Active Data Warehouses: Complementing OLAP with Analysis Rules

DISSERTATION
zur Erlangung des akademischen Grades
eines Doktors der Sozial- und Wirtschaftswissenschaften

Eingereicht an der Johannes Kepler Universität Linz
Institut für Wirtschaftsinformatik
Data & Knowledge Engineering

Betreuung:
o.Univ.-Prof. Dipl.-Ing. Dr. Michael Schrefl
a.Univ.-Prof. Mag. Dr. Werner Retschitzegger

Verfaßt von: **Mag. Thomas Thalhammer**

Linz, im November 2001

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, daß ich die vorliegende Dissertation selbständig verfaßt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe. Diese Dissertation habe ich bisher weder im Inland noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt.

Linz, am 5. November 2001

Danksagung

An dieser Stelle möchte ich mich bei all jenen bedanken, die zum Gelingen dieser Arbeit beigetragen haben. Zuallererst sei Michael Schrefl erwähnt, der mit seinem unerschöpflichen Ideenreichtum und einem schier unendlichen Vorrat an Energie in zahllosen Diskussionen einen ganz wesentlichen Beitrag zum Erfolg dieser Arbeit geleistet hat. Bei Prof. Werner Retschitzegger bedanke ich mich für die Übernahme der Zweitbegutachtung.

Weiters möchte ich mich bei allen Kolleginnen und Kollegen am Institut für Wirtschaftsinformatik der Universität Linz (Forschungsschwerpunkt Data & Knowledge Engineering), Ingeborg, Margit, Martin, Günter, Stephan und Roman, für das sehr angenehme Umfeld, in dem diese Arbeit entstehen konnte, bedanken.

Besonderer Dank gebührt meinen Eltern Brigitte und Johann, die den Grundstein für meine Ausbildung gelegt haben. Sie haben mich über all die Jahre hinweg unterstützt und gefördert und mich in meinen Entscheidungen bekräftigt. Ein liebevolles Dankeschön gilt Doris, die in den letzten Monaten für mich und meine Arbeit soviel Verständnis gezeigt hat.

Thomas Thalhammer

Kurzfassung

In den vergangenen Jahren wurden *Ereignis-Bedingungs-Aktions-Regeln* erfolgreich in der Entwicklung von transaktionsverarbeitenden Systemen (z.B., Lagerhaltungssysteme) verwendet. Solche Regeln basieren auf der Idee, daß beim Eintritt eines Ereignisses eine spezifizierte Bedingung geprüft wird und – falls diese Bedingung erfüllt ist – eine entsprechende Aktion ausgeführt wird ohne daß dafür eine Benutzerinteraktion erforderlich ist. Der Anwendungsbereich dieser Regeln erstreckt sich von der Erledigung systemspezifischer Aufgaben (z.B., Sicherstellung der Integrität von Datenbanken) bis zur Erledigung applikationsspezifischer Aufgaben (z.B., Realisierung von Geschäftsregeln eines Anwendungsprogrammes) in transaktionsverarbeitenden Systemen. Obwohl Abläufe (z.B., standardisierte Analysen) innerhalb von Entscheidungsunterstützungssystemen (Data Warehouses und OLAP-Systeme) ebenfalls mit Ereignis-Bedingungs-Aktions-Regeln automatisiert werden können, wurden diese dort bisher nicht eingesetzt.

Diese Dissertation beschäftigt sich mit der automatischen Ausführung von strukturierten und halbstrukturierten Entscheidungsprozessen innerhalb von Data Warehouses unter Verwendung von Ereignis-Bedingungs-Aktions-Regeln. Solche Data Warehouses werden *Aktive Data Warehouses* genannt, da gemäß den spezifizierten Regeln beim Eintritt eines Ereignisses Daten im Data Warehouse analysiert werden und anschließend eine "Entscheidung" getroffen wird. Umgesetzt werden solche Entscheidungen, indem durch das Data Warehouse in einem operativen Datenbanksystem eine Transaktion ausgeführt wird. Der Schwerpunkt solcher Ereignis-Bedingungs-Aktions-Regeln, die fortan *Analyseregeln* genannt werden, liegt (i) in der Nachahmung/Automatisierung des Analyseprozesses eines Endbenutzers (Analyst) und (ii) in der Festlegung, wie Entscheidungen getroffen werden.

Das soeben skizzierte Basismodell für Analyseregeln wird um ein Framework zur Modellierung von komplexen Entscheidungsprozessen erweitert. Die Erweiterungen beziehen sich insbesondere auf (i) die flexible Modellierung von Entscheidungskriterien, (ii) alternative Ansätze zur Komposition von Gesamtentscheidungen aus Teilentscheidungen und (iii) auf die Bestimmung von Aktivitätsparametern. Die Semantiken des Basismodells sowie der genannten Erweiterungen werden deklarativ und prozedural definiert. Weiters wird ein Ansatz zur Implementierung von Analyseregeln des Basismodells mittels Standard-Datenbanktechnologien vorgeschlagen.

Abstract

In the recent years, Event-Condition-Action (ECA) rules have been successfully used for developing on-line transaction processing systems (e.g., warehouse management systems). ECA rules are based upon the principle that when an event occurs and a specified condition is satisfied, then a particular action will be carried out without requiring any interaction with the user. The field of application of ECA rules in on-line transaction processing systems ranges from internal tasks (e.g., ensuring the integrity of databases) to external tasks (e.g., realizing the business rules of a particular application program). Although practically relevant, ECA rules have not been utilized for automatizing various processes (e.g., routine analysis tasks) within decision-support systems (data warehouses and OLAP systems) so far.

This thesis proposes an approach to carry out routine decision tasks and semi-routine decision tasks automatically within data warehouses using ECA rules. Such systems are called *active data warehouses*, since these systems analyze the data warehouse as reaction to occurred events. The "decisions" that are generated by these rules will be realized by executing transactions in on-line transaction processing systems. The emphasis of such ECA rules, which we call *analysis rules* from now on, is (i) on automatizing analyses originally carried out manually by users (analysts) and (ii) on defining how decisions will be generated.

This basic approach of analysis rules is extended by a framework to specify complex decision tasks. These extensions are (i) flexible modeling of decision criteria, (ii) alternative approaches to specify decision-making models, and (iii) determining the bindings of action parameters. We define the semantics of the basic approach and of the extended approach declaratively and procedurally and propose a basic implementation of analysis rules using off-the-shelf database technology.

Contents

Part I

The Need for Active Data Warehouses

Chapter 1

Introduction

Contents

1.1 Motivation

In the recent years, computer based information systems have been developed to support two kinds of tasks almost every business is faced with: (1) Carrying out the day-to-day business (i.e., operational business processes such as sales, marketing, production, etc.) as efficiently as possible. (2) Making decisions about these entities (i.e., articles, stores, customers, etc.) of these business processes. Information systems that support the first kind of tasks are called *on-line transaction processing (OLTP) systems* [CD97]. Such systems are typically used by business clerks to carry out their day-to-day business tasks. From a technical point of view, these systems are designed to support short and simple transactions (e.g., register the sale of a particular item) that access a small number of records but that may be initiated by a large number of users concurrently. Hence, the underlying databases have been designed to allow for high transaction throughput. Since these databases reflect only a narrow window of "current data", operations like inspecting and analyzing historical data and making predictions are only badly supported. Information systems that support the second kind of tasks are called *on-line analytical processing (OLAP) systems* [CD97]. From a conceptual point of view, these systems are designed to give business analysts (also known as "knowledge workers") insight into historical data since the underlying databases collect long histories of data. Business analysts use front-end *OLAP tools* to manually browse, inspect, and analyze these data. From an implementational point of view, the un-

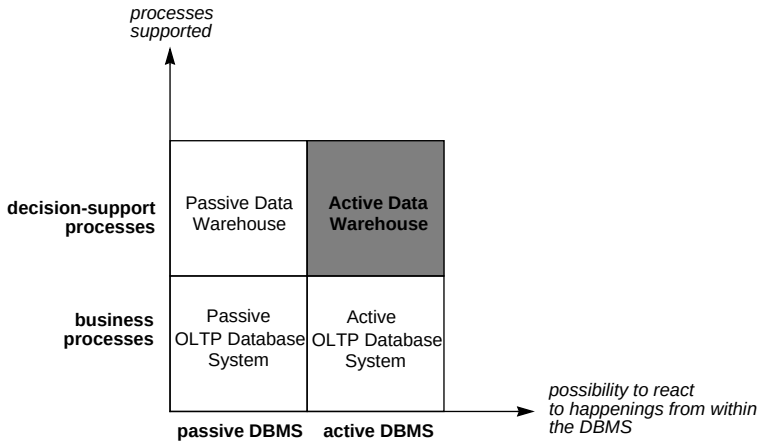

Figure 1.1: Paradigms in Database Systems Development

derlying database systems (the "data warehouses") are designed to answer complex ad-hoc queries very quickly.

Orthogonally to what we described above, *database management systems* have undergone several evolutions in the recent decade. One of these evolutions was the integration of reactive capabilities within database systems. A conventional database systems is *passive*, i.e., every action that may be carried out on data provided by a database system must be explicitly initiated by a user. In other words, the database system could not autonomously react to happenings. With the integration of Event-Condition-Action (ECA) rules in database systems, database designers were able to specify not only data structures and passive behavior but also to specify some "happenings of interest" upon which the database can react autonomously. The principal idea behind ECA rules is that when such a "happening of interest" – the event – occurs (e.g., price of an article changed, etc.) and a specified condition holds, then a particular action should be carried out without asking the user. The expressiveness of an active database system heavily depends on (i) what kinds of events are supported (e.g., DML operation, clock, external, etc.), (ii) what kind of data model is used by the database system (i.e., relational, extended relational, object-relational, object-oriented, deductive), and (iii) what kinds of ECA rules are supported (i.e., global rules, class-specific rules, instance-specific rules).

In the recent years, active database systems have been used especially to automatize various applications [CW96a] supporting operational business processes of OLTP systems such as *internal applications* (e.g., checking integrity constraints, maintaining materialized views, mastering data integration), *extended applications* (e.g., workflow management, extended transaction models), and *external applications* (e.g., business rules). In the domain of decision-support activities, concepts of active database systems remained unconsidered so far. Hence, all activities concerned with analyzing data and making decisions must be explicitly invoked by analysts. An *active data warehouse* (ADW) fills exactly this gap (cf.

Figure 1.1). ECA rules are employed to analyze data collected by the data warehouse and to invoke (or not to invoke) transactions in OLTP systems as their "decisions".

1.2 Overview

A data warehouse (DW) extracts, transforms, and loads (ETL) data from OLTP systems autonomously without user interaction. After the data is loaded, analysts perform interactive analyzes using OLAP tools to find solutions for different kinds of decision tasks. In the case of "non-routine decision tasks" analysts query the data warehouse to create different scenarios for solving a business problem. Decision making is accomplished by choosing the "most effective" scenario or by choosing the scenario with the "fewest side-effects", etc. Such tasks do not occur regularly and/or there are no generally accepted decision criteria. Examples for "non-routine decision tasks" can be found in strategic business management (e.g., the decision to set up a new brand or the decision to change a business policy). In the case of "routine decision tasks", analysts extract information from the data warehouse to solve well-structured problems by applying generally accepted procedures in decision making. Such decision tasks occur frequently at predictive timepoints. Examples can be found in the areas of product assortment (e.g., change price, withdraw product), customer relationship management (e.g., grant a discount to special customers), and in many administrative areas (e.g., accept/reject a paper for a conference). Finally, "semi-routine decision tasks" typically occur when routine problems require non-routine treatment (e.g., if a paper is rated contradictory it is discussed by the program committee before it will be accepted or rejected). Although today's OLAP and DW systems efficiently support ETL processes and interactive data analysis, they do not offer the possibility to automatize analysis processes for "routine decision tasks" and the routinizable elements of "semi-routine decision tasks".

The major contributions of this thesis are as follows:

Architecture. We describe a novel architecture, the *active data warehouse*, which offers the possibility to automatize decision making of routine decision tasks and the routinizable elements of semi-routine decision tasks. An active data warehouse exports decisions automatically back to OLTP systems. In doing so, such systems realize a closed-loop decision approach, which we call *active data warehouse cycle* or *ADW cycle* for short (cf. Figure 1.2).

Data Model. We propose a multidimensional data model that supports specification, definition, and automatic execution of analysis tasks. We further present a mapping of this data model to the relational data model in order to provide a common basis for extending existing relational data warehouses with reactive capabilities.

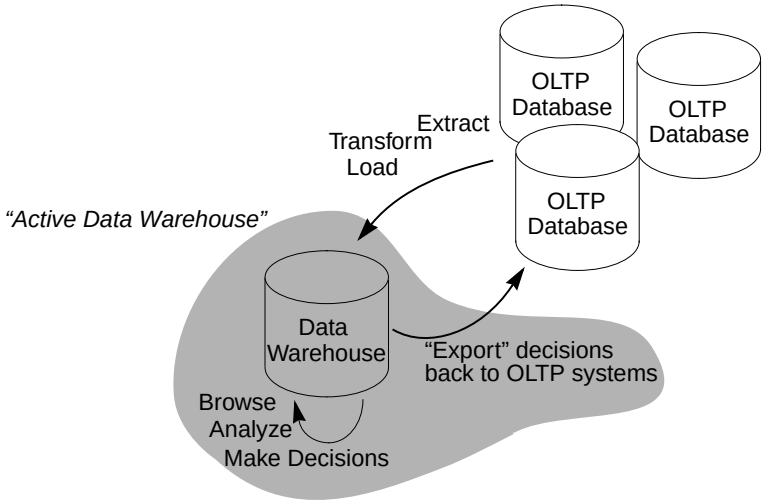

Figure 1.2: Active Data Warehouse and Active Data Warehouse Cycle

Basic Analysis Rules. The emphasis of this thesis is on automatizing decision tasks by adopting the idea of ECA rules from active database systems. ECA rules that mimic the work of an analyst are called *analysis rules*. Analysis rules are modeled from the perspective of an analyst who specifies (1) the timepoints at which the analysis rule has to be "fired", (2) the cubes, which have to be analyzed (the *analysis graph*) by using certain criteria for decision making (the *decision steps*), and (3) the action that will be executed if a decision criteria is satisfied. Although analysis rules are founded on the same principle structure as ECA rules, there are some major differences to conventional ECA systems [PD99, DPW98, FP98, RPS98, CCS94, ZB98, Han96]:

1. Active database systems offer a wide range of events from different sources (e.g., insert/update/delete, behavior invocation, transaction, abstract situations, program exceptions, clock, external calls) [PD99, CM94, GD93]. In an initial approach, an active data warehouse uses a very simple event model to describe the timepoints at which analyzes should be carried out automatically. Analysis rules may be fired at the occurrence of an *absolute temporal event* (e.g., Oct. 15th, 2000), a *periodic temporal event* (e.g., at the end of every quarter), or a *relative temporal event* (e.g., two weeks after a product's price was changed). "Reference point" for relative temporal events are *OLTP method events*, which represent happenings in OLTP systems (e.g., a product's price has been changed). This event model is sufficient to describe the timepoints at which routine decision tasks should be carried out automatically.

2. The action part of an ECA rule may be an arbitrary sequence of insert/update/delete statements, behavior invocations, transactional statements, external calls, or some alternative statements using do-instead [PD99]. An analysis rule's action consists of a single statement, which is the directive to execute a transaction for some entities

in an OLTP database.

3. The most important difference between ECA rules and analysis rules can be found in the role of a rule's condition. The optional condition of an ECA rule is a simple boolean predicate or a query, which prevents the rule from unnecessary action execution. An analysis rule utilizes a set of conditions to perform multidimensional analyses, which were formerly carried out manually by analysts. The conditions of an analysis rule are mandatory. To accomplish the task of analyzing data and making decisions, analysis rules mimic the "top-down" approach an analyst follows in analyzing a certain problem. "Top-down" means that measures for analysis are initially inspected using very coarse grained dimension levels. If a decision cannot be made at the current dimension level, more detailed analysis using finer dimension levels will be carried out. Following this approach, the analyst creates a hierarchy of cubes that is called *analysis graph*. For each cube, the analyst defines a decision step, which has two conditions as its parts. The first condition is used to examine, whether the rule's action can be executed for an entity. The second condition is used to examine, whether analysis should be continued for an entity at subsequent cubes if the first condition is not fulfilled.

Extended Analysis Rules. We propose several extensions to the above sketched basic approach of analysis rules. These extensions concern the decision-making capabilities of analysis rules as follows: (i) logical events (i.e., events that occur when the database arrives at a particular state), (ii) specifying alternative criteria for decision making, (iii) alternative approaches to integrate the decisions of sub-cubes, (iv) specifying the parameter bindings of transactions flexibly.

Rule Implementation. We propose a mapping of basic analysis rules to the relational data model using off-the-shelf database technology (i.e., SQL and triggers) in order to be "compatible" with existing relational data warehouses.

1.3 Outline

The remainder of this thesis is organized in five parts. Part 1 (Chapters 1 - 3) motivates the need for active data warehouses, describes a typical business scenario for which active data warehouses may be the right technology, and explains the technologies that enable active data warehouses. Part 2 (Chapters 4 and 5) presents a basic approach to analysis rules in active data warehouses. This approach may be considered as the "kernel" technology that may be provided by any active data warehouse. Part 3 (Chapters 6 and 7) motivates and presents several extensions to the basic approach. Part 4 (Chapters 8 and 9) presents the implementation of the basic approach. Part 5 (Chapter 10) concludes this thesis and

highlights the most challenging questions that are left up to future work. The following list gives a more detailed description of the successive chapter's contents.

- Chapter 2 sketches a case study from retail trade that will be used to explain the concepts of active data warehouses and analysis rules in the remainder of this thesis.
- Chapter 3 presents the "enabling technologies" (i.e., active database systems, data warehouses, and on-line analytical processing) of active data warehouses. We further sketch a logical architecture of an active data warehouse system.
- Chapter 4 introduces an approach to specify multidimensional data structures and to query multidimensional data as required by analysis rules.
- Chapter 5 presents a basic approach for specifying and evaluating analysis rules. This approach may be considered as the "kernel" of an active data warehouse.
- Chapter 6 motivates several extensions to the basic approach. These extensions concern the decision-making capabilities of analysis rules.
- Chapter 7 introduces the syntax and semantics of these extensions.
- Since many data warehouses are built upon relational database technology, Chapter 8 shows how passive multidimensional data – as specified in Chapter 4 – can be mapped to the relational data model.
- Chapter 9 shows how analysis rules can be realized on top of a relational data warehouse using off-the-shelf database technology.
- Chapter 10 concludes this thesis and discusses challenging questions in the realm of active data warehouses that will be investigated in the future.

Chapter 2

Active Data Warehouses - A Case Study from Retail Trade

This chapter outlines a case study in which an active data warehouse may be used to complement manual OLAP analyzes. This scenario is settled in retail trade, which is a typical area in which active data warehouses may be used to automatize decision making processes.

The Setting. Consider a retail chain consisting of several chain stores that are located in different Austrian cities and a headquarter that maintains the centralized information system infrastructure. This information system infrastructure basically consists of (i) a sales database that supports online processing of sales transactions effected in the various chain stores and (ii) a data warehouse that periodically collects data from the sales database to provide analysts access to long histories of sales figures and offerings. Figure 2.1 depicts the global setting of this retail chain store. In the following we describe the data stored in the sales database and in the data warehouse and we outline the transactions that may be executed in the sales database and the analytical tasks that are carried out using the data from the data warehouse.

The Sales Database. The following object types constitute the sales database:

- *Store:* Each chain store is represented by an instance of object type Store. Chain stores are located in a particular city and belong to a particular region.
- *Article:* An instance of object type Article represents the description (e.g., name, price, producer, category) of an item that may be sold in the chain stores of the retail chain. For each region, the range of articles to be offered is the same in all chain stores of that region.

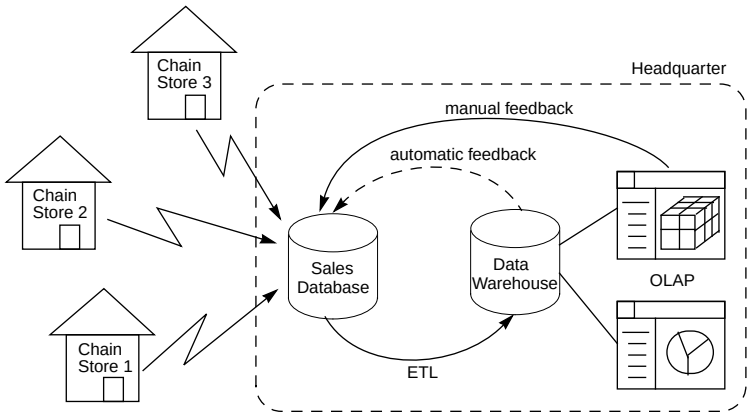

Figure 2.1: Retail Data Warehouse Scenario

- *Sale:* When a particular article is sold in a chain store, the corresponding sales data (article, store, date, quantity, revenue) will be transmitted to the sales database where these data will be registered.

Typical transactions that may be invoked in the sales database for the instances of object type Article are:

- *Register Sale:* To register the sale of a particular article, transaction registerSale may be invoked. This transaction requires the parameters store, date, quantity, and revenue.

- *Cancel Sale:* To cancel the sale of a particular article, transaction cancelSale may be invoked. This transaction requires the parameters store, date, quantity, and revenue.

- *Launch Article:* To offer a particular article in the chain stores of a particular region, transaction launchOnMarket may be invoked[1]. This transaction requires the region as parameter.

- *Withdraw Article:* To withdraw a particular article from the chain stores of a particular region, transaction withdrawFromMarket may be invoked. This transaction requires the region as parameter.

- *Change Price:* To change the price of a particular article, transaction changePrice may be invoked. This transaction requires the ratio (old price/new price) as parameter.

[1]Note that "market" and "region" are treated as synonyms here.

The Data Warehouse. Every night, the data warehouse loads new data from the sales database (depicted by the bold arrow labeled ETL in FIgure 2.1). The data warehouse provides information about *sales* and *offerings* to analysts as follows:

- *Sales:* Sales figures (i.e., quantities and revenues) may be inspected per article, store, and day. Further, articles may be viewed per category or per producer. Stores may be viewed per city and per region. Days may be viewed (i) per month, quarter, and year or alternatively (ii) per week and year.
- *Offerings:* Offering periods are also registered in the data warehouse. Offerings may be inspected per article, region, start date of the offering period, and end date of the offering period.

