NULL, ReferentialConstraint) | 10 | |
| RC16 | (”Referential constraint between tables ’Einkommen’ and ’Quartal’”, DIMENSION, ALL_TYPES, ALL_TYPES, UR10, FR16, {AD16}, ReferentialConstraint) | 10 | 11 |
| RC17 | (”Referential constraint between tables ’Einkommen’ and ’Strassenbereich’”, DIMENSION, ALL_TYPES, ALL_TYPES, UR14, FR17, {AD15}, ReferentialConstraint) | 10 | 11 |
| RC18 | (”Referential constraint between tables ’Verkauf’ and ’Ort_des_Verkaufs’”, DIMENSION, ALL_TYPES, ALL_TYPES, UR05, FR18, {AD01}, ReferentialConstraint) | 10 | 11 |
| RC19 | (”Referential constraint between tables ’Verkauf’ and ’Tag’”, DIMENSION, ALL_TYPES, ALL_TYPES, UR15, FR19, {AD02}, ReferentialConstraint) | 10 | 11 |
| RC20 | (”Referential constraint between tables ’Verkaufszahl’ and ’Artikel’”, DIMENSION, ALL_TYPES, ALL_TYPES, UR01, FR20, {AD03}, ReferentialConstraint) | 10 | 11 |
| RC21 | (”Referential constraint between tables ’Verkaufszahl’ and ’Tag’”, DIMENSION, ALL_TYPES, ALL_TYPES, UR15, FR21, {AD05}, ReferentialConstraint) | 10 | 11 |
| RC22 | (”Referential constraint between tables ’Verkaufszahl’ and ’Ort_des_Verkaufs’”, DIMENSION, ALL_TYPES, ALL_TYPES, UR05, FR22, {AD04},ReferentialConstraint) | 10 | 11 |
| RC23 | (”Referential constraint between tables ’Verkauftes_Produkt’ and ’Artikel’”, DIMENSION, ALL_TYPES, ALL_TYPES, UR01, FR23, {AD06, AD07, AD08}, ReferentialConstraint) | 10 | 11 |
| RC24 | ("Referential constraint between tables ’Verkauftes_Produkt’ and ’Ort_des_Verkaufs’", DIMENSION, ALL_TYPES, ALL_TYPES, UR05, FR24, {AD09, AD10, AD11}, ReferentialConstraint) | 10 | 11 |
| RC25 | (”Referential constraint between tables ’Verkauftes_Produkt’ and ’Tag’”, DIMENSION, ALL_TYPES, ALL_TYPES, UR15, FR25, {AD12, AD13, AD14}, ReferentialConstraint) | 10 | 11 |
| RC26 | ("Referential constraint between tables ’MTMOrt_des_VerkaufsArtikel’ and ’Artikel’", ASSOCIATION, ALL_TYPES, ALL_TYPES, UR01, FR26, NULL, ReferentialConstraint) | 14 | |
| RC27 | ("Referential constraint between tables ’MTMOrt_des_VerkaufsArtikel’ and ’Ort_des_Verkaufs’", ASSOCIATION, ALL_TYPES, ALL_TYPES, UR05, FR27, NULL, ReferentialConstraint) | 14 | |

| **Table** | | | |
|---|---|---|---|
| TA01 | (”Artikel”,DIMENSION,{CO03,CO09,CO31,CO56},{UK01, UK21},{FK01},{},Table) | 2 | 3,4,5, 10 |
| TA02 | (”Jahr”,DIMENSION,{CO10,CO32},{UK02, UK22},{},{},Table) | 2 | 3,5 |
| TA03 | (”Einkommen”,FACT,{CO07,CO58,CO65},{UK03},{FK16, FK17},{},Table) | 2 | 3,4, 10 |
| TA04 | (”Monat”,DIMENSION,{CO11,CO33,CO59},{UK04, UK23},{FK03},{},Table) | 2 | 3,4,5, 10 |
| TA05 | (”Ort_des_Verkaufs”,DIMENSION,{CO12,CO34,CO28,CO29,CO27, CO26,CO63,CO69, CO77},{UK05},{FK04,FK05},{TC01, TC02, TC03, TC04, TC05}, Table) | 2 | 3,4,7, 10 |
| TA06 | (”MTMOrt_des_VerkaufsArtikel”,ASSOCIATION,{CO04,CO35, CO50},{UK06, UK24},{FK26, FK27},{},Table) | 2 | 3,4,5, 14 |
| TA07 | (”Produktfamilie”,DIMENSION,{CO13,CO36,CO57},{UK07, UK25},{FK06},{},Table) | 2 | 3,4,5, 10 |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | | | |
|---|---|---|---|
| TA08 | (”Produktgruppe”,DIMENSION,{CO14,CO37,CO55},{UK08, UK26},{FK07},{},Table) | 2 | 3,4,5, 10 |
| TA09 | (”Produktkategorie”,DIMENSION,{CO15,CO38},{UK09, UK27},{},{},Table) | 2 | 3,4,5 |
| TA10 | (”Quartal”,DIMENSION,{CO16,CO39,CO47},{UK10, UK28},{FK08},{},Table) | 2 | 3,4,5, 10 |
| TA11 | (”Region”,DIMENSION,{CO17,CO40,CO62,CO70},{UK11, UK40},{FK09},{},Table) | 2 | 3,4,5, 10 |
| TA12 | (”Staat”,DIMENSION,{CO19,CO41},{UK12, UK30},{},{},Table) | 2 | 3,4,5 |
| TA13 | (”Stadt”,DIMENSION,{CO20,CO42,CO54,CO60,CO71},{UK13, UK31},{FK10, FK11},{},Table) | 2 | 3,4,5 ,10 |
| TA14 | (”Strassenbereich”,DIMENSION,{CO21,CO43,CO64},{UK14, UK32},{FK12},{},Table) | 2 | 3,4,5, 10 |
| TA15 | (”Tag”,DIMENSION,{CO22,CO44,CO49,CO72},{UK15, UK33},{FK13, FK14},{},Table) | 2 | 3,4,5, 10 |
| TA16 | (”Verkauf”,FACT,{CO08,CO51,CO66},{UK16},{FK18, FK19},{},Table) | 2 | 3,4, 10 |
| TA17 | (”Verkaufsbezirk”,DIMENSION,{CO23,CO45,CO61},{UK17, UK34},{FK15},{},Table) | 2 | 3,4,5, 10 |
| TA18 | (”Verkaufszahl”,FACT,{CO01,CO05,CO52,CO67},{UK18},{FK20, FK21, FK22},{},Table) | 2 | 3,4,10 |
| TA19 | (”Verkauftes_Produkt”,FACT,{CO02,CO06,CO25,CO30, CO53,CO68},{UK19},{FK23, FK24, FK25},{},Table) | 2 | 3,4,10 |
| TA20 | (”Woche”,DIMENSION,{CO24,CO46},{UK20, UK35},{},{},Table) | 2 | 3,4,5 |
| TA21 | (”Filialkategorie”,DIMENSION,{CO73, CO75, CO78},{UK38},{FK02},{},Table) | 2 | 3,4,5, 10 |
| TA22 | (”Filialoberkategorie”,DIMENSION,{CO74, CO76},{UK39},{},{},Table) | 2 | 3,4,5 |

| **TableConstraint** | | | |
|---|---|---|---|
| TC01 | (”Integrity rule for table ’Ort_des_Verkaufs’”, ”Flaeche IS NULL OR Type = ’Kaufhaus’”, TA05, TableConstraint) | 7 | |
| TC02 | (”Integrity rule for table ’Ort_des_Verkaufs’”, ”Gesamtflaeche IS NULL OR Type = ’Kaufhaus’”, TA05, TableConstraint) | 7 | |
| TC03 | (”Integrity rule for table ’Ort_des_Verkaufs’”, ”Filialart IS NULL OR Type = ’Filiale’”, TA05, TableConstraint) | 7 | |
| TC04 | (”Integrity rule for table ’Ort_des_Verkaufs’”, ”Filialleiter IS NULL OR Type = ’Filiale’”, TA05, TableConstraint) | 7 | |
| TC05 | (”Table constraint for table ’Ort_des_Verkaufs’”,"Filialkategorie_FK IS NULL OR Type=’Filiale’", TA05, TableConstraint) | 10 | |

| **UniqueKey** | | | |
|---|---|---|---|
| UK01 | (”Primary key of table ’Artikel’”, TRUE, TA01, (CO31), UR01, UniqueKey) | 4 | |
| UK02 | (”Primary key of table ’Jahr’”, TRUE, TA02, (CO32), UR02, UniqueKey) | 4 | |
| UK03 | (”Primary key of table ’Einkommen’”, TRUE, TA03, (CO58,CO65), UR03, UniqueKey) | 4 | |
| UK04 | (”Primary key of table ’Monat’”, TRUE, TA04, (CO33), UR04, UniqueKey) | 4 | |
| UK05 | (”Primary key of table ’Ort_des_Verkaufs’”, TRUE, TA05, (CO34), UR05, UniqueKey) | 4 | |
| UK06 | (”Primary key of table ’MTMOrt_des_VerkaufsArtikel’”, TRUE, TA06, (CO35), UR06, UniqueKey) | 4 | |
| UK07 | (”Primary key of table ’Produktfamilie’”, TRUE, TA07, (CO36), UR07, UniqueKey) | 4 | |
| UK08 | (”Primary key of table ’Produktgruppe’”, TRUE, TA08, (CO37), UR08, UniqueKey) | 4 | |
| UK09 | (”Primary key of table ’Produktkategorie’”, TRUE, TA09, (CO38), UR09, UniqueKey) | 4 | |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | | | |
|---|---|---|---|
| UK10 | (”Primary key of table ’Quartal’”, TRUE, TA10, (CO39), UR10, UniqueKey) | 4 | |
| UK11 | (”Primary key of table ’Region’”, TRUE, TA11, (CO40), UR11, UniqueKey) | 4 | |
| UK12 | (”Primary key of table ’Staat’”, TRUE, TA12, (CO41), UR12, UniqueKey) | 4 | |
| UK13 | (”Primary key of table ’Stadt’”, TRUE, TA13, (CO42), UR13, UniqueKey) | 4 | |
| UK14 | (”Primary key of table ’Strassenbereich’”, TRUE, TA14, (CO43), UR14, UniqueKey) | 4 | |
| UK15 | (”Primary key of table ’Tag’”, TRUE, TA15, (CO44), UR15, UniqueKey) | 4 | |
| UK16 | (”Primary key of table ’Verkauf’”, TRUE, TA16, (CO51,CO66), UR16, UniqueKey) | 4 | |
| UK17 | (”Primary key of table ’Verkaufsbezirk’”, TRUE, TA17, (CO45), UR17, UniqueKey) | 4 | |
| UK18 | (”Primary key of table ’Verkaufszahl’”, TRUE, TA18, (CO05,CO52,CO67), UR18, UniqueKey) | 4 | |
| UK19 | (”Primary key of table ’Verkauftes_Produkt’”, TRUE, TA19, (CO06,CO53,CO68), UR19, UniqueKey) | 4 | |
| UK20 | (”Primary key of table ’Woche’”, TRUE, TA20, (CO46), UR20, UniqueKey) | 4 | |
| UK21 | (”Conceptual key of table ’Artikel’”, FALSE, TA01, (CO03), NULL, UniqueKey) | 5 | |
| UK22 | (”Conceptual key of table ’Jahr’”, FALSE, TA02, (CO10), NULL, UniqueKey) | 5 | |
| UK23 | (”Conceptual key of table ’Monat’”, FALSE, TA04, (CO11), NULL, UniqueKey) | 5 | |
| UK24 | (”Conceptual key of table ’MTMOrt_des_VerkaufsArtikel’”, FALSE, TA06, (CO04,CO50), NULL, UniqueKey) | 5 | |
| UK25 | (”Conceptual key of table ’Produktfamilie’”, FALSE, TA07, (CO13), NULL, UniqueKey) | 5 | |
| UK26 | (”Conceptual key of table ’Produktgruppe’”, FALSE, TA08, (CO14), NULL, UniqueKey) | 5 | |
| UK27 | (”Conceptual key of table ’Produktkategorie’”, FALSE, TA09, (CO15), NULL, UniqueKey) | 5 | |
| UK28 | (”Conceptual key of table ’Quartal’”, FALSE, TA10, (CO16), NULL, UniqueKey) | 5 | |
| UK29 | (”Conceptual key of table ’Ort_des_Verkaufs’”, FALSE, TA05, (CO12), NULL, UniqueKey) | 5 | |
| UK30 | (”Conceptual key of table ’Staat’”, FALSE, TA12, (CO19), NULL, UniqueKey) | 5 | |
| UK31 | (”Conceptual key of table ’Stadt’”, FALSE, TA13, (CO54), NULL, UniqueKey) | 5 | |
| UK32 | (”Conceptual key of table ’Strassenbereich’”, FALSE, TA14, (CO21), NULL, UniqueKey) | 5 | |
| UK33 | (”Conceptual key of table ’Tag’”, FALSE, TA15, (CO22), NULL, UniqueKey) | 5 | |
| UK34 | (”Conceptual key of table ’Verkaufsbezirk’”, FALSE, TA17, (CO23), NULL, UniqueKey) | 5 | |
| UK35 | (”Conceptual key of table ’Woche’”, FALSE, TA20, (CO24), NULL, UniqueKey) | 5 | |
| UK36 | (”Primary key of table ’Filialkategorie’”, TRUE, TA21, (CO75), UR21, UniqueKey) | 4 | |
| UK37 | (”Primary key of table ’Filialoberkategorie’”, TRUE, TA22, (CO76), UR22, UniqueKey) | 4 | |
| UK38 | (”Conceptual key of table ’Filialkategorie’”, FALSE, TA21, (CO73), NULL, UniqueKey) | 5 | |
| UK39 | (”Conceptual key of table ’Filialoberkategorie’”, FALSE, TA22, (CO74), NULL, UniqueKey) | 5 | |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | | | |
|---|---|---|---|
| UK40 | (”Conceptual key of table ’Region’”, FALSE, TA11, (CO17), NULL, UniqueKey) | 5 | |
| **UniqueKeyRole** | | | |
| UR01 | (”Role of primary key of table ’Artikel’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK01, {RC07, RC20, RC23, RC26}, UniqueKeyRole) | 4 | 10,14 |
| UR02 | (”Role of primary key of table ’Jahr’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK02, {}, UniqueKeyRole) | 4 | |
| UR03 | (”Role of primary key of table ’Einkommen’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK03, {}, UniqueKeyRole) | 4 | |
| UR04 | (”Role of primary key of table ’Monat’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK04, {RC13}, UniqueKeyRole) | 4 | 10 |
| UR05 | (”Role of primary key of table ’Ort_des_Verkaufs’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK05, {RC18, RC22, RC24, RC27}, UniqueKeyRole) | 4 | 10,14 |
| UR06 | (”Role of primary key of table ’MTMOrt_des_VerkaufsArtikel’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK06, {}, UniqueKeyRole) | 4 | |
| UR07 | (”Role of primary key of table ’Produktfamilie’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK07, {}, UniqueKeyRole) | 4 | |
| UR08 | (”Role of primary key of table ’Produktgruppe’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK08, {RC01,RC08}, UniqueKeyRole) | 4 | 10 |
| UR09 | (”Role of primary key of table ’Produktkategorie’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK09, {RC06}, UniqueKeyRole) | 4 | 10 |
| UR10 | (”Role of primary key of table ’Quartal’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK10, {RC03, RC16}, UniqueKeyRole) | 4 | 10 |
| UR11 | (”Role of primary key of table ’Region’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK11, {RC10,RC15}, UniqueKeyRole) | 4 | 10 |
| UR12 | (”Role of primary key of table ’Staat’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK12, {RC09}, UniqueKeyRole) | 4 | 10 |
| UR13 | (”Role of primary key of table ’Stadt’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK13, {RC05,RC12}, UniqueKeyRole) | 4 | 10 |
| UR14 | (”Role of primary key of table ’Strassenbereich’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK14, {RC17}, UniqueKeyRole) | 4 | 10 |
| UR15 | (”Role of primary key of table ’Tag’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK15, {RC19, RC21, RC25}, UniqueKeyRole) | 4 | 10 |
| UR16 | (”Role of primary key of table ’Verkauf’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK16, {}, UniqueKeyRole) | 4 | |
| UR17 | (”Role of primary key of table ’Verkaufsbezirk’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK17, {RC11}, UniqueKeyRole) | 4 | 10 |

Fortsetzung auf der folgenden Seite

Fortsetzung von der letzten Seite

| | | | |
|---|---|---|---|
| UR18 | (”Role of primary key of table ’Verkaufszahl’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK18, {}, UniqueKeyRole) | 4 | |
| UR19 | (”Role of primary key of table ’Verkauftes_Produkt’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK19, {}, UniqueKeyRole) | 4 | |
| UR20 | (”Role of primary key of table ’Woche’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK20, {RC14}, UniqueKeyRole) | 4 | 10 |
| UR21 | (”Role of primary key of table ’Filialkategorie’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK36, {RC04}, UniqueKeyRole) | 4 | 10 |
| UR22 | (”Role of primary key of table ’Filialoberkategorie’”, 1, FULL, REFERENTIALRULE_CASCADE, REFERENTIALRULE_CASCADE, TRUE, TRUE, UK37, {RC02}, UniqueKeyRole) | 4 | 10 |

Tabelle A.3: *LCD of SQL*–Objekte im Beispiel

# Anhang B

# Evaluation

In diesem Anhang ist als vertiefende Information zu der in Kapitel 12 beschriebenen Evaluation das dokumentierte konzeptionelle Datenschema dargestellt.

## B.1 Faktklassen

Abbildung B.1 zeigt die Faktklassen des Schemas und ihre Dimensionen. Wichtigste Faktklasse ist „Fall“, die einen Inzidenz– bzw. Sterbefall wiedergibt. Zu einem solchen Fall können mehrere Tätigkeiten, Therapien und Familienanamnesen gehören, die ebenfalls jeweils eine Faktklasse bilden. Für die Auswertungen sind als Vergleichsmaßstab Populationen wichtig, insbesondere „Standardpopulationen“ und „Vergleichspopulationen“, die jeweils als eigene Faktklassen mit einer gemeinsamen abstrakten Oberklasse dargestellt sind.