Using the data in the data warehouse, analysts carry out various analytical tasks, three of which we will explain below:

Scenario 1: *Reducing the price of an article.* Twenty days after a soft drink has been launched on a market, analysts compare the quantities sold during this period with a standardized indicator. This indicator requires the total quantities sold during the twenty-day period not to drop below a threshold of 10000 sold items. If the analyzed sales figures are below this threshold, the price of the newly launched soft drink will be reduced by 15 %. Hence, analysts typically invoke transaction changePrice in the sales database for these articles.

Scenario 2: *Withdrawing articles from a market.* At the end of every quarter, high priced soft drinks that are offered in Upper Austrian stores will be analyzed. If the sales figures of a high priced soft drink have continuously dropped, the article will be withdrawn from the Upper Austrian market. Analysts inspect the sales figures of articles effected in the current quarter per region and city and per quarter and month. Trend, average, and variance measures are used as indicators in decision making. Analysts will invoke transaction withdrawFromRegion in the sales database.

Scenario 3: *Setting up a new chain store.* The decision whether to set up a new chain store in a particular region is a much more complex task than the two aforementioned. Such a decision is primarily derived from strategic business policies. Although having no particular regular schedule and no standardized indicators exist for this kind decision tasks, the data warehouse will be used to support decision making (e.g., using historic sales and offering data from that region).

An active data warehouse may be used to automatize analytical tasks for which well-established decision procedures exist. Hence analysts can specify decision procedures for scenario 1 and scenario 2 once to be carried out automatically by the data warehouse, which

also invokes the corresponding transactions in the sales database (depicted by the dashed arrow labeled "automatic feedback" between the data warehouse and the sales database in Figure 2.1). The analytical tasks of scenario 3 will be still performed manually by analysts since there exists no regular schedule and no standardized indicators for decision making.

In the remainder of this thesis, we will introduce theoretical and practical foundations of active data warehouses that will be exemplified using the above sketched case study.

Chapter 3

Enabling Active Data Warehouses

Contents

This chapter describes the enabling technologies of active data warehouses – *active database systems*, *data warehouses*, and *on-line analytical processing* – and outlines active data warehouses from an architectural point of view.

1. *Active Database Systems.* An active database system can automatically (i.e., without explicit user interaction) react to events. Active database systems support Event-Condition-Action (ECA) rules, which are fired (i.e., the condition is evaluated and –

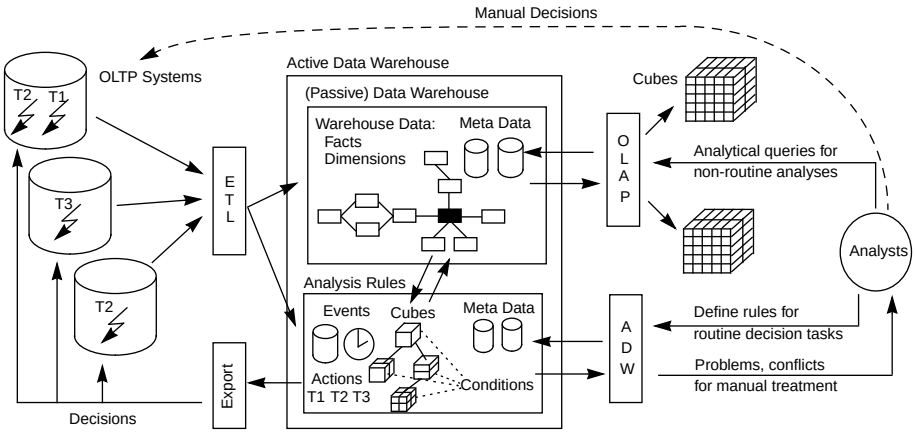

Figure 3.1: Conceptual architecture of Active Data Warehouses

if the condition holds – the action is executed) by the active database system when an instance of a rule's event occurs.

2. *Data Warehouses.* A data warehouse is a decision-support database that provides an integrated view of data loaded from operational (OLTP) systems. In contrast to operational systems, whose underlying data models emphasize the relationships among business entities, data warehouses draw the focus on business entities which are of primer interest in decision making. Such business entities are represented hierarchically as the "dimensions" for analysis. Data in a data warehouse is non-volatile, i.e., data can only be added to the data warehouse but existing data cannot be deleted from the data warehouse. Hence, a data warehouse is time-variant since it records histories of data instead of overriding that data.

3. *On-Line Analytical Processing.* On-line analytical processing belongs to a category of software technology that views the data in the data warehouse from a multidimensional conceptual perspective, which is well-suited for the interaction with domain experts that have no expertise in database systems, i.e., user-analysts. This conceptual perspective also provides conceptual operations for interactive navigation and analysis (i.e., drilldown, rollup, slice & dice) of multidimensional data. OLAP systems come along with different alternative architectures using relational database systems, multidimensional database systems, or an architectural mixture of both as their data repository.

An *active data warehouse* (ADW) integrates the three technologies to automatize decision processes, which are typically carried out by analysts on a repetitive schedule and whose decision criteria are well-established. The "glue" among these technologies are *analysis rules*. They realize the basic ECA rule structure from active database systems, but carry out complex OLAP analyzes on warehouse data instead of evaluating simple conditions

as it is the case with ECA rules in OLTP systems. Figure 3.1 depicts the conceptual architecture of an active data warehouse. An Extract-Transform-Load (ETL) software layer is responsible for loading data from operational (OLTP) systems into the passive data warehouse repository. Further, basic events that occurred in OLTP systems (e.g., article was launched on a market) are loaded to the active data warehouse repository. These events are used to create more complex types of events (i.e., relative temporal events). Within the active data warehouse repository, temporal events (e.g., end of quarter, etc.) may be used to determine the timepoints at which analysis rules should be fired. The passive data warehouse repository is responsible to answer two types of requests: (1) Ad-hoc OLAP requests that are issued by analysts using conventional OLAP tools. (2) Predefined OLAP requests that are issued by analysis rules when these rules are executed. A separate meta-data repository provides information about events, cubes, conditions, and actions. Automatically generated "decisions" (e.g., reduce the price of an article by 5 %) are exported back to OLTP systems, where transactions are executed to realize these decisions.

The remainder of this chapter is organized as follows: Section 3.1 describes the concepts of active database systems. Section 3.2 characterizes (passive) data warehouses and establishes two basic requirements that must be fulfilled by dimensional data to be used for automatic decision making in an active data warehouse. Section 3.3 explains the technology behind on-line analytic processing (OLAP). Finally, Section 3.4 presents a logical architectures for active data warehouses.

3.1 Active Database Systems

Before we can give a definition of the term *active database system*, we first need to clarify the terms *database*, *database management system*, and *database system*.

"A *database* is a collection of related data." [EN00]

"A *database management system (DBMS)* is a collection of programs that enables users to create and maintain a database." [EN00]

"A *database system* is the combination of a database and the corresponding database management system." [EN00]

Conventional database systems are *passive*, i.e., they execute only operations that are explicitly invoked by users or by application programs. But many applications demand reactions upon changes to the database state or the database environment without further user action.

"An *active database system* monitors the database state and executes predefined actions if a given condition holds after a specified event has occurred. Events to be detected, conditions to be checked, and actions to be initiated are defined explicitly in the database

Event	**Type** $\subset$ {Primitive, Composite} **Source** $\subset$ {Structure Operation, Behavior Invocation, Transaction, Abstract, Exception, Clock, External} **Consumption Mode** $\subset$ {Recent, Chronicle, Cumulative, Continuous} **Granularity** $\subset$ {Member, Subset, Set} **Role** $\in$ {Mandatory, Optional, None}
Condition	**Role** $\in$ {Mandatory, Optional, None} **Context** $\subset$ {DB_T, $Bind_E$, DB_E, DB_C}
Action	**Options** $\subset$ {Structure Operation, Behavior Invocation, Abort, Inform, External, Do Instead} **Context** $\subset$ {DB_T, $Bind_E$, $Bind_C$, DB_E, DB_C, DB_A}

Table 3.1: Dimensions for the knowledge model

schema using some form of *event-condition-action rules (ECA rules).*" [Obe98]

ECA rules rely on the basic structure "ON Event IF Condition DO Action". The constituting components of ECA rules – event, condition, and action – together with ECA rules themselves enrich the schema of a database. These components are described by the *knowledge model* of an active database system. This is in contrast with the *execution model* of an active database, which defines how a given set of ECA rules is treated at run-time. Paton and Diaz [PD99] provide a framework with which active database systems may be described and compared. Rather than using a particular rule language, they introduce "dimensions" to classify the knowledge model and the execution model of an active database system. These dimensions are explained in the subsequent sections.

3.1.1 Knowledge Model

The knowledge model of an active database system indicates *what* can be said about ECA rules in the system. Table 3.1 presents the dimensions with which the structural components of an ECA rule may be classified [PD99].

Event

The event part of an ECA rule describes the happenings that – when occurred – cause the rule's condition to be evaluated and the action to be executed if the condition holds. The *type* of an event can be (1) primitive or (2) composite. A primitive event is raised by a single, low-level occurrence of one of the following sources: *structure operation* (e.g., insert/update/delete), *behavior invocation* (e.g., a particular message is sent to an object), *transaction* (i.e., begin, abort, commit, rollback), *abstract* (e.g., some information entered by the user), *exception* (i.e., a particular exception was raised by the system), *clock* (i.e., absolute – 6th September 2001, 9:30, relative – 2 weeks after an article's price

was changed, periodic – every quarter), *external* (i.e., happening outside the database). A composite event is raised by some combination of primitive and/or composite events using a range of event operators. These event operators may be further classified [Obe98] into *boolean operators* (i.e., and, or not), *history operators* (i.e., sequence, historical conjunction), and *interval-based operators* (i.e., notOccurredIn – interval-based negation, occursDuring – monitoring interval, $*$ – interval-based sequence, $[n_1, n_2]$ – interval-based counting sequence).

Detecting composite events is based on the cartesian product of event occurrences that are needed for composition. Since this is the most general "context" for composite event detection, it is commonly referred to as *general context* in literature [CKAK94] on active database systems. *Consumption modes* [CKAK94] are used to select the subsets of composite event occurrences in the general context, that are relevant to particular types of applications, leaving the remaining composite event occurrences undetected. In the *recent context*, a composite event is raised with the most recent occurrences of the contributing events. In the *chronicle* context, a composite event is raised with the oldest occurrences of the contributing events that have not yet contributed to an older occurrence of the composite event. In the *continuous context*, a composite event is raised for every possible combination of occurrences of contributing events that have not yet contributed to an older occurrence of the composite event. In the *cumulative context*, a composite event is raised that collects all the occurrences of the contributing events that have not yet contributed to an older occurrence of the composite event.

The *granularity* of an event indicates whether an event is defined for every object in a *set*, for given *subsets*, or for specific *members* of a set. Finally, the *role* of an event indicates whether specifying the event part of a rule is *mandatory*, *optional*, or *none*. In conventional ECA systems, specifying the event part of a rule is mandatory. If the event is optional and no event was specified, the triggering of the rule is directly derived from the rule's condition part. Such rules are then called *condition-action* (CA) rules. Their execution differs substantially from executing ECA rules, since the task of determining when to evaluate the condition is shifted from the user to the system.

Condition

The condition part of an ECA rule is evaluated after the rule's event occurred. The *role* of a condition indicates whether or not it has to be specified. Usually, specifying a condition is optional or a dummy condition `true` may be specified instead.

Evaluating the different components of a rule is not effected in isolation from the database or from each other. Hence, processing a single rule can potentially be associated with four different database states: DB_T – the state of the database at the start of the transaction within which the rule is executed; DB_E – the state of the database when the event took place; DB_C – the state of the database when the condition is evaluated; DB_A – the state

Condition-Mode $\subset$ {Immediate, Deferred, Detached}
Action-Mode $\subset$ {Immediate, Deferred, Detached}
Transition Granularity $\subset$ {Tuple, Set}
Cycle policy $\subset$ {Iterative, Recursive}
Priorities $\in$ {Dynamic, Numerical, Relative, None}
Scheduling $\in$ {All Parallel, All Sequential, Saturation, Some}
Error handling $\subset$ {Abort, Ignore, Backtrack, Contingency}

Table 3.2: Dimensions for the execution model

of the database when the action is executed. Besides these four database states, bindings associated with the event (Bind_E) may be of interest to condition evaluation especially when event and condition are not evaluated in quick succession. The *context* indicates the setting (DB_T, DB_E, DB_C, Bind_E) in which the condition is evaluated.

Action

When a rule's condition evaluates to true, the rule's action is executed. The *options* of an action represent the different kinds of operations that may be executed. These operations range from simple *structure operations* that typically occur in relational database systems, *behavior invocations* that may be specified in object-relational or object-oriented database systems, *abort* the transaction within which the rule is executed, *inform* the user, make some *external* procedure call, perform some alternative action *instead* of the action that caused the rule's event to occur.

The *context* within which an action is executed may be DB_T, DB_E, DB_C, and DB_A. Further, bindings associated with the condition (Bind_C) may be of interest to action execution when condition and action are not evaluated in quick succession.

3.1.2 Execution Model

The execution model of an active database system determines *how* a set of ECA rules behaves at run-time. The dimensions describing the execution model are presented in Table 3.2.

Depending on the kind of application, executing an ECA rule at run-time may be carried out in a single atomic unit or may be split up into several phases. Most active database systems introduce *coupling modes* to provide a higher degree of flexibility in rule execution. The *immediate* mode specifies that condition (action) is evaluated (executed) immediately after the event (condition). The *deferred* mode specifies that the condition (action) is evaluated at the end of the triggering transaction as the last operation before the commit.

The *detached* mode specifies that the condition (action) is evaluated (executed) in an independent transaction.

The *transition granularity* of an ECA rule describes the relationship between event occurrence and rule instantiation. In the case of *tuple*, each individual event occurrence causes a separate rule to be instantiated and executed. In the case of *set*, a collection of event occurrences causes a rule to be instantiated and executed.

The *cycle policy* addresses the question of what happens when events are signaled due to condition evaluation or action execution. In the case of an *iterative* cycle policy, events signaled during condition evaluation or action execution are combined with those from the original event source and are subsequently consumed by rules from this single repository of signaled events. In the case of a *recursive* cycle policy, condition evaluation and action execution is suspended such that any immediate rules monitoring the signaled event can be processed at the earliest opportunity.

Priorities are used to determine the selection of the next rule to be fired. The possibilities are (1) *dynamic* approaches, which prioritize rules according to the time of event occurrence, (2) *numerical* approaches, which assign each rule an absolute value (e.g., based on the rule creation time or defined by the user), and (3) *relative* approaches, which explicitly specify that a given rule must be fired before another when both are triggered at the same time.

The *scheduling* of rules addresses the question of what happens when multiple rule instantiations are triggered at the same time. Possible solutions are (1) to fire all rule instantiations in *parallel*, (2) to fire all rule instantiations *sequentially* using one of the priorities mentioned above, or (3) to fire all rule instantiations of a specific rule before any other rules are considered (*saturation*).

Finally, the *error handling* of rules addresses the problem to treat errors during rule execution. Possible solutions are (1) *abort* the transaction in which the error occurred, (2) *ignore* the rule that raised the error and continue processing other rules, (3) *backtrack* to the state of the database when rule processing started and restart rule processing or continue the transaction without rule processing, and (4) adopt some *contingency* plan to recover from the error.

3.1.3 Applications of ECA Rules in Database Systems

Today, ECA rules support many different applications of active database systems. We classify these applications into *internal*, *extended*, and *external* applications [CW96a].

Internal Applications

Database internal applications use ECA rules to implement conventional system-supported facilities of database management systems. Two important examples are (1) integrity maintenance and (2) view maintenance and data integration [CW96b, EG98].

Integrity Maintenance. Maintaining integrity within a database was one of the first applications of active rules in commercial database systems. An integrity constraint is a declarative specification of some condition that must be satisfied by a valid database. ECA rules are used here (1) to identify the operations that may cause a violation (represented by the rule's event), (2) to check whether the integrity constraint still holds (represented by the rule's condition), and (3) to take particular action either to repair the integrity violation, roll back the statement that caused the violation, or roll back the entire transaction (represented by the rule's action).

View Maintenance and Data Integration. Maintaining materialized views is another important field of application to be realized with ECA rules. A materialized view is a relation whose tuples are determined once by a query and are then stored in a separate relation. Whenever the relations to which the query refers (i.e., the base relations) change, this must be propagated to the materialized view. Similar problems exist in the field of data integration, where data from independent data sources need to be integrated in a central repository. ECA rules are used here (1) to identify the changes to base relations (represented by the rule's event), (2) to examine whether these changes also affect the materialized view (represented by the rule's condition), and (3) to propagate these changes to the materialized view (represented by the rule's action).

Extended Applications

Extended applications implement mechanisms that support novel or nonstandard database tasks. Examples of extended applications are workflow management systems and transaction models that extend conventional ACID (atomicity, consistency, isolation, durability) [EN00] transactions.

Workflow Management. Workflow management systems have been introduced to design, execute, and monitor business processes that involve multiple steps of processing. ECA rules may be used here to control the dynamic aspects of such systems [KLRSR95], e.g., ordering of activities, selection of agents in response to state changes, and managing worklists associated with agents.

Extended Transaction Models. Extended transaction models offer concepts to split up a long-running (parent) transaction (consuming several hours up to days) into several nested sub-transactions that are executed under the control of the parent transaction. Dayal et al [DHL90] proposed an approach to control sub-transactions using ECA rules that "chain" related sub-transactions and that are responsible for handling failures and exceptions.

Hence, ECA rules are used to realize control flows and to check integrity constraints among transactions.

External Applications

Database external applications represent real-world domains within the database. Active rules are used here to obtain automatic reactive behavior in the context of the specific problem domain [CW96a]. Such rules are typically called *business rules*. Lang et al [LOS97] proposed an approach to specify business rules using a graphical notation in the context of active object-oriented database systems. Using this approach, business rules of any real-world domain are represented by such high-level representations that may be mapped to an ECA rule implementation model [Obe98].

3.2 Data Warehouses (DW)

"A data warehouse is a *subject oriented*, *integrated*, *non-volatile*, and *time variant* collection of data in support of management's decisions". Using this definition, which was coined by W. H. Inmon in [Inm96], we will explain the differences between OLTP systems and data warehouses. At the end of this section, we will identify two requirements that must be fulfilled by an active data warehouse system to bridge the gap between operational data and dimensional data to enable automatic analyzes in the data warehouse and automatic transaction execution in operational systems.

3.2.1 Subject Orientation

The first distinguishing feature of data warehouse data is the emphasis of the *subjects* for which decisions are made. Operational systems are organized around the functions of a company. Applied to our running example in retail trade, these functions are procurement, stock-keeping, marketing, and sales. Such systems have been designed to support operational business processes efficiently, i.e., to represent the transactional data of the business process, to avoid redundancies when storing these transactional data, and to achieve a high throughput of relatively simple insert/update/delete transactions that may be issued by many users often at the same time. During the last 30 years, many database design notations have been developed that address exactly these requirements. The two most prominent representatives are the Entity-Relationship Model (ERM) by P. Chen [Che75] and the Unified Modeling Language (UML) by I. Jacobson, J. Rumbaugh, and G. Booch [RJB99]. Both design notations explicitly distinguish between entities (or classes), which basically represent non-transactional data (e.g., products, customers, stores), and relationships (or associations), which represent transactional data (e.g., quantities sold and

revenues earned). The ERM, which may be regarded as the first modeling technique in the realm of the relational data model, only supports modeling of object structure but leaves the design of object behavior an open issue. The UML, which represents the last development in the evolution of modeling techniques, not only supports the design of object structure but also supports the design of object behavior (i.e., processes, states, and flow of control) as required by object-relational and object-oriented database management systems. We have chosen the UML as design notation to describe operational data models, since object structure and transactional object behavior are relevant in our further discussions.

Example: Figure 3.2 depicts a UML class diagram that represents the data model of the operational retail sales database. In UML, classes are depicted by rectangles (the class name is displayed in the upper pane, attributes are displayed in the center pane, and methods are displayed in the lower pane) and associations are depicted by arcs. Every Article has a unique articleId (uniqueness is depicted by stereotype $\ll$ pkAttr $\gg$) and is described by attributes name, price per unit, and producer. Methods registerSale, cancelSale, launchOnMarket, withdrawFromMarket, changePrice, and currentPriceEquals can be invoked for Articles. These methods represent the observable behavior of an Article and are considered to be transactions. Each Article is produced by one Producer (identified by its name) and belongs to exactly one Category (e.g., soft drinks, coffee, cereals, etc.). A Category has a unique name and is described by attribute description. Chain stores have a unique storeId and are described by the name of the store's owner. Every chain store is located in a particular City (identified by its zip-code). A City belongs to one geographical Region (identified by attribute name). This data model basically represents two kinds of transactional data that describe business processes in the real world: First, the data model describes data about *offerings*, which are represented by the binary bi-directional association between class Article and class Region. The multiplicity (M:N) of this association indicates that an article may be offered in stores that belong to several geographical regions and it indicates that the stores of a particular region can offer several articles at the same time. Second, the data model describes data about *sales*, which are represented by association class Sale and by the unlabeled ternary association that relates object classes Article, Day, and Store. When an article is sold in a store, then the quantity sold, and the revenue will be recorded. Note that an instance of class Sale represents the total quantities and revenues of an article per store and day. The constraint depicted below class Sale requires that an article may be sold only in a store in which that article is also offered.

In a data warehouse, the focus is drawn away from the different business processes of a company towards the various subjects that are of primary interest to decision makers. In the area of retail trade, the subjects of interest are products, customers, suppliers, and the various stores in which products are offered and sold. Business decisions that

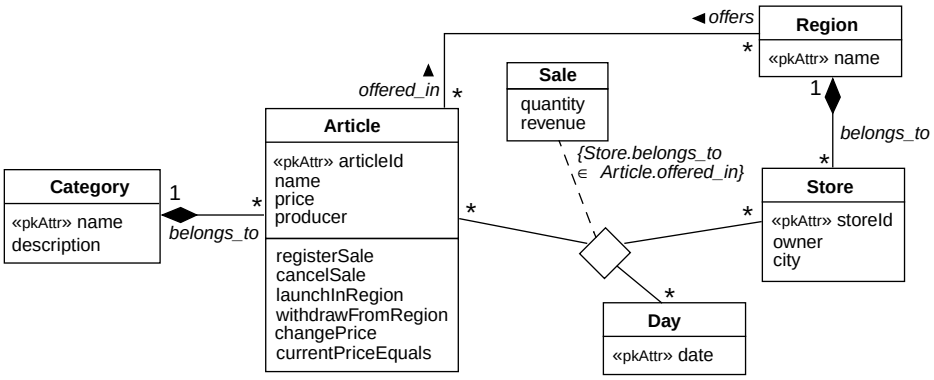

Figure 3.2: UML diagram describing the objects in the OLTP sales database

concern these subjects aim at goals such as improving profits, reducing costs. Thus, typical questions that should be solved when analyzing a data warehouse are for example whether to change the price of an article or whether to grant a discount to a particular customer. The subjects of a data warehouse are also called the data warehouse's *dimensions*, which represent the non-transactional data from operational systems. Unlike operational systems, a data warehouse typically provides several "levels of granularities" within each dimension. For a precise description of the organization of these "levels of granularities" one has to distinguish between *level instances* and *dimension levels*. Dimension levels collect level instances describing a dimension at the same level of granularity. The dimension levels of a dimension are organized in a directed, acyclic graph, in which nodes constitute dimension levels and arcs denote the *direct* subordinate relationship (denoted by $<$) between two dimension levels. Two dimension levels L_i and L_j are called direct/indirect subordinate and direct/indirect superordinate, respectively, if there is an arc/path between L_i and L_j. If L_i is the subordinate and L_j is the superordinate level, then every instance l_i of L_i belongs to exactly one instance l_j of L_j, thus l_i *rolls up* to l_j, which will be denoted by $L_i < L_j$. Conversely, every instance l_j of L_j comprises several instances l_i^1 ... l_i^n of L_i, thus l_j *drills down* to l_i^1 ... l_i^n. This emphasis of dimensional data in the design of data warehouses has led to the development of various conceptual modeling approaches [GMR98, KS95, SBHD98, TP98, TBC99], which are jointly called *conceptual dimensional data models*.

Example: Figure 3.3 depicts the conceptual dimensional model of our retail store chain, which represents the non-transactional data from the operational sales database. The three dimensions of interest – Product, Location, and Time – are depicted by dashed ovals. The levels of each dimension are depicted by rectangles within the oval representing the respective dimension; level hierarchies are depicted by arrows that point from a subordinate dimension level to its direct superordinate dimension level. Dimension Product consists of hierarchies Article $\rightarrow$ Category and Article $\rightarrow$ Producer; dimension Location consists of hierarchy Store $\rightarrow$ City $\rightarrow$ Region; dimension

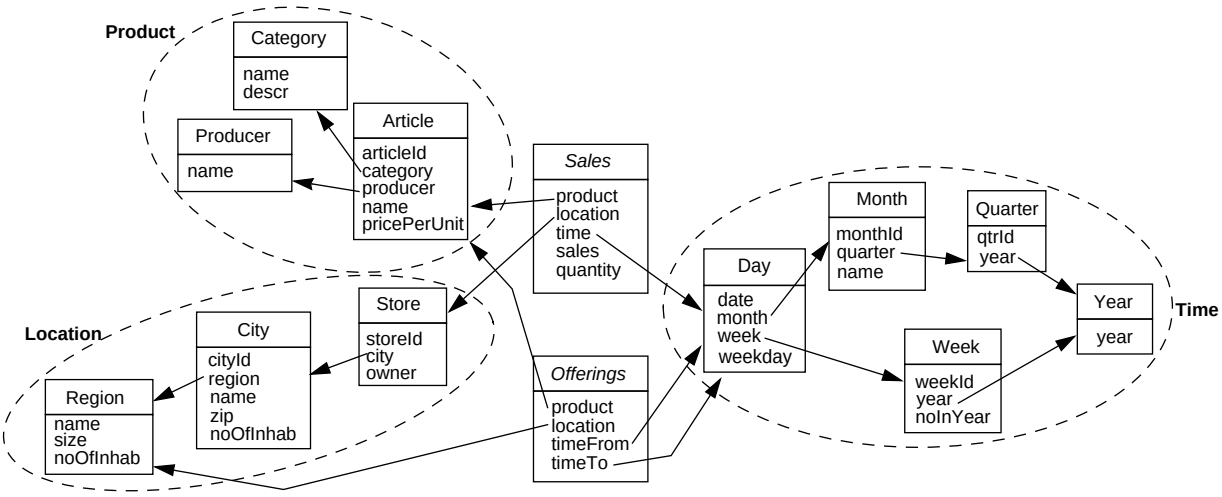

Figure 3.3: Dimensional model representing the retail data warehouse schema

Time consists of hierarchies Day → Month → Quarter → Year and Day → Week → Year. The two rectangles labeled Sales and Offerings will be explained in the next example.

The "glue" between the dimensions of a data warehouse and the measurable attributes of business processes (i.e., the *measures*) are the *facts* of the data warehouse. Facts represent the transactional data from operational systems. In the area of retail trade, a fact provides information about quantities of sold articles per store and per day. When analyzing data for decision support, an analyst aggregates the measure values along the dimension hierarchies with which the corresponding measures are connected.

Example: In Figure 3.3, transactional data concerning sales are represented by fact Sales, which is depicted by a rectangle in the center of the data model. The lower part of this rectangle depicts the linkage between measures sales and quantity and dimensions Product, Location, and Time. Transactional data concerning offerings are represented by fact Offerings, which relates dimensions Product, Location, and Time to describe that a particular article was offered in a particular store during a particular period of time (timeFrom and timeTo represent the first and the last day of that period). Note that this fact relates dimensions without providing any measures.

3.2.2 Integration

The second feature of data warehouse data is the *integration* of data from different operational systems. A typical data warehouse environment consists of several operational systems and a single database system that represents the data warehouse. In the majority

of cases, operational systems have been developed independently and the data warehouse has been introduced long after the operational systems have been put into work. Although operational systems may represent the same objects from the real world, their implementations may differ substantially since each system may use different encodings, naming schemes, physical attributes, measurements of attributes, etc. Applied to the area of retail trade, there may exist a system supporting the retail chain's sales activities and a system supporting the retail chain's marketing activities. Both systems represent the same products in the real world, but each system uses different identifiers and each system represents different aspects of the same products in real world.

The data warehouse serves two purposes in this context: (1) To provide an integrated view of data that exist in different operational systems and that represent the same objects of the real world. (2) To provide a unified view of data that exist in different operational systems and that represent different objects of the real world. The challenge of the first purpose is to resolve the various structural conflicts that may exist among the schemas of the various operational systems. Figure 3.4 provides a classification of such conflicts, which have been identified by W. Kim [KS91] in the field of multi-database systems. This classification and the proposed conflict resolutions [KCGS93] may be used to realize transformations of operational data before loading these data into the data warehouse. The challenge of the second purpose is to establish a "global picture" of the company's dimensions and facts and to offer these data to decision makers within a single data warehouse. This challenge addresses an architectural question as to whether create the data warehouse bottom up by defining departmental *data marts* using separate database systems that need to be integrated afterwards or as to whether create the data warehouse top down by defining all data marts in a single system, which supports sharing of joint dimensions. While the first approach offers the advantage of turning a data warehouse project quickly into practice, its disadvantage is the lack of enterprise-wide integration of data. Conversely, the advantage of the second approach is the full integration of data, its disadvantage are long running projects.

3.2.3 Non-Volatility

The third distinguishing feature of data warehouse data is its *non-volatility*. In operational systems, data is regularly accessed and updated on a per-record-basis, i.e., transactions are invoked for individual records only. The data warehouse loads masses of operational data and provides these data for read-only accesses to analysts, who will initiate transactions in operational systems as the conclusions of their analyzes. Applied to the area of retail trade, an analyst may decide to withdraw a particular article from chain stores that belong to a particular geographical region after analyzing the sales figures of that article using the data provided by the data warehouse.

1. Entity-vs.-Entity
   (a) One-to-One Entity
   (b) Many-to-Many Entities
2. Attribute-vs.-Attribute
   (a) One-to-One Attribute
   (b) Many-to-Many Attributes
3. Entity-vs.-Attributes
4. Different representations for the same data
   (a) Different Expressions denoting the same Information
   (b) Different Units
   (c) Different Precisions

Figure 3.4: Classification of Conflicts according to W. Kim

3.2.4 Time Variance

The last feature of data warehouse data is the emphasis of the time dimension, which characterizes almost every fact. Whenever a transaction that modifies the state of an operational system commits, old data is replaced by new data. If the OLTP system is not explicitly prepared for evolving its data over time, old data is lost at the moment it is replaced by new data since an OLTP system typically stores "current data" only. In the area of retail trade, such replacements may be the changing of an article's price (i.e., the old price is replaced by the new price) or the withdrawal of an article from a particular region (i.e., the article is no longer offered in the stores of a particular geographical region). A data warehouse offers a significant longer time horizon by providing a separate time dimension that may be used for versioning factual data (e.g., the validity of offerings is described by the two attributes timeFrom and timeTo in Figure 3.3) and for versioning dimensional data (e.g., it may be relevant to provide a history of the prices of an article). Since the time dimension provides different levels and hierarchies (e.g., Day $\rightarrow$ Week $\rightarrow$ Year and Day $\rightarrow$ Month $\rightarrow$ Quarter $\rightarrow$ Year), the time dimension may be also used for aggregating measures.

3.2.5 Requirements

In an active data warehouse, analyzes are carried out automatically and – as a conclusion of these analyzes – a transaction may be executed in one or in several operational systems. Two basic requirements must be fulfilled by a data warehouse system to be extended with

analysis rules:

1. **Support for events and actions.** This requirement addresses the need to extend the data warehouse schema with events, which are used to trigger analysis rules, and with (trans-)action specifications, which are used to identify concrete transactions to be executed in an operational system.

2. **Compatibility of dimensional data with operational data.** This requirement addresses the need to identify operational data in OLTP systems from within the data warehouse such that the execution of a particular transaction in the OLTP system can be initiated by the data warehouse.

These two requirements will not contradict the separation of operational data from dimensional data, which has been discussed above. Information about events explicitly introduces meta-data about decision making procedures in the data warehouse. Transaction specifications are specified for the subjects of decision making. But since concrete transactions are executed only in OLTP systems, non-volatility of data warehouse data will be ensured. Finally, the compatibility of dimensional data with operational data requires the data warehouse designer to provide a mapping from integrated dimensional data back to operational data. Since this mapping represents additional meta-data, the data warehouse schema will be further enriched.

3.3 On-Line Analytic Processing (OLAP)

"On-Line Analytical Processing (OLAP) is a category of software technology that enables analysts, managers and executives to gain insight into data through fast, consistent, interactive access to a wide variety of possible views of information that has been transformed from raw data to reflect the real dimensionality of the enterprise as understood by the user." [Cou00] The term *On-Line Analytical Processing (OLAP)* was first coined by E.F. Codd et al [CCS93] who realized that the relational data model, which was introduced to support operational business processes, is ill-suited as a first-class data model for decision support activities carried out by business executives. The mostnotable disadvantage of the relational data model in the realm of decision support is the lacking ability to consolidate, view, and analyze data according to multiple dimensions, in ways that make sense to one or more specific analysts at any given point in time.

3.3.1 Characterizing OLAP

OLAP tools are basically the means to provide the multidimensional view on data. These tools may be used to access data stored in several operational systems or they may be used

to access the data stored in a data warehouse (cf. Section 3.2). Hence, E.F. Codd et al proposed twelve rules to evaluate OLAP products [CCS93]:

1. *Multidimensional Conceptual View.* The analysts' conceptual view of OLAP models should be multidimensional, which facilitates model design and analysis. Such a multidimensional data model enables analysts to manipulate multidimensional data more easily. Note that this rule focuses on how data is *viewed* by analysts. From an implementation perspective, multidimensional data may be realized using the relational data model (i.e., *relational OLAP*), using a "native" multidimensional data model (i.e., *multidimensional OLAP*), or using a "hybrid" approach (i.e., combining relational OLAP with multidimensional OLAP).

2. *Transparency.* OLAP should be provided within the context of a true open systems architecture, allowing the analytical tool to be embedded anywhere the user-analyst desires, without adversely impacting the functionality of the host tool.

3. *Accessibility.* The OLAP system should access only the data actually required to perform the indicated analysis. This rule should ensure that there exists a common logical schema that integrates possible heterogeneous physical data stores (e.g., flat files, relational databases, legacy systems). Whenever a query is posed, necessary conversions will be made automatically using the information provided by the logical schema. If OLAP tools are used on top of a data warehouse, accessibility is satisfied due to the integrated view on operational data provided by the data warehouse (cf. Section 3.2.2).

4. *Consistent Reporting Performance.* Reporting performance should not degrade as the number of dimensions or the size of the database increases. The rationale for this rule is that if analysts perceive unsatisfactory reporting performance, required information will be typically presented in ways other than originally desired.

5. *Client-Server Architecture.* The predominant architecture of today's computer systems is that data is stored on high-performance mainframe systems but is accessed via personal computers. OLAP tools should support this architecture, which allows that various clients can be attached to a server within minimum effort and integration programming.

6. *Generic Dimensionality.* Every data dimension must be equivalent in both its structure and operational capabilities. Due to symmetry of dimensions, any additional operational capability that is granted to a selected dimension may be granted to any dimension.

7. *Dynamic Sparse Matrix Handling.* An important aspect of multidimensionality is efficient handling of sparsity (i.e., missing cells as a percentage of possible cells). If not handled properly, sparsity may cause complex computations with long response times and inefficient storage structures.

8. *Multi-User Support.* Analytic databases are accessed from a variety of users which have different information needs and which may inspect different parts of an analytic database. OLAP tools must provide concurrent access (i.e., retrieval and update), integrity, and security to such analytic databases.

9. *Unrestricted Cross-dimensional Operations.* To allow unrestricted cross-dimensional operations, a computationally complete language must be available to specify the various formulae needed.

10. *Intuitive Data Manipulation.* Consolidation path re-orientation, drilling down across columns or rows, zooming in or out, and other manipulation inherent in the consolidation path should be accomplished via direct action upon the cells of the analytical model. The analyst's view of the dimensions defined in the analytical model should contain all information necessary to effect these interactions.

11. *Flexible Reporting.* This rule aims at the capability to present data flexibly, i.e., comparisons of data may be arranged by any grouping occurring naturally in the enterprise. Hence, rows, columns, and page headings of reports must be capable of containing an arbitrary number of dimensions in any order and provide means of showing the inter-consolidation path relationships between the members of these dimensions.

12. *Unlimited Dimensions and Aggregation Levels.* The OLAP system should support an arbitrary number of dimensions, at least fifteen to twenty, each having an unlimited number of user-defined aggregation levels within any hierarchy.

Besides these twelve rules by E. F. Codd, Pendse and Creeth [PC95], the founders of the "OLAP Report Project", introduced a different characterization of OLAP with the intention "to be short and easy to remember". This characterization is known as the five-term acronym *FASMI* (Fast Analysis of Shared Multidimensional Information) [PC95]:

1. *Fast* means that the system is targeted to deliver most responses to users within about five seconds, with the simplest analyses taking no more than one second and very few taking more than 20 seconds.

2. *Analysis* means that the system can cope with any business logic and statistical analysis that is relevant for the application and the user, and keep it easy enough for the target user.

3. *Shared* means that the system implements all the security requirements for confidentiality (possibly down to cell level) and, if multiple write access is needed, concurrent update locking at an appropriate level.

4. *Multidimensional* means that the system must provide a multidimensional conceptual view of the data, including full support for hierarchies and multiple hierarchies, as this is certainly the most logical way to analyze businesses and organizations. A specific minimum number of dimensions that must be handled is not defined since it is too application dependent.

5. *Information* means that measuring the capacity of an OLAP system depends on how much input data it can handle, not how many Gigabytes it takes to store the data. The capacity of an OLAP system is determined by many factors such as data duplication, RAM required, disk space utilization, performance, integration with data warehouses, etc.

Since an active data warehouse supports OLAP tasks, all of the characteristics mentioned above are of relevance in this field but need to be viewed from the following perspective: Analysts design OLAP tasks once for multiple automatic execution by an active data warehouse. Hence, the characteristics mentioned above are classified as follows:

1. Characteristics concerning the user's view of data (i.e., multidimensionality, transparency, multi-user support, data manipulation) are important to let analysts carry out their tasks properly.

2. Characteristics concerning reporting performance and flexibility of the underlying data model are important for automatic execution of OLAP tasks.

Note that this thesis proposes an approach to extend data warehouses with the capabilities of automatic decision making. Topics related to guaranteeing performance requirements or measuring the performance of multidimensional analyzes may be relevant in this concern but are beyond the scope of this thesis.

3.3.2 OLAP Data and OLAP Operations

The primary purpose of a data model supporting OLAP is to provide a multidimensional conceptual view on data and to offer high-level operations that may be used to navigate and analyze data.

OLAP Data. Due to its multidimensional nature, OLAP data is represented using the metaphor of a *data cube* (or *cube* for short). A cube defines the dimensions that may be used to analyze factual data, as provided by a data warehouse (see Section 3.2). Each dimension of a cube is populated by the instances of a dimension level, which represent the cube's facts at a particular granularity. A cube's cells (i.e., the intersection of level instances that populate the cube's dimensions) hold the facts described by the cube. Although we usually

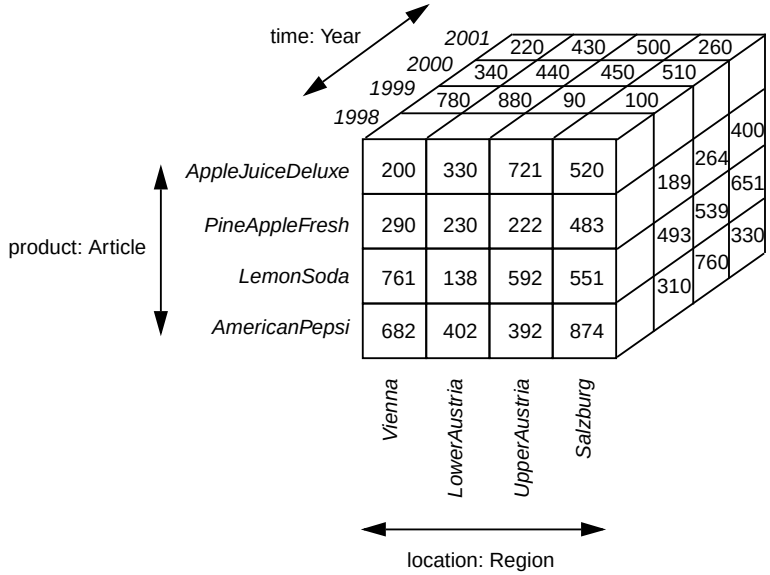

Figure 3.5: 3-dimensional cube representing sales per Article, Region, and Year

think of cubes as 3-D geometric structures, in data warehousing and OLAP systems the data cube is n-dimensional.

Example: Figure 3.5 depicts a 3-dimensional cube with dimensions product, location, and time that describes sales (in 1000 ATS). Dimension product is populated with the instances of dimension level Article, dimension location is populated with the instances of dimension level Region, and dimension time is populated with the instances of dimension level Year. Hence, sales figures are represented for each combination of instances of Article, Region, and Year.

OLAP Operations. To inspect and analyze multidimensional data, OLAP tools offer high-level conceptual operations that may be applied directly on data cubes using information provided by individual dimension levels or using the hierarchical order of dimension levels. The result of applying these operations is a new cube. The following high-level conceptual operations are commonly distinguished in literature [CD97, HK00]:

- *Rollup & Drilldown.* The *rollup* operation performs aggregation on the fact values of a data cube by climbing up a dimension hierarchy of one of the cube's dimensions. The *drilldown* operation is the reverse of *rollup*.
- *Slice & Dice.* The *slice* operation performs a selection on one of the cube's dimensions, thus retrieving the set of cells that comply with the selection condition. Selection predicates may be formulated using the attributes of the level instances that populate

one of the cube's dimension. The *dice* operation is a variation of the *slice* operation; it defines a subcube by performing a selection on two or more of the cube's dimensions.

- *Drill Across.* The *drill across* operation executes queries involving more than one cube. To be able to apply this operation, the cubes on which *drill across* is applied must "share" at least one of their dimensions. This operation is sometimes also called *cube join.*

- *Pivot.* The *pivot* operation rotates the cube's dimension axes visually in order to provide an alternative presentation of the data. The information provided by the new cube is exactly the same as the information provided by the cube on which *pivot* was applied.

3.3.3 Related Work on Data Models supporting OLAP

In literature, several multidimensional data models and multidimensional query languages in support of OLAP have been proposed. In the following, we briefly outline the most important of these data models and query languages along the lines of [SBH99, VS99]:

- Gray et al [GCB+97] introduced the *data cube* operator, which is the generalization of the GROUP BY-clause of SQL. This operator expands a relational table, by computing the aggregations over all possible subspaces created from the combinations of the attributes of such a relation. Note that this operator introduces a "flavor of multidimensionality" in the relational data model since many data warehouses are built upon relational database technology. While the data cube operator is applied on conventional relational data, the remaining data models below are either "real" extensions to the relational data model or define a completely new data model using arrays to represent multidimensional data.

- Li and Wang [LW96] formalize a multidimensional data model (1) by introducing multidimensional (MD) cubes that consist of a number of relations and (2) by defining an algebraic query language on MD cubes, called *grouping algebra.*

- Gingras and Lakshmanan [GL98] propose an extension to the relational algebra, called *restructuring relational algebra*, and an extension to SQL, called nD-SQL. The restructuring relational algebra is used to create the data warehouse, which integrates data from various operational systems having different schemas. nD-SQL is an extension to SQL which offers constructs to specify drilldown, rollup, and cube aggregations.

- Jagadish et al [JLS99] propose the SQL($\mathcal{H}$) data model, which is an extension to the relational data model that gives a first-class status to the notion of dimension hierarchies. A *hierarchy schema* is a triple $\mathbf{H} = (G, \mathcal{A}, \sigma)$ such that G is a rooted

DAG (root is a special node *All*), $\mathcal{A}$ is an attribute set, and σ is a function $\sigma : G \rightarrow 2^{\mathcal{A}}$ that assigns each node $u \in G$ an attribute set $\sigma(u) \subseteq \mathcal{A}$. Further, the structured query language (SQL) is extended by a DIMENSIONS-clause to support queries using such hierarchies. Tables specified in the DIMENSIONS-clause represent dimension levels. In the WHERE-clause of an SQL($\mathcal{H}$)-query, tuples from the fact table may be rolled up directly to tuples of some dimension level L by specifying the $<$-relationship (denoted as $=<<$). This avoids multiple joins between the fact table and the tables representing dimension levels that roll up to L.