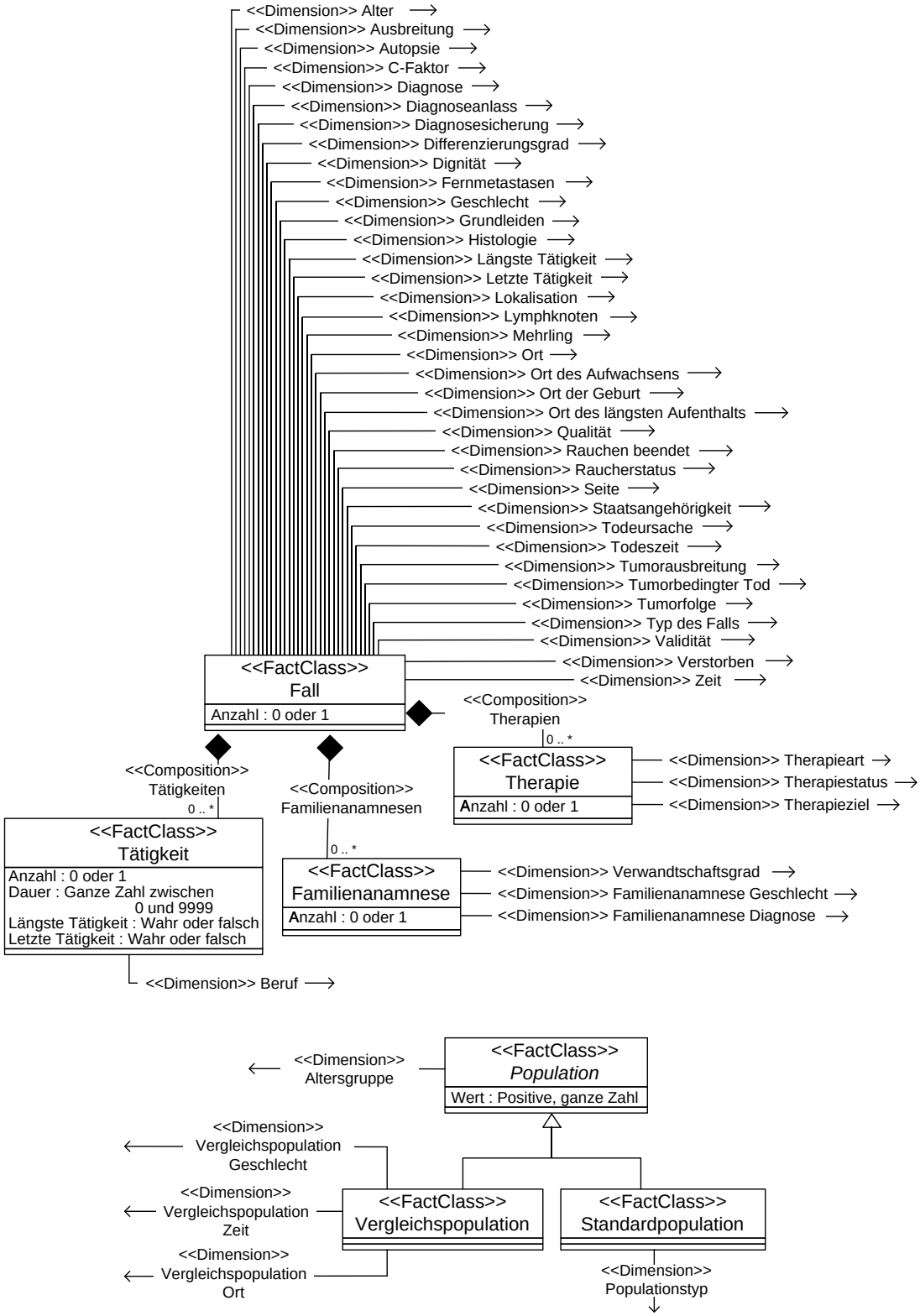



Abbildung B.1: Konzeptionelle Modellierung: Faktklassen

## B.2 DataClass Aufzählungstyp

Die Datenklasse „Aufzählungstyp“ stellt einen diskreten Wertebereich zur Verfügung, wobei jeder mögliche Wert aus einem ein Zeichen langen *Kürzel* (Datentyp „AbkürzungsTyp“) und einem dieses Kürzel beschreibenden Langtext besteht. Verwendung findet diese in Abbildung B.2 dargestellte Datenklasse in diversen Dimensionen in den Unterabschnitten B.4 bis B.38.

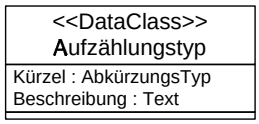



Abbildung B.2: Konzeptionelle Modellierung: Datenklasse Aufzählungstyp

## B.3 Dimension Alter

Die Dimension „Alter“ besitzt als Ebene der feinsten Granularität eine Altersangabe, diese wird für verschiedene Auswertungszwecke zu den Hierarchieebenen „Altersgruppen für standardisierte Mortalität“ (zur Berechnung der standardisierten Mortalität, Werte sind „35-64“ sowie „65+“), „Kumulative Altersgruppen“ (zur Berechnung der kumulativen Inzidenzrate, Werte sind hier „0–64“ und „0–74“) und zu „Fünfjahresaltersgruppen“ (Werte sind hier „0–4“, „5–9“, ..., „80–84“ sowie „85+“) verdichtet.

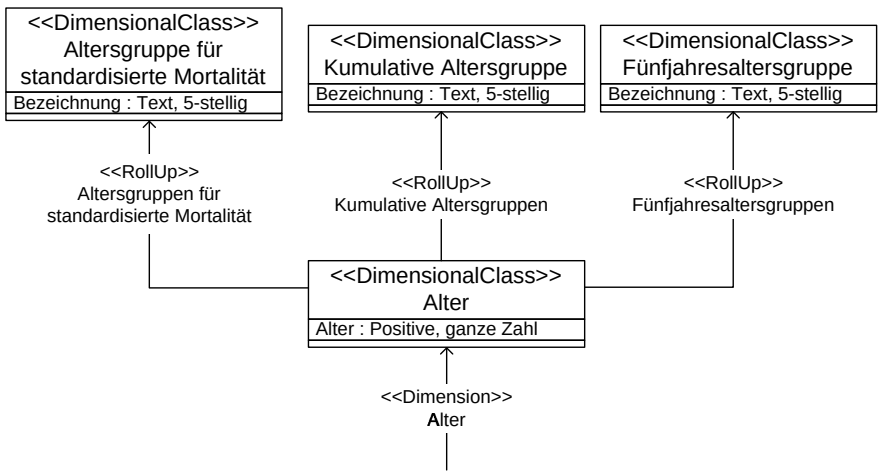



Abbildung B.3: Konzeptionelle Modellierung: Dimension Alter

## B.4 Dimension Ausbreitung

Die Dimension „Ausbreitung“ gibt die externe Ausbreitung des Tumors an. Zulässige Wertepaare sind „0, In Situ“, „1, Lokal begrenzt“, „2, Regionäre Lymphknoten/Nachbarschaft“, „3, Fernmetastasen“, „4, Systemerkrankung“ sowie „9, Fehlende Angabe/Unbekannt“.

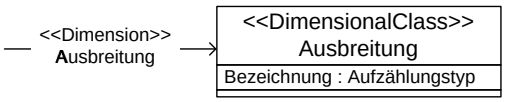



Abbildung B.4: Konzeptionelle Modellierung: Dimension Ausbreitung

## B.5 Dimension Autopsie

Die Dimension „Autopsie“ gibt Auskunft über eine durchgeführte Autopsie. Zulässige Wertepaare sind „1, Autopsie durchgeführt“, „2, Autopsie nicht durchgeführt“ und „9, Fehlende Angabe/Sonstige“.

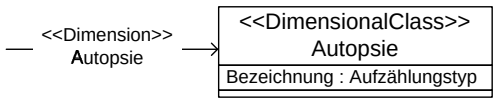



Abbildung B.5: Konzeptionelle Modellierung: Dimension Autopsie

## B.6 Dimension Beruf

Die Dimension „Beruf“ beschreibt den Beruf des Patienten in Form des vierstelligen Berufscodes nach der Klassifikation der Bundesanstalt für Arbeit; „9911“ steht für „Unbekannt“. Berufe können zu Berufsgruppen verdichtet werden.
Die Dimension wird von der Faktklasse „Fall“ zur Angabe doppelt genutzt — zur Angabe der letzten sowie der am längsten ausgeübten Tätigkeit. Ebenso wird die Dimension von der Faktklasse „Tätigkeit“ zur Angabe des Berufes genutzt.

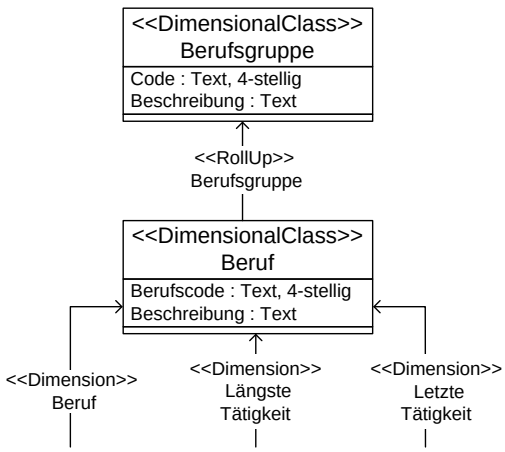



Abbildung B.6: Konzeptionelle Modellierung: Dimension Beruf

## B.7 Dimension C–Faktor

Die Dimension „C–Faktor“ gibt Auskunft über den die Tumorausbreitung beschreibenden C–Faktor. Zulässige Wertepaare sind „1, Klinisch“, „2, Spezielle Diagnostik, evtl. auch Biopsie/Zytologie“, „3, Chirurgische Exploration mit Biopsie/Zytologie“, „4, Tumorresektion mit pathologischer Untersuchung“, „5, Autoptisch“ und „9, Fehlende Angabe/Unbekannt“.

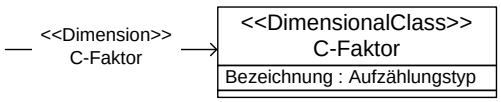



Abbildung B.7: Konzeptionelle Modellierung: Dimension C–Faktor

## B.8 Dimension Diagnose

Die Dimension „Diagnose“ beschreibt die Diagnose in Form eines ICD–Codes sowie des Langtextes (Attribut „Bezeichnung“) und der Angabe des Typs der Diagnoseklassifikation.
Für Überblicksauswertungen findet eine Zusammenfassung nach laufenden Zehnergruppen gemäß ICD–Code statt (*DimensionalClass*–Objekt „Diagnosezehnergruppe“).

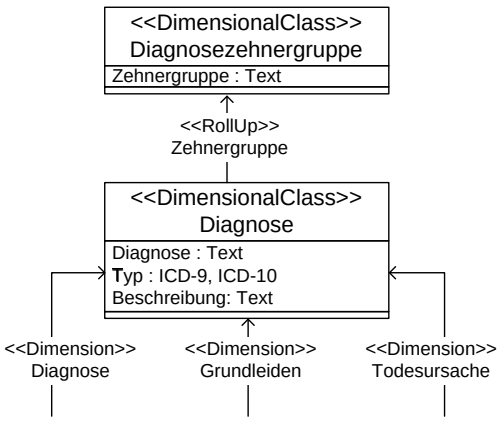



Abbildung B.8: Konzeptionelle Modellierung: Dimension Diagnose

## B.9 Dimension Diagnoseanlass

Die Dimension „Diagnoseanlass“ beschreibt die Ursache der der Diagnose zugrundeliegenden Untersuchung. Zulässige Wertepaare sind „1, Beschwerden“, „2, Früherkennung“, „3, Arbeitsmedizinische Untersuchung“, „4, Nachsorge–Untersuchung“, „5, Zufallsbefund“, „6, Zufallsbefund bei Autopsie“, „7, Sonstiges“ und „9, Fehlende Angabe/Unbekannt“.

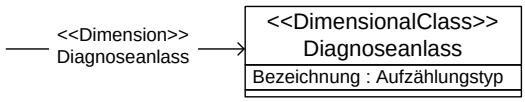



Abbildung B.9: Konzeptionelle Modellierung: Dimension Diagnoseanlass

## B.10 Dimension Diagnosesicherung

Die Dimension „Diagnosesicherung“ beschreibt die Art des Diagnosenachweises. Zulässige Wertepaare sind „1, Klinisch“, „2, Spezielle Diagnostik“, „3, Zytologisch“, „4, Histologisch“, „5, Autoptisch“, „6, Sonstiges“ und „9, Fehlende Angabe/Unbekannt“.

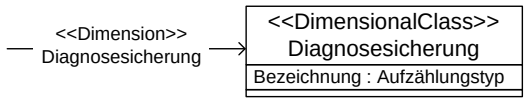



Abbildung B.10: Konzeptionelle Modellierung: Dimension Diagnosesicherung

## B.11 Dimension Differenzierungsgrad

Die Dimension „Differenzierungsgrad“ beschreibt den Differenzierungsgrad, der die Detaillierungsebene der Unterscheidbarkeit angibt.
Zulässige Werte sind „1, Gut differenziert (G1)“, „2, Mäßig differenziert (G2)“, „3, Schlecht differenziert (G3)“, „4, Undifferenziert (G4)“, „5, T–Zell–Lymphom“, „6, B–Zell–Lymphom“, „7, Null–Zell–Lymphom“, „8, NK–Zell–Lymphom“, „9, Fehlende Angabe/Nicht bestimmbar/Nicht zutreffend“, „16, Low Grade (G1 oder G2)“, „17, Medium Grade (G2 oder G3)“, „18, High Grade (G3 oder G4)“ und „19, Grenzfall bzw. Borderline – nur bei Ovar“.

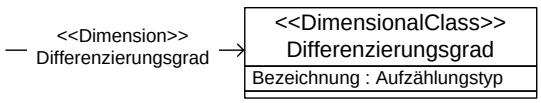



Abbildung B.11: Konzeptionelle Modellierung: Dimension Differenzierungsgrad

## B.12 Dimension Dignität

Die Dimension „Dignität“ beschreibt den Malignitätsgrad (Grad der Bösartgkeit) aus dem ICD–0–Morphologie–Schlüssel. Zulässige Wertepaare sind „0, Gutartig“, „1, Unbestimmter Charakter / Unsicher, ob bös– oder gutartig“, „2, In situ“, „3, Bösartig/Primärsitz“, „6, Bösartig/Metastase“ und „9, Fehlende Angabe/Unbekannt“.

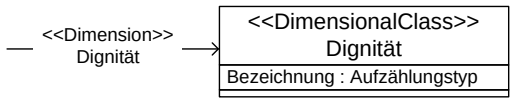



Abbildung B.12: Konzeptionelle Modellierung: Dimension Dignität

## B.13 Dimension Fernmetastasen

Die Dimension „Fernmetastasen“ beschreibt das Vorliegen von Fernmetastasen, die sog. M–Angabe. Zulässige Werte sind „0“, „1“ und „Unbekannt“.

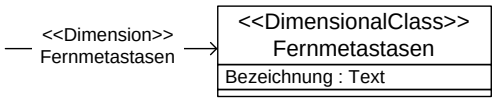



Abbildung B.13: Konzeptionelle Modellierung: Dimension Fernmetastasen

## B.14 Dimension Geschlecht

Die Dimension „Geschlecht“ beschreibt das Geschlecht des Patienten, auf den sich die Meldung bezieht. Zulässige Wertepaare sind „1, Männlich“, „2, Weiblich“ und „9, Fehlende Angabe/Sonstige“.

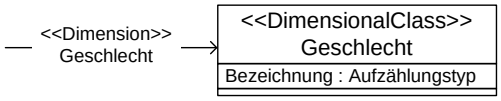



Abbildung B.14: Konzeptionelle Modellierung: Dimension Geschlecht

## B.15 Dimension Histologie

Die Dimension „Histologie“ beschreibt die Histologie nach ICD–0. Diese besteht aus einem vierstelligen Zifferncode und dem zugehörigen Langtext.

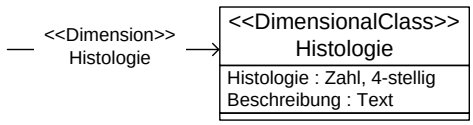



Abbildung B.15: Konzeptionelle Modellierung: Dimension Histologie

## B.16 Dimension Lokalisation

Die Dimension „Lokalisation“ beschreibt die Lokalisation nach dem Tumorlokalisationsschlüssel zur ICD–0 mit mindestens einer Nachkommastelle. Somit ergibt sich ein maximal sechsstelliger Code und der zugehörige Langtext.

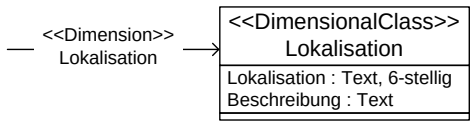



Abbildung B.16: Konzeptionelle Modellierung: Dimension Lokalisation

## B.17 Dimension Lymphknoten

Die Dimension „Lymphknoten“ beschreibt den Lymphknotenbefund, die sog. N–Angabe. Zulässige Werte sind „1“, „2“, „3“, und „Unbekannt“.

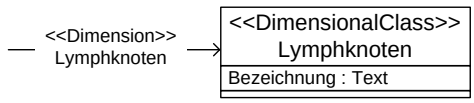



Abbildung B.17: Konzeptionelle Modellierung: Dimension Lymphknoten

## B.18 Dimension Mehrling

Die Dimension „Mehrling“ beschreibt, ob der Patient Einzelkind oder Mehrling ist. Zulässige Wertepaare sind „0, Kein Mehrling“, „1, Eineiiger Mehrling“, „2, Zweieiiger Mehrling“, „3, Mehrling, unbekannt, ob ein– oder zweieiig“ und „9, Fehlende Angabe/Unbekannt“.

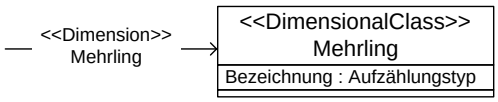



Abbildung B.18: Konzeptionelle Modellierung: Dimension Mehrling

## B.19 Dimension Ort

Die Dimension „Ort“ beschreibt die geographischen Verhältnisse. Die Ortsdimension wird von der Faktklasse „Fall“ vierfach genutzt — zur Bestimmung des Wohnortes, des Geburtsortes und des Ortes des Aufwachsens. Kleinste Einheit ist dabei ein „Gebiet“, i. d. R. Gemeinden, es kann sich aber auch um „Teilgemeinden“ etc. handeln, dies wird über das Attribut „Typ“ beschrieben.
Weiterhin wird ein Gebiet durch die Gemeindekennziffer und den Langtext bestimmt ist. Darauf bauen hierarchisch Landkreise, Regierungsbezirke und Bundesländer auf, wobei jede Ebene durch das hierarchische Kennziffernsystem und einen Bezeichner beschrieben wird. Weiterhin besitzt die Hierarchieebene „Lankreis“ ein Attribut Urbanisierungsgrad, das einen der Werte „Städtisch“, „Ländlich“ und „Gemischt“ annehmen kann.

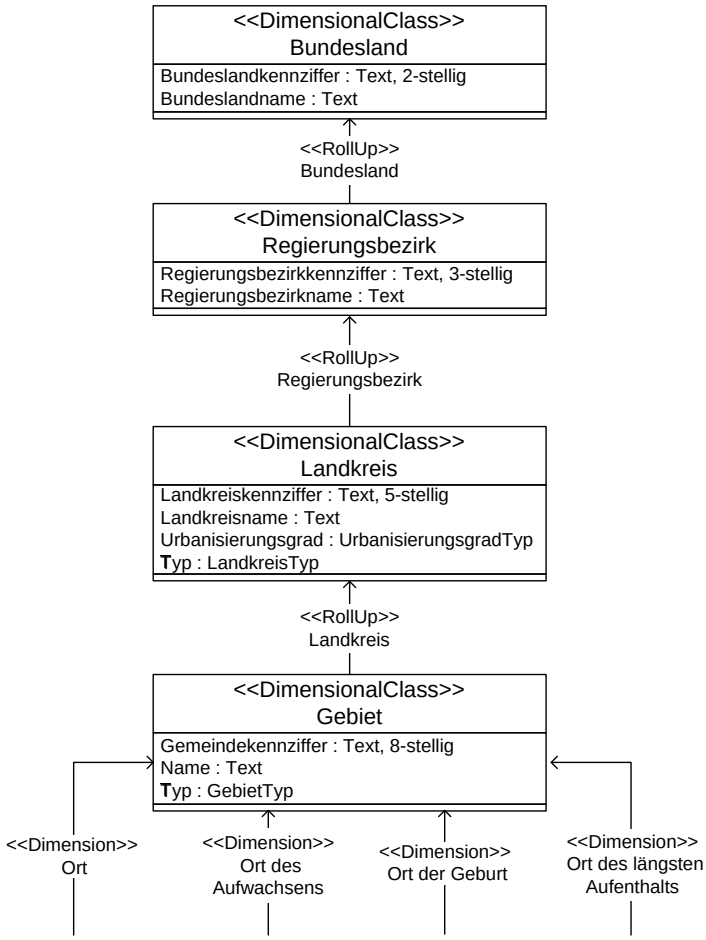



Abbildung B.19: Konzeptionelle Modellierung: Dimension Ort

## B.20 Dimension Populationstyp

Die Dimension „Populationstyp“ gibt den Typ einer Population in Form eines Bezeichners an.

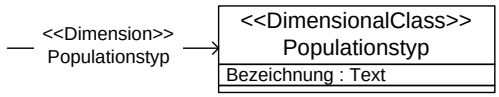



Abbildung B.20: Konzeptionelle Modellierung: Dimension Typ

## B.21 Dimension Qualität

Die Dimension „Qualität“ beschreibt, ob es sich um einen „DCO–Fall“ oder „DCN–Fall“ handelt. Diese Attribute werden benötigt, um Aussagen über die Vollständigkeit der erfassten Daten treffen zu können und somit Rückschlüsse auf die Güte der Datenanalysen geben können.
Zulässige Werte sind „1, DCO–Fall“, „2, DCN–Fall“ und „3, Sonstiges“.

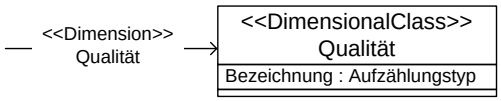



Abbildung B.21: Konzeptionelle Modellierung: Dimension Qualität

## B.22 Dimension Rauchen Beendet

Die Dimension „Rauchen Beendet“ beschreibt, wann der Patient das Rauchen beendet hat. Zulässiger Wert ist neben einer Jahresangabe auch der Wert „0“ für unbekannt, Raucher und Nichtraucher.

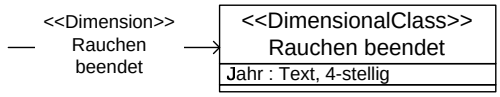



Abbildung B.22: Konzeptionelle Modellierung: Dimension Rauchen Beendet

## B.23 Dimension Raucherstatus

Die Dimension „Raucherstatus“ gibt Auskunft über das Rauchverhalten des Patienten. Zulässige Werte sind „1, Nichtraucher“, „2, Exraucher“, „3, Raucher“ und „9, Fehlende Angabe/Unbekannt“.

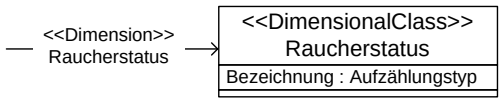



Abbildung B.23: Konzeptionelle Modellierung: Dimension Raucherstatus

## B.24 Dimension Seite

Die Dimension „Seite“ gibt die Körperseite an, an der die Erkrankung aufgetreten ist. Zulässige Werte sind „1, Rechts“, „2, Links“, „3, Beidseits“ und „8, Trifft nicht zu“ sowie „9, Fehlende Angabe/Unbekannt“.

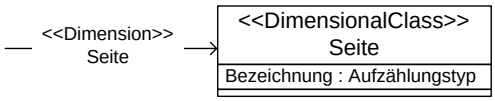



Abbildung B.24: Konzeptionelle Modellierung: Dimension Seite

## B.25 Dimension Staatsangehörigkeit

Die Dimension „Staatsangehörigkeit“ gibt die aktuelle Staatsangehörigkeit des Patienten an. Diese besteht aus dem Ländercode und dem Land im Klartext, gemäß der Kodierung nach der Klassifikation des Statistischen Bundesamtes. „999“ ist der Wert für „Unbekannt“.

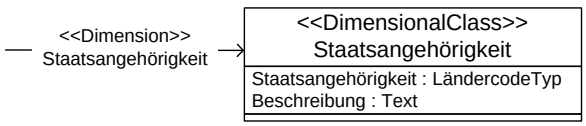



Abbildung B.25: Konzeptionelle Modellierung: Dimension Staatsangehörigkeit

## B.26 Dimension Therapieart

Die Dimension „Therapieart“ beschreibt den Typ der Therapie. Zulässige Wertepaare sind „1, Operation“, „2, Radiatio“, „3, Chemotherapie“, „4, Hormontherapie“, „5, Immuntherapie“, „6, Knochenmarktransplantation“, „7, Sonstige“ und „9, Fehlende Angabe/Unbekannt“.

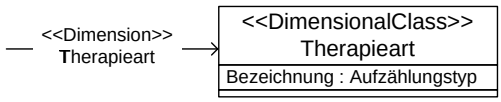



Abbildung B.26: Konzeptionelle Modellierung: Dimension Therapieart

## B.27 Dimension Therapiestatus

Die Dimension „Therapiestatus“ gibt Auskunft über den aktuellen Zustand der Therapie. Zulässige Wertepaare sind „1, Durchgeführt“, „2, Nicht durchgeführt“, „3, Vorgesehen“ und „4, Verweigert“.

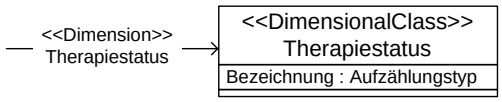



Abbildung B.27: Konzeptionelle Modellierung: Dimension Therapiestatus

## B.28 Dimension Therapieziel

Die Dimension „Therapieziel“ beschreibt den mit der Therapie verfolgten Zweck. Zulässige Wertepaare sind „1, Kurativ“, „2, Palliativ“, „3, Adjuvant“, „4, Supportiv“, „5, Neoadjuvant“, „6, Explorativ“, „7, Sonstige“ und „9, Fehlende Angabe/Unbekannt“.

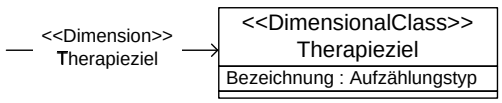



Abbildung B.28: Konzeptionelle Modellierung: Dimension Therapieziel

## B.29 Dimension Todeszeit

Die Dimension „Todeszeit“ macht eine Angabe zur Zeit des Todes. Sie wird dabei in der Form Monat/Jahr angegeben, und benutzt die gleichen Klassen und Verdichtungspfade wie die Dimension „Zeit“ (siehe Abbildung im Abschnitt B.39).

## B.30 Dimension Tumorausbreitung

Die Dimension „Tumorausbreitung“ beschreibt die Ausbreitung des Tumors, die sog. T–Angabe. Zulässige Werte sind Zulässige Werte sind „is“, „a“, „0“, „1“, „2“, „3“, „4“ und „X“.

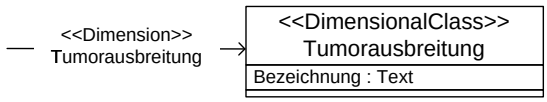



Abbildung B.29: Konzeptionelle Modellierung: Dimension Tumorausbreitung

## B.31 Dimension Tumorbedingter Tod

Die Dimension „Tumorbedingter Tod“ beschreibt die mögliche Todesfolge aufgrund des Tumors. Zulässige Wertepaare sind „1, Tod durch diesen Tumor bedingt“, „2, Tod nicht durch diesen Tumor bedingt“, „3, Fraglich, ob Tod durch diesen Tumor bedingt“ und „9, Fehlende Angabe/Unbekannt/Nicht verstorben“.

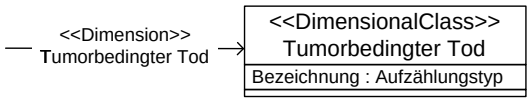



Abbildung B.30: Konzeptionelle Modellierung: Dimension Tumorbedingter Tod

## B.32 Dimension Tumorfolge

Die Dimension „Tumorfolge“ beschreibt, ob es sich um einen Erst– oder Folgetumor handelt. Zulässige Werte sind „1, Erster Tumor“, „2, Zweiter Tumor“ und „3, Weiterer Tumor“.

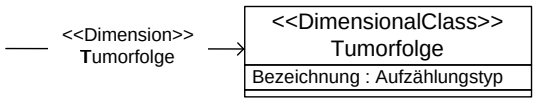



Abbildung B.31: Konzeptionelle Modellierung: Dimension Tumorfolge

## B.33 Dimension Typ des Falles

Die Dimension „Typ des Falles“ beschreibt, ob es sich um einen Inzidenz– oder einen Sterbefall handelt.

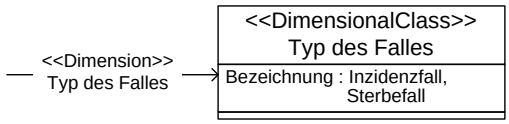



Abbildung B.32: Konzeptionelle Modellierung: Dimension Typ des Falles

## B.34 Dimension Validität

Die Dimension „Validität“ macht eine Angabe zur Zuverlässigkeit des Falles. Zulässige Wertepaare sind „1, Valide/Unauffällig/Noch unbearbeitet“, „2, Auffällig, aber in Nachfrage bestätigt“, „3, Ungewöhnlich und (trotz) Nachfrage ungeklärt“ und „4, Invalide/Unmöglich und ungeklärt“.

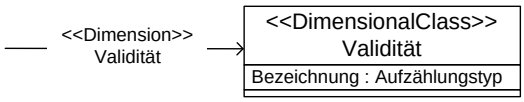



Abbildung B.33: Konzeptionelle Modellierung: Dimension Validität

## B.35 Dimension Vergleichspopulation Ort

Die Dimension „Vergleichspopulation Ort“ macht eine Ortsangabe, auf die sich die entsprechende Population bezieht. Da die Bezugsgrössen nicht unbedingt mit denen der Ortshierarchie in Abschnitt B.19 übereinstimmen müssen, können keine dimensionalen Klassen gemeinsam genutzt werden. Die Verbindung für spätere Auswertungen wird in Abschnitt B.40 durch Verwenden eines *Dimensional-Mapping*–Objektes realisiert.

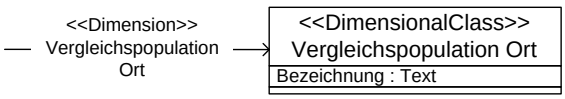



Abbildung B.34: Konzeptionelle Modellierung: Dimension Vergleichspopulation Ort

## B.36 Dimension Vergleichspopulation Zeit

Die Dimension „Vergleichspopulation Zeit“ macht eine Zeitangabe, auf die sich die entsprechende Population bezieht. Die Bezugsgrösse ist meistens ein Jahr. Da dies aber prinzipiell auch anders sein kann, wird eine eigene dimensionale Klasse verwendet und nicht auf die Zeit–Hierarchie aus Abschnitt B.39 zurückgegriffen.
Auch eine Verbindung in Form eines *DimensionalMapping*–Objektes ist nicht notwendig, da keine verbindenden Auswertungen vorgenommen werden. Vielmehr geht der Zeitaspekt der Vergleichspopulation als Parameter in die Auswertung ein.

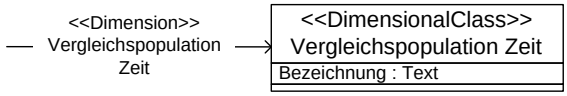



Abbildung B.35: Konzeptionelle Modellierung: Dimension Vergleichspopulation Zeit

## B.37 Dimension Verstorben

Die Dimension „Verstorben“ macht eine Angabe zum tumorbedingten Tod. Zulässige Wertepaare sind „1, Patient verstorben“, „2, Patient nicht verstorben“ und „3, Fehlende Angabe/Unbekannt“.

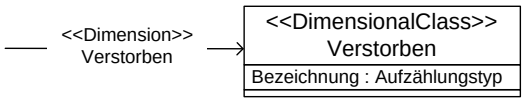



Abbildung B.36: Konzeptionelle Modellierung: Dimension Verstorben

## B.38 Dimension Verwandtschaftsgrad

Die Dimension „Verwandtschaftsgrad“ beschreibt der Grad der Verwandtschaft von Familienangehörigen zum Festhalten von Familienanamnesen. Zulässige Werte sind „0, Enkel“, „1, Kinder“, „2, Geschwister“, „3, Eltern“, „4, Neffen“, „5, Großeltern“, „6, Onkel/Tanten“, „7, Cousins/Cousinen“, „8, Sonstige“, „9, Fehlende Angabe/Unbekannt“.

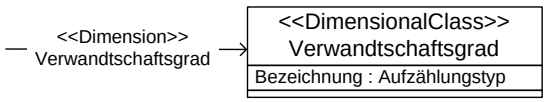



Abbildung B.37: Konzeptionelle Modellierung: Dimension Verwandtschaft

## B.39 Dimension Zeit

Die Dimension „Zeit“ macht eine Angabe zum Zeitpunkt des Falles. Sie wird dabei in der Form Monat/Jahr angegeben, es existieren Verdichtungspfade zu den auswertungsrelevanten Hierarchiebenen „Jahr“ und „Dreijahresgruppe“.

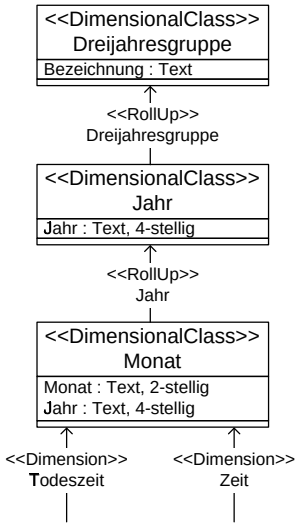



Abbildung B.38: Konzeptionelle Modellierung: Dimension Zeit

## B.40 DimensionalMapping

Um den Zusammenhang zwischen der Ortsdimension von Vergleichspopulationen aus Abschnitt B.35 und der Ortshierarchie aus Abschnitt B.19 herzustellen, wird zwischen den Ortsobjekten der Vergleichspopulationen und allen Hierarchieebenen der Ortshierarchie jeweils eine dimensionale Abbildung definiert. Das Resultat ist in Abbildung B.39 zu sehen. Als Eingabeparameter dient jeweils das Attribut „Bezeichnung“, Resultat ist jeweils die identifizierende Kennziffer.

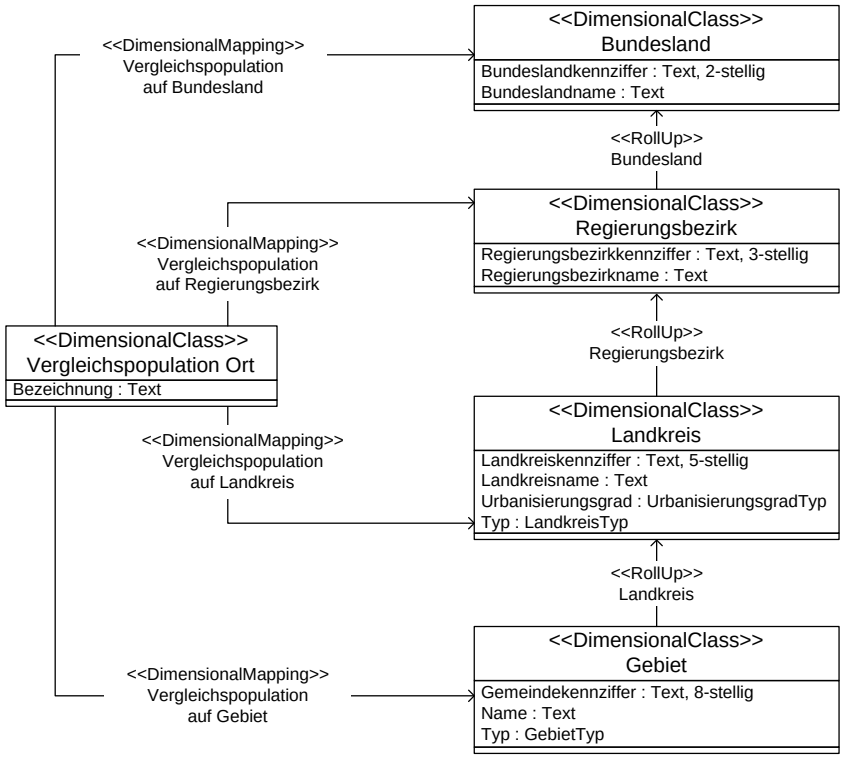



Abbildung B.39: Konzeptionelle Modellierung: Zwischendimensionale der Ortshierarchien

# Glossar

Dieses Glossar führt alle im Rahmen der Arbeit verwendeten wichtigen Termini auf. Diese entstammen vor allem den Bereichen Data Warehousing und multidimensionale Datenmodelle, ebenso wurden elementare Begriffe aus den Gebieten Datenbanken und Objektorientierung in das Glossar aufgenommen, ergänzt um im Zuge der Arbeit definierter Ausdrücke.