- Gyssens and Lakshmanan [GL97] introduce n-dimensional tables and a mapping to the relational data model. OLAP operations are expressed by an algebra that provides classical relational operators as well as operators for restructuring, classification, and summarization.

- Agrawal et al [AGS97] propose a multidimensional data model together with an associated algebra. Data is organized in one or more hypercubes, which are defined as triples $(D, E(C), N)$. D is a set of dimension names, $E(C)$ is a mapping from the domains of the dimensions in D to the cell values of cube C, and N is an n-tuple containing the names of measures of C. The algebra provides a set of operations that may be used for navigation. *Push* and *Pull* are used to achieve a symmetric treatment of measures and dimension members. *Push(C, D_i)* converts the members of dimension D_i of cube C into measures. *Pull(C, D_i, i)* is the converse of the *Pull* operator. It creates a new cube C_{new} with an additional dimension D containing the i-th element of the measures as its dimension members. *Destroy dimension (C, D_i)* removes dimension D_i from cube C. *Restriction(C, P, D_i)* evaluates a predicate P for a given dimension D_i of some cube C and removes the elements of D_i that do not satisfy P. *Join(C_1, C_2*, where (i) C_1 consists of m dimensions, (ii) C_2 consists of n dimensions, and (iii) C_1 and C_2 have k dimensions in common, creates a new cube C with $m + n - k$ dimensions. The measures of C_1 and C_2 are mapped to C using function f_{elem}.

- Datta and Thomas [DT99] propose a multidimensional data model and an algebra which are similar to the approach taken on by Agrawal et al [AGS97]. This approach also treats measures and dimensions symmetrically. Further operators are: The *aggregation* operator performs aggregation on one or more dimensions in a similar way as achieved by the relational aggregation functions SUM, COUNT, or MAX. The *partition* operator maps the cells of a cube into meaningful groups to be later used by aggregation. The *cartesian product* operator can be used to combine any two cubes, even if these cubes do not share any dimension.

- Vassiliadis [Vas98] proposes a multidimensional data model which distinguishes *basic cubes*, which store data physically, and *cubes*, which may be derived from basic cubes using cube operations *level climbing* (i.e., rollup), *packing* (i.e., partition [DT99]),

function application (i.e., aggregation [DT99]), *projection* (i.e., destroy dimension [AGS97]), and *dicing* (i.e., restriction [AGS97]).

- Lehner et al [Leh98] proposes a multidimensional data model that supports classification hierarchies and features. The approach distinguishes primary and secondary multidimensional objects (MO). A primary MO represents the structure of a data cube according to the classification hierarchies (i.e., measures, dimensions, aggregations, selection conditions). A secondary MO is used to formalize the extension of the conventional multidimensional model using features (i.e., describing attributes). Further, an algebra is defined that offers the following operations: *sub-cube selection* (i.e., slicing & dicing), *drill-down*, *roll-up*, *aggregation*, and *cell-oriented operations* that allow applying unary operations (e.g., abs, sign) to the cells of a cube. In [BL97] an SQL-like declarative query language (called *cube query language*) that aims at this data model and algebra is presented.

- Cabbibo and Torlone [CT98a] propose a multidimensional data model that defines facts in so-called *f-tables*, which are relations that contain a tuple for each cell of the data cube containing a value. Dimensions are defined by a graph, containing dimension levels as the nodes of the graph. In [CT98c], they propose an operational approach to query f-tables which is equivalent with the above-mentioned algebras. A visual query language to be used by user-analysts is proposed in [CT98b].

- Pedersen and Jensen [PJ99] propose an "extended multidimensional data model" that supports (i) explicit hierarchies in dimensions, (ii) symmetric treatment of dimensions and measures (i.e., dimensions can be converted to measures and vice versa), (iii) multiple hierarchies in each dimension, (iv) support for correct aggregation, (v) non-strict hierarchies, (vi) many-to-many relationships between facts and dimensions, (vii) handling change and time (i.e., slowly changing dimensions), (viii) handling uncertainty (i.e., probabilities are added to old factual data in order to be counted correctly together with new factual data), and (ix) describing facts using different levels of granularity (i.e., facts may be registered at different levels of granularities). An algebra is proposed that consists of basic operators such as *selection*, *projection*, *union*, *difference*, *rename*, and *aggregate formation*. Extensions to support handling of time – called the *valid-timeslice operator* – and to support handling of uncertainty – called the *cut-off operator* – are further proposed.

3.3.4 Architectures for OLAP

In the recent years, several alternative architectures for OLAP systems – relational OLAP (ROLAP), multidimensional OLAP (MOLAP), hybrid OLAP (HOLAP), and desktop OLAP (DOLAP) – have emerged. These architectures have in common that they rely on a 3-tier separation of presentation, processing, and storage of multidimensional data, which is illustrated in Figure 3.6. This separation satisfies rule 5 (client-server architecture)

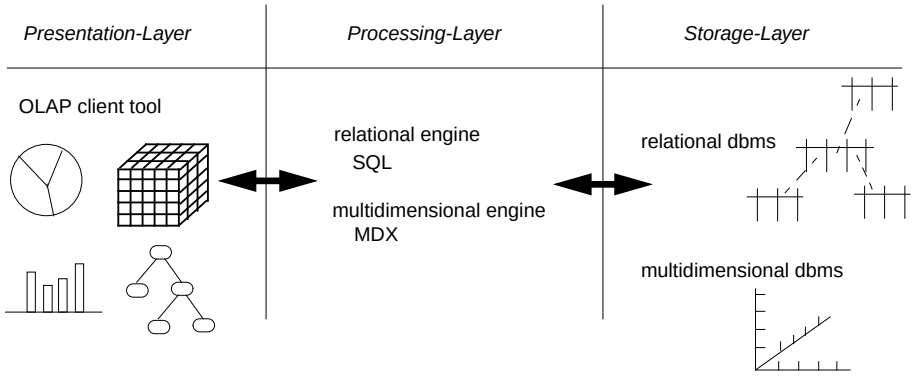

Figure 3.6: 3-Tier Architecture of OLAP systems

imposed by E. F. Codd. The distinguishing features are basically (1) *storage technologies* (i.e., relational database, multidimensional database, and client-based files) and (2) *processing technologies* (i.e., SQL, multidimensional server engine, client multidimensional engine) [PC95]. In the following we will outline the four alternative architectures.

Relational OLAP (ROLAP). In a relational OLAP setting, the storage and the query processing layers are both realized using relational database technology. The structured query language (SQL) [MS01], which is the standard data definition and data manipulation language for relational databases, is used to define the data warehouse schema and to process multidimensional queries. Advanced analytic functions [Red97] (e.g., moving average, moving sum, ranking, tertile, ratio to report, etc.) and extensions to the SQL GROUP BY-clause [GCB+97] improve the analytic capabilities of ROLAP systems. The relational data model is a generally accepted data model that is widely used to support OLTP and OLAP systems. The major benefits of relational database systems are the standardized query language SQL and the great variety of physical access structures such as B-trees, bitmap indexes, hashing, etc.

Multidimensional OLAP (MOLAP). In a multidimensional OLAP setting, the storage and query processing layers are both realized using proprietary multidimensional implementations. Storing multidimensional data is realized using arrays. Processing queries is realized using proprietary languages such as multidimensional expressions (MDX) from Microsoft [Nol99] and multidimensional database APIs such as OLE-DB for OLAP from Microsoft [Mic01] or MDAPI from OLAP council [Cou97]. Since multidimensional database systems and query processing systems are optimized towards supporting OLAP architectures and multidimensional data efficiently, they typically outperform an equivalent ROLAP architecture.

Hybrid OLAP (HOLAP). In a hybrid OLAP setting, the storage and query processing layers are realized using relational and multidimensional database technologies. Relational databases are used to store detailed data while multidimensional databases are used to give access to pre-aggregated data. Queries are answered by accessing the multidimensional database first, and if no proper data cube is available, the query is answered by accessing the relational database. Hybrid OLAP integrates the advantages of ROLAP and MOLAP systems but at the cost of a more complex system architecture causing increased maintenance efforts.

Desktop OLAP (DOLAP). In a desktop OLAP setting, the presentation layer (i.e., the desktop computer that runs the OLAP client) is empowered with an additional storage and query processing layer. The conventional storage layer is realized using a conventional ROLAP, MOLAP, or HOLAP architecture as described above. The additional layer, which provides access to only a subset of data stored in the conventional storage layer, typically answers requests from a single client. Due to the increased system load on the desktop computer, the additional storage layer is realized using flat files instead of a separate database management system. A DOLAP architecture is particularly useful in a distributed setting with slow network communication or if the data warehouse/OLAP server is only accessible at particular timepoints.

3.4 A Logical Architecture for Active Data Warehouses

Active data warehouses combine active database systems, data warehouses, and on-line analytical processing to automatize decision processes that occur on a regular schedule and for which well-established decision criteria exist. The technology to automatize these decision processes are *analysis rules*, which rely on the basic ECA rule structure borrowed from active database systems. The *event part* of an analysis rule represents the timepoints at which decision processes should be initiated. In an initial approach, we only consider temporal events (i.e., absolute temporal events, relative temporal events, and periodic temporal events) to be used as the initiators for rule execution. The *condition part* of an analysis rule represents the multidimensional analyzes that are usually carried out manually using some OLAP tool. In contrast to conventional ECA rules, the condition is not a simple boolean predicate but consists of hierarchically organized predicates that are evaluated incrementally. The condition part is designed to mimic the top-down analytical work of an analyst. The *action part* of an analysis rule is the directive to execute a particular transaction in an OLTP system, representing the decision for which analyzes have been carried out. When an analysis rule is executed, parameters needed by the OLTP transaction are determined. These parameters are passed over to the OLTP transaction when the decisions of an active data warehouse are exported to OLTP systems.

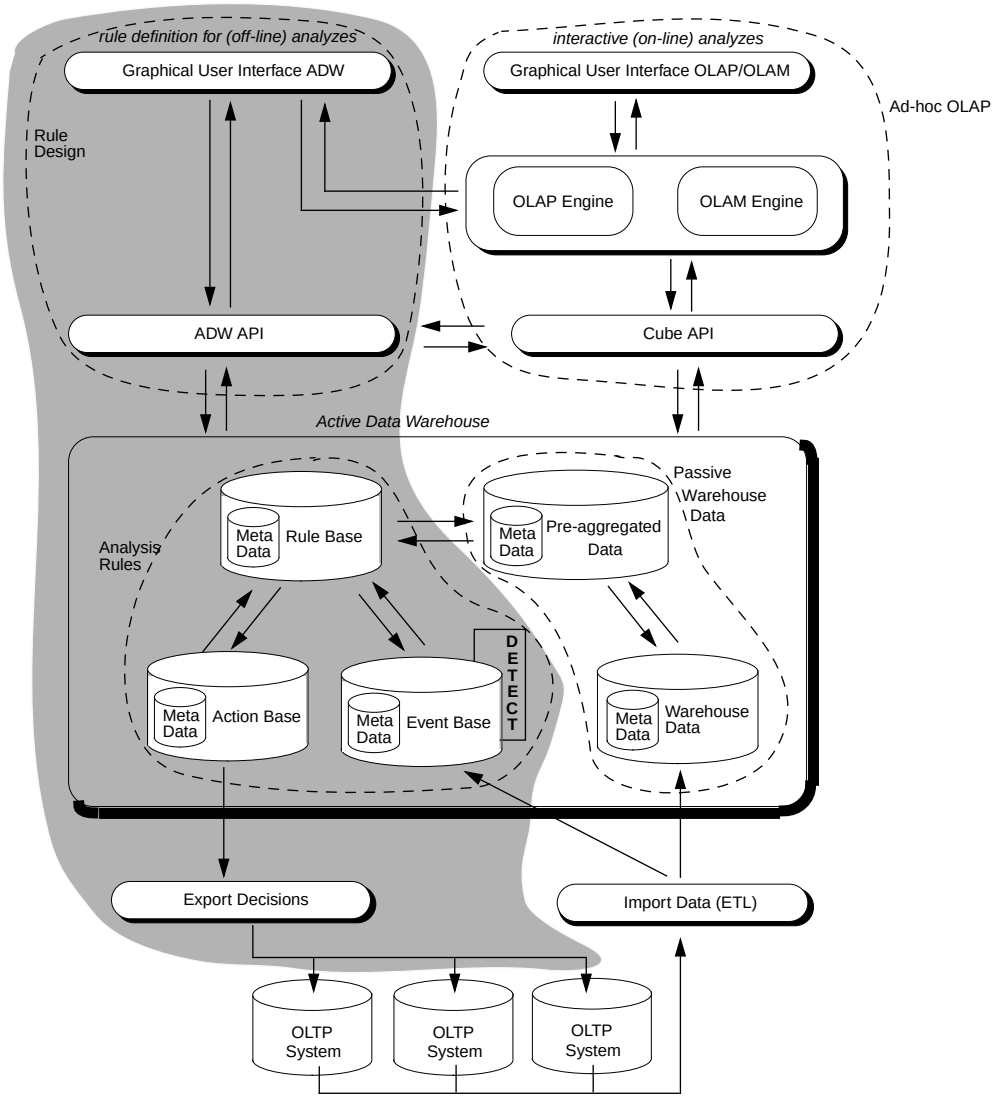

Figure 3.7: Logical Architecture for Active Data Warehouses

Hence, active data warehouses realize a "closed feedback loop" in which data is loaded from OLTP systems, decisions are generated within the active data warehouse, and these decisions are exported back to OLTP systems by invoking transactions. One such loop, which consists of importing data from OLTP systems, executing analysis rules, and exporting generated decisions back to OLTP systems, is called *ADW cycle*. Figure 3.7 depicts the logical architecture of an active data warehouse and the various components that realize such an ADW cycle. Arrows indicate the flow of data among these components. The shaded area highlights the components that extend the conventional (passive) data warehouse/OLAP architecture. In the following, we will explain the components that constitute the passive and the active part of this architecture.

- *Import Data (ETL).* Data is extracted, loaded, and transformed (ETL) from OLTP systems to be integrated in the data warehouse. Besides loading conventional passive warehouse data that represent dimension levels and facts, happenings that occurred in OLTP systems (so-called "OLTP method events") are loaded to the event data repository. OLTP method events are used in the active data warehouse as the "initiators" of relative temporal events, which may trigger analysis rules.

- *Passive Warehouse Data.* The passive warehouse data repository maintains data about dimensions, levels, and facts. This basic warehouse data is used to answer OLAP queries and to provide pre-aggregated data that speeds up such queries. In a relational data warehouse, pre-aggregated data are realized by "materialized views" or "summary tables" whereas in a multidimensional data warehouse, pre-aggregated data is realized by physically stored "cubes".

- *Ad-hoc OLAP.* To browse, inspect, and analyze data in the data warehouse, analysts use OLAP tools that provide a graphical user interface. These tools typically access the data warehouse repository using a *cube API*, which represents warehouse data in a uniform multidimensional data model. An *OLAP engine* supports typical browsing tasks within data cubes such as *drilldown*, *rollup*, *slice & dice*, or *pivot*. Recent OLAP tools also offer extended functions for data mining (e.g., clustering, association rules, decision trees, etc) [HK00], which are supported by a separate OLAM (on-line analytical mining) engine.

- *Analysis Rules.* In the active data warehouse, analysis rules are provided by a separate *rule base*, in which executable rule definitions and meta data about these rules are stored. Since analysis rules carry out multidimensional analyzes in a similar way as analysts do manually, the data warehouse repository that maintains pre-aggregated data ("cubes") is accessed by each analysis rule. Analysis rules are fired upon the occurrence of events, which are maintained by a separate *event base*. The event base is responsible for detecting and signaling event occurrences to analysis rules. Transactions to be executed in OLTP systems are provided by a separate *action base*, which (i) maintains the specifications of these transactions and (ii) maintains the

parameter bindings which have been determined by analysis rules (i.e., change the price of article X by 10 %).

- *Rule Design.* Similar to carrying out ad-hoc OLAP/OLAM tasks, analysts access the rule base using a graphical user interface with which they can define new rules, update existing rules, inspect generated decisions, and modify the event base. The programming interface that gives access to the active data warehouse repository ("ADW API"), presents rules, actions, and events independent from their implementation in a particular warehouse. Since analysts specify cubes and multidimensional analyzes to be carried out by analysis rules, the OLAP engine is accessed for presentation purposes whereas the cube API is accessed to define the cubes that need to be analyzed by analysis rules.

- *Export Decisions.* At the end of every ADW cycle, decisions are exported back to OLTP systems where corresponding transactions are invoked. Since data in the data warehouse is not fully synchronized with data in OLTP system, consistency checks need to be carried out prior to executing transactions in OLTP systems.

In the remainder of this thesis, we will concentrate on the definition, specification, and implementation of analysis rules. We will not touch topics related to importing events or exporting decisions. Further, we will not discuss topics related to rule design. Note that a basic assumption of our approach is that the length of one ADW cycle remains constant (e.g., one day). This assumption may be reconsidered in the future, which may lead to a more dynamic approach to active data warehouses.

Part II

A Basic Approach to Analysis Rules

Chapter 4

Multidimensional Data

Contents

This chapter lays the foundations for active data warehouses in two respects: First, in Section 4.1, we will introduce a data model and a language for specifying basic multidimensional data (i.e., levels and facts) in a data warehouse environment. Second, in Section 4.2, we will introduce an approach for specifying multidimensional analyzes on top of this data model. This approach will be adopted later in this thesis to specify multidimensional analyzes that will be carried out automatically by analysis rules in active data warehouses.

4.1 Basic Multidimensional Data Structures

This section introduces a logical data model for multidimensional data that will be used throughout this thesis. Note that this data model is used to specify multidimensional data as required by an active data warehouse. It is independent from any concrete implementation data model (e.g., arrays, relations) but it can be easily translated into such a data model. Although several logical data models have been proposed for data warehousing and OLAP in the recent years [Vas98, Leh98, CT98a, LW96, GL97, DT99, AGS97, PJ99], we

have chosen to introduce a new data model that particularly meets the needs of multidimensional analyzes in active data warehouses (i.e., hierarchical top-down analysis of cubes, see Section 4.2). We utilize schemas to describe the data in a data warehouse.

4.1.1 Dimension Levels

The common structure L of the instances of a dimension level, denoted by $k(L)$, is described by a conceptual dimensional model, the *level schema* $L = (I : dom(I), R_1 : dom(R_1), ..., R_m : dom(R_m), D_1 : dom(D_1), ..., D_n : dom(D_n)), n > 1, m \geq 0$, where $dom(I)$ is the set of unique level identifiers of $k(L)$, $dom(R_i)$ is the set of unique level identifiers $k(L_i)[I]$ of some other dimension level L_i to which the instances of L *roll up*, and $dom(D_i)$ are values to further describe the level instances $k(L)$ concerning a particular aspect D_i (e.g., prices of articles). As required by level schema L, the level instances $k(L)$ of L, which have been loaded from OLTP systems, provide values for *identifying attribute* I, for *rollup attributes* R_1 ... R_k, and for *describing attributes* D_1 ... D_k. The identifying attribute I is used to identify level instances of the dimension level, i.e., $\nexists l_i, l_j \in k(L) : i \neq j, l_i[I] = l_j[I]$. It can be compared with a primary key attribute in the relational data model or with an object identifier in an object-oriented data model. Note that in the future we write L instead of $k(L)$ if $k(L)$ is understood. The $<$-relationship between dimension levels is defined by the rollup attributes of each dimension level. A rollup attribute $R{:}L_j[I]$ of level schema L_i defines a direct subordinate relationship, denoted by $<$, between dimension levels L_i and L_j. The $<$-relationship can be depicted as a directed acyclic graph (DAG) whose nodes are dimension levels and directed arcs are directed subordinate-superordinate relationships. This graph must possess exactly one top element L_{sup} and one bottom element L_{inf}. Note that the top-level dimension level L_{sup} is implicit for each dimension path. If two or more dimension paths originate at the same dimension level L_{inf}, they also "share" the same implicit top-level dimension level. To provide for a unique name of top level dimension levels, we use string ALL together with the name of L_{inf} in plural (e.g., ALL-Articles, ALL-Stores, ALL-Days). A general property of dimension levels that belong to the same dimension path is that these dimension levels are comparable with respect to their position in the dimension hierarchy, i.e., $L_i = L_j$, or $L_i > L_j$ (i.e., L_i is "more general" than L_j), or $L_i < L_j$ (i.e., L_i is "more specific" than L_j). We will refer to this property later in this thesis.

Example: Figure 4.1 (a) depicts the $<$-relationships among dimension levels Article, Category, and Producer. Attributes are not depicted. The two $<$-relationships between Article and Category and between Article and Producer are represented by rollup attributes category:Category and producer:Producer at level schema Article.

Rollup attributes can be compared with foreign key attributes in the relational data model or with single-valued object attributes in an object-oriented data model. The value of

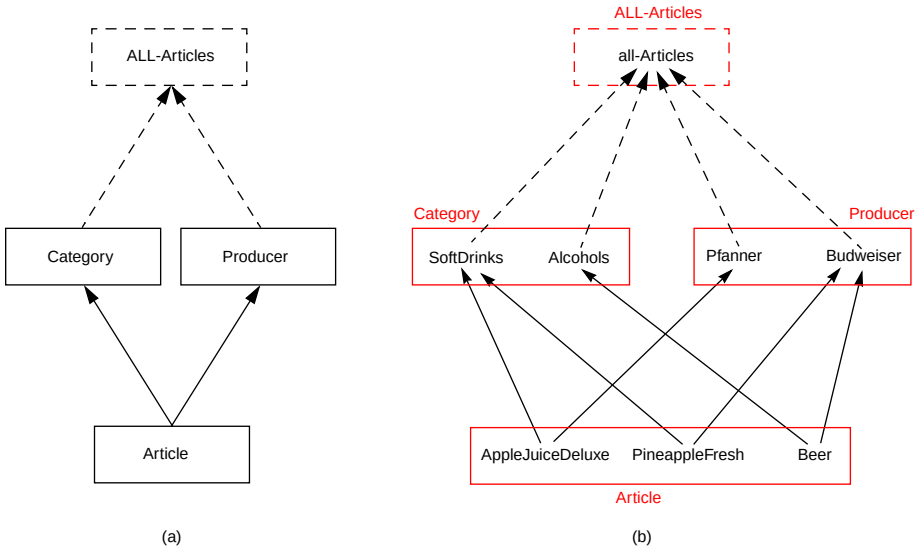

Figure 4.1: Dimension Levels and Level Instances

rollup attribute R of a particular level instance l_i establishes the $<$-relationship between l_i and its immediate superordinate level instance l_j in the domain of R, i.e., $l_i[R] = l_j[I]$. This is called l_i rolls up to l_j and is denoted as $l_i < l_j$. For each top-level dimension level L_{sup}, there exists a single implicit level instance l_{sup} to which all level instances of subordinate dimension levels roll up. To provide a unique naming also for such implicit level instances, we write string all together with the name of L_{inf} in plural (e.g., all-Articles, all-Stores, all-Days). Since a top-level dimension level exists implicitly, the $<$-relationships between this dimension level and its subordinate dimension levels are also implicit and are not represented by separate rollup attributes.

Example: Figure 4.1 (b) depicts level instances AppleJuiceDeluxe, PineappleFresh, and Beer of dimension level Article, level instances SoftDrinks and Alcohols of dimension level Category, and level instances Pfanner and Budweiser of dimension level Producer. The arrow that points from AppleJuiceDeluxe to SoftDrinks indicates the value of rollup attribute category for article AppleJuiceDeluxe (i.e., AppleJuiceDeluxe rolls up to SoftDrinks), the arrow that points from AppleJuiceDeluxe to Pfanner indicates the value of rollup attribute producer (i.e., AppleJuiceDeluxe rolls up to Pfanner). The remaining arrows indicate the values of the two rollup attributes for articles PineappleFresh, AmericanPepsi, Beer, and Whisky accordingly. Since the two dimension paths Article $<$ Producer and Article $<$ Category both originate at dimension level Article, their common top-level dimension level is ALL-Articles, which is depicted by dashed arrows and a dashed rectangle.

Intuitively, we expect that the hierarchy of level instances is consistent with regard to the following:

1. A level instance of a dimension level rolls up to exactly one level instance of each direct superordinate dimension level (i.e., dimension paths are *strict*, cf. [PJ99]).

2. A level instance is not superordinate to itself.

3. If a level instance is rolled up to the same dimension level along different paths, it is rolled up to the same level instance (i.e., dimension paths are *partitioning*, cf. [PJ99]).

Therefore the definitions of level schemas must obey the following properties:

1. For any two different rollup attributes R_i and R_j of dimension level $L \in \mathcal{L}$[1]: $dom(R_i) \neq dom(R_j)$. Since each rollup attribute of a level instance has exactly one non-null value, this property ensures that the rollup attributes of L refer to exactly one level instance of each direct superordinate dimension level.

   Example: Consider the time dimension in Figure 3.3. Assume that level instance 10-Oct-2001 (i.e., value of the identifying attribute date, further attributes are not shown) of dimension level Day rolls up to the superordinate level instance WK-41-2001 of dimension level Week using rollup attribute week:Week. The above property is violated if the level schema of Day defines another rollup attribute wk: Week, which relates 10-Oct-2001 with level instance WK-03-2001.

2. $\nexists L \in \mathcal{L} : L < ... < L$, i.e., dimension paths are free from cycles.

   Example: The property above is violated if the schema of dimension level Year defines a rollup attribute 1stMon: Month, since then Year $<$ Month $<$ Quarter $<$ Year is satisfied.

3. Consider the two dimension paths $L < ... < L' < L'''$, $L < ... < L'' < L'''$ and the level instances $l \in L, l' \in L'$, $l'' \in L'', , l''' \in L''', l''' \in L'''$ with $l < ... < l' < l'''$ and $l < ... < l'' < l'''$. Then $l''' = l'''$. Hence $l'[R'''] = l''[R''']$ with $dom(L'[R''']) = L'''[I]$ and $dom(L''[R''']) = L'''[I]$. If two different dimension paths exist between the two dimension levels L and L''' then each level instance of L must roll up to the same level instance of L''' on both paths.

   Example: As depicted in Figure 3.3, the two dimension paths along which days can be rolled up coincide at dimension level Year, i.e., Day $<$ Week $<$ Year and Day $<$ Month $<$ Quarter $<$ Year. Consider that level instance Q-4-2001 of dimension level Quarter and level instance 2001 of dimension level Year belong to the two dimension paths 10-Oct-2001 $<$ WK-4-2001 $<$ 2001 and 10-Oct-2001 $<$ Oct-2001 $<$ Q-4-2001 $<$ 2001. The above property is violated if these two dimension paths, which originate at level instance 10-Oct-2001, destinate at different level instances of Year.

[1] $\mathcal{L}$ denotes the set of dimension levels that are available in the data warehouse.

A describing attribute typically represents additional information that describes the instances of an OLTP object type and that is needed for analytical purposes in the data warehouse (e.g., for selections such as Region.noOfInhabitants > 2.000,000). Figure 4.4 represents the syntax to specify level schemas. Consider only the first four production rules; the remaining production rules will be explained later.

Example: Figure 4.2 presents the specifications of level schemas Article, Category, and Producer as described by the conceptual dimensional model of Figure 3.3.

```
DEFINE LEVEL Article
  IDENTIFIED BY articleId: STRING
  ROLLUP TO category: Category, producer: Producer
  DESCRIBED BY name: STRING, pricePerUnit: NUMERIC;

DEFINE LEVEL Category
  IDENTIFIED BY name: STRING
  DESCRIBED BY descr: STRING;

DEFINE LEVEL Producer
  IDENTIFIED BY name: STRING;
```

Figure 4.2: Specifications of level schemas Article, Category, and Producer

4.1.2 Base Cubes

A base cube $b(B)$ represents the facts of a conceptual dimensional model using *base cube schema* $B = (D_1 : dom(D_1), ..., D_k : dom(D_m), M_1 : dom(M_1), ..., M_n : dom(M_n))$, where $dom(D_i)$ is the set of unique level identifiers $L_i[I]$ of some other dimension level L_i to which the cells of $b(B)$ *roll up*. The instances of a base cube, called the *cells* $c_i, ..., c_n$ represent transactional data which have been loaded from OLTP systems. While the measures of a fact are represented by numeric or categorical *measure attributes* M_i in the base cube schema, the dimensions of a fact are represented by dimension attributes $D_i : L_i[I]$, which relate the measure attributes with dimension levels L_i. A particular cell c_i is identified by the values of all of its dimension attributes, thus $\nexists c_i, c_j \in B : c_i \neq c_j, c_i[D_1] = c_j[D_1] \wedge ... \wedge c_i[D_k] = c_j[D_k]$. A particular instance of a measure attribute M_i for a given cell c_i is called the *measure value* of $c_i[M_i]$. The bottom three production rules of Figure 4.4 represent the syntax to specify base cube schemas.

Example: Figure 4.3 shows the specifications of base cube Sales with schema (product: Article, location: Store, time: Day, sales: NUMERIC, quantity: NUMERIC) and of base cube Offerings with schema (product: Article, location: Region, timeFrom: Day, timeTo: Day) as described by the conceptual dimensional model of Figure 3.3.

```
DEFINE BASE CUBE Sales
  DIMENSIONS product: Article, location: Store, time: Day
  MEASURES sales: NUMERIC, quantity: NUMERIC;

DEFINE BASE CUBE Offerings
  DIMENSIONS product: Article, location: Region, timeFrom: Day, timeTo: Day;
```

Figure 4.3: Specifications of base cube schemas Sales and Offerings

4.2 Multidimensional Analyzes

Analyzing the data in a data warehouse is typically carried out with the help of an OLAP tool, that visualizes data multidimensionally (i.e., "cube") and that assists the analyst in navigating along dimension paths by providing some high-level operations such as "rollup" (i.e., aggregate measure values of a cube along the hierarchies of the cube's dimensions), "drilldown" (i.e., de-aggregate measure values of a cube by navigating to subordinate dimension levels of the cube's dimensions), or "slice" (i.e., select the subset of a cube's cells that satisfy a particular selection condition). Such tools split up the analytical task of a decision maker into two steps:

Step 1: Determine the data that are relevant to decision making, i.e., create the cubes that provide information concerning the decision task.

Step 2: "Analyze" these cubes, i.e., evaluate some conditions on the measure values of the cells of these cubes to identify those cases for which a definite decision can be made and to identify those cases for which further analyzes are necessary.

We adopted this approach of decision making in our model, since it reduces the complexity of decision making. So far, all query languages that have been proposed for data warehousing and OLAP [AGS97, Vas98, DT99, GCB+97, LW96, RSC98, BL97, GL98, CT98c, JLS99] do not address this separation. The resulting queries are thus complex and hard to understand. Since the typical OLAP user is not an experienced database programmer, our approach not only helps in structuring decision tasks but also increases the acceptance of end users. Nevertheless, the mentioned query language approaches may be used in a later phase to implement the cubes and analyzes as specified using our approach. In Section 4.2.1, we describe how cubes are created, whereas in Section 4.2.2, we describe how analyzes can be specified by referring to these cubes.

Syntax

```
LevelDef       ::=  DEFINE LEVEL levelName
                    IDENTIFIED BY IdAttrDef
                    [ ROLLUP TO RollAttrDef { "," RollAttrDef}]
                    [ DESCRIBED BY DescAttrDef { "," DescAttrDef}] ";".
IdAttrDef      ::=  idAttrName ":" dataType.
RollAttrDef    ::=  rollAttrName ":" levelName.
DescAttrDef    ::=  descAttrName ":" dataType.
BaseCubeDef    ::=  DEFINE BASE CUBE baseCubeName
                    DIMENSIONS DimAttrDef { "," DimAttrDef}
                    [ MEASURES MeasAttrDef { "," MeasAttrDef} ] ";".
DimAttrDef     ::=  dimAttrName ":" levelName.
MeasAttrDef    ::=  measAttrName ":" dataType.
```

Figure 4.4: Syntax to specify level schemas and base cube schemas

4.2.1 Specifying Cubes

A *cube* arranges data in the way users think of their enterprise, i.e., dimensions and facts. Although the physical metaphor of cube lets one assume that the number of dimensions is restricted to be exactly three, in data warehousing and OLAP, "cube" is commonly referred to as an n-dimensional (n $\geq$ 1) representation of data. In our approach, we distinguish between two kinds of cube: (1) Cubes that store factual data, which has been loaded from OLTP systems. (2) Cubes that represent cells from base cubes by rolling up the values of a base cube's dimension attributes to level instances of a superordinate dimension level or by selecting a subset of cells of any given cube.

The first kind of cube is called *base cube*. We introduced this kind of cube in Section 4.1.2 and will thus not refer to base cubes in this section. The second kind of cube is derived from base cubes and is simply called *cube*. Such cubes are particularly used to specify the data on which analyzes of analysis rules in an active data warehouse should be carried out.

Specifying cubes in the process of decision support activities is carried out *incrementally*, i.e., new cubes are derived from existing cubes by applying OLAP operators. An OLAP operator is a conceptual operation on a cube $r(R)_{old}$ in order to create a new cube $r(R)_{new}$ that provides measures (a) using a more detailed dimension level as provided by $r(R)_{old}$,

(b) using a less detailed dimension level as provided by $r(R)_{old}$, or (c) using a subset of the level instances of a dimension level provided by $r(R)_{old}$.

Cubes

Specifying cubes incrementally requires an appropriate representation schema, which can be afterwards translated into a particular data model. Although there exists a basic agreement on fundamental multidimensional concepts in literature, the various definitions of *cube* [GCB+97, AGS97, Vas98] are diverging and are not sufficient to *specify* cubes incrementally but they may be used to *realize* an incrementally specified cube. In our data model, every cube $r(R)$ consists of a *cube definition* and a *cube extension*, which are defined below.

Cube Definition. The cube definition is a 4-tuple $\langle R, b(B), C, AGG \rangle$ that represents the following components:

1. The cube schema $R = (D_1 : dom(D_1), ..., D_n : dom(D_n), M_1 : dom(M_1), ..., M_k : dom(M_k))$, which defines the dimension attributes D_i and the measure attributes M_i; $dom(D_i)$ is the set of unique level identifiers $L_i[I]$ of some dimension level L_i to which the cells of $r(R)$ refer in dimension attribute D_i; $dom(M_i)$ defines the numeric or categorical domain of measure attribute M_i. Note that we consider only numeric attributes in this thesis.

2. The base cube $b(B)$ (B is the base cube schema), from which cells are taken for aggregation.

3. The set of conditions C, which determine the subset of cells of $b(B)$ satisfying the condition $\bigwedge_{cond \in C} cond$ for aggregation (denoted as $b(B)^C$), whereby each condition $cond_i$ restricts the domain of a particular dimension attribute D_i by an expression over one dimension level in the dimension hierarchy to which the domain of D_i belongs. Note that if some condition $cond_i$ is defined upon some higher-level dimension level L_i, the cells of $b(B)$ are rolled up to the level instances of L_i that satisfy $cond_i$ for determining the cells of $b(B)^C$.

4. The set of aggregation functions AGG that contains pairs $\langle M_i, f_i \rangle$, one for each measure attribute M_i of R where f_i is the definition of the aggregation function that is applied on a single measure attribute of the base cube or on an arithmetic expression of measure attributes of the base cube.

Example: Suppose that an analyst needs to inspect quarterly sales figures of articles that were sold in Upper Austrian stores. For this purpose, cube SalesArticlesUpperAustriaQuarters2001 is defined as follows:

1. The *schema* of this cube is (P: Article, L: Region, T: Quarter, SalesTotal: ATS). Attributes P, L, and T are dimension attributes; attribute SalesTotal is a measure attribute with a numeric domain.
2. *Base cube* Sales provides the cells for aggregation.
3. *Condition* L:Region.name = 'Upper Austria' restricts the domain of dimension attribute L to region Upper Austria; condition T:Year.year = '2001' restricts the domain of dimension attribute T to year 2001. Consequently, only cells that belong to region Upper Austria and that refer to year 2001 will be taken from base cube Sales.
4. The set of *aggregation functions* contains the aggregation function for measure attribute SalesTotal, i.e., ⟨ SalesTotal, SUM(sales) ⟩. SUM represents the aggregation function that will be evaluated using the projected values of measure attribute sales taken from the cells of base cube Sales.

Note that cube schema R possesses a dimension attribute for every dimension attribute that is provided by base cube schema B. If a particular dimension attribute D_i^B of B is not important for analysis by cube $r(R)$, the domain of the corresponding dimension attribute D_i^R in R refers to the set of identifying attributes of the top-level dimension level L_{sup} (e.g., ALL-Articles). In such a situation, the cells of $b(B)$ are rolled up to the single level instance l_{sup} (e.g., all-Articles) for dimension attribute D_i^R.

Example: The cells of cube SalesArticlesUpperAustriaQuarters2001 (cf. previous example) describe sales totals per Article, Region, and Quarter restricted to region Upper Austria and to year 2001. As a variation of this example assume that the granularity of the time dimension (represented by dimension attribute T) is not important for this analysis. Hence, dimension attribute T refers to level instance all-Days of top-level dimension level ALL-Days. Hence the cells of this cube would describe sales totals per Article and Region without considering days, weeks, months, etc. of sales.

Cube Extension. The set of cells that represent the extension of some cube $r(R)$ are derived from the cells of base cube $b(B)$. A basic assumption of this approach is that base cubes provide complete information, i.e., there are no missing cells and no null values of measure attributes of some cells. Before we can determine the set of cells in $r(R)$ that are derived from base cube $b(B)$, we need to introduce the $<$-relationship ("rolls up") between pairs of cells c_S, c_R. This is needed to determine the set of cells of base cube $b(B)$ that are mapped to a single cell of cube $r(R)$, which is represented by the $<$-relationship. Some cell $c_S \in s(S)$ "rolls up" to some other cell $c_R \in r(R)$, denoted as $c_S < c_R$, if and only if $c_S[D_i] \leq c_R[D_i] \forall i \in (1..n)$. The predicate *rolls-up-to-from*$(c_R, s(S))$ (short *rolls-up*$(c_R, s(S))$) denotes the set of cells of cube $s(S)$ that roll up to cell c_R, i.e., *rolls-up*$(c_R, s(S)) = \{c_S \in s(S) \mid c_S < c_R\}$. To be used later in this thesis, we extend the predicate *rolls-up* to sets of cells U, i.e., *rolls-up*$(U, s(S)) = \bigcup_{u \in U}$ *rolls-up*$(u, s(S))$.

Example: Consider base cube Sales with schema (product:Article, location:Store, time:Day, ...) and cube SalesArticlesUpperAustriaQuarters2001 whose schema was defined in the example above. Note that we only consider dimension attributes in this example. Assume that cells ⟨ AJD, S-0001, 3-Jan-2001 ⟩, ⟨ AJD, S-0002, 1-Feb-2001 ⟩, and ⟨ AJD, S-0003, 5-Apr-2001 ⟩ belong to cube Sales whereas cells ⟨ AJD, Upper Austria, Q1-2001 ⟩ and ⟨ AJD, Upper Austria, Q2-2001 ⟩ belong to cube SalesArticlesUpperAustriaQuarters2001. Further assume that the two stores S-0001 and S-0002 roll up to region Upper Austria (i.e., S-0001 < Upper Austria and S-0002 < Upper Austria) and that 3-Jan-2001 < Q1-2001, 1-Feb-2001 < Q1-2001, and 5-Apr-2001 < Q2-2001. For pairs of cells c_{Sales}, $c_{\text{SalesUpperAustriaQuarters2001}}$, the following <-relationships exist: ⟨ AJD, S-0001, 3-Jan-2001 ⟩ < ⟨ AJD, Upper Austria, Q1-2001 ⟩, ⟨ AJD, S-0002, 1-Feb-2001 ⟩ < ⟨ AJD, Upper Austria, Q1-2001 ⟩, and ⟨ AJD, S-0003, 5-Apr-2001 ⟩ < ⟨ AJD, Upper Austria, Q2-2001 ⟩. Thus, *rolls-up*(⟨ AJD, Upper Austria, Q1-2001 ⟩, Sales) = { ⟨ AJD, S-0001, 3-Jan-2001 ⟩, ⟨ AJD, S-0002, 1-Feb-2001 ⟩ } and *rolls-up*(⟨ AJD, Upper Austria, Q2-2001 ⟩, Sales) = { ⟨ AJD, S-0003, 5-Apr-2001 ⟩ }.

Now, the set of cells that belong to cube $r(R)$ are derived from the cells of the base cube $b(B)$ as follows:

1. Determine the set of cells in the projection of $r(R)$ onto the dimension attributes $D_1, ..., D_n$ that satisfy condition $\bigwedge_{cond \in C} cond$, i.e., $r(R)^C_{dim} \subseteq dom(D_1) \times ... \times dom(D_n)$. Evaluating the conditions of C is as follows:
   (a) If some condition $cond_i$ is defined over dimension level L_i whose level instances also define the domain of the corresponding dimension attribute D_i in cube schema R, select those cells in the projection of $r(R)$ onto the dimension attributes $D_1, ..., D_n$ that satisfy $cond_i$.
   (b) If some condition $cond_i$ is defined over some dimension level L_i whose level instances are superordinate to the level instances in the domain of D_i, the cells of $r(R)^C_{dim}$ are determined as the set of cells in the projection of $r(R)$ onto the dimension attributes $D_1, ..., D_n$, where the dimension values of D_i roll up to level instances of L_i satisfying $cond_i$.
   (c) If some condition $cond_i$ is defined over some dimension level L_i whose level instances are sub ordinate to the level instances in the domain of D_i, the cells of $r(R)^C_{dim}$ are determined as the cells in the projection of $r(R)$ onto the dimension attributes $D_1, ..., D_n$, where the level instances of L_i satisfying $cond_i$ roll up to the dimension values of D_i.
2. The measure values m_1, ..., m_k of a cell $c_R \in r(R)$ are calculated by applying the aggregation functions $< f_1, M_{f_1} >, ..., < f_k, M_{f_k} > \in AGG$ on *rolls-up*($c_{R^C_{dim}}$, $b(B)^C$) such that $r(R) = \{c_R \mid \exists c_{R^C_{dim}} \in r(R)^C_{dim} : c_R[D_i] = c_{R^C_{dim}}[D_i]\ \forall i = (1..n) \wedge c_R[M_j] = f_j(\textit{rolls-up}(c_{R^C_{dim}}, b(B)^C)[M_{f_j}])\ \forall j = (1..k)\}$.

Example: Again consider base cube Sales and cube SalesArticlesUpperAustriaQuarters2001 with schema (P: Article, L: Region, T: Quarter, SalesTotal: ATS). Further assume that the three level instances AJD, PAF, and LES represent the extension of dimension level Article. The extension of cube SalesArticlesUpperAustriaQuarters2001 is now determined as follows:

1. The cartesian product among the level instances of dimension level Article, Region, and Quarter is created, i.e., ⟨ AJD, Lower Austria, Q1-2001 ⟩, ⟨ AJD, Upper Austria, Q1-2001 ⟩, ..., ⟨ LES, Upper Austria, Q4-2001 ⟩.
2. From this set of cells, only those are selected that satisfy conditions L:Region.name = 'Upper Austria' and T:Year.year = '2001'. The first condition is evaluated directly using the cells of the cartesian product. The second condition refers to the higher-level dimension level Year to which level instances of Quarter roll up. Hence, only those cells of the cartesian product will be selected whose dimension values at dimension attribute T roll up to level instance 2001, which belongs dimension level Year.
3. The measure attribute SalesTotal of such a cell c_i will be determined by evaluating aggregation function SUM on measure attribute sales of the cells from base cube Sales that roll up to c_i, e.g., SalesTotal = SUM(*rolls-up*(⟨ AJD, Upper Austria, Q1-2001 ⟩, sales)) = SUM(⟨ AJD, S-0001, 3-Jan-2001, 2500 ⟩, ⟨ AJD, S-0002, 1-Feb-2001, 1300 ⟩, ⟨ AJD, S-0001, 3-Jan-2001, 2500 ⟩), sales) = 3800. Evaluation of aggregation function SUM will be carried out correspondingly for the remaining cells.

OLAP Operators

OLAP operators are used to specify cubes incrementally, i.e, a new cube $r'(R')$ is derived from an existing cube $r(R)$ by applying an OLAP operator on $r(R)$. Note that, although *specified* incrementally, cubes will be *created* bottom-up from the base cube as described in the previous section. Figure 4.10 shows the syntax to specify cubes incrementally. We will explain this syntax using various examples below.