**Abfragewerkzeuge:** Werkzeugklasse aus dem ↑Front End–Bereich eines ↑Data Warehouse–Systems. ∼ verwenden zwischen Datenbank und Benutzer eine Zwischenschicht, die es durch „Point and Click"–Bedienung ermöglicht, Anfragen zu formulieren und so dem Endbenutzer das Formulieren komplexer SQL–Anfragen abnimmt.

**Abgeleitetes Attribut:** Wird aus den Werten anderer ↑Attribute berechnet. ∼e können nicht direkt geändert werden und werden durch eine Berechnungsoperation implementiert oder gesetzt.

**Abstrakte Klasse:** ↑Klasse, die gemeinsame Merkmale ihrer Unterklassen festlegt, wobei von der Klasse selbst keine ↑Instanzen existieren dürfen.

**Abstraktionsebene:** Synonym für ↑Beschreibungsebene.

**Additivität:** Eigenschaft einer ↑Kennzahl, die angibt, bez. welcher ↑Dimension welche ↑Verdichtungsoperatoren angewendet werden dürfen.

**Ad–Hoc–Anfrage:** Während der Nutzung „spontan" formulierte und interaktiv gestellte Anfrage an ein ↑Datenbanksystem.

**Änderungsanomalie:** Bei Änderungsoperationen in einer ↑Datenbank auftretende Inkonsistenz aufgrund von redundanter Datenhaltung.

**Aggregation:**
1. Synonym für ↑Verdichtung.
2. Spezielle Form der ↑Assoziation, die eine *Ganzes–Teil*–Beziehung zwischen zwei ↑Klassen beschreibt. Gegenüber einer ↑Komposition ist die als *Teil* betrachtete Klasse jedoch nicht von der als *Ganzes* aufgefassten Klasse abhängig.

**Aggregationsebene:** Synonym für ↑Hierarchieebene.

**Aggregationsfunktion:** Synonym für ↑Verdichtungsoperator.

**Aggregation!–soperator:** Synonym für ↑Verdichtungsoperator.

**Aggregierbarkeit:** Synonym für ↑Additivität.

**Aggregierte Daten:** Mittels ↑Aggregation ermittelte Daten.

**Alternativer Verdichtungspfad:** ↑Multiple Hierarchie, die wieder zusammengeführt wird.

**Analyse:** Untersuchung und Präsentation von Daten mit Hilfe unterschiedlicher, z. B. statistischer Methoden.

**Analysebereich:** Synonym für ↑Front End–Bereich.

**Analysewerkzeug:** Softwaresystem, das die ↑Analyse unterstützt.

**Annotation:** Synonym für ↑Notiz.

**Annotiertes Schema:** Um statistische Informationen über die ↑Extension wie Volumen und Zu– oder Abhnahmeraten angereichertes ↑Schema.

**Anteilige Verrechnung:** Bei einer ↑Verdichtung kann ein Element einer ↑Hierarchieebene zu mehreren Elementen der nächst höheren Ebene mittels einer Berechnungsvorschrift zugeordnet werden.

**Archiv–Datenbank:** Datenbank in einem ↑Data Warehouse–System, in der für die aktuelle Datenanalyse nicht mehr relevante Daten abgelegt werden.

**Assoziation:** Semantische Beziehung zwischen zwei oder mehreren ↑Klassen, ↑Objekten, Typen oder anderen Elementen eines Schemas.

**Assoziationsrolle:** Rolle, die ein Typ oder eine ↑Klasse in einer ↑Assoziation spielt, d. h. eine Rolle repräsentiert eine Klasse in einer Assoziation. Gewöhnlich befinden sich Assoziationen zwischen zwei Klassen. Eine Klasse kann aber auch eine Assoziation zu sich selbst haben; in diesem Fall sind die beiden Enden der Assoziation nur durch die Rollenangabe zu unterscheiden.

**Attribut:**
1. Beschreibung einer bestimmten Eigenschaft der ↑Objekte einer ↑Klasse
2. Beschreibung einer bestimmten Eigenschaft der ↑Entitäten eines ↑Entitätstyps.

**Back End–Bereich:** Teil des ↑Data Warehouse–Systems. Der ∼ umfasst die zwischen den ↑Datenquellen und dem ↑Data Warehouse angesiedelten Komponenten.

**Basisklasse:** Synonym für ↑Oberklasse.

**Berichtswerkzeug:** Werkzeugtyp aus dem ↑Front End–Bereich eines ↑Data Warehouse–Systems. ∼e erzeugen mittels vordefinierter, eventuell parametrisierter Abfragen Auswertungen der Daten, reichern diese eventuell um einfache arithmetische Operationen an und repräsentieren sie in Form von Berichten. Diese können tabellarisch oder in Form von Diagrammen dargestellt sein.

**Beschreibungsebene:** Zustand im ↑Entwurfsprozess.

**Beschreibungsformalismus:** Darstellungsform für ↑Entwurfsdokumente.

**Beziehung:** Abhängigkeit zwischen zwei Modellelementen.

**Bidirektionale Assoziation:** Beidseitig direkt navigierbare ↑Assoziation, d. h. eine Assoziation, bei der von beiden beteiligten ↑Assoziationsrollen zur jeweils anderen direkt navigiert werden kann.

**Constraint:** Semantische Bedingung bzw. Restriktion.

**Cursor:** Ermöglicht die satzweise Verarbeitung einer Menge von Datensätzen mit Hilfe eines Zeigers.

**Data Mart:** „Kleines“ ↑Data Warehouse, das sich auf einzelne abteilungsspezifische oder geschäftsprozessorientierte Unternehmensausschnitte beschränkt.

**Data Migration:** ↑Transformation, die dem Zweck der Vereinheitlichung dienen.

**Data Mining:** (Semi–)automatische Analyse eines großen oder komplexen Datenbestandes mit dem Ziel, neue, signifikante Muster oder Trends zu entdecken, die sonst unerkannt bleiben.

**Data Mining–Werkzeuge:** Werkzeuge, die ↑Data Mining unterstützen.

**Data Warehouse (DWH):** Physische ↑Datenbank, in der sowohl ↑Schema als auch Daten integriert sind. Das Schema ist analyseorientiert ausgelegt, einmal im ∼ gespeicherte Daten werden nicht mehr modifiziert. Die Daten im ∼ sind typischerweise, aber nicht notwendig historisiert.

**Data Warehouse–Manager:** Verwaltungskomponente des ↑Data Warehouse–Systems, der den ↑Data Warehouse–Prozess steuert.

**Data Warehousing:** Umfasst den dynamischen Vorgang der Datenverarbeitung in einem ↑Data Warehouse–System von der Extraktion in den Datenquellen bis hin zur Auswertung.

**Data Warehouse–Prozess:** Synonym für ↑Data Warehousing.

**Data Warehouse–System (DWS):** Umfasst alle für das ↑Data Warehousing notwendigen Hardware– und Softwarekomponenten (außer den Datenquellen) sowie deren Zusammenspiel.

**Datenanalyse:** Synonym für ↑Analyse.

**Datenbank (DB):** Strukturierte Sammlung von Daten, die mit Hilfe eines ↑Datenbankmanagementsystems verwaltet wird.

**Datenbankentwurf:** Prozess der Modellierung eines vorgegebenen Weltausschnitts mit dem Ziel der Erzeugung eines ↑Schemas in der formalen Sprache des ↑Datenmodells.

**Datenbanksystem (DBS):** Kombination eines ↑Datenbankmanagementsystems mit mindestens einer ↑Datenbank.

**Datenbankmanagementsystem (DBMS):** Aus einer Speicherungs- und einer Verwaltungskomponente bestehendes Programm. Die Speicherungskomponente erlaubt, Daten und ihre Beziehungen abzulegen; die Verwaltungskompenente stellt Funktionen und Sprachmittel zur Pflege und Verwaltung der Daten zur Verfügung.

**Datendefinitionssprache:** Sprache zum Erstellen und Verändern eines ↑Schemas.

**Datenintegration:** Prozess der Transformation, Bereinigung und Konsolidierung von aus heterogenen ↑Datenquellen extrahierten Daten. Integrierte Daten werden im ↑Operational Data Store abgelegt.

**Datenmodell:** Formale Sprache zur strukturierten Beschreibung von Daten und ihren Beziehungen sowie von Operationen auf diesen Daten. Besondere Bedeutung kommt dem ↑Entity Relationship–Modell, dem ↑objektrelationalen und dem ↑objektorientierten ∼ zu.

**Datenquelle:** Organisationsinterne oder –externe ↑Datenbank oder andere Daten speichernde Komponente (z. B. Flat Files, WWW–Seiten etc.), aus der Daten für das ↑Data Warehousing entnommen werden.

**Datenschema:** Langform für ↑Schema.

**Datentyp:** Gegenüber ↑Objekten besitzen Ausprägungen eines ∼s keine Identität, sondern nur eine Wertgleichheit.

**Datenwürfel:** Kern–Metapher des ↑multidimensionalen Datenmodells, das zur Auswertung relevante ↑Fakten mit den Beschreibungsdaten in einem mehrdimensionalen Datenraum („Würfel“) anordnet. Jede Art der Beschreibungsdaten wird im Sinne des ∼s als ↑Dimension bezeichnet.

**Detaildaten:** Daten mit der feinsten verfügbaren ↑Granularität.

**Dimension:** ↑Qualifizierende Eigenschaft eines ↑Fakts.

**Dimensionshierarchie:** Langform für ↑Hierarchie.

**Dimensionalität:** Anzahl qualifizierender Eigenschaften eines ↑Faktes.

**Dimensionselement:** ↑Objekt innerhalb einer ↑Dimension.

**Diskriminierendes Attribut:** ↑Attribut einer ↑Relation oder ↑Klasse, das zur Typunterscheidung des ↑Tupels oder ↑Objektes dient.

**Domäne:** Benannte Menge von Werten.

**Drilling:** Zusammenfassende Bezeichnung für ↑Roll–Up und ↑Drill–Down.

**Drill–Down:** Umkehrung einer ↑Roll–Up–Operation.

**DWS–Manager:** Administrationswerkzeug, das für die Steuerung und Überwachung der einzelnen, im ↑Data Warehouse–System stattfindenden Prozesse zuständig ist.

**Eigenschaftswert:** Synonym für ↑Elementeigenschaft.

**Einfachverbung:** Jede ↑Unterklasse erbt nur von genau einer direkten ↑Oberklasse.

**Elementeigenschaft:** Aus einem Schlüsselwort (dem sog. *Tag*) und einem dazugehörigen Wert (dem sog. *Value*) bestehender Erweiterungsmechanismus der UML. Der Unterschied zum ↑Stereotyp besteht darin, dass durch ein Stereotyp das Metamodell um ein neues Element erweitert wird. Mit ∼n hingegen können einzelne Ausprägungen bestehender Modellelemente (z.B. eine bestimmte Operation) um bestimmte Eigenschaften erweitern.

**Entität:** Individuelles Objekt der realen oder der Vorstellungswelt. Sofern eine Beziehung zwischen Entitäten eine Bedeutung in der realen oder in der Vorstellungswelt hat, kann auch ein individuelles Exemplar einer solchen Beziehung als Entität aufgefasst werden.

**Entitätstyp:** Menge von ↑Entitäten.

**Entity Relationship–Modell (ERM):** Formale Sprache zur Beschreibung von statischen Strukturen der Anwendungswelt. Es dient zumeist als Grundlage des ↑Datenbankentwurfs von ↑herkömmlichen Datenbanken.

**Entwurfsdokument:** Resultat eines ↑Entwurfsschrittes.

**Entwurfsmethodik:** Strukturierter Ansatz, der unter Verwendung bestimmter Vorgehensweisen, Techniken, Werkzeuge und Dokumentationen den ↑Entwurfsprozess einer DB unterstützt.

**Entwurfsprozess:** Abfolge von ↑Entwurfsdokumenten. Jedes Entwurfsdokument wird mit den Mitteln eines ↑Beschreibungsformalismus verfasst und gehört zu einer ↑Beschreibungsebene. Zwischen zwei Beschreibungsebenen erfolgt ein ↑Entwurfsschritt.

**Entwurfsschritt:** Im ↑Entwurfsprozess Übergang zwischen zwei ↑Beschreibungsebenen.

**ETL–Prozess:** Prozess, der die Datenverarbeitung von den ↑Datenquellen bis zum ↑Data Warehouse beschreibt. Mit „Transformation“ ist dabei ↑Datenintegration gemeint.

**Extension:** Konkrete Ausprägung einer ↑Datenbank.

**Extraktion–Transformation–Laden–Prozess:** Langform für ↑ETL–Prozess.

**Fakt:** Objekt, das ↑quantifzierende und ↑qualifizierende Eigenschaften besitzt. Die quantifizierenden Eigenschaften beinhalten für die Organisation relevante Daten, die während der ↑Datenanalyse weitergehend untersucht werden können. Qualifizierende Eigenschaften dienen der näheren Beschreibung der quantifizierenden Eigenschaften, wodurch diese eine Bedeutung erhalten.

**Faktattribut:** Bestandteil eines ↑Fakts.

**Fremdschlüssel:** ↑Attribut oder Attributkombination, das/die in einer anderen ↑Relation ↑Primärschlüssel ist.

**Extraktion:** Selektieren von Daten aus den ↑Datenquellen und deren Bereitstellung zur ↑Datenintegration.

**Front End–Bereich:** Bereich eines ↑Data Warehouse–Systems, die der Untersuchung und Präsentation von Daten dient. Die dabei eingesetzten Methoden und Werkzeuge reichen vom einfachen Reporting mit ↑Abfragewerkzeugen bis hin zu komplexen Analysemethoden wie z. B. ↑On–Line Analytical Processing und ↑Data Mining.

**Funktionale Abhängigkeit:** Für eine ↑Relation R mit ↑Attributen X und Y heißt Y funktional abhängig von X, genau dann, wenn jeder X–Wert in R genau einen Y–Wert in R bestimmt.

**Ganzes–Teil–Beziehung:** Synonym für ↑Aggregation.

**Generalisierung:** Modellierungskonstrukt, bei dem gleiche Eigenschaften verschiedener ↑Klassen nur einmal für eine gemeinsame ↑Basisklasse modelliert werden. Siehe auch ↑Vererbung.

**Geordnete Assoziation:** ↑Assoziation, bei der die Objektverbindungen in bestimmter Weise geordnet sind.

**Gerichtete Assoziation:** ↑Assoziation, bei der von der einen beteiligten ↑Assoziationsrolle zur anderen direkt navigiert werden kann, nicht aber umgekehrt.

**Granularität:** Stufe des Verdichtungsgrades von Daten innerhalb einer ↑Hierarchie. Dabei haben ↑Detaildaten den niedrigsten Verdichtungsgrad und die feinste $\sim$ . Zusammengefasste Daten haben entsprechend einen höheren Verdichtungsgrad und damit eine gröbere $\sim$ .

**Gruppierung:** Synonym für ↑Verdichtung.

**Herkömmliche Datenbank:** Synonym für ↑OLTP–Datenbank.

**Hierarchie:** Menge aufeinander aufbauender ↑Hierarchieebenen.

**Hierarchieebene:** Auswertungsorientierte Zusammenfassung der Daten in einer ↑Dimensionen.

**Hybrides On–Line Analytical Processing (HOLAP):** ↑OLAP–System, in dem die Daten sowohl in einem ↑Multimensionalen DBMS als auch einem ↑Relationalen DBMS gehalten werden.

**Identität:** Eigenschaft eines ↑Objektes, die es von allen anderen Objekten unterscheidet und die nicht verändert werden kann.

**Index:** Ein $\sim$ ist eine physische Datenstruktur, die die Zugriffsgeschwindigkeit auf die in der Datenbank gespeicherten Daten erhöht.

**Informationssystem (i. e. S.):** Computergestützter Teil eines ↑Informationssystems (i. w. S.). ∼e sind Datenorganisationsformen auf Computern, die bestimmte Vorgänge und Abläufe in Kommunikationsprozessen unterstützen.

**Informationssystem (i. w. S.):** Gesamte informationsverarbeitende Teilsystem einer Organisation. Es besteht wiederum aus Teilsystemen, wie Anwendungssystemen und Aktenarchiven. Man kann das ∼ in das computergestützte ↑(Informationssystem (i. e. S.) und das nichtcomputergestützte Informationssystem unterteilen.

**Instanz:** Synonym für ↑Objekt.

**Integrität:** Unter ∼ einer ↑Datenbank werden alle Aspekte zusammengefasst, die im weitesten Sinne mit der Korrektheit der Daten in der ↑Datenbank zu tun haben.

**Integritätsbedingung:** Synonym für ↑Konsistenzbedingung.

**Kategorie A–Metadaten:** ↑Metadaten eines ↑LCD of SQL–Schemas, die sich auf ein Objekt beziehen.

**Kategorie B–Metadaten:** ↑Metadaten eines ↑LCD of SQL–Schemas, die sich auf zwei Objekte beziehen.

**Kenngröße:** Synonym für ↑Fakt.

**Kennzahl:** Synonym für ↑Faktattribut.

**Klasse:** Zusammenfassung einer ↑Klassendefinition und der Menge der ↑Objekte, die nach diesem Schema erzeugt wurden.

**Klassenbibliothek:** Sammlung von ↑Klassen.

**Klassendefinition:** Definition der Merkmale von ↑Objekten. Eine ∼ beschreibt die Objekte einer ↑Klasse durch ein Schema, nach dem ↑Instanzen dieser Klasse erzeugt und manipuliert werden. Dieses Schema besteht aus dem Klassennamen sowie den ↑Attributen und ↑Methoden der Klasse.

**Klassendiagramm:** Zeigt eine Menge von ↑Klassen und ihre ↑Beziehungen.

**Komposition:** Spezielle Form der ↑Assoziation, die eine *Ganzes–Teil*–Beziehung zwischen zwei Komponenten beschreibt. Gegenüber einer ↑Aggregation sind die Instanzen der als *Teil* betrachteten Komponente von der *Ganzes*–Komponente abhängig.

**Konsistenz:** Beinhaltet die logische Korrektheit bzgl. vorgegebener ↑Konsistenzbedingungen der Daten und damit die Übereinstimmung des Inhaltes der ↑Datenbank mit der Datenbeschreibung.

**Konsistenzbedingung:** Logische Formel über die in der ↑Datenbank gespeicherten ↑Entitäten.

**Konventionelle Datenbank:** Synonym für ↑OLTP–Datenbank.

**Laden:** Schritt innerhalb des ↑Data Warehouse–Prozesses, in dem Daten aus dem ↑Operational Data Store in das analyseorientierte Schema des ↑Data Warehouse übernommen werden. Mit diesem Schritt ist häufig eine ↑Verdichtung der Daten verbunden.

**Ladewerkzeug:** Werkzeug, das den Prozess des ↑Ladens unterstützt.

**LCD of SQL:** Physisches relationales ↑Datenmodell, das elementare Konstrukte des ↑SQL–Standards und der am weitesten verbreiteten ↑Datenbankmanagementsysteme enthält.

**Maßzahl:** Synonym für ↑Kennzahl.

**Materialisierte Sicht:** Redundantes, physisches Abspeichern einer ↑Sicht.

**Materialisierung:** Synonym für ↑Materialisierte Sicht.

**Mehrfachhierarchien:** Synoym für ↑Multiple Hierarchie.

**Mehrfachvererbung:** Form der ↑Vererbung, bei der eine ↑Klasse mehrere direkte ↑Oberklassen besitzt. Eine Klasse, die durch $\sim$ definiert wird, erbt die ↑Merkmale aller Oberklassen. Dabei können Konflikte entstehen, falls ein Merkmal in verschiedenden Oberklassen definiert ist oder eine indirekte Oberklasse als mehrfache Oberklasse auftritt.

**Merkmal:** Oberbegriff für die charakteristischen Eigenschaften eines ↑Objektes, d. h. dessen ↑Attribute und ↑Methoden.

**Messprozess:** Synonym für ↑Messverfahren.

**Messung:** Anwendung eines ↑Messverfahrenes auf ein ↑Untersuchungsobjekt.

**Messverfahren:** Menge von Tätigkeiten zur Ermittlung eines Größenwertes, der eine spezifische Eigenschaft eines ↑Untersuchungsobjektes beschreibt.

**Metadaten:** Jede Form von Informationen über Daten. Dies betrifft in einem ↑Data Warehouse System neben den Schemadaten (Metadaten i. e. S.) auch den ↑Data Warehouse–Prozess beschreibende Daten wie Transformationsregeln, Daten zur Sicherheit, Daten über Herkunft und Güte etc., also Metadaten i. w. S.

**Metadaten–Manager:** Synonym für ↑Repository–Manager.

**Metadaten–Repository:** ↑Datenbankmanagementsystem zur Verwaltung von ↑Metadaten.

**Metaklasse:** ↑Klasse, deren ↑Instanzen wiederum Klassen sind.

**Methode:** Operation, die ein ↑Objekt ausführen kann.

**Metrik:** Abbildung(–svorschrift) von einem ↑Untersuchungsobjekt auf Zahlen oder Symbole mit dem Ziel, eine spezifische Eigenschaft des Untersuchungsobjektes zu charakterisieren.

**Monitor:** Werkzeug, das den Prozess des ↑Monitoring unterstützt.

**Monitoring:** Tätigkeit innerhalb des ↑Data Warehouse–Prozesses, in dem die ↑Datenquellen überwacht werden, um zu extrahierende Daten zu bestimmen.

**Multidimensionale Datenbank (MDB):** Auf Grundlage eines ↑Multidimensionalen Datenmodells aufgebaute ↑Datenbank.

**Multidimensionales Datenbanksystem (MDBS):** Auf Grundlage eines ↑Multidimensionalen Datenmodells basierendes ↑Datenbanksystem.

**Multidimensionales Datenbankmanagementsystem (MDBMS):**
↑Datenbankmanagementsystem, das die Verwaltung ↑multidimensionaler Datenbanken ermöglicht.

**Multidimensionales Datenmodell:** ↑Datenmodell, das die Modellierung der Daten zu Analysezwecken ermöglicht. Wesentliches Charakteristikum sind die Unterscheidung der Daten in ↑Fakten und ↑Dimensionen sowie die Möglichkeit der Bildung von ↑Hierarchien auf den ↑Dimensionen.

**Multidimensional OLAP (MOLAP):** Mit ∼ werden ↑Datenbanksysteme bezeichnet, deren Daten in einem ↑MDBMS gehalten werden.

**Multidimensionales Schema:** ↑Schema, das mit den Mitteln eines ↑multidimensionalen Datenmodells erstellt wurde.

**Multiple Hierarchie:** Spezialform der ↑Hierarchie, bei der auf eine ↑Hierarchieebene alternativ mehrere folgen können. ∼n erlauben somit Verzweigungen innerhalb einer Hierarchie.

**Multiplizität:** Beschreibt die Anzahl von ↑Objekten, die an einer ↑Assoziation oder ↑Komposition beteiligt sein können.

**Navigation:** Die Betrachtung von Zugriffsmöglichkeiten auf ↑Objekte (und ihre ↑Attribute und ↑Methoden) innerhalb einer Menge von Objekten. Direkt navigierbar werden solche Zugriffe genannt, die ohne Umwege möglich sind.

**Navigierbarkeit:** Synonym für ↑Navigation.

**Nestung:** Fasst mehrere Objekte zu einem Objekt zusammen.

**Nicht–vollständige Verdichtung:** ↑Verdichtung, an der nicht alle Instanzen einer ↑Hierarchieebene teilnehmen.

**Normalform:** Eine ↑Relation ist in einer bestimmten ∼ , wenn sie eine Menge von Eigenschaften erfüllt. Definiert sind die 1. bis 5. Normalform und die Boyce–Codd Normalform.

**Notiz:** Kommentar zu einem Diagramm oder einem oder mehreren Elementen ohne semantische Wirkung.

**Nullwert:** Ausgezeichneter Datenwert, der Element jeder ↑Domäne ist. Steht meistens für „nicht vorhanden“ oder „nicht bekannt“.

**Oberklasse:** ↑Klasse, deren ↑Attribute und ↑Methoden durch ↑Vererbung an ↑Unterklassen übertragen werden. In Abhängigkeit von der Anzahl der ↑Vererbungsstufen spricht man auch von den direkten und indirekten Oberklassen einer Klasse.

**Object Constraint Language (OCL):** Definiert eine Sprache zu Beschreibung von Zusicherungen, Invarianten, Vor- und Nachbedingungen und ↑Navigation.

**Objekt:** Zusammenfassung einer Datenstruktur und der darauf anwendbaren ↑Methoden zu einer Einheit. Ein ∼ besitzt eine Struktur (Objektstruktur), einen Zustand (Objektzustand), ein Verhalten (Objektverhalten) und eine ↑Identität.

**Objektdiagramm:** Diagramm, das ↑Objekte und ihre ↑Beziehungen untereinander zu einem bestimmten Zeitpunkt zeigt.

**Objektidentität:** Synonym für ↑Identität.

**Objektorientiertes Datenmodell:** ↑Datenmodell, das objektorientierte Konstrukte wie Klassen, Vererbung etc. zur Verfügung stellt.

**Objektrelationales Datenmodell:** ↑Datenmodell, das Aspekte des ↑Relationenmodells und ↑objektorientierten Modells verbindet.

**On–Line Analytical Processing (OLAP):** ∼ ist eine interaktive, explorative Datenanalyse auf Grundlage eines multidimensionalen Datenmodells.

**On–Line Transaction Processing (OLTP):** Arbeitsprozess, der von den klassischen, operativen, transaktionsorientierten Datenbankanwendungen verfolgt wird.

**OLAP–Werkzeug:** Werkzeug, das ↑OLAP ermöglicht.

**OLTP–Datenbank:** ↑Datenbank, die das ↑OLTP unterstützt.

**Operational Data Store (ODS):** Physische ↑Datenbank, die konsolidierte, feingranulare Daten speichert. Daten können nicht nur hinzugefügt, sondern auch modifiziert werden. Im Mittelpunkt steht die Bereitstellung konsolidierter Daten für ein ↑Data Warehouse. Das Schema des ∼ ist im Gegensatz zum ↑Data Warehouse nicht vorrangig analyseorientiert.
Manchmal greifen auch ↑Analysesysteme auf das ∼ durch (↑Reach–Through), um die Auswertung von Daten auf feingranularer Ebene zu ermöglichen.

**Operative Datenbank:** Synonym für ↑OLTP–Datenbank.

**Optionales Attribut:** ↑Attribut, das nicht immer einen konkreten Wert besitzen muss, sondern statt dessen auch undefiniert sein kann.

**Pivotisierung:** Synonym für ↑Rotation.

**Primärschlüssel:** ↑Attribut oder Attributkombination, das/die eindeutig ein ↑Tupel einer ↑Relation kennzeichnet.

**Projekt:** Menge von ↑Projektinformationen.

**Projektinformationen:** Menge von Schemata und ↑Prozessschritten.

**Prozessschritt:** Synonym für ↑Entwurfsschritt.

**Qualifizierende Eigenschaft:** Eigenschaft, z. B. ein ↑Attribut oder eine ↑Assoziation, die ein ↑Dimensionselement oder ↑Fakt beschreibt.

**Qualitätssicherung:** Gesamtheit aller planbaren Maßnahmen und Hilfsmittel, die bewusst dazu eingesetzt werden, um die Anforderungen an den Entwicklungs– und Wartungsprozess und an ein Softwareprodukt zu erfüllen.

**Quantifizierende Eigenschaft:** Eigenschaft eines ↑Fakts, die für die ↑Datenanalyse wichtigen (numerischen) Daten enthält.

**Quelle:** Kurzform für ↑Datenquelle.

**Reach–Through:** Durchgriff von ↑Analysewerkzeugen auf den ↑Operational Data Store oder die ↑Datenquellen.

**Redundanz:** Mehrfache Speicherung desselben Sachverhalts in einer ↑Datenbank.

**Referentielle Integrität:** Stellt sicher, dass alle Werte eines ↑Attributes oder einer Attributkombination, das/die als ↑Fremdschlüssel definiert ist, in einer anderen ↑Relation (als ↑Primärschlüssel) vorhanden sind.

**Relation:** Tabelle, die einen eindeutig festgelegten Namen besitzt. Der Tabelleninhalt besteht aus einer Menge von ↑Tupeln.

**Relationales DBMS (RDBMS):** Auf dem ↑Relationenmodell basierendes ↑DBMS.

**Relationenmodell:** ↑Datenmodell, das sowohl Daten als auch Datenbeziehungen in Form von Tabellen ausdrückt.

**REMUS:** Logische relationales ↑Datenmodell, das in Ergänzung zu herkömmlichen relationalen Datenmodellen zahlreiche ↑Metadaten enthält, die in ↑Kategorie A– und ↑Kategorie B–Metadaten unterteilt werden.

**Repository:** Kurzform für ↑Metadaten–Repository.

**Repository–Manager:** Verwaltungskomponente eines ↑Repository, fungiert als Schnittstelle zum Repository.

**ROLAP (Relational ↑OLAP):** ↑OLAP-Systeme, deren Daten in einem ↑RDBMS gehalten werden.

**Roll–Up:** Operation in ↑multidimensionalen Datenmodellen, Synonym für ↑Verdichtung.

**Rolle:** Synonym für ↑Assoziationsrolle.

**Rotation:** Umfasst das Drehen eines ↑Datenwürfels, so dass dem Benutzer eine spezifische Sicht angezeigt wird. Zudem beinhaltet der Begriff ∼ das Ein- und Ausblenden von ↑Dimensionen. Beim Ausblenden einer Dimension werden die dargestellten Daten geeignet verdichtet (siehe ↑Aggregation).

**Satz:** Ein ∼ ist eine logische oder physische Einheit von Daten.

**Schema:** Die mit den Mitteln eines ↑Datenmodells festgelegte Struktur einer ↑Datenbank.

**Schlüssel:** ↑Attribut oder Attributkombination, womit Elemente (↑Entitäten oder ↑Objekte) in einer Menge von Elementen ausgezeichnet werden können.

**Schneeflockenschema:** Relationale Repräsentation multidimensionaler Daten, wobei diese in Fakt– und Dimensionstabellen gespeichert werden und die Dimensionstabellen normalisiert sind.

**Sicht:** Spezifischer Ausschnitt der Daten einer ↑Datenbank.

**Slice and Dice:** Benutzergesteuerte Erforschung eines Datenbestandes. Der Anwender kann während dieses Vorganges Teile eines ↑Datenwürfels selektieren, Datenwerte ↑aggregieren oder transformieren, unterschiedliche ↑Datenwürfel miteinander verknüpfen oder einen Würfel aus verschiedenen Perspektiven betrachten.

**Spezialisierung:** Definition einer neuen ↑Klasse als ↑Unterklasse einer oder mehrerer anderer Klassen (↑Oberklassen). Aufgrund der ↑Vererbung besitzt die neue Klasse alle ↑Merkmale ihrer Oberklassen. In ihrer ↑Klassendefinition können jedoch zusätzliche Merkmale definiert (↑Erweiterung) oder geerbte Merkmale überschrieben (↑Redefinition) werden. Die neue Klasse stellt deshalb eine ∼ ihrer Oberklassen dar. Eine Unterklasse muss jedoch nicht unbedingt zusätzliche Merkmale definieren oder geerbte Merkmale redefinieren, sie kann auch ausschließlich die Vereinigung der Merkmale ihrer Oberklassen bilden.

**Standardisierte Annotation:** Synonym für ↑Elementeigenschaft.

**Standardwert:** Wert, mit dem eine neu erzeugte Datenstruktur vorbelegt wird, ohne dass der Entwickler ihn explizit angeben muss. ∼e sind i. Allg. vom Typ der zu initialisierenden Datenstruktur abhängig.

**Stereotyp:** Dient zur (werkzeug–, projekt–, unternehmens– oder methodenspezifischen) Erweiterung vorhandener Modellelemente der ↑UML, d. h. ihres Metamodells. Entsprechend der mit der Erweiterung definierten Semantik wird das Modellierungselement, auf das der ∼ angewendet wird, direkt semantisch beeinflusst.

**Sternschema:** Relationale Repräsentation multidimensionaler Daten, in der diese in Fakt– und Dimensionstabellen gespeichert werden, wobei die Dimensionstabellen denormalisiert sind.

**Subklasse:** Synonym für ↑Unterklasse.

**Subschema:** Teil eines ↑Schemas.

**Surrogat:** Eindeutiger, meistens vom System vergebener Identifikator eines Datensatzes.

**Tagged Value:** Englische Bezeichnung und Synonym für ↑Elementeigenschaft.

**Transformation:** Schritt innerhalb des ↑Data Warehouse–Prozesses, in dem Daten mit Hilfe einer bestimmten Vorschrift umgewandelt werden.

**Transformationswerkzeug:** Werkzeug, das den Prozess der ↑Transformation unterstützt.

**Tupel:** Abkürzung für n–Tupel, also ein Element aus einem n–stelligen kartesischen Produkt. Es bezeichnet ein Element einer ↑Relation.

**Unbalancierte Hierarchie:** ↑Hierarchie, in der bei Zuordnung von Elementen einer Hierarchieebene zur nächsthöheren (oder nächstniedrigeren) Ebene nicht immer zugehörige Elemente existieren.

**Ungerichtete Assoziation:** ↑Assoziation, deren Richtung nicht festgelegt ist, d. h. unspezifiziert ist. In der ↑UML wird die ∼ häufig auch mit der ∼ bidirektionalen Assoziation gleichgesetzt.

**Unified Modeling Language (UML):** Eine von der Object Management Group (OMG) standardisierte Notation und Semantik zur Visualisierung, Konstruktion und Dokumentation von Modellen für die objektorientierte Softwareentwicklung.

**Unterklasse:** ↑Klasse, deren ↑Merkmale durch ↑Vererbung aus ein oder mehreren anderen Klassen (↑Oberklassen) übernommen werden. In Abhängigkeit von der Anzahl der ↑Vererbungsstufen spricht man auch von den direkten und indirekten ∼n einer Klasse.

**Untersuchungsobjekt:** Gegenstand einer ↑Messung.

**Verbund:** Operator, der zwei ↑Relationen über einen gemeinsamen Ausdruck verbindet und eine Resultatrelation erzeugt.

**Verdichtung:** Zusammenfassen von Daten mittels einer Berechnungsvorschrift. In ↑multidimensionalen Datenmodellen werden ∼en über ↑quantifizierende Attribute entlang einer ↑Dimension betrachtet.

**Verdichtungsoperator:** Berechnungsvorschrift, wie z. B. *sum*, *avg* oder *max*, mit der die betrachteten Daten bei Durchführung einer ↑Verdichtungs–Operation zusammengefasst werden können.

**Verdichtungspfad:** Synonym für ↑Hierarchie.

**Vererbung:** Beziehung zwischen ↑Klassen, durch die eine Klasse (↑Unterklasse) die Merkmale einer (↑Einfachvererbung) oder mehrerer (↑Mehrfachvererbung) anderer Klassen (↑Oberklassen) übernimmt.

**Verteilte Datenbanken:** ↑Datenbank, deren Daten auf mindestens zwei Rechner verteilt sind; diese sind durch ein Rechnernetz untereinander verbunden und auf jedem Rechner des Netzes stehen alle Daten zur Verfügung.

**Verteiltes Datenbankmanagementsystem:** ↑Datenbankmanagementsystem, das ↑verteilte Datenbanken verwalten kann.

**Workload:** Menge von gewichteten Aufgaben, die auf einer ↑Datenbank ausgeführt werden.

**Würfel:** Kurzform für ↑Datenwürfel.

**Würfelzelle:** Kleinstes Element innerhalb eines ↑Datenwürfels, kann durch ↑Dimensionselemente der feinsten ↑Granularität adressiert werden.

**Zelle:** Kurzform für ↑Würfelzelle.

**Zerlegung:** Aufteilung einer Menge in Teilmengen, wobei drei Eigenschaften erfüllt sein müssen:
(i) Die leere Menge darf nicht Element der Zerlegung sein.
(ii) Die Vereinigung aller Zerlegungselemente muss wieder die Menge ergeben.
(iii) Zwei verschiedene Elemente aus der Zerlegung müssen disjunkt sein.

# Literaturverzeichnis


[AAD+96] Agarwal, Sameet, Rakesh Agrawal, Prasad Deshpande, Ashish Gupta, Jeffrey F. Naughton, Raghu Ramakrishnan und Sunita Sarawagi: *On the Computation of Multidimensional Aggregates.* In: Vijayaraman, T. M., Alejandro P. Buchmann, C. Mohan und Nandlal L. Sarda (Herausgeber): *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 1996, Mumbai (Bombay, Indien)*, Seiten 506–521. Morgan Kaufmann, 1996.

[ABH+98] Appelrath, Hans-Jürgen, Marit Beyer, Holger Hinrichs, Joachim Kieschke, Kirsten Panienski, Martin Rohde, Alexander Scharnofske, Wilfried Thoben, Ina Wellmann, Frank Wietek und Ludger Zachewitz: *CARLOS – Tätigkeitsbericht für den Zeitraum 1.1.1998–31.12.1998.* OFFIS, Oldenburg (Deutschland), Dezember 1998.