During incremental specification of a cube $r(R')$ from cube $r(R)$, an analyst may find it necessary to extend the schema of cube $r(R')$ beyond those attributes carried over from $r(R)$ with additional measure attributes $M_{new_1}, ..., M_{new_n}$, which are appended to the list of measure attributes taken from cube schema R to yield cube schema R'. For each such newly defined measure attribute, an additional aggregation function $\langle M_{new}, f_{new} \rangle$ is added to the set of aggregation functions AGG. The syntax to specify a cube's schema and to extend a cube's schema with an additional measure attribute is provided in Figure 4.10 by production rules CubeSchemaDef and SchemaExtDef, respectively.

If applied on cube $r(R)$, an OLAP operator copies the cube definition of $r(R)$ (i.e., schema, base cube, conditions, aggregation functions) into the cube definition of the new cube $r'(R')$ and afterwards modifies the components of the cube definition of $r'(R')$ according to the

semantics of the OLAP operator. Since base cubes do not define conditions, aggregation functions, and base cubes, an OLAP operator that refers to some base cube $b(B)$ is applied to an intermediate cube $s(S)$ that represents $b(B)$. Cube $s(S)$ is defined as follows:

- $S = (D_1 : dom(D_1), ..., D_m : dom(D_m), M_1 : dom(M_1), ..., M_n : dom(M_n)$ with $dom(S[D_i]) = dom(B[D_i])\ \forall i = (1..m)$ and $dom(S[M_j]) = dom(B[M_j])\ \forall j = (1..n)$
- $b(B)$ is the base cube
- $C = \emptyset$
- $AGG = \emptyset$

Example: Consider cube Sales with schema (product:Article, location:Store, time:Day, sales:ATS, quantity:NUMERIC). Intermediate cube Sales′, which represents base cube Sales, consists of schema P:Article, L:Store, T:Day, SalesTotal:NUMERIC, base cube Sales, an empty set of conditions, and an empty set of aggregation functions.

Rollup. OLAP operator *Rollup*$(r(R), \{\langle D_i, L_i\rangle, ..., \langle D_k, L_k\rangle\})$ defines a new cube $r'(R')$, where $r(R)$ represents the existing cube and $\{\langle D_i, L_i\rangle, ..., \langle D_k, L_k\rangle\}$ represents a non-empty set of pairs $\langle D_i, L_i\rangle$ each identifying a particular dimension attribute D_i of R and R' and the new dimension level L_i of D_i in R', i.e., $dom(R'[D_i]) = L_i$ for which the condition $dom(R[D_i]) < dom(R'[D_i])$ must hold. Base cube $b(B)$, aggregation functions AGG, and conditions C are the same in $r(R)$ and $r'(R')$.

Example: Consider cube SalesArticlesRegionsQuarters with schema (P: Article, L: Region, T: Quarter, SalesTotal: ATS), base cube Sales, and aggregation function ⟨ SalesTotal, SUM(sales), 0, sales ⟩. OLAP operator *Rollup*(SalesArticlesRegionsQuarters, {⟨ T, Year ⟩}) defines a new cube SalesArticlesRegionsYears with schema (P: Article, L: Region, T: Year, SalesTotal: ATS), base cube Sales, and aggregation function ⟨ SalesTotal, SUM(sales) ⟩. The upper statement in Figure 4.5 shows how cube SalesArticlesRegionsYears is specified upon cube SalesArticlesRegionsQuarters. The lower statement in The upper part of Figure 4.5 presents how cube SalesArticlesRegionsYears is specified upon base cube Sales. The cube schema is given in parentheses next to the cube's name. Dimension attributes P, L, and T refer to the corresponding dimension attributes of base cube Sales; measure attribute SalesTotal is specified from scratch. Hence, the schema of intermediate cube Sales' is (P: Article, L: Store, T: Day, SalesTotal: ATS) and the schema of cube SalesArticlesRegionsYears is (P: Article, L: Region, T: Year, SalesTotal: ATS) after applying OLAP operator *Rollup*(Sales′, {⟨ L, Region ⟩, ⟨ T, Year ⟩}). The lower part of Figure 4.5 presents the corresponding cube definition.

```
DEFINE CUBE SalesArticlesRegionsYears AS
  ROLLUP T TO Year
  FROM SalesArticlesRegionsQuarters;

or

DEFINE CUBE SalesArticlesRegionsYears
(P product, L location, T time, SalesTotal SUM(sales)) AS
  ROLLUP L TO Region,
         T TO Year
  FROM Sales;
```

cube definition of SalesArticlesRegionsYears

- *schema* R = (P: Article, L: Region, T: Year, SalesTotal: ATS)
- *base cube* sales
- *conditions* $C = \emptyset$
- *aggregation functions* AGG = {⟨ SalesTotal, SUM(sales) ⟩}

Figure 4.5: Alternative ways to specify cube SalesArticlesRegionsYears using ROLLUP

Drilldown. OLAP operator *Drilldown*$(r(R), \{\langle D_i, L_i\rangle, ..., \langle D_k, L_k\rangle\})$ defines a new cube $r'(R')$, where $r(R)$ represents the existing cube and $\{\langle D_i, L_i\rangle, ..., \langle D_k, L_k\rangle\}$ represents a non-empty set of pairs $\langle D_i, L_i\rangle$ each identifying a particular dimension attribute D_i of R and R' and the new dimension level L_i of D_i in R', i.e., $dom(R'[D_i]) = L_i$ for which the condition $dom(R'[D_i]) < dom(R[D_i])$ must hold. Base cube $b(B)$, aggregation functions AGG, and conditions C are the same in $r(R)$ and $r'(R')$.

Example: Consider cube SalesArticlesRegionsYears with schema (P: Article, L: Region, T: Year, SalesTotal: ATS), base cube Sales, and aggregation function ⟨ SalesTotal, SUM(sales) ⟩. OLAP operator *Drilldown*(SalesArticlesRegionsYears, {⟨ T, Week ⟩}) defines a new cube SalesArticlesRegionsWeeks with schema (P: Article, L: Region, T: Week, SalesTotal: ATS), base cube Sales, and aggregation function ⟨ SalesTotal, SUM(sales) ⟩. The upper part of Figure 4.6 presents the specification of this cube. The lower part of Figure 4.6 presents the corresponding cube definition.

Slice. OLAP operator *Slice*$(r(R), \{\ cond_1, ..., cond_n\})$ defines a new cube $r'(R')$, where $r(R)$ represents the existing cube and $\{$ cond$_1$, ..., cond$_n\}$ represents a non-empty set of conditions that will be added to the conditions C' of $r'(R')$, i.e., $C' = C \cup \{\ cond_1, ...,$

```
DEFINE CUBE SalesArticlesRegionsWeeks AS
  DRILLDOWN T TO Week
  FROM SalesArticlesRegionsYears;
```

cube definition of SalesArticlesRegionsWeeks

- *schema* R = (P : Article, L : Region, T : Week, SalesTotal : ATS)
- *base cube* sales
- *conditions* $C = \emptyset$
- *aggregation functions* AGG = {⟨ SalesTotal, SUM(sales) ⟩}

Figure 4.6: Specification of cube SalesArticlesRegionsWeeks using DRILLDOWN

$cond_n\}$. Cube schema R', base cube $b(B)$, and aggregation functions AGG are the same in $r(R)$ and $r'(R')$.

A *Slice*-condition may be specified on the attributes of a dimension level L_i that belongs to a dimension path of some dimension attribute $R[D_i]$ (i.e., $dom(R'[D_i]) < L_i$, $dom(R'[D_i]) = L_i$, or $dom(R'[D_i]) > L_i$). The syntax to specify a *Slice*-condition is defined by production rule DimConditionDef in Figure 4.10. We distinguish three patterns for specifying *Slice*-conditions:

1. Restrict the domain of a dimension attribute using a constant value.

   Example: Consider cube SalesArticlesRegionsQuarters with schema (P: Article, L: Region, T: Quarter, SalesTotal: ATS), base cube Sales, an empty set of conditions, and aggregation function ⟨ SalesTotal, SUM(sales) ⟩. OLAP operator *Slice*(SalesArticlesRegionsQuarters, { L:Region.name = 'Upper Austria', T:Year.year = '2001' }) defines a new cube SalesArticlesUpperAustriaQuarters2001 with schema (P: Article, L: Region, T: Week, SalesTotal: ATS), base cube Sales, conditions { L:Region.name = 'Upper Austria', T:Year.year = '2001' }, and aggregation function ⟨ SalesTotal, SUM(sales) ⟩. The upper part of Figure 4.7 presents the syntactical specification of this cube. The lower part of Figure 4.7 presents the corresponding cube definition.

2. Restrict the domain of a dimension attribute by referring to another dimension attribute of the same cell (i.e., *self-join*).

   Example: Consider base cube Offerings. OLAP operator *Slice*(Offerings, { timeFrom:Year.year = timeTo:Year.year }) creates a new cube OfferingsSameYear, which

represents offerings of articles per region. A cell c_i belongs to this cube if it describes an offering period that begins (i.e., timeFrom) and ends (i.e., timeTo) in the same year.

3. Restrict the domain of a dimension attribute by referring to the dimension attribute of another cube (i.e., *drill-across*).

   Example: Consider cube SalesArticlesRegionsQuarters and base cube Offerings. OLAP operator *Slice* (SalesArticlesRegionsQuarters, { P:Article.articleId = Offerings.product:Article.articleId, L:Region.name = Offerings.region:Region.name, T:Day.date $\geq$ Offerings.timeFrom:Day.date, T:Day.date $\leq$ Offerings.timeTo:Day.date }) creates a new cube ValidSalesArticlesRegionsQuarters, which represents sales of articles per region and per quarter. A cell c_i belongs to this cube if it describes the sale of an article in a particular region in which the article has also been offered (the date of the sale is also restricted to the offering period). Although the active domains of cubes SalesArticlesRegionsQuarters and ValidSalesArticlesRegionsQuarters are identical, the passive domain of cube ValidSalesArticlesRegionsQuarters (i.e., cartesian product of Articles, Regions, and Quarters) is a subset of the passive domain of cube SalesArticlesRegionsQuarters (i.e., cartesian product of Articles, Regions, and Quarters restricted to the facts provided by base cube Offerings).

Note that our syntax allows specifying conditions using the short notation D_i = value if value is taken from the domain of dimension attribute $D_i : L_i$, which represents the identifiers of the level instances of L_i. In our data model, such a condition is represented as D_i:L_i.idAttr = value as described above.

Example: The first *Slice*-condition in Figure 4.7 restricts the domain of dimension attribute L (dimension level Region) to level instances with identifyer 'Upper Austria'. It is specified using the short notation L = 'Upper Austria', but is represented in our data model as L:Region.name = 'Upper Austria'. In contrast, the second *Slice*-condition restricts the domain of dimension attribute T (dimension level Quarter) to such quarters, that roll up to year 2001 (dimension level Year). A short notation is not applicable here.

Intersection. To define OLAP operator *Intersection*, which may be applied to two cubes, we first need to introduce function *most specific* (msc:$\mathcal{L} \times \mathcal{L} \rightarrow \mathcal{L}$) and schema operator *msc-merge* ($\oplus\downarrow$). Function *most specific* (msc) determines the most specific dimension level of two comparable dimension levels L_1 and L_2 as

$$msc(L_1, L_2) = \begin{cases} L_1 & \text{if } L_1 < L_2 \\ L_2 & \text{if } L_1 > L_2 \\ L_1 & \text{if } L_1 = L_2 \end{cases}$$

```
DEFINE CUBE SalesArticlesUpperAustriaQuarters2001 AS
  SLICE L = 'Upper Austria' AND T:Year.year = '2001'
  FROM SalesArticlesRegionsQuarters;
```

cube definition of SalesArticlesUpperAustriaQuarters2001

- *schema* R = (P : Article, L : Region, T : Quarter, SalesTotal : ATS)
- *base cube* sales
- *conditions* C = {L : Region.name = 'Upper Austria', T : Year.year = '2001'}
- *aggregation functions* AGG = {⟨ SalesTotal, SUM(sales) ⟩}

Figure 4.7: Specification of cube SalesArticlesUpperAustriaQuarters2001 using SLICE

Operator $\oplus\downarrow$ merges two cube schemas S_1 and S_2 such that a new most specific cube schema S_3 will be created, denoted as $S_3 = S_1 \oplus\downarrow S_2$, iff (1) $S_1[D_i] = S_2[D_i]\ \forall i \in (1..n)$ and (2) $dom(S_1[D_i])$ is comparable with $dom(S_2[D_i])\ \forall i \in (1..n)$. In such a situation the cube schemas S_1 and S_2 are also comparable. The new cube schema S_3 is defined as $(D_1 : msc(dom(S_1[D_1]), dom(S_2[D_1])), ..., D_n : msc(dom(S_1[D_n]), dom(S_2[D_n])), X_1, ..., X_r, Y_1, ..., Y_s, Z_1, ..., Z_t)$, where (1) $X_1, ..., X_r$ represent the measure attributes that are defined in S_1 and S_2 (Notice that each X_i must be defined upon the same aggregation function in S_1 and S_2), (2) $Y_1, ..., Y_s$ represent the measure attributes that are defined in S_1 but not in S_2, and (3) $Z_1, ..., Z_t$ represent the measure attributes that are defined in S_2 but not in S_1.

Example: Consider cube SalesArticlesCitiesMonths with schema (P: Article, L: City, T: Month, SalesTotal, SalesAvg) and cube SalesArticlesRegionsQuarters with schema (P: Article, L: Region, T: Quarter, SalesTotal, MaxSale). If the two cube schemas will be merged ($\oplus\downarrow$), the resulting cube schema will be (P: Article, L: City, T: Month, SalesTotal, SalesAvg, MaxSale). Figure 4.8 illustrates the this operation graphically. Note that only dimension paths that are relevant to this example are depicted. Bold lines indicate the granularity of the respective cube schemas.

OLAP operator $Intersection(\{r_1(R_1), ..., r_n(R_n)\})$ defines a new cube $r'(R')$, where $\{r_1(R_1), ..., r_n(R_n)\}$ represents the non-empty set of cubes that should be merged. The components of $r'(R')$ are defined as follows:

- $R' = R_1 \oplus\downarrow \ ... \oplus\downarrow \ R_n$
- $b(B)' = b(B)_i$

Figure 4.8: Merging the schemas of cubes SalesArticlesRegionsQuarters and SalesArticlesCitiesMonths

- $AGG' = AGG_1 \cup ... \cup AGG_n$
- $C' = C_1 \cup ... \cup C_n$

Example: The upper part of Figure 4.9 shows the syntactical specification of cube SalesArticlesUpperAustrianCitiesMonths2001 using OLAP operator *Intersection*({ SalesArticlesCitiesMonths, SalesUpperAustriaQuarters2001 }). Cube SalesArticlesCitiesMonths is defined upon schema (P: Article, L: City, T: Month, SalesTotal, SalesAvg), base cube Sales, an empty set of conditions, and aggregation functions { ⟨ SalesTotal, SUM(sales) ⟩, ⟨ SalesAvg, AVG(sales) ⟩ }. The definition of cube SalesUpperAustriaQuarters2001 (presented in the lower part of Figure 4.9) is the same as in the examples above. Cube SalesArticlesUpperAustrianCitiesMonths2001 is defined upon schema (P: Article, L: City, T: Month, SalesTotal, SalesAvg), base cube Sales, conditions { L:Region.name = Upper Austria, T:Year.year = 2001 }, and aggregation functions { ⟨ SalesTotal, SUM(sales) ⟩, ⟨ SalesAvg, AVG(sales) ⟩ }.

```
DEFINE CUBE SalesArticlesUpperAustrianCitiesMonths2001 AS INTERSECTION
  FROM SalesArticlesCitiesMonths, SalesUpperAustriaQuarters2001;
```

cube definition of SalesArticlesUpperAustrianCitiesMonths2001

- *schema* R = (P : Article, L : City, T : Month, SalesTotal : ATS)
- *base cube* sales
- *conditions* C = {L : Region.name = 'Upper Austria', T : Year.year = '2001'}
- *aggregation functions* AGG = {⟨ SalesTotal, SUM(sales) ⟩}

Figure 4.9: Specification of cube SalesArticlesUpperAustrianCitiesMonths2001 using INTERSECTION

Syntax

CubeDef	::=	DEFINE CUBE cubeName (CubeSchemaDef | SchemaExtDef) AS (RollupDef | DrilldownDef | SliceDef | IntersectionDef) ";".
CubeSchemaDef	::=	"(" DimAliasDef { "," DimAliasDef } "," MeasAggDef { "," MeasAggDef } ")".
SchemaExtDef	::=	"(" MeasAggDef { "," MeasAggDef } ")".
DimAliasDef	::=	dimAttrAlias baseCubeDimAttrName.
MeasAggDef	::=	measAttrAlias AggFunction BaseCubeMeasExprDef.
BaseCubeMeasExprDef	::=	"(" (baseCubeMeasAttrName | BaseCubeMeasExprDef arithmeticOp BaseCubeMeasExprDef) ")".
RollupDef	::=	ROLLUP dimAttrAlias TO levelName {"," dimAttrAlias TO levelName } FROM baseCubeName ";".
DrilldownDef	::=	DRILLDOWN dimAttrAlias TO levelName {"," dimAttrAlias TO levelName } FROM baseCubeName";".
SliceDef	::=	SLICE DimConditionDef {AND DimConditionDef } FROM baseCubeName ";".
DimConditionDef	::=	DimAttrRefDef compOper constantValue | DimAttrRefDef compOper DimAttrRefDef | DimAttrRefDef compOper ExtCubeDimAttrRefDef.
DimAttrRefDef	::=	dimAttrAlias[":"levelName"."attrName].
ExtCubeDimAttrRefDef	::=	extCubeName"."DimAttrRefDef.
IntersectionDef	::=	INTERSECTION FROM $baseCubeName_1$ "," $baseCubeName_2$ { "," $baseCubeName_i$ }.

Figure 4.10: Syntax to specify cubes incrementally

4.2.2 Analyzing Cubes

After a cube has been defined, an analyst can carry out its analyzes using the cells of this cube either by referring directly to the measure attributes of this cube or, for more complex analyzes, by first deriving "auxiliary cubes" from this cube and then by specifying separate formulas over the measure attributes of cells of those auxiliary cubes. Examples of such analyzes are to evaluate an analytical function (e.g., RANK()) using the measure values of cells, grouping the cells of a cube, comparing some cells of a cube with some cells of another cube, etc. To specify such analyzes, we introduce a declarative analysis language for writing *cube analysis statements*.

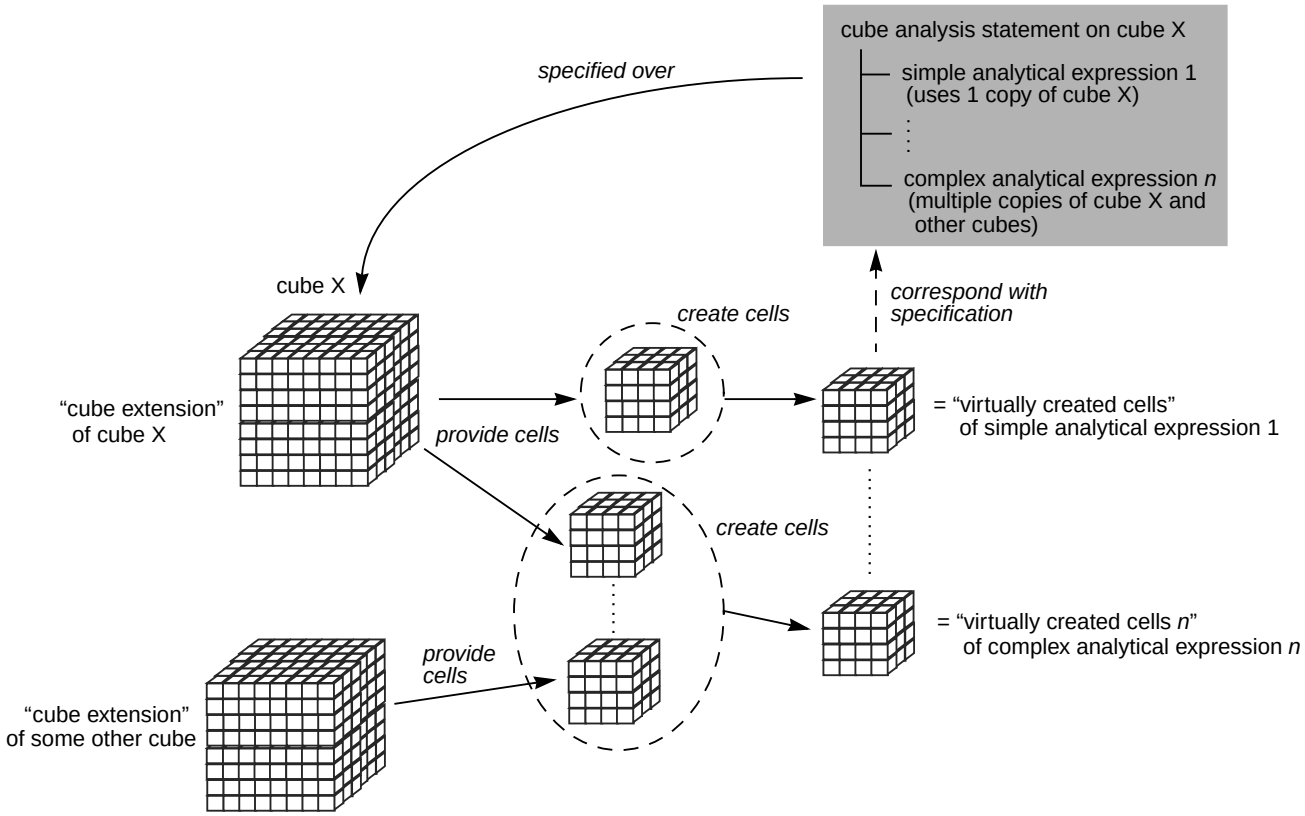

Figure 4.11: Principal Idea behind Cube Analysis Statements/Analytical Expressions

The basic idea behind a cube analysis statement is that an analyst selects a particular cube whose cells are then analyzed by an arbitrary number of formulas, which we call *analytical expressions*. An analytical expression may be compared with a query that creates "virtual cells" upon the cells of the cube for which the cube analysis statement is specified. In its simplest form, an analytical expression uses a single auxiliary cube which is identical to the main cube (i.e., the cube for which the cube analysis statement is specified). Such analytical expressions are referred to as *simple analytical expression*. A *complex analytical*

expression uses several auxiliary cubes, which are generated by OLAP operators SLICE and/or ROLLUP from the main cube or from any other cube. The analytical expression may relate the cells of these cubes and apply formulas to these cells so related. Figure 4.11 presents the principal idea behind cube analysis statements. Note that the data upon which analytical expressions are defined are the cells of the cube to which the cube analysis statement refers, while the data upon which a cube is defined are the cells of this cube's *base cube*. In the following, we will explain analytical expressions informally using various examples.

Basic Concepts

In its simplest form, an analytical expression implicitly refers that copy of the cube for which the cube analysis statement is defined, whereby for each cell of this cube a virtual cell is created. The specification of such an analytical expression only consists of a formula, which defines the measure attribute for these cells in the same way as a cube's measure attributes are defined. The dimension attributes of these virtual cells are taken from the dimension attributes of the cube's cells. The formula is evaluated for each cell of the cube's expanded active domain. The syntax to specify cube analysis statements is shown in Figure 4.17. Clause ANALYZE identifies the cube whose cells will be used to create the virtual cells of subsequent analytical expressions, which are specified in squared brackets ([...]).

Example: Consider cube SalesArticlesRegionsQuarters, which defines measure attribute SalesTotal for its cells. The upper part of Figure 4.12 presents a cube analysis statement, which consists of a single analytical expression. This analytical expression refers to measure attribute SalesRank using function RANK(SalesTotal). This function determines the rank of cells (i.e., 1, 2, 3, ...) according to the values of measure attribute SalesTotal of cube SalesArticlesRegionsQuarters. The lower part of Figure 4.12 presents the result of evaluating this analytical expression.

Advanced Concepts

When analyzing data for decision support, analysts inspect not only the measure values of a cube's cells separately, they also (i) compare the measure values of a subset of cells (i.e., "slicing"), (ii) they aggregate the measure values of cells up to some higher levels of granularities (i.e., "rollup"), or (iii) they relate the cells of a cube with the cells of another cube (i.e., "drill across") in order to get a better insight into business data. In the following, we introduce syntactical extensions to analytical expressions with which analysts may perform such advanced analyzes.

```
ANALYZE SalesArticlesRegionsQuarters [ SalesRank RANK(SalesTotal); ]
```

SalesArticlesRegionsQuarters2001

P:Article	L:Region	T:Quarter	SalesTotal	SalesRank
AJD	UpperAustria	Q1-2001	6075	2
AJD	UpperAustria	Q2-2001	6200	1
AJD	UpperAustria	Q3-2001	0	99
AJD	UpperAustria	Q4-2001	5980	3
AJD	LowerAustria	Q1-2001	4100	5
AJD	LowerAustria	Q2-2001	0	99
AJD	LowerAustria	Q3-2001	0	99
AJD	LowerAustria	Q4-2001	4320	4
...	...	...	...	...

Figure 4.12: Simple analytical expression

Identifying cells. A single cell is identified through the values of its dimension attributes since the combination of these values is unique among the cells of a cube. In an analytical expression, an individual cell is identified by specifying a *Slice*-condition for each dimension attribute of the cube using clause SLICE (to specify *Slice*-conditions, see the definition of OLAP operator *Slice* in Section 4.2.1). Hence, a formula is not evaluated for all the cells of the cube but only for the cell that has been identified instead. If not all dimension attributes have been specified in the SLICE-clause, the *set* of cells that are identified through these coordinates are selected instead.

Example: Consider the first analytical expression in Figure 4.13. The three *Slice*-conditions specify for each dimension attribute of cube SalesArticlesRegionsQuarters2001 a single value and thus identify the cell that represents the SalesTotal of article AppleJuiceDeluxe in region Upper Austria of quarter Q1-2001. These values are the dimension values of the virtual cell that is selected by this analytical expression.

If the formula of a virtual cell's measure attribute is defined upon the measure values of several individual cells, clause FOR CELLS is used to assign a unique name – the *cell identifier* – to each required cell. For each cell identifier, the analyst may specify a separate SLICE-clause. If all dimension attributes have been specified in the SLICE-clause, the cell identifier refers to a single cell and is thus called *singleton cell identifier*. If not all dimension attributes of a cell identifier are specified, the corresponding cell identifier refers to a set of cells and is thus called *set cell identifier*. For a simple analytical expression (i.e., no cell identifiers) there is an implicit 1:1 mapping between the dimension attributes of cells from the cube and the dimension attributes of the corresponding virtual cells. Thus there is no need to specify the dimension attributes of the virtual cell separately. For a

complex analytical expression (i.e., several cell identifiers specified), the various cells that are identified by cell identifiers need to be mapped to a single virtual cell. Thus, the dimension attributes of the virtual cells need to be specified separately in a similar way as the schema of a cube is specified. Note that Slice-conditions may be specified using two alternative approaches: (i) For each cell identifier *id*, clause SLICE may be specified to restrict the dimension attributes of the cell identified by *id*. We refer to these *Slice*-conditions as the "inner" *Slice*-conditions. (ii) If several cell identifiers are used, then the "outer" SLICE-clause (i.e., the SLICE clause that is specified before the FOR CELLS-clause is specified) may be used to relate the dimension attributes of several cells (i.e., joining cells).

Example: Consider the second analytical expression in Figure 4.13, which determines the SalesRatio for article AppleJuiceDeluxe in region Upper Austria as the quotient between the SalesTotals of quarters Q2-2001 and Q1-2001. These two cells are identified by singleton cell identifiers s1 and s2 in clause FOR CELLS. Both cell identifiers refer to cells that represent SalesTotals of article AppleJuiceDeluxe in region Upper Austria, but s1 represents quarter Q1-2001 while s2 represents quarter Q2-2001. Alternatively, outer *Slice*-conditions may be used to restrict the dimension attributes of cell identifier s2 with the value of the corresponding dimension attribute of cell identifier s1 as specified in the third analytical expression in Figure 4.13. Although the two cells that are identified by s1 and s2 are described by three dimension attributes, the resulting virtual cell is described only by dimension attributes P and L, which represent dimension values that are common to s1 and s2.

Aggregations. The levels of granularities of a cube are defined by the domains of the cube's dimension attributes. The measure values of a cube's cell c_i are calculated upon the measure values of those cells from the base cube that roll up to c_i. A cube does not provide higher levels of granularities such that further aggregations that calculate cross-tab or sub-totals are not provided by a cube as it is the case in [GCB+97]. The rationale for this approach is that the complete set of sub-totals (i.e., given N dimensions the number of sub-totals is 2^N) is seldomly needed by analysts and is also expensive to compute. To specify such aggregates using our approach, an analyst may choose one of the following alternatives: (1) The analyst may define a new cube for each higher level of granularity using OLAP operator *Rollup*. (2) The analyst may define an analytical expression for each higher level of granularity using OLAP operator *Rollup* in the same way as used to create a new cube. Aggregations are specified by using an aggregation function such as SUM, AVG, or COUNT in the definition of a measure attribute in conjunction with clause ROLLUP, which is specified after the (optional) outer SLICE-clause.

Example: The upper box of Figure 4.14 shows the analytical expressions of a cube analysis statement that refers to cube SalesArticlesRegionsQuarters2001. Note that the cells

```
ANALYZE SalesArticlesRegionsQuarters2001 [

  (SalesTotal (SalesTotal))
           SLICE P = 'AJD'
             AND L = 'Upper Austria'
             AND T = 'Q1-2001';

  (P s1.P, L s1.L, SalesRatio (s2.SalesTotal/s1.SalesTotal))
           FOR CELLS s1 SLICE P = 'AJD'
                           AND L = 'Upper Austria'
                           AND T = 'Q1-2001,'
                      s2 SLICE P = 'AJD'
                           AND L = 'Upper Austria'
                           AND T = 'Q2-2001';

  (P s1.P, L s1.L, SalesRatio (s2.SalesTotal/s1.SalesTotal))
           SLICE s1.P = s2.P AND s1.L = s2.L
           FOR CELLS s1 SLICE P = 'AJD'
                           AND L = 'Upper Austria'
                           AND T = 'Q1-2001',
                      s2 SLICE T = 'Q2-2001';   ]
```

Figure 4.13: Identifying cells

of this cube provide sales figures per Article, Region, and Quarter. The first analytical expression aggregates the SalesTotals of cells that roll up to virtual cells of granularity P:Article, L:Region, and T:ALL-Days (i.e., ROLLUP T TO ALL-Days) using formula SUM(SalesTotal). These virtual cells represent sales per article and region for all quarters. Similarly, the second analytical expression aggregates the SalesTotals of cells that roll up to a virtual cell of granularity P:Article, L:ALL-Stores, and T:ALL-Days (i.e., ROLLUP L TO ALL-Stores, T TO ALL-Days) using formula SUM(SalesTotal), which represents sales per article for all regions and for all quarters. Finally, the third analytical expression aggregates the SalesTotals of all cells of cube SalesArticlesRegionsQuarters2001 into a single value, which represents the SalesTotal of all articles in all regions of year 2001. The lower box of Figure 4.14 depicts cube SalesArticlesRegionsQuartes2001 and the resulting cells of the corresponding analytical expressions.

To let analysts specify more complex formulas such as the quotient between (1) an article's SalesTotal of a particular region and quarter and (2) the SalesTotal of all articles that have been sold in that region and that quarter etc., cell identifiers and aggregations can be used jointly within the definition of a decision variable. The ROLLUP-clause may be specified for each set cell identifier separately or it may be specified after the outer SLICE-clause to define the granularity of the virtual cell. In the formula that defines a measure attribute of a virtual cell, an aggregation function may be specified to refer to the measure values

```
ANALYZE SalesArticlesRegionsQuarters2001 [

  (SubTotalAR SUM(SalesTotal))
                ROLLUP T TO ALL-Days;

  (SubTotalA SUM(SalesTotal))
                ROLLUP L TO ALL-Stores,
                       T TO ALL-Days;

  (CrossTab SUM(SalesTotal))
                ROLLUP P TO ALL-Articles,
                       L TO ALL-Stores,
                       T TO ALL-Days;   ]
```

SalesArticlesRegionsQuarters2001

P:Article	L:Region	T:Quarter	SalesTotal	SubTotalAR	SubTotalA	CrossTab
AJD	UpperAustria	Q1-2001	6075			
AJD	UpperAustria	Q2-2001	6200			
AJD	UpperAustria	Q3-2001	0			
AJD	UpperAustria	Q4-2001	5980			
AJD	UpperAustria	all-Days		18255		
AJD	LowerAustria	Q1-2001	4100			
AJD	LowerAustria	Q2-2001	0			
AJD	LowerAustria	Q3-2001	0			
AJD	LowerAustria	Q4-2001	4320			
AJD	LowerAustria	all-Days		8420		
AJD	all-Stores	all-Days			26675	
PAF	UpperAustria	Q1-2001	2600			
PAF	UpperAustria	Q2-2001	3100			
PAF	UpperAustria	Q3-2001	3400			
PAF	UpperAustria	Q4-2001	3440			
PAF	UpperAustria	all-Days		12740		
PAF	LowerAustria	Q1-2001	0			
PAF	LowerAustria	Q2-2001	0			
PAF	LowerAustria	Q3-2001	3200			
PAF	LowerAustria	Q4-2001	0			
PAF	LowerAustria	all-Days		3200		
PAF	all-Stores	all-Days			15940	
all-Articles	all-Stores	all-Days				42615

Figure 4.14: Aggregating measure values for calculating sub-totals and cross-tab

of the set of cells that are identified by the set cell identifier. Note the difference in syntax and semantics of aggregation functions when used in conjunction with set cell identifiers. Syntactically, such an aggregation function is specified as cellId.AGG(measureAttr) (cf. the second branch of non-terminal MeasAggDef in Figure 4.17). The semantics of such an expression is that the aggregation function AGG is applied on the values of measure attribute measureAttr for the set of cells identified by cellId. When used within a formula of a measure attribute, cellID.AGG(measureAttr) refers to the projection of aggregated measure values AGG(measureAttr).

Example: Figure 4.15 shows the analytical expressions of another cube analysis statement that refers to cube SalesArticlesRegionsQuarters2001. The first analytical expression defines cell identifier s1, which refers to the cell that represents the sales of article AppleJuiceDeluxe in region Upper Austria and in quarter Q1-2001, and cell identifier s2, which represents the sales of all articles in region Upper Austria and quarter Q1-2001. Aggregation function SUM computes the SalesTotal for the cells that are identified by s2. Measure attribute SalesRatio represents the quotient between the sales of article AppleJuiceDeluxe in Upper Austria and in Q1-2001 and the sales of all articles in that region and that quarter. Hence, this analytical expression generates a single virtual cell as its result. Measure attribute SalesRatio of the second analytical expression is defined upon the same formula but several virtual cells are created for AppleJuiceDeluxe, one for each combination of region and quarter. The third analytical expression provides virtual cells not only for AppleJuiceDeluxe but for all articles.

Drilling Across. So far, only cells of the cube to which the cube analysis statement refers have been analyzed by analytical expressions. To relate the cells of two cubes, which is commonly referred to as *drilling across*, several operators have been introduced in literature. If applied to analytical expressions, drilling across is realized (1) by defining a cell identifier that represents cells of the cube of the cube analysis statement, (2) by defining a cell identifier representing the cells of the cube to which drill across should be performed, and (3) by specifying *Slice*-conditions that relate the dimension attributes of the two cell identifiers in the outer SLICE clause. Syntactically, a cell identifier that refers to the cells of some other cube consists of a unique name and a cube that serves as the domain from which cells are taken.

Example: Consider the analytical expression in Figure 4.16, which defines the set of virtual cells that represent the SalesTotals of AppleJuiceDeluxe per region and quarter. This analytical expression defines virtual cells that represent the same granularity as provided by cube SalesArticlesRegionsQuarters2001. This analytical expression retrieves those sales of article AppleJuiceDeluxe for which no official offerings exist in year 2001 (i.e., an article has been sold already some days before it was officially offered or the article has been still sold several days after the official offering period).

```
ANALYZE SalesArticlesRegionsQuarters2001 [

  (P s1.P, L s1.L, T s1.T, SalesRatio (s1.SalesTotal / SUM(s2.SalesTotal)))
                SLICE s1.L = s2.L AND s1.T = s2.T
                FOR CELLS s1 SLICE P = 'AJD'
                             AND L = 'Upper Austria'
                             AND T = 'Q1-2001',
                         s2;

  (P s1.P, L s1.L, T s1.T, SalesRatio (s1.SalesTotal / s2.SUM(SalesTotal)))
                SLICE s1.L = s2.L AND s1.T = s2.T
                FOR CELLS s1 SLICE P = 'AJD',
                         s2 ROLLUP P TO ALL-Articles;

  (P s1.P, L s1.L, T s1.L, SalesRatio (s1.SalesTotal / s2.SUM(SalesTotal)))
                SLICE s1.L = s2.L AND s1.T = s2.T
                FOR CELLS s1,
                         s2 ROLLUP P TO ALL-Articles;   ]
```

Figure 4.15: Combining cell identification with aggregation

The two cell identifiers s1 and o1 refer to cells of cubes SalesArticlesRegionsDays2001 and Offerings, respectively. Since we consider only year 2001, cell identifier o1 refers only to cells that represent offering periods that start or end in 2001. Note that dimension attributes timeFrom and timeTo represent Days. Hence, we use notation timeFrom:Year.year = 2001 to restrict these days to year 2001. In the outer SLICE-clause, cells identified by s1 and o1 are "joined" using predicate s1.P = o1.P AND s1.L = o1.L. Condition s1.T <= o1.timeFrom OR s1.T >= o1.timeTo identifies those cells of cube SalesArticlesRegionsDays2001, that exceed the official offering period.

```
ANALYZE SalesArticlesRegionsDays2001 [
  (P s1.P, L s1.L, T s1.T, SalesTotal (s1.SalesTotal))
                SLICE s1.P = o1.P AND s1.L = o1.L AND
                      (s1.T <= o1.timeFrom OR s1.T >= o1.timeTo)
                FOR CELLS s1 SLICE P = 'AJD',
                         o1:Offerings SLICE timeFrom:Year.year = '2001' OR
                                            timeTo:Year.year = '2001';   ]
```

Figure 4.16: Joining the cells of cube SalesArticlesRegionsQuarters2001 with cells of base cube Offerings

Syntax

```
CubeAnalysisStmtDef   ::=  ANALYZE cubeName "["
                             AnalyticalExprDef { AnalyticalExprDef } "]".
AnalyticalExprDef     ::=  VirtualCellSchemaDef [VirtualCellBodyDef] ";".

VirtualCellSchemaDef  ::=  "(" { DimAliasDef "," } MeasAggDef
                             { "," MeasAggDef }.
VirtualCellBodyDef    ::=  [OuterSliceDef] [RollupDef]
                             [FOR CELLS CellIdDef { "," CellIdDef }].
OuterSliceDef         ::=  SLICE DimConditionDef
                             { (AND | OR) [NOT] DimConditionDef }.
CellIdDef             ::=  CellIdNameDef [InnerSliceDef] [RollupDef].

InnerSliceDef         ::=  SLICE InnerSliceCondDef
                             { (AND | OR) [NOT] InnerSliceCondDef }.
InnerSliceCondDef     ::=  DimAttrRefDef compOper constantValue
                            DimAttrRefDef compOper DimAttrRefDef.
RollupDef             ::=  ROLLUP [cellIdName "."]dimAttrName TO levelName.

CellIdNameDef         ::=  cellIdName[":"otherCubeName].

DimAliasDef           ::=  dimAttrAlias cellIdName "." dimAttrName.

MeasAggDef            ::=  measAttrAlias (AggFunction MeasExprDef
                             | cellIdName "." AggFunction MeasExprDef ).
DimConditionDef       ::=  DimAttrRefDef compOper constantValue
                             | DimAttrRefDef compOper DimAttrRefDef.
DimAttrRefDef         ::=  [cellIdName "."] dimAttrAlias [":" levelName "." attrName].

MeasExprDef           ::=  "(" [cellIdName "."] measAttrName
                             | MeasExprDef arithmeticOp MeasExprDef) ")".
```

Figure 4.17: Syntax to specify analytical expressions

Evaluating Analytical Expressions

Analytical expressions are evaluated in a similar way as it is the case with SQL-queries. In the following we explain how the virtual cells of analytical expressions are created.

Simple Analytical Expressions. A simple analytical expression builds a single unnamed auxiliary cube from the main cube (i.e., the cube for which the cube analysis statement is specified). As such it has no FOR CELLS-clause. This can be compared with an SQL-query having a single table in its FROM-clause. A simple analytical expression is evaluated as follows:

1. Construction of the auxiliary cube from the main cube:
   (a) *Slice*-conditions are evaluated over the auxiliary cube that represents the main cube.
   (b) The remaining cells will be rolled up to the dimension levels specified in clause ROLLUP. This means that the values of dimension attributes will be replaced with dimension values to which these cells roll up. Hence, there may be several cells having the same combination of dimension values.
   (c) Cells are grouped according to their dimension values.
2. Evaluation of formulas:
   (a) Aggregation functions as specified in the analytical expression will be evaluated on the measure attributes of these groups of cells.

Complex Analytical Expressions. A complex analytical expression defines one or several auxiliary cubes. In its FOR CELLS-clause, cell identifiers refer to the cells of these auxiliary cubes to be used for relating them and applying formulas to the so-related cells. If compared with an SQL-query, every cell identifier ranges over the tuples of a separate query that is nested in the FROM-clause of the outer SQL-query. Note that there is no difference between cell identifiers that represent the cells of the cube for which the analytical expression is defined an cell identifiers that represent the cells of some other cube (i.e., *drill-across*). Creating the virtual cells of a complex analytical expression is as follows:

1. Construction of auxiliary cubes from the main cube and from other cubes as specified in the definition of cell identifiers:
   (a) For each cell identifier, follow the steps as described for simple analytical expressions above. Note that in this step, only "inner" *Slice*-conditions are evaluated.

2. Combination of the cells of auxiliary cubes:

   (a) Create the cartesian product among the virtual cells referenced by the different cell identifiers.

   (b) Evaluate "outer" *Slice*-conditions.

   (c) The remaining cells will be rolled up to the dimension levels specified in clause ROLLUP (i.e., replace the values of dimension attributes with the corresponding dimension values to which these cells roll up). Note that if an "inner" ROLLUP-clause was specified, the measure values of these cells cannot be used directly. Instead, aggregation functions that are qualified with the cell identifier are used instead (e.g., s1.SUM(SalesTotal), cf. Figure 4.15).

   (d) Cells are grouped according to their dimension values.

3. Evaluation of formulas:

   (a) Evaluate the aggregation functions to determine the measure values as defined in the analytical expression.

4.3 Summary

In this chapter, we have laid the foundations to specify multidimensional analyzes as needed automatize decision tasks using analysis rules.

1. Data in the data warehouse is represented using the multidimensional paradigm, which classifies data into dimension levels, base cubes, and cubes. Dimension levels represent non-transactional data from OLTP systems, i.e., business entities such as products, regions, etc. Base cubes represent transactional data from OLTP systems, i.e., business processes such as sales, costs, etc. Cubes are a means to aggregate or to de-aggregate transactional data from base cubes using dimension levels other than provided by base cubes or to select a subset of cells taken from an existing cube. A cube is created using one of the conceptual OLAP operators *Drilldown*, *Rollup*, *Slice*, or *Intersection*. We further introduced a declarative multidimensional analysis language – *cube analysis statements* – with which an analyst can specify advanced multidimensional analyzes (e.g., selecting individual cells of a cube, comparing the cells of different cubes, grouping cells of a cube, etc.).

2. Since data warehouses store huge amounts of data, processing queries efficiently is a key issue in data warehousing. We briefly reviewed recently proposed approaches to materialize pre-aggregated data (so-called materialized views) that can be used to speed up queries. We only considered approaches that are applicable to our data warehouse architecture.

In the next chapters, we will define the syntax and the semantics of analysis rules on a conceptual level. Specifying analyzes to be automatically carried out by analysis rules is founded on the concepts introduced in this chapter.