[ACPT99] Atzeni, Paolo, Stefano Ceri, Stefano Paraboschi und Riccardo Torlone: *Database Systems – Concepts, Languages and Architectures.* McGraw–Hill, 1999.

[AFH+97] Appelrath, Hans-Jürgen, Jörg Friebe, Elke Hinrichs, Holger Hinrichs, Ina Hoting, Joachim Kieschke, Kirsten Panienski, Jens Rettig, Alexander Scharnofske, Wilfried Thoben und Frank Wietek: *CARLOS – Tätigkeitsbericht für den Zeitraum 1.1.1997–31.12.1997.* OFFIS, Oldenburg (Deutschland), Dezember 1997.

[Alh98] Alhir, Sinan Si: *UML in a Nutshell.* O'Reilly–Verlag, London (England), 1998.

[ALW98] Albrecht, Jens, Wolfgang Lehner und Hartmut Wedekind: *Normal Forms for Multidimensional Databases.* 10th International Conference on Scientific and Statistical Data Management, Juli 1998.

[AM97a] Anahory, Sam und Dennis Murray: *Data Warehouse: Planung, Implementierung und Administration.* Addison–Wesley, Juni 1997.

[AM97b] Anahory, Sam und Dennis Murray: *Data Warehousing in the Real World: A Practical Guide for Building Decision Support Systems.* Addison–Wesley, Juni 1997.

[Arb99] Arbeitsgemeinschaft bevölkerungsbezogener Krebsregister in Deutschland: *Krebs in Deutschland – Häufigkeiten und Trends.* Arbeitsgemeinschaft bevölkerungsbezogener Krebsregister in Deutschland, 1999.

[Bal98] Balzert, Helmut: *Lehrbuch der Software–Technik.* Spektrum Akademischer Verlag, Heidelberg (Deutschland), 1998.

[BBC+99] Bernstein, Philip A., Thomas Bergstraesser, Jason Carlson, Shankar Pal, Paul Sanders und David Shutt: *Microsoft Repository Version 2 and the Open Information Model.* Information Systems, 24(2):71–98, 1999.

[BCJJ98] Busborg, Frank, Jens G. Borch Christiansen, Kristian M. Jensen und Lars R. Jensen: *Data Warehouse Modeling: The Nykredit Case Study.* Technischer BerichtDat5 Report/Part I, Aalborg University, Computer Science Department, Aalborg (Dänemark), 1998.

[BCN92] Batani, Carlo, Stefano Ceri und Shamkant Navathe: *Conceptual Database Design. An Entity–Relationship–Approach.* Redwood City, 1992.

[BD99] Borst, Thomas und Matthias Diedrich: *Terabyte Warehousing mit Oracle8.* In: Scherrer, W. (Herausgeber): *Vortragsband zur 12.Jahrestagung der Deutschen Oracle Anwender–Konferenz 1999 (DOAG99)*, Seiten 324–333, Fellbach (Deutschland), November 1999.

[BE99a] Böhnlein, Michael und Achim Ulbrich–vom Ende: *Using the Conceptual Data Models of the Operational Information Systems for the Construction of Initial Data Warehouse Structures*. In: Sinz, Elmar J. (Herausgeber): *Proceedings der MobiS–Fachtagung 1999*, Bamberg (Deutschland), Oktober 1999.

[BE99b] Böhnlein, Michael und Achim Ulbrich-vom Ende: *Deriving Initial Data Warehouse Structures from the Conceptual Data Models of the Underlying Operational Information Systems.* In: *DOLAP '99, ACM Second International Workshop on Data Warehousing and OLAP, November 1999, Kansas City (Missouri, USA), Proceedings,* Seiten 12–16. ACM Press, 1999.

[BES98] Becker, Jörg, Lars Ehlers und Reinhard Schütte: *Grundsätze ordnungsmäßiger Modellierung – Konzeption, Vorgehensmodelle, technische Realisierung, Nutzen.* http://www.wi.uni-muenster.de/is/mitarbeiter/isresc/resc_Statutstagung.pdf, 1998.

[BFM99] Beeri, Catriel, Anna Formica und Michele Missikoff: *Inheritance hierarchy design in object-oriented databases.* Data & Knowledge Engineering Journal (DKE), 30(3):191–216, Juli 1999.

[BG01] Bauer, Andreas und Holger Günzel (Herausgeber): *Data Warehouse–Systeme — Architektur, Entwicklung, Anwendung.* dpunkt–Verlag, 2001.

[BH98] Becker, Jörg und Roland Holten: *Fachkonzeptuelle Spezifikation von Führungsinformationssystemen.* Wirtschaftsinformatik, 40(6):483–492, Dezember 1998.

[BKK96] Berchtold, Stefan, Daniel A. Keim und Hans-Peter Kriegel: *The X–tree : An Index Structure for High-Dimensional Data.* In: Vijayaraman, T. M., Alejandro P. Buchmann, C. Mohan und Nandlal L. Sarda (Herausgeber): *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 1996, Mumbai (Bombay, Indien),* Seiten 28–39. Morgan Kaufmann, 1996.

[BKSS90] Beckmann, Norbert, Hans-Peter Kriegel, Ralf Schneider und Bernhard Seeger: *The R*–Tree: An Efficient and Robust Access Method for Points and Rectangles.* In: Garcia-Molina, Hector und H. V. Jagadish (Herausgeber): *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Mai 1990, Atlantic City (New Jersey, USA),* Seiten 322–331. ACM Press, 1990.

[BLT86] Blakeley, José A., Per-Åke Larson und Frank Wm. Tompa: *Efficiently Updating Materialized Views*. In: Zaniolo, Carlo (Herausgeber): *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data, Washington, D.C. (USA), Mai 1986,* Seiten 61–71. ACM Press, 1986.

[BM72] Bayer, Rudolf und Edward M. McCraight: *Organization and Maintenance of Large Ordered Indexes.* Acta Informatica, 1:173–189, 1972.

[BPT97] Baralis, Elena, Stefano Paraboschi und Ernest Teniente: *Materialized Views Selection in a Multidimensional Database.* In: Jarke, Matthias, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos und Manfred A. Jeusfeld (Herausgeber): *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 1997, Athen (Griechenland),* Seiten 156–165. Morgan Kaufmann, August 1997.

[BRS95] Becker, Jörg, Michael Rosemann und Reinhard Schütte: *Grundsätze ordnungsmäßiger Modellierung.* Wirtschaftsinformatik, 37(5):435–445, Oktober 1995.

[Bru91] Bruce, Thomas A.: *Designing Quality Databases With Idef1X Information Models.* Dorset House Publishing, 1991.

[BS96] Becker, Jörg und Reinhard Schütte (Herausgeber): *Handelsinformationssysteme*. Verlag Moderne Industrie, 1996.

[BS97] Berson, Alex und Stephen J. Smith: *Data Warehousing, Data Mining, and OLAP*. McGraw–Hill, 1. Auflage, 1997.

[Bul96] Bulos, Dan: *A New Dimension.* Database Programming & Design 6/1996, Seiten 33–37, 1996.

[Bun01] Bundesministerium für Bildung und Forschung: *Analyse und Evaluation der Softwareentwicklung in Deutschland.* Studie, 2001.

[Bur98] Burkett, William C.: *Database Schema Design Quality Principles.* http://www-scf.usc.edu/ wcb/dmq/dmqmain.html, 1998.

[CBS98] Connolly, Thomas, Carolyn Begg und Anne Strachan: *Database Systems.* Addison–Wesley, 2. Auflage, 1998.

[CD97] Chaudhuri, Surajit und Umeshwar Dayal: *An Overview of Data Warehousing and OLAP Technology*. SIGMOD Record, 26(1), März 1997.

[Che76] Chen, Peter P.: *The Entity–Relationship Model — Toward a Unified View of Data*. TODS, 1(1):9–36, 1976.

[CI98] Chan, Chee Yong und Yannis E. Ioannidis: *Bitmap Index Design and Evaluation.* In: Haas, Laura M. und Ashutosh Tiwary (Herausgeber): *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, Juni 1998, Seattle (Washington, USA)*, Seiten 355–366. ACM Press, 1998.

[CLF99] Chan, Goretti K. Y., Qing Li und Ling Feng: *Design and Selection of Materialized Views in a Data Warehousing Environment: A Case Study.* In: *DOLAP '99, ACM Second International Workshop on Data Warehousing and OLAP, November 1999, Kansas City (Missouri, USA), Proceedings*, Seiten 42–47. ACM Press, 1999.

[CM00] Claxton, John C. und Peter A. McDougall: *Measuring the Quality of Models.* TDAN – www.tdan.com, 3(14), 2000.

[Com01] Computer Associates Inc.: *Homepage Firma Computer Associates.* http://www.ca.com/, 2001.

[Con97] Conrad, Stefan: *Föderierte Datenbanksysteme – Konzepte der Datenintegration.* Springer–Verlag, 1997.

[CT98a] Cabibbo, Luca und Riccardo Torlone: *A Logical Approach to Multidimensional Databases.* In: Hans-Jörg Schek and Fèlix Saltor and Isidro Ramos and Gustovo Alonso (Herausgeber): *Advances in Database Technology - EDBT'98, 6th International Conference on Extending Database Technology, Valencia (Spanien), März 1998, Proceedings*, Band 1377 der Reihe *LNCS*, Seiten 183–197. Springer, März 1998.

[CT98b] Cabibbo, Luca und Riccardo Torlone: *Un Quadro Metodologico per la Costruzione e l'Uso di un Data Warehouse (In Italienisch)*. In: *Sesto Convegno Nazionale su Sistemi Evoluti per Basi di Dati (SEBD '98), September 1998, Ancona (Italien), Proceedings*, Seiten 123–140, 1998.

[Dat99] Data Mart Consulting GmbH: *Einsatzfelder von Technologien und Anwendungen im Data Warehouse Umfeld bei verschiedenen betriebswirtschaftlichen Aufgabenstellungen*. Studie, Data Mart Consulting GmbH der TU Darmstadt, 1999.

[Dat00] Dataquest–Studie, Veröffentlicht in ComputerZeitung 20/2000: *Oracle und IBM liegen Kopf an Kopf – Der globale Datenbankmarkt 1999*, 2000.

[Dat01] Datanamic Inc.: *Homepage Firma Datanamic*. http://www.datanamic.com, 2001.

[DG98] Dodge, Gary und Tim Gorman (Herausgeber): *Oracle8 Data Warehousing*. John Wiley & Sons, Inc., 1998.

[DHP+99] Deifel, Bernhard, Ursula Hinkel, Barbara Paech, Peter Scholz und Veronika Thurner: *Die Praxis der Softwareentwicklung: Eine Erhebung*. Informatik Spektrum, 22(1):24–36, Februar 1999.

[DNR+97] Deshpande, Prasad, Jeffrey F. Naughton, Karthikeyan Ramasamy, Amit Shukla, Kristin Tufte und Yihong Zhao: *Cubing Algorithms, Storage Estimation, and Storage and Processing Alternatives for OLAP*. Data Engineering Bulletin, 20(1):3–11, 1997.

[Dor99] Dorendorf, Stefan: *Die fünf großen W der Datenbankreorganisation bei relationalen Datenbank–Management–Systemen*. In: Frank Hüsemann and Klaus Küspert and Frank Mäurer (Herausgeber): *11. Workshop Grundlagen von Datenbanken, Arbeitskreis Grundlagen von Informationssystemen im GI–Fachausschuß 2.5, Luisenthal (Deutschland), Mai 1999*, Jeaner Schriften zur Mathematik und Informatik, Math/Inf/99/16, Seiten 12–16. Friedrich–Schiller–Universität Jena, 1999.

[DR00] Do, Hong Hai und Erhard Rahm: *On Metadata Interoperability in Data Warehouses*. Technischer Bericht Institut für Informatik, Universität Leipzig, Leipzig (Deutschland), Juni 2000.

[EKN96] EKN, Epidemiologisches Krebsregister Niedersachsen: *Krebs in Niedersachsen – Jahresbericht mit den Daten von 1996*. Niedersächsiches Ministerium für Frauen, Arbeit und Soziales (Hannover)/OFFIS (Oldenburg), 1996.

[EKN01] EKN, Epidemiologisches Krebsregister Niedersachsen: *Homepage Krebsregister Niedersachsen*. http://www.krebsregister-niedersachsen.de/, 2001.

[FGM97] Formica, Anna, Hans Dietmar Gröger und Michele Missikoff: *Object–Oriented Database Schema Analysis and Inheritance Processing: A Graph–Theoretic Approach*. Data & Knowledge Engineering Journal (DKE), 24(2):157–181, 1997.

[FGM98] Formica, Anna, Hans Dietmar Gröger und Michele Missikoff: *An Efficient Method for Checking Object–Oriented Database Schema Correctness*. TODS, 23(3):334–369, 1998.

[FST88] Finkelstein, Sheldon J., Mario Schkolnick und Paolo Tiberio: *Physical Database Design for Relational Databases*. TODS, 13(1):91–128, 1988.

[Gar98] Gardner, Stephen R.: *Building the Data Warehouse*. CACM, 41(9):52–60, 1998.

[GG98] Gabriel, Roland und Peter Gluchowski: *Grafische Notationen für die semantische Modellierung multidimensionaler Datenstrukturen in Management Support Systemen.* Wirtschaftsinformatik, 40(6):493–502, Dezember 1998.

[GGC97] Gluchowski, Peter, Roland Gabriel und Peter Chamoni: *Management–Support–Systeme : Computergestützte Informationssysteme für Führungskräfte und Entscheidungsträger*. Springer, Berlin (Deutschland), 1. Auflage, 1997.

[Gho91] Ghosh, Sakti P.: *Statistical Relational Databases: Normal Forms.* Transactions on Knowledge and Data Engineering, 3(1):55–64, 1991.

[GHRU97] Gupta, Himanshu, Venky Harinarayan, Anand Rajaraman und Jeffrey D. Ullman: *Index Selection for OLAP*. In: Gray, Alex und Per-Åke Larson (Herausgeber): *Proceedings of the Thirteenth International Conference on Data Engineering, April 1997, Birmingham (England)*, Seiten 208–219. IEEE Computer Society, 1997.

[GL95] Griffin, Timothy und Leonid Libkin: *Incremental Maintenance of Views with Duplicates.* In: Carey, Michael J. und Donovan A. Schneider (Herausgeber): *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, Mai 1995, San Jose (Kalifornien, USA)*, Seiten 328–339. ACM Press, 1995.

[GL97] Gyssens, Marc und Laks V. S. Lakshmanan: *A Foundation for Multi–dimensional Databases.* In: Jarke, Matthias, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos und Manfred A. Jeusfeld (Herausgeber): *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 1997, Athen (Griechenland)*, Seiten 106–115. Morgan Kaufmann, 1997.

[Gli00] Glinz, Martin: *Vorlesungsskript Requirements Engineering.* Institut für Informatik, Universität Zürich, Zürich (Schweiz), Oktober 2000.

[Glu96] Gluchowski, Peter: *Modellierung multidimensionaler Datenstrukturen.* Folien, DWE'96 Data Warehousing Europe, München, Juni 1996.

[GM99] Gupta, Himanshu und Inderpal Singh Mumick: *Selection of Views to Materialize Under a Maintenance Cost Constraint.* In: Beeri, Catriel und Peter Buneman (Herausgeber): *Database Theory - ICDT '99, 7th International Conference, Januar 1999, Jerusalem (Israel), Proceedings*, Band 1540 der Reihe *Lecture Notes in Computer Science*, Seiten 453–470. Springer, 1999.

[GMR98a] Golfarelli, Matteo, Dario Maio und Stefano Rizzi: *Conceptual Design of Data Warehouses from E/R Schemes.* In: *CIKM '98, Proceedings of the of Hawaii International Conference On System Sciences, Januar 1998, Maui (Hawai, USA)*, Seiten 81–88, 1998.

[GMR98b] Golfarelli, Matteo, Dario Maio und Stefano Rizzi: *The Dimensional Fact Model: A Conceptual Model for Data Warehouses.* International Journal of Cooperative Information Systems (IJCIS), 7(2–3):215–247, Juni/September 1998.

[GMR00] Golfarelli, Matteo, Dario Maio und Stefano Rizzi: *Applying Vertical Fragmentation Techniques in Logical Design of Multidimensional Databases.* In: Kambayashi, Yahiko, Mukesh K. Mohania und A. Min Tjoa (Herausgeber): *Data Warehousing and Knowledge Discovery, Second International Conference, DaWaK 2000, September 2000, London (England), Proceedings*, Seiten 11–23, London (England), 2000.

[GMRR01] Gupta, Ashish, Inderpal Singh Mumick, Jun Rao und Kenneth A. Ross: *Adapting Materialized Views after Redefinitions: Techniques and a Performance Study.* Information Systems, 26(5):323–362, 2001.

[GMS93] Gupta, Ashish, Inderpal Singh Mumick und V. S. Subrahmanian: *Maintaining Views Incrementally.* In: Buneman, Peter und Sushil Jajodia (Herausgeber): *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Mai 1993, Washington, D.C. (USA)*, Seiten 157–166. ACM Press, 1993.

[Gün00] Günzel, Holger: *Data Warehouse – Eine Basis für Data Mining?* Datenbank–Rundbrief, 1(25):6–10, April 2000.

[GR98] Golfarelli, Matteo und Stefano Rizzi: *Methodological Framework for Data Warehouse Design.* In: *DOLAP '98, ACM First International Workshop on Data Warehousing and OLAP, November 1998, Bethesda (Maryland, USA), Proceedings*, Seiten 3–9. ACM Press, 1998.

[Gup97] Gupta, Himanshu: *Selection of Views to Materialize in a Data Warehouse.* In: Afrati, Foto N. und Phokion Kolaitis (Herausgeber): *Database Theory – ICDT '97, 6th International Conference, Delphi, (Griechenland), Januar 1997, Proceedings*, Band 1186 der Reihe *Lecture Notes in Computer Science*, Seiten 98–112. Springer, 1997.

[Gut84] Guttman, Antonin: *R–Trees: A Dynamic Index Structure for Spatial Searching.* In: Yormark, Beatrice (Herausgeber): *SIGMOD'84, Proceedings of Annual Meeting, Juni 1984, Boston (Massachusetts, USA)*, Seiten 47–57. ACM Press, 1984.

[HA01] Hinrichs, Holger und Thomas Aden: *An ISO 9001: 2000 Compliant Quality Management System for Data Integration in Data Warehouse Systems.* In: Theodoratos, Dimitri, Joachim Hammer, Manfred A. Jeusfeld und Martin Staudt (Herausgeber): *Proceedings of International Workshop DMDW'01 (Design and Management of Data Warehouses), Juni 2001, Interlaken (Schweiz)*, 2001.

[Har99a] Harren, Arne: *$_m$UML – Einsatz der Unified Modeling Language für das konzeptionelle Design von Data Warehouse-Datenbanken*. In: Gesellschaft für Informatik (Herausgeber): *Proceedings Informatik–Tage 99, November 1998, Bad Schussenried (Deutschland)*, Seiten 99–101, 1999.

[Har99b] Harren, Arne: *Konzeptionelles Data Warehouse–Design.* Diplomarbeit, Universität Oldenburg, Fachbereich Informatik, 1999.

[Her99] Herden, Olaf: *ISMUS - An Integration Strategy for Multidimensional Schemas.* In: Richta, Karel (Herausgeber): *Proceedings of 19th Annual Conference on the Current Trends in Databases and Information Systems (Datasem99), Oktober 1999, Brno (Tschechien)*, 1999.

[Her01a] Herden, Olaf: *Measuring Quality of Database Schema by Reviewing – Concept, Criteria, Tool*. In: Fernando Brito and Brian Henderson-Sellers and Mario Pattini and Geert Poels and Houari A. Sahraoui (Herausgeber): *5th International ECOOP Workshop on Quantitative Approaches in Object–Oriented Software Engineering (QAOOSE), Juni 2001, Budapest (Ungarn), Proceedings*, Seiten 59–70. Wettelijk Depot, 2001.

[Her01b] Herden, Olaf: *ODAWA – Physisches Datenbank–Design.* Technischer Bericht OFFIS, Verfügbar unter http://odawa.offis.uni-oldenburg.de, Oldenburg (Deutschland), Juli 2001.

[Her01c] Herden, Olaf: *ODAWA – Transformationsbeschreibung MML nach REMUS*. Technischer Bericht OFFIS, Verfügbar unter http://odawa.offis.uni-oldenburg.de, Oldenburg (Deutschland), Juni 2001.

[HH99] Harren, Arne und Olaf Herden: *MML und $_m$UML – Sprache und Werkzeug zur Unterstützung des konzeptionellen Data Warehouse–Designs.* In: (Herausgeber): *2. Workshop Data Mining und Data Warehousing als Grundlage moderner entscheidungsunterstützender Systeme (DMDW99), September 1999, Magdeburg (Deutschland), Proceedings,* Seiten 57–67, 1999.

[HHM01] Herden, Olaf, Jens Happe und Jürgen Meister: *ODAWA – Benutzerhandbuch*. Technischer Bericht OFFIS, Verfügbar unter http://odawa.offis.uni-oldenburg.de, Oldenburg (Deutschland), Erscheint Ende 2001.

[Hin00] Hinrichs, Holger: *Statistical Quality Control of Warehouse Data*. In: Albertas Caplinskas (Herausgeber): *Proceedings of the 4th IEEE Internatl. Baltic Workshop (Baltic DB&IS 2000), Mai 2000, Vilnius (Litauen),* Seiten 125–139, 2000.

[Hin01] Hinrichs, Holger: *Datenqualitätsmanagement in Data Warehouse–Umgebungen.* In: Heuer, Andreas, Frank Leymann und Denny Priebe (Herausgeber): *Datenbanksysteme in Büro, Technik und Wissenschaft (BTW'01), Proceedings, März 2001, Oldenburg (Deutschland),* Seiten 187–206, Oldenburg (Deutschland), 2001.

[HK99] Holten, Roland und Ralf Knackstedt: *Fachkonzeptuelle Modellierung von Führungsinformationssystemen am Beispiel eines filialisierenden Handelsunternehmens.* In: Sinz, Elmar J. (Herausgeber): *Proceedings der MobiS–Fachtagung, Oktober 1999, Bamberg (Deutschland),* 1999.

[HK01] Herden, Olaf und Joachim Kieschke: *ODAWA – Evaluation.* Technischer Bericht OFFIS, Verfügbar unter http://odawa.offis.uni-oldenburg.de, Oldenburg (Deutschland), Juni 2001.

[HLV00] Hüsemann, Bodo, Jens Lechtenbörger und Gottfried Vossen: *Conceptual Data Warehouse Design.* In: Jeusfeld, Manfred A., H. Shu, Martin Staudt und Gottfried Vossen (Herausgeber): *Proceedings of International Workshop DMDW'00 (Design and Management of Data Warehouses), Juni 2000, Stockholm (Schweden),* 2000.

[Hol99] Holten, Roland: *A Framework for Information Warehouse Development Process.* In: *Arbeitsberichte des Instituts für Wirtschaftsinformatik, Nr. 67,* Münster (Deutschland), Mai 1999.

[Hol00] Holten, Roland: *Entwicklung einer Modellierungstechnik für Data Warehouse–Fachkonzepte.* In: Schmidt, Herrad (Herausgeber): *Proceedings der MobiS–Fachtagung 2000, Oktober 2000, Siegen (Deutschland),* 2000.

[HRU96] Harinarayan, Venky, Anand Rajaraman und Jeffrey D. Ullman: *Implementing Data Cubes Efficiently.* In: Jagadish, H. V. und Inderpal Singh Mumick (Herausgeber): *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Juni 1996, Montreal (Quebec, Kanada),* Seiten 205–216. ACM Press, 1996.

[HS00] Heuer, Andreas und Gunter Saake: *Datenbanken – Konzepte und Sprachen.* International Thomsen Publishing, Bonn (Deutschland), 2. Auflage, 2000.

[HSB00a] Hahn, Karl, Carsten Sapia und Markus Blaschka: *Automatically Generating OLAP Schemata from Conceptual Graphical Models.* In: *DOLAP '00, ACM Third International Workshop on Data Warehousing and OLAP, November 2000, Washington DC (USA), Proceedings,* Seiten 22–27. ACM Press, 2000.

[HSB00b] Hahn, Karl, Carsten Sapia und Markus Blaschka: *Automatically Generating OLAP Schemata from Conceptual Graphical Models.* Technischer BerichtFORWISS Technical Report FR-2000-002, FORWISS, München (Deutschland), Oktober 2000.

[Huy00] Huyn, Nam: *Speeding up View Maintenance Using Cheap Filters at the Warehouse.* In: Wang, Xiaoyang Sean, Ge Yu und Hongjun Lu (Herausgeber): *Proceedings of the 16th International Conference on Data Engineering, Februar/März 2000, San Diego (Kalifornien, USA)*. IEEE Computer Society, 2000.

[IEE83] IEEE: *IEEE Standard Glossary of Software Engineering Terminology*, 1983.

[Inm96] Inmon, William H.: *Building the Data Warehouse.* John Wiley & Sons, Inc., 2. Auflage, 1996.

[JJQV99] Jarke, Matthias, Manfred A. Jeusfeld, Christoph Quix und Panos Vassiliadis: *Architecture and Quality in Data Warehouses: An Extended Repository Approach*. Information Systems, 24(3):229–253, 1999.

[JL99] Jürgens, Marcus und Hans-J. Lenz: *Tree Based Indexes vs. Bitmap Indexes – a Performance Study*. In: Gatziu, Stella, Manfred A. Jeusfeld, Martin Staudt und Yannis Vassiliou (Herausgeber): *Proceedings of International Workshop DMDW'99 (Design and Management of Data Warehouses), Juni 1999, Heidelberg (Deutschland)*, 1999.

[JLVV00] Jarke, Matthias, Maurizio Lenzerini, Yannis Vassiliou und Panos Vassiliadis: *Fundamentals of Data Warehouses.* Springer–Verlag, Berlin, Heidelberg (Deutschland), 2000.

[Jon99] Jones, Richard M.: *Introduction to MFC Programming with Visual C++*. Microsoft Technologie Series, 1999.

[Jos96] Josuttis, Nicolai: *Die C++–Standardbibliothek. Eine detaillierte Einführung in die vollständige ANSI/ISO–Schnittstelle.* Addison–Wesley, 1996.

[JT98] Jaworski, Ramon und Andreas Totok: *Modellierung von multidimensionalen Datenstrukturen mit ADAPT*. Berichte des Instituts für Wirtschaftswissenschaften der Technischen Universität Braunschweig, Juli 1998.

[Kem99] Kemper, Hans-Georg: *Architektur und Gestaltung von Management–Unterstützungs-Systemen.* B. G. Teubner, 1999.

[Ken99] Kenner, Andrea: *Wie viele Dimensionen hat ein Würfel?* Informatik/Informatique, 20(1):3–7, 1999.

[Kim96] Kimball, Ralph: *The Data Warehouse Toolkit.* John Wiley & Sons, Inc., 1996.

[KN99] Küspert, Klaus und Jan Nowitzky: *Partitionierung von Datenbanktabellen.* Informatik Spektrum, 22(2):146–147, April 1999.

[KRRT98] Kimball, Ralph, Laura Reeves, Margy Ross und Warren Thornthwaite: *The Data Warehouse Life Cycle Toolkit.* John Wiley & Sons, Inc., 1998.

[Lec01] Lechtenbörger, Jens: *Data Warehouse Schema Design.* Dissertation, Westfälische Wilhelms–Universität Münster (Deutschland), Fachbereich Mathematik und Informatik, Juni 2001.

[Lei97] Leighton, F.Thomson: *Einführung in parallele Algorithmen und Architekturen - Gitter, Bäume und Hypercubes.* Thomson Publishing International, 1997.

[LH99] Lee, Minsoo und Joachim Hammer: *Speeding up Warehouse Physical Design Using a Randomized Algorithm.* In: Gatziu, Stella, Manfred A. Jeusfeld, Martin Staudt und Yannis Vassiliou (Herausgeber): *Proceedings of International Workshop DMDW'99 (Design and Management of Data Warehouses), Juni 1999, Heidelberg (Deutschland)*, 1999.

[LMSS95] Lu, James J., Guido Moerkotte, Joachim Schü und V. S. Subrahmanian: *Efficient Maintenance of Materialized Mediated Views*. In: Carey, Michael J. und Donovan A. Schneider (Herausgeber): *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, Mai 1995, San Jose (Kalifornien, USA)*, Seiten 340–351. ACM Press, 1995.

[LQA97] Labio, Wilbert Juan, Dallan Quass und Brad Adelberg: *Physical Database Design for Data Warehouses.* In: Gray, Alex und Per–Åke Larson (Herausgeber): *Proceedings of the Thirteenth International Conference on Data Engineering, April 1997, Birmingham (England)*, Seiten 277–288. IEEE Computer Society, 1997.

[LRT96] Lehner, Wolfgang, Thomas Ruf und Michael Teschke: *CROSS–DB: A Feature-Extended Multidimensional Data Model for Statistical and Scientific Databases.* In: *CIKM '96, Proceedings of the Fifth International Conference on Information and Knowledge Management, November 1996, Rockville (Maryland, USA)*, 1996.

[LS97] Lenz, Hans–Joachim und Arie Shoshani: *Summarizability in OLAP and Statistical Data Bases*. In: Ioannidis, Yannis E. und David M. Hansen (Herausgeber): *Ninth International Conference on Scientific and Statistical Database Management, Proceedings, August 1997, Olympia (Washington, USA)*. IEEE Computer Society, 1997.

[LS99] Ling, Tok Wang und Eng Koon Sze: *Materialized View Maintenance Using Version Numbers*. In: Chen, Arbee L. P. und Frederick H. Lochovsky (Herausgeber): *Database Systems for Advanced Applications, Proceedings of the Sixth International Conference on Database Systems for Advanced Applications (DASFAA), April 1999, Hsinchu (Taiwan)*, Seiten 263–270. IEEE Computer Society, 1999.

[LW96] Li, Chang und Xiaoyang Sean Wang: *A Data Model for Supporting On-Line Analytical Processing.* In: *CIKM '96, Proceedings of the Fifth International Conference on Information and Knowledge Management, November 1996, Rockville (Maryland, USA)*, Seiten 81–88. ACM Press, 1996.

[LY01] Liang, Weifa und Jeffrey X. Yu: *Revisit on View Maintenance in Data Warehouses.* In: Wang, Xiaoyang Sean, Ge Yu und Hongjun Lu (Herausgeber): *Advances in Web–Age Information Management, Second International Conference, WAIM 2001, Juli 2001, Xi'an (China), Proceedings*, Nummer 2118 in *LNCS*. Springer, 2001.

[Mar98] Martin, Wolfgang: *Data Warehousing, Data Mining, OLAP*. Thomson Publishing International, Bonn (Deutschland), 1998.

[MBS00] Mertens, Holger, Carsten Bange und Heiko Schinzer: *BARC Studie: Data Warehouse.* BARC (Business Application Research Center), Würzburg (Deutschland), 2000.

[MBS01] Mertens, Holger, Carsten Bange und Heiko Schinzer: *BARC Studie: OLAP and Business Intelligence.* BARC (Business Application Research Center), Würzburg (Deutschland), 2001.

[Met99a] Meta Data Coalition (MDC): *Open Information Model – Version 1.1*. Meta Data Coalition, Homepage http://www.mdc.com, August 1999.

[Met99b] Meta Group: *1999 Data Warehouse Marketing Trends/Opportunities.* Studie, Meta Group, 1999.

[Met00] Meta Group: *SPEX Business Intelligence: EIS/DSS and Query Tools 2000.* Studie, Meta Group, 2000.

[MHH01] Meister, Jürgen, Olaf Herden und Jens Happe: *ODAWA – Dokumentation der Implementierung.* Technischer Bericht OFFIS, Verfügbar unter http://odawa.offis.uni-oldenburg.de, Oldenburg (Deutschland), Erscheint Ende 2001.

[Mic91] Michalewicz, Zbigniew (Herausgeber): *Statistical and Scientific Datatbases.* Ellis Horwood Series in Computers and Their Applications, 1991.

[Mic95] Microstrategy, Inc.: *The Case for Relational OLAP*. White Paper, 1995.

[Mic99] MicroStrategy, Inc.: *Microstrategy Handbuch,* 1999.

[MK98] Muto, Seigo und Masaru Kitsuregawa: *Improving Main Memory Utilization for Array-Based DataCube Computation.* In: *DOLAP '98, ACM First International Workshop on Data Warehousing and OLAP, November 1998, Bethesda (Maryland, USA), Proceedings*, Seiten 28–33. ACM Press, 1998.

[MQM97] Mumick, Inderpal Singh, Dallan Quass und Barinderpal Singh Mumick: *Maintenance of Data Cubes and Summary Tables in a Warehouse.* In: Joan Peckham (Herausgeber): *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, Mai 1997, Tucson (Arizona, USA)*, Seiten 100–111. ACM Press, 1997.

[MS94] Moody, Daniel L. und Graeme G. Shanks: *What Makes a Good Data Model? Evaluating the Quality of Entity Relationship Models.* In: Pericles Loucopoulos (Herausgeber): *Proceedings 13th Intl. Conference on Entity Relationship Approach (ER'94)*, Seiten 94–111, Manchester (England), Dezember 1994. Springer.

[MWM99] Munneke, Derek, Kirsten Wahlstrom und Mukesh K. Mohania: *Fragmentation of Multidimensional Databases.* In: Roddick, John F. (Herausgeber): *Database Systems 99, Proceedings of the 10th Australasian Database Conference, ADC '99, Januar 1999, Auckland (Neuseeland)*, Band 21 der Reihe *Australian Computer Science Communications*, Seiten 153–164. Springer, 1999.

[OAE00] O'Gorman, Kevin, Divyakant Agrawal und Amr El Abbadi: *On the Importance of Tuning in Incremental View Maintenance: An Experience Case Study.* In: Kambayashi, Yahiko, Mukesh K. Mohania und A. Min Tjoa (Herausgeber): *Data Warehousing and Knowledge Discovery, Second International Conference, DaWaK 2000, September 2000, London (England), Proceedings*, Nummer 1874 in *LNCS*, Seiten 77–82. Springer, 2000.

[Obj01] Object Management Group (OMG): *Common Warehouse Meta Model (CWM) Specification – Version 1.0.* Object Management Group, Homepage http://www.omg.org, Februar 2001.

[Ora01] Oracle Inc.: *Homepage Firma Oracle.* http://www.oracle.com/, 2001.

[OV99] Oszu, M. Tamer und Patrick Valduriez: *Principles of Distributed Database Systems.* Prentice–Hall, Englewood Cliffs, New Jersey (USA), 2. Auflage, 1999.

[Ovu98] Ovum: *Ovum Evaluates: Data Warehousing Tools and Strategies.* Ovum Ltd., London (England), 1998.

[PJ99] Pedersen, Torben Bach und Christian S. Jensen: *Multidimensional Data Modeling for Complex Data.* In: *Proceedings of the 15th International Conference on Data Engineering (ICDE), März 1999, Sydney (Australien)*, Seiten 336–345. IEEE Computer Society, 1999.

[PMR99] Peralta, Veronika, Adriana Marotta und Raul Ruggia: *Designing Data Warehouses Through Schema Transformation Primitives.* In: *Conceptual Modeling – ER'99, 18th International Conference on Conceptual Modeling, Demonstration and Posters ER'99, Proceedings, November 1999, Paris (Frankreich)*, Seiten 13–14, 1999.

[PS94] Pagel, Bernd-Uwe und Hans-Werner Six: *Software Engineering.* Addison–Wesley, Bonn (Deutschland), 1994.

[QW97] Quass, Dallan und Jennifer Widom: *On-Line Warehouse View Maintenance.* In: Peckham, Joan (Herausgeber): *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, Mai 1997, Tucson (Arizona, USA)*, Seiten 393–404. ACM Press, 1997.

[Rah94] Rahm, Erhard: *Mehrrechner–Datenbanksysteme*. Addison–Wesley, Bonn (Deutschland), 1. Auflage, 1994.

[Rah01] Rahm, Erhard: *Vorlesungsskript Data Warehouse.* Institut für Informatik, Universität Leipzig, Leipzig (Deutschland), April 2001.

[Rat97a] Rational Software Corporation und UML Partners: *Object Constraint Language Specification. Version 1.1*. Object Management Group. OMG Document ad/97-08-08, September 1997.

[Rat97b] Rational Software Corporation und UML Partners: *UML Notation Guide. Version 1.1*. Object Management Group. OMG Document ad/97-08-05, September 1997.

[Rat99a] Rational Software Corporation: *The Unified Modeling Language, Version 1.3*, Juni 1999.

[Rat99b] Rational Software Corporation: *UML Metamodel Abstract Syntax v1.3 R20*, Januar 1999.

[Rau97] Rautenstrauch, Claus: *Modellierung und Implementierung von Data Warehouse Systemen.* In: Scherrer, W. (Herausgeber): *Vortragsband zur 10.Jahrestagung der Deutschen Oracle Anwender–Konferenz 1997 (DOAG97), Fellbach (Deutschland), November 1997*, 1997.

[RG94] Reingruber, Michael und William W. Gregory: *The Data Modeling Handbook: A Best–Practice Approach to Building Quality Data Models.* John Wiley & Sons, Inc., 1994.

[Ris92] Rishe, Naphtali: *Database Design: The Semantic Modeling Approach.* Mc Graw–Hill, 1992.

[Ris93] Rishe, Naphtali: *A Methodology and Tool for Top–Down Relational Database Design.* Data and Knowledge Engineering, 10(1):259–291, 1993.

[RL85] Roussopoulos, Nick und Daniel Leifker: *Direct Spatial Search on Pictorial Databases Using Packed R–Trees.* In: Navathe, Shamkant B. (Herausgeber): *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data, Mai 1985, Austin (Texas, USA)*, Seiten 17–31. ACM Press, 1985.

[RMF+00] Ramsak, Frank, Volker Markl, Robert Fenk, Martin Zirkel, Klaus Elhardt und Rudolf Bayer: *Integrating the UB-Tree into a Database System Kernel.* In: Abbadi, Amr El, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter und Kyu-Young Whang (Herausgeber): *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 2000, Kairo (Ägypten)*, Seiten 263–272. Morgan Kaufmann, 2000.

[Rou98] Roussopoulos, Nick: *Materialized Views and Data Warehouses.* SIGMOD Record, 27(1):21–26, 1998.

[RS91] Rozen, Steve und Dennis Shasha: *A Framework for Automating Physical Database Design.* In: Lohman, Guy M., Amílcar Sernadas und Rafael Camps (Herausgeber): *17th International Conference on Very Large Data Bases, Proceedings, September 1991, Barcelona (Spanien),* Seiten 401–411. Morgan Kaufmann, 1991.

[RS99] Rautenstrauch, Claus und Andre Scholz: *Vom Performance Tuning zum Software Performance Engineering am Beispiel datenbankbasierter Anwendungssysteme.* Informatik Spektrum, 22(4):261–275, August 1999.

[RSS96] Ross, Kenneth A., Divesh Srivastava und S. Sudarshan: *Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time*. In: Jagadish, H. V. und Inderpal Singh Mumick (Herausgeber): *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Juni 1996, Montreal (Quebec, Kanada),* Seiten 447–458. ACM Press, 1996.

[Ruf97] Ruf, Thomas (Herausgeber): *Scientific & Statistical Datatbases – Datenbankeinsatz in der multidimensionalen Datenanalyse.* Vieweg–Verlag, Braunschweig (Deutschland), 1997.

[Rum93a] Rumbaugh, James: *Disinherited! Examples of misuse of inheritance.* JOOP, 3(2):22–24, Februar 1993.

[Rum93b] Rumbaugh, James: *On the horns of the modeling dilemma: choosing among alternate modeling constructs.* JOOP, 3(11):8–17, November 1993.

[RW99] Rohde, Martin und Frank Wietek: *Das Datenschema für das Epidemiologische Krebsregister Niedersachsen.* Technischer Bericht OFFIS, Oldenburg (Deutschland), 1999.

[Saa96] Saarland, Krebsregister: *Morbidität und Mortalität an bösartigen Neubildungen im Saarland.* Statistisches Landesamt Saarland, Saarbrücken (Deutschland), 1996.

[SAP97] SAP AG: *Business Information Warehouse – Technologie.* SAP AG, Walldorf (Deutschland), 1997.

[Sap99] Sapia, Carsten: *On Modeling and Predicting Query Behavior in OLAP Systems.* In: Gatziu, Stella, Manfred A. Jeusfeld, Martin Staudt und Yannis Vassiliou (Herausgeber): *Proceedings of International Workshop DMDW'99 (Design and Management of Data Warehouses), Juni 1999, Heidelberg (Deutschland),* 1999.

[Sap00a] Sapia, Carsten: *PROMISE: Modeling and Predicting User Query Behaviour in Online Analytical Processing Applications.* Technischer BerichtFORWISS Technical Report FR-2000-001, FORWISS, München (Deutschland), Juni 2000.

[Sap00b] Sapia, Carsten: *PROMISE: Predicting Query Behaviour to Enable Predictive Caching Strategies for OLAP Systems.* In: *DAWAK 2000, 2nd International Conference on Data Warehousing and Knowledge Discovery (DAWAK 2000), September 2000, Greenwich (England), Proceedings.* Springer LNCS, 2000.

[SBH00] Sapia, Carsten, Markus Blaschka und Gabriele Höfling: *GraMMi: The Design and Implementation of a Generic Metadata-driven Graphical Modeling Tool*. In: *Proceedings of the of 33rd Hawaii International Conference On System Sciences (HICSS–33), Januar 2000, Maui (Hawai, USA),* Seiten 81–88, 2000.

[SBHD98a] Sapia, Carsten, Markus Blaschka, Gabriele Höfling und Barbara Dinter: *An Overview of Multidimensional Data Models for OLAP*. Technischer BerichtFORWISS Technical Report FR-1998-001, FORWISS, München (Deutschland), Januar 1998.

[SBHD98b] Sapia, Carsten, Markus Blaschka, Gabriele Höfling und Barbara Dinter: *Extending the E/R Model for the Multidimensional Paradigm.* In: *Proceedings of the International Workshop on Data Warehouse and Data Mining, November 1998, Singapur*, 1998.

[SBM99] Stonebraker, Michael, Paul Brown und Dorothy Moore: *Object–relational DBMSs — Tracking the Next Great Wave*. Morgan Kaufmann, 2 Auflage, Oktober 1999.

[Sch99] Schütte, Reinhard: *Vergleich alternativer Ansätze zur Bewertung der Informationsmodellqualität.* IS Architekturen, 5:39–48, Oktober 1999.

[Sin88] Sinz, Elmar J.: *Das Strukturierte Entity–Relationship Model (SER–Modell).* Angewandte Informatik, 30(5):191–202, 1988.

[Sno95] Snodgrass, Richard (Herausgeber): *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers, Boston (USA), Dordrecht (Niederlande), London (Großbritannien), 1995.

[Som00] Sommerville, Ian: *Software Engineering.* Addison–Wesley, 2000.

[SRF87] Sellis, Timos K., Nick Roussopoulos und Christos Faloutsos: *The R+–Tree: A Dynamic Index for Multi-Dimensional Objects.* In: Stocker, Peter M., William Kent und Peter Hammersley (Herausgeber): *VLDB'87, Proceedings of 13th International Conference on Very Large Data Bases, September 1987, Brighton (England)*, Seiten 507–518. Morgan Kaufmann, 1987.

[SS00] Scholz, Andre und Andreas Schmietendorf: *Aspekte des Performance Engineering – Aufgaben und Inhalte.* In: Dumke, Rainer, Claus Rautenstrauch, Andreas Schmietendorf und Andre Scholz (Herausgeber): *Tagungsband des 1. Workshops Performance Engineering in der Softwareentwicklung (PE2000), Mai 2000, Darmstadt (Deutschland)*, Seiten 33–40, 2000.

[SSL01] Seemann, Achim, Bernd Schmalzridt und Peter Lehmann (Herausgeber): *SAP Business Information Warehouse.* Galileo Press, Bonn (Deutschland), 2001.

[Ste00] Stewart, Nancy: *Data Warehousing and Business Intelligence Market Forecast 2001–2005*. Studie, Firma Survey.com, 2000.

[TB88] Tompa, Frank Wm. und Jose A. Blakeley: *Maintaining Materialized Views Without Accessing Base Data.* Information Systems, 13(4):393–406, 1988.

[TBC99] Tryfona, Nectaria, Frank Busborg und Jens G. Borch Christiansen: *starER: A Conceptual Model for Data Warehouse Design.* In: *DOLAP '99, ACM Second International Workshop on Data Warehousing and OLAP, November 1999, Kansas City (Missouri, USA), Proceedings*, Seiten 3–8. ACM Press, 1999.

[Teo90] Teorey, Toby J.: *Database Modeling and Design: The Entity–Relationship Approach*. Morgan Kaufmann, 1990.

[Tha00] Thalheim, Bernhard: *Entity–Relationship Modeling.* Springer, 2000.

[The01] Theodoratos, Dimitri: *Detecting Redundant Materialized Views in Data Warehouse Evolution.* Information Systems, 26(5):363–381, 2001.

[Tho97] Thomsen, Erik: *OLAP Solutions – Building Multidimensional Information Systems.* John Wiley & Sons, Inc., 1. Auflage, 1997.

[TKS01] Tsois, Aris, Nikos Karayannidis und Timos Sellis: *MAC : Conceptual Data Modeling for OLAP*. In: Theodoratos, Dimitri, Joachim Hammer, Manfred A. Jeusfeld und Martin Staudt (Herausgeber): *Proceedings of International Workshop DMDW'01 (Design and Management of Data Warehouses), Juni 2001, Interlaken (Schweiz)*, 2001.

[TM75] Trembly, J.P. und R. Manohar: *Discrete Mathematical Structures with Applications to Computer Science.* McGraw Hill Book Company, New York (USA), 1975.

[TS97] Theodoratos, Dimitri und Timos K. Sellis: *Data Warehouse Configuration.* In: Jarke, Matthias, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos und Manfred A. Jeusfeld (Herausgeber): *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 1997, Athen (Griechenland)*, Seiten 126–135. Morgan Kaufmann, 1997.

[Tur96] Turau, Volker: *Algorithmische Graphentheorie.* Oldenbourg–Verlag, München (Deutschland), Dezember 1996.

[URT99] Uchiyama, Hidetoshi, Kanda Runapongsa und Toby J. Teorey: *A Progressive View Materialization Algorithm.* In: *DOLAP '99, ACM Second International Workshop on Data Warehousing and OLAP, November 1999, Kansas City (Missouri, USA), Proceedings.* ACM Press, 1999.

[VGD99] Vavouras, Athanasios, Stella Gatziu und Klaus R. Dittrich: *The SIRIUS Approach for Refreshing Data Warehouses Incrementally.* In: Buchmann, Alejandro P. (Herausgeber): *Datenbanksysteme in Büro, Technik und Wissenschaft (BTW'99), Proceedings, März 1999, Freiburg (Deutschland)*, Seiten 80–96, 1999.

[VVS00] Vetterli, Thomas, Anca Vaduva und Martin Staudt: *Metadata Standards for Data Warehousing: Open Information Model vs. Common Warehouse Metamodel.* SIGMOD Record, 29(3):68–75, 2000.

[WB98] Wu, Ming-Chuan und Alejandro P. Buchmann: *Encoded Bitmap Indexing for Data Warehouses.* In: *Proceedings of the Fourteenth International Conference on Data Engineering, Februar 1998, Orlando (Florida, USA)*, Seiten 220–230. IEEE Computer Society, 1998.

[Wed74] Wedekind, Hartmut: *On the Selection of Access Paths in a Data Base System. Database Management.* North–Holland, 1974.

[Wes00] Westerman, Paul: *Data Warehousing: Using the Wal–Mart Model.* Morgan Kaufmann, 1. Auflage, 2000.

[Wie00] Wietek, Frank: *Intelligente Analyse multidimensionaler Daten in einer visuellen Programmierumgebung und deren Anwendung in der Krebsepidemiologie.* Dissertation, Universität Oldenburg (Deutschland), Fachbereich Informatik, Verfügbar unter http://docserver.bis.uni-oldenburg.de/publikationen/dissertation/2000/wieint00/wieint00.html, März 2000.

[Win99] Winter, Robert: *Data Warehousing Beyond Tools and Data: Justification, Organization, and Structured Development of Data Warehousing Applications.* In: *Proceedings of 3rd Internatl. Conference on Business Information Systems(BIS'99), April 1999, Posen (Polen)*, Seiten 125–134, 1999.

[Woo99] Wood, Chuck: *OLE DB and ODBC Developer's Guide.* IDG Books Worldwide, 1999.

[Wor01] World Wide Web Consortium (W3C): *Sematic Web*. http://www.w3.org/2001/sw/, September 2001.

[YG98] Yellen, Jay und Jonathan L. Gross: *Graph Theory & Its Applications.* CRC Press, Dezember 1998.

[YKL97] Yang, Jian, Kamalakar Karlapalem und Qing Li: *Algorithms for Materialized View Design in Data Warehousing Environment.* In: Jarke, Matthias, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos und Manfred A. Jeusfeld (Herausgeber): *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 1997, Athen (Griechenland)*, Seiten 136–145. Morgan Kaufmann, August 1997.

[ZDN97] Zhao, Yihong, Prasad Deshpande und Jeffrey F. Naughton: *An Array–Based Algorithm for Simultaneous Multidimensional Aggregates.* In: Peckham, Joan (Herausgeber): *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, Mai 1997, Tucson (Arizona, USA)*, Seiten 159–170. ACM Press, 1997.

[ZGHW95] Zhuge, Yue, Hector Garcia–Molina, Joachim Hammer und Jennifer Widom: *View Maintenance in a Warehousing Environment.* In: Carey, Michael J. und Donovan A. Schneider (Herausgeber): *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, Mai 1995, San Jose (Kalifornien, USA)*, Seiten 316–327. ACM Press, 1995.

[ZS99] Zurek, Thomas und Markus Sinnwell: *Data Warehousing Has More Colours Than Just Black & White.* In: *Proceedings of the Industrial Session of VLDB'99, September 1999, Edinburgh (Schottland)*, 1999.

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| ACM | Association for Computing Machinery |
| ADAPT | Application Design for Analytical Processing Technologies |
| ADO | Access Data Objects |
| BARC | Business Application Research Center |
| CACM | Communications of the ACM |
| CASE | Computer Aided Software Engineering |
| COLAP | Client OLAP |
| CWM | Common Warehouse Metamodel |
| DAG | Directed Acyclic Graph |
| DB | Datenbank |
| DBS | Datenbanksystem |
| DBMS | Datenbank Managementsystem |
| DFM | Dimensional Fact Model |
| DNF | Dimensional Normal Form |
| DOLAP | Desktop OLAP |
| DWH | Data Warehouse |
| DWS | Data Warehouse System |
| EKN | Epidemiologische Krebsregister Niedersachsen |
| EPK | Ereignisorientierte Prozessketten |
| ERM | Entity Relationship–Modell |
| ETL | Extraktion Transformation Laden |
| FORWISS | Bayrisches Forschungszentrum für wissensbasierte Systeme |
| GIS | Geographisches Informationssystem |
| GMNF | Generalized Multidimensional Normal Form |
| GoM | Grundsätze ordnungsgemäßer Modellierung |
| GUI | Graphische Benutzungsschnittstelle |
| HOLAP | Hybrid OLAP |
| HTML | Hypertext Markup Language |
| IBM | International Business Machines |
| IEEE | Institute of Electrical and Electronic Engineers |
| IS | Informationssystem |

| | |
|---|---|
| LCD of SQL | Lowest Common Denominator of SQL |
| LNCS | Lecture Notes in Computer Science |
| MAC | Multidimensional Aggregation Cube |
| MADEIRA | Modelling Analyses of Data in Epidemiological InteRActive Studies |
| MDB | Multidimensionale DB |
| MDBMS | Multidimensionales DBMS |
| MDBS | Multidimensionales DBS |
| MDC | Meta Data Coalition |
| MERM | Multidimensional Entity Relationship–Modell |
| MFC | Microsoft Foundation Classes |
| MML | Multidimensional Modeling Language |
| MNF | Multidimensional Normal Form |
| MOLAP | Multidimensional OLAP |
| $_m$UML | Multidimensional UML |
| OCL | Object Constraint Language |
| ODAWA | OFFIS Tools for Data Warehousing |
| ODBC | Open Database Connectivity |
| ODS | Operational Data Store |
| OFFIS | Oldenburger Forschungs– und Entwicklungsinstitut für Informatik–Werkzeuge und –Systeme |
| OID | Objektidentität |
| OIM | Open Information Model |
| OLAP | On–Line Analytical Processing |
| OLE | Object Linking and Embedding |
| OLTP | On–Line Transaction Processing |
| OMG | Object Management Group |
| QS | Qualitätssicherung |
| RDBMS | Relational DBMS |
| REMUS | Relational Schema for Multidimensional Purpose |
| ROLAP | Relational OLAP |
| SERM | Semantisches Entity Relationship–Modell |
| SIGMOD | Special Interest Group on Managament of Data |
| SQL | Structured Query Language |
| TODS | Transactions on Database Systems |
| UML | Unified Modeling Language |
| VLDB | Very Large Database |
| W3C | World Wide Web Consortium |
| XML | Extensible Markup Language |

# Index

# Lebenslauf

**Persönliche Daten**

| | |
|---|---|
| Name: | Olaf Herden |
| Geburtsdatum: | 20.09.1968 |
| Geburtsort: | Nordenham |
| Wohnort: | Beowulfsweg 3<br>26131 Oldenburg |
| Familienstand: | ledig |
| Staatsangehörigkeit: | deutsch |

**Schulbesuch**

| | |
|---|---|
| 1975 - 1979 | Besuch der Grundschule Süd in Nordenham |
| 1979 - 1981 | Besuch der Orientierungsstufe Süd in Nordenham |
| 1981 - 1988 | Besuch des Gymnasiums in Nordenham<br>Abschluss: Allgemeine Hochschulreife |

**Wehrdienst**

| | |
|---|---|
| 1988 - 1989 | Wehrdienst in Oldenburg |

**Studium**

| | |
|---|---|
| 1989 - 1996 | Studium der Informatik mit Nebenfach Betriebswirtschaftslehre an der Carl von Ossietzky Universität Oldenburg<br>Vertiefungsfach: Praktische Informatik<br>Abschluss: Diplom-Informatiker |

**Berufstätigkeit**

| | |
|---|---|
| seit 1996 | Wissenschaftlicher Mitarbeiter im FuE-Bereich "Betriebliches Informations- und Wissensmanagement" am Oldenburger Forschungs- und Entwicklungsinstitut für Informatik-Werkzeuge und -Systeme (OFFIS) bei Prof. Dr. H.-J. Appelrath |
| Dezember 2001 | Promotion zum Doktor der Ingenieurswissenschaften am Fachbereich Informatik der Carl von Ossietzky Universität Oldenburg |

Oldenburg, den 16.2.2002