Chapter 5

Analysis Rules – A Primer

Contents

Today's OLAP and DW systems automatize ETL (extract, transform, load) processes and support interactive data analysis efficiently, but these systems do not offer the possibility to automatize analysis processes that are carried out periodically using well-established decision procedures. This chapter presents a basic model of *analysis rules*, which are a means to automatize such analysis processes with respect to making a decision in an OLTP system. Analysis rules have been originally introduced in [ST00] and extended in [TSM01]. Analysis rules adopt the idea of Event-Condition-Action (ECA) rules from active database systems to automatize well-structured decision tasks that can be solved by carrying out analyzes in the data warehouse. The event that triggers an analysis rule is typically an entry in an analyst's calendar indicating the necessity to analyze some objects (e.g., articles) in the data warehouse. The condition to be evaluated are the multidimensional

analyzes an analyst will carry out for these objects. The action is a particular transaction in an OLTP system, which the analyst will execute for those objects that satisfied the condition. Analysis rules support the knowledge worker in carrying out decision tasks, which we classify as follows:

1. In the case of "non-routine decision tasks", analysts query the data warehouse to create different scenarios to solve ill-structured business problems. Decision making is accomplished by choosing the "most effective" scenario or by choosing the scenario with the "fewest side-effects", etc. Such tasks do not occur regularly and/or there are no generally accepted decision criteria. Examples for "non-routine decision tasks" can be found in strategic business management. Typical questions to be answered are "How many more widgets will we sell in the next year if we drop the price by 10 %?", "How much more of product X can we produce in a month if we suspend production of product Y for 3 months?", or "Is it beneficial to set up a new brand in region Z?".

2. In the case of "routine decision tasks", analysts extract information from the data warehouse to solve well-structured business problems by applying generally accepted procedures in decision making. Such decision tasks occur frequently at predictive timepoints. Examples can be found in the areas of product assortment (e.g., change price, withdraw product), customer relationship management (e.g., grant a discount to special customers), and in many administrative areas (e.g., accept/reject a paper for a conference).

3. In the case of "semi-routine decision tasks", analysts try to make a decision for problems to which routine decision making procedures were not applicable (e.g., if a paper is rated contradictory it is discussed by the program committee before it will be accepted or rejected). Hence, well-established decision making procedures need to be complemented with non-routinizable decision making procedures that cannot be generalized to be applicable to other cases.

Analysis rules are well-suited to automatize "routine decision tasks" and the routinizable elements of "semi-routine decision tasks". Once specified by an analyst, an analysis rule is executed onwards autonomously by the active data warehouse system such that analysts can concentrate on solving "non-routine decision tasks" and the non-routinizable elements of "semi-routine decision tasks". Analysis rules have several immediate impacts on the quality of decision support activities: First, in specifying an analysis rule, implicit knowledge about decision making procedures will be transformed into explicit rules, which is then available to a large community of data warehouse users. Thus, existing rules can be easily adapted to changing requirements and can be used further as patterns to solve new or similar decision tasks. Second, the active data warehouse is responsible for monitoring events and choosing the right time window for executing rules such that the overall system

load remains low (e.g., during night). Finally, analysts can concentrate on more difficult problems (i.e., non-routine decision tasks) for which no decision procedures exist so far.

The remainder of this chapter is organized as follows: Section 5.1 gives a short overview of analysis rules. Section 5.2 introduces the knowledge model of an active data warehouse. Section 5.3 introduces multidimensional analyzes to be carried out by analysis rules and defines the semantics of decision making. Finally, Section 5.5 summarizes the main contributions of this chapter.

5.1 An Overview of Analysis Rules

A conventional data warehouse provides multidimensional data to be analyzed manually using interactive OLAP tools. In an active data warehouse, *events*, *actions*, and *analysis rules* as a whole need to be provided and maintained along with conventional multidimensional data. These additional components are part of the meta knowledge of an active data warehouse, which comprises all information that must be available such that analysts can specify rules to automatize "routine decision tasks". This section introduces analysis rules and argues that analysis rules need to be specified from an object-oriented perspective and from a multidimensional perspective.

An *analysis rule* is a 7-tuple $\langle L^{prim}, E, C^{prim}, AG, DS, A^{OLTP}, C^{OLTP} \rangle$, where

1. L^{prim} is the *primary dimension level*, for whose level instances the analysis rule will generate decisions,
2. E is the *event*, which triggers rule processing,
3. C^{prim} is the *primary condition*, which selects some objects of the primary dimension level to be analyzed by the analysis rule,
4. AG is the *analysis graph*, whose nodes represent the cubes for analysis,
5. DS is the set of *decision steps*, which represent multidimensional conditions that must be satisfied by a level instance of the primary dimension level in order to generate a decision for that object,
6. A^{OLTP} is the *action*, which represents a particular transaction to be executed in an OLTP system if an object satisfies the multidimensional conditions,
7. C^{OLTP} is a list of *conditions* that must hold in an OLTP system for each exported decision prior to executing the rule's action in that OLTP system.

Figure 5.1 presents the EBNF syntax to specify analysis rules. The FOR-clause is used to specify the primary dimension level and a variable that represents an instance of the

primary dimension level. The ON-clause is used to specify the rule's event and a variable that represents the triggering event instance. The optional BIND-clause is used to associate the event instance represented by the event variable with an instance of the primary dimension level. The rule's primary condition is specified using the optional IF-clause. The analysis graph is specified by clause USING CUBES. Decision steps are specified after the BEGIN-clause. The rule's action is referenced in the TO EXECUTE-clause where also the action's parameters are specified. Finally, the optional SUBJECT TO-clause is used to specify conditions as boolean method calls that must be satisfied before an exported decision is executed in an OLTP system. The non-terminal symbols RootCubeDef, InnerCubeDef, and DecisionStepDef will be explained later in this chapter.

Specifying an analysis rule requires analysts to view the data in the data warehouse from an object-oriented perspective and from a multidimensional perspective. The main reasons for viewing data from an *object-oriented perspective* are twofold:

1. The objects that have to be analyzed using analysis rules need to be associated with some methods that will be used as the actions of analysis rules.

2. Any method invocation in an OLTP systems may represent some happening of interest to analyze data. Hence, these happenings may then be loaded as *OLTP method events* into the data warehouse to trigger analysis rules.

Viewing data from a *multidimensional perspective* is natural to the way an analyst inspects cubes and specifies conditions when analyzing these cubes. The outcome of multidimensional analyzes is the decision (1) as to whether to execute a particular method in an OLTP system, (2) as to whether to carry out further manual analyzes, or (3) as to whether to abandon analyzing the data warehouse for a particular object.

5.1.1 Object-Oriented Perspective

The object-oriented perspective facilitates identifying (1) which objects have to be analyzed by an analysis rule in the data warehouse in regard of (2) whether some transaction should be executed for these objects in an OLTP database. Since analysis rules analyze data in the data warehouse but may execute actions for these data in OLTP systems, analysis rules can be specified only for dimension levels that represent data loaded from OLTP systems. To avoid confusions, a dimension level's corresponding object type in an OLTP system is called *OLTP object type* from now on. An analysis rule is a behavioral aspect of the dimension level, whose level instances will be subject to decision making of this analysis rule. This dimension level is referred to as the analysis rule's *primary dimension level* which is specified in the FOR-clause of the analysis rule; the corresponding dimension is called *primary dimension*. The dimensions of the data warehouse other than the primary dimension are referred to as *analysis dimensions*.

Syntax

```
AnalysisRuleDef   ::=  DEFINE ANALYSIS RULE ruleName
                         FOR primDimVar":"primDimLevelName
                         ON eventVar":"eventTypeName
                         [BIND eventVar"."attrName TO primDimVar]
                         [IF PrimCondDef]
                         USING CUBES RootCubeDef { InnerCubeDef }
                         BEGIN DecisionStepDef { DecisionStepDef }
                         TO EXECUTE ActionDef
                         [SUBJECT TO BoolMethodDef]
                       END ruleName.
PrimCondDef       ::=  "(" primDimVar"."attrName CompOperDef PrimCondValDef
                         [(AND | OR) [NOT] PrimCondDef] ")".
CompOperDef       ::=  ("=" | "<" | ">" | "<>" | "<=" | ">=") .
PrimCondValDef    ::=  (constant | EventVarAttr).
ActionDef         ::=  PrimDimOLTPQual"."methodName [ParameterDef].
BoolMethodDef     ::=  "(" PrimDimOLTPQual.boolMethodName [ParameterDef]
                         [("AND" | "OR") [NOT] BoolMethodDef] ")".
PrimDimQual       ::=  (primDimVar | PrimDimOLTPQual).
PrimDimOLTPQual   ::=  "("primDimVar"@OLTP)".
ParameterDef      ::=  PARAMETERS parameterName "=" ParamValDef
                       { "," parameterName "=" ParamValDef }.
ParamValDef       ::=  (constant | PrimDimVarAttr | EventVarAttr).
PrimDimVarAttr    ::=  PrimDimQual"."attrName.
EventVarAttr      ::=  eventVar"."attrName.
```

Figure 5.1: Syntax to specify analysis rules

Example: Scenarios 1 and 2 of our case study (cf. Chapter 2) will be modeled as the two analysis rules ChangePriceOfSoftDrinksAR (scenario 1) and RemoveHighPricedSoftDrinksAR (scenario 2) at dimension level Article, which represents the primary dimension level of both analysis rules. The two analysis rules may use Location and Time as analysis dimensions.

An active data warehouse may execute transactions in an OLTP system as the results of executing analysis rules. These transactions are the methods of OLTP object types. An analysis rule may refer in its action part to such OLTP methods that are defined for the primary dimension level's corresponding OLTP object type. The action part of an analysis rule is specified by clause TO EXECUTE, in which the analyst may refer to a single OLTP method. Parameters are specified using the optional PARAMETERS-clause (cf. Figure5.1). To be able to reference such a method in the action part of a rule, the interface of the transaction needs to be part of the meta data of the data warehouse.

Example: The decision tasks of analysis rule ChangePriceOfSoftDrinksAR and of analysis rule RemoveHighPricedSoftDrinksAR are to decide whether to reduce the price of an article and whether to withdraw an article from a market, respectively. Such decisions can be carried out by invoking the two methods changePrice and withdrawFromMarket, on the respective instances of OLTP object type Article. The interfaces of these methods are part of the meta knowledge of the active data warehouse in order to be used in the action part of analysis rules that were defined for dimension level Article.

Active data warehouses use temporal events to determine the timepoints at which analysis rules will be fired. We distinguish *absolute temporal events*, *periodic temporal events*, and *relative temporal events*. Absolute temporal events and periodic temporal events, which are also called *calendar events*, are globally available and therefore can be used to trigger analysis rules that are specified for any dimension level. Detecting the occurrence of a relative temporal event is initiated with the detection of a method event that occurred in an OLTP system (*OLTP method event*). The relative temporal event will then occur a specified time period after its initiating OLTP method event occurred. Occurrences of OLTP method events must be imported into the active data warehouse before being used to detect occurrences of relative temporal events. Since OLTP method events occur for the instances of a particular dimension level, a relative temporal event is available locally to the dimension level for which the initiating OLTP method event occurred. If a calendar event is used to trigger an analysis rule, then the analysis rule will be fired for each level instance of the primary dimension level[1]. If the triggering event is a relative temporal event, then the analysis rule will be fired only for the level instance for which the initiating OLTP method event occurred. This must be specified by clause BIND ... TO ..., which requires

[1]Note although an analysis rule is specified as if it were fired for each level instance of the primary dimension level separately, an efficient implementation will execute an analysis rule for a *set* of level instances.

a level instance of the primary dimension level to be identical with the level instance for which the triggering relative temporal event occurred. If clause BIND ... TO ... was not specified although the analysis rule's event is a relative temporal event, the rule will be fired for all level instances of the primary dimension level as it is the case with calendar events. The set of level instances of the primary dimension level for which the rule fires is further reduced by evaluating the primary condition for these level instances. The analysis rule will carry out its multidimensional analyzes only for the set of level instances of the primary dimension level that satisfy the primary condition.

```
DEFINE ANALYSIS RULE ChangePriceOfSoftDrinksAR FOR a:Article

ON 20pl:TwentyDaysPastLaunch BIND 20pl.levelId TO a
IF (a.category = 'Soft Drinks')

USING CUBES

  QuantitiesAllLocationsAllTimes
   (product P, location L, time T, SUM(quantity) TotalQty) PRIMARY P AS
    ROLLUP L TO ALL-Locations, T TO ALL-Times
    FROM Sales;

BEGIN

  ANALYZE QuantitiesAllLocationsAllTimes
     TRIGGER ACTION IF TotalQty < 10000
     DETAIL ANALYSIS IF TotalQty < 20000

TO EXECUTE (a@OLTP).changePrice PARAMETERS ratio = 0.85
SUBJECT TO ((a@OLTP).currentPriceEquals PARAMETERS price = a.pricePerUnit)

END ChangePriceOfSoftDrinksAR
```

Figure 5.2: Analysis Rule ChangePriceOfSoftDrinksAR

Example: Analysis rule ChangePriceOfSoftDrinksAR will be triggered by an instance of relative temporal event TwentyDaysPastLaunch, which is referenced by variable 20pl. This event occurs twenty days after the occurrence of an OLTP method event of type ArticleLaunch, which represents the invocation of OLTP method launchOnMarket for an Article. Figure 5.2 shows the complete specification of analysis rule ChangePriceOfSoftDrinksAR (scenario 1). The level instance of the primary dimension level Article, for which the rule is executed, will be referenced by variable a. The binding between an instance of dimension level Article and the triggering event instance referenced by 20pl is expressed by clause BIND 20pl.levelId TO a. The primary condition will consider article a, for which the analysis rule was triggered, for further multidimensional analyzes if a belongs to product category "Soft Drinks".

5.1.2 Multidimensional Perspective

Using the multidimensional perspective, analysts specify the multidimensional analyses that are necessary in decision making. An important aspect of this approach is that an analysis rule emulates exactly the way an analyst inspects and analyzes multidimensional data. When an analyst queries the data warehouse to make a decision whether to execute a transaction in an OLTP system, he follows an *incremental top-down approach* in creating and analyzing cubes. First, the analyst creates a very "coarse grained" cube that describes the objects for which a decision should be made (i.e., the level instances of the primary dimension level). Second, in carrying out multidimensional analyzes the analyst compares, aggregates, transforms, etc. the cells of this cube. As a conclusion the analyst can determine for each level instance of the primary dimension level (1) whether a decision can be made (e.g., change the price of an article) or (2) whether further analysis using a more detailed cube is necessary. In the first case, the analyst will mark level instances of the primary dimension level as "positively decided" if a particular transaction should definitely be carried out in an OLTP system or level instances are marked as "negatively decided" if a particular transaction should definitely not be carried out. In the latter case, the analyst creates a more detailed cube for the level instances of the primary dimension level for which further analyses are necessary and then continues his analyses. Sometimes, there may be several alternative ways to create and analyse more detailed cubes. In such a situation, several more detailed cubes will be derived directly from the same "starting point". Analyzes are continued until for each level instance of the primary dimension level a definite decision was made or until no further cubes are available to be analyzed.

Analysis rules follow the same approach in decision making. The cubes that are necessary for analysing level instances of the primary dimension level will be specified by the analyst. These cubes constitute the rule's *analysis graph*, which is used during each execution of the analysis rule. The n dimensions of each cube of the analysis graph are classified into one "primary dimension", which represents the level instances of the primary dimension level, and $n-1$ "analysis dimensions", which represent the multidimensional space for analysis.

Example: Analysis rule ChangePriceOfSoftDrinksAR (cf. Figure 5.2) defines cube QuantitiesAllLocationsAllTimes, which represents the total quantity of articles that has been sold at all locations and at all times. Since the rule is defined for the level instances of dimension level Article, which is also the primary dimension level, the remaining dimensions Location and Time are the analysis dimensions. This rule will be fired twenty days after a new soft drink was launched. Hence, cube QuantitiesAllLocationsAllTimes represents exactly the data that belong to the twenty-day period of the newly launched soft drink.

Every cube of the analysis graph will be "analyzed" by a *decision step*, which is a cube analysis statement (cf. Section 8.2) extended by (1) names for each analytical expression (called the *decision variables*) and (2) conditions for decision making. A decision variable

is a name that identifies the cells of an analytical expression in order to be used in the conditions of the decision step. The conditions of a decision step fulfill two tasks:

Task 1: Select the level instances of the primary dimension level whose cells comply with the *trigger action condition*, which represents the condition that must be satisfied in order to execute the rule's action (e.g., withdraw an article if the sales total of the last quarter is below ATS 100000).

Task 2: Select the level instances of the primary dimension level whose cells comply with the *detail analysis condition*, which represents the condition to continue analysis for a level instance of the primary dimension level at finer grained cubes (e.g., continue analysis if the sales total of the last quarter is below ATS 500000).

If the two conditions specified overlap, the *trigger action condition* is given precedence over the *detail analysis condition*. The rationale for this conflict resolution strategy is that whenever a decision step decides to execute the rule's action for a level instance of the primary dimension level, then the level instance will be withdrawn from the decision making process and thus no further analyses will be necessary. If a particular level instance does not satisfy both conditions, *action execution is dismissed* for that level instance which means that the level instance is not analyzed further. Figure 5.3 depicts this top-down approach for analyzing data. The left part of that figure depicts the cubes of the analysis graph, whereas in the right part decisions steps are depicted indicating the set of objects for which the rule's action should be executed (TrgA), the set of objects for which more detailed analyzes should be carried out (DtlA), and the set of objects that are dismissed from action execution (DsmA).

Example: As specified by analysis rule ChangePriceOfSoftDrinksAR, soft drinks will be analyzed using the cells of cube QuantitiesAllLocationsAllTimes. The decision task of this analysis rule is to decide as to whether change the price of a soft drink. Analysis is carried out as follows: If a soft drink's sold quantity is below the threshold of 10000 (*trigger action condition*), then the analysis rule will reduce the price of this soft drink by invoking method changePrice with parameter ratio = 85 % in the OLTP system. If the total quantity sold is above the threshold of 20000 items, then action execution will be dismissed for the soft drink. For the case that the total quantity sold is between 10000 and 20000 (*detail analysis condition*), then further analysis is necessary. Since this analysis rule uses a single cube, "further analysis" means that the problem will be forwarded to the analyst for manual treatment. Clause SUBJECT TO of analysis rule ChangePriceOfSoftDrinksAR will be explained later in this chapter.

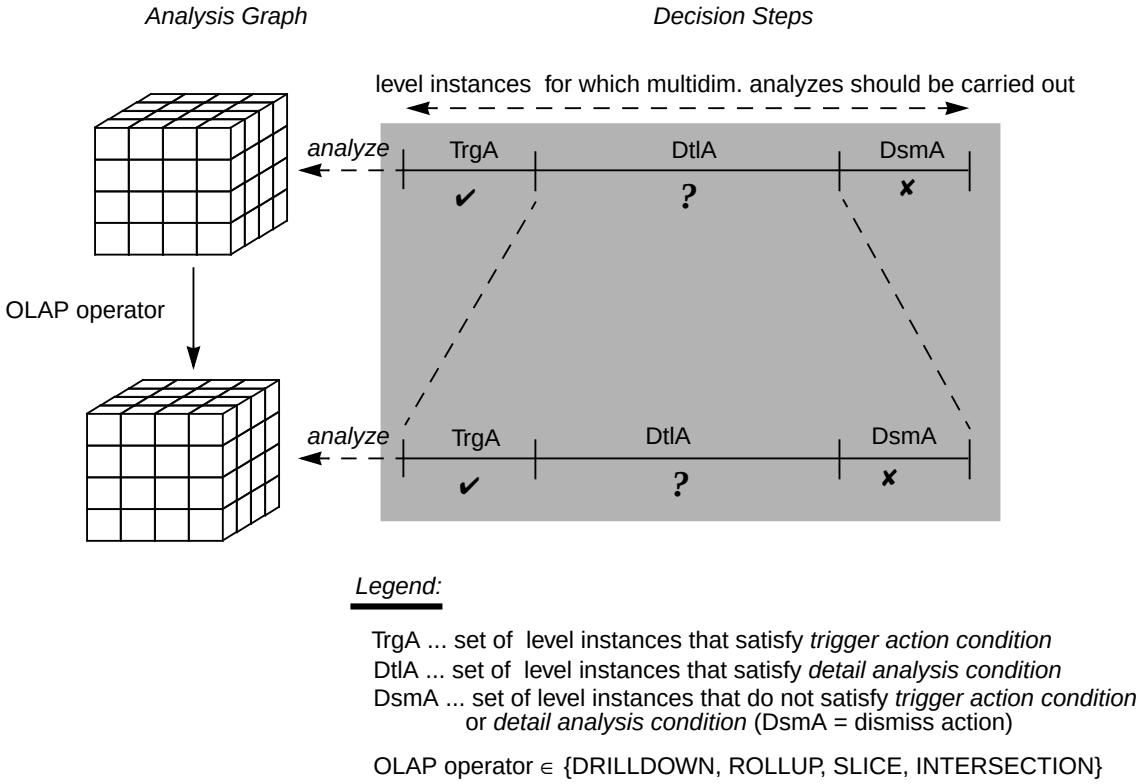

Figure 5.3: Top-down analysis

5.2 Event, Primary Condition, and Action

This section describes the conventional ECA rule parts — the event, the primary condition, and the action — for analysis rules. We explain these components by using the categories of the classification framework from Paton and Diaz [PD99]. Note that the components of a rule (event, primary condition, and action) treat data in the data warehouse from an object-oriented perspective, which has been discussed in Section 5.1.1. Following this perspective, analysts specify rules for the level instances of a particular dimension level, called the *primary dimension level*.

5.2.1 Event

Events are used to specify the timepoints at which analysis rules should be carried out. These timepoints may be characterized as follows: Routine decision tasks are carried out (1) on the basis of a fixed schedule using a calendar (e.g., "at the end of every quarter ...") or (2) as a reaction to recent happenings in the data warehouse's source systems (e.g., "twenty days after the price of an article was changed ..."). Since carrying out analysis

	OLTP Method Ev.	*Relative Temp. Ev.*	*Calendar Ev.*
Source	External Behavior Invocation	OLTP Method Event, Clock	Clock
Granularity	Member	Member	Set
Type	Primitive	Primitive	Primitive
Role	Mandatory		

Table 5.1: Classification of Events

rules is basically determined by a clock, the event model of an active data warehouse is more primitive than the event model of active database systems [PD99, CGMR95, CM94]. Our event model provides three kinds of events: (1) *OLTP method events*, (2) *relative temporal events*, and (3) *calendar events*. OLTP method events describe basic happenings in the data warehouse's sources. Relative temporal events are used to define a temporal distance between such a basic happening and carrying out an analysis rule. Calendar events represent fixed points in time at which an analysis rule may be carried out. Structurally, every event instance is characterized by an occurrence time and by an event identifier. In its event part, an analysis rule refers to a calendar event or to a relative temporal event. Table 5.1 provides an overview of events in active data warehouses using the categories of Paton and Diaz [PD99]. The following subsections describe these events and the detection of their instances in an active data warehouse in more detail.

OLTP Method Events

An OLTP method event describes a happening in the data warehouse's source systems that is of interest to analysis rules in the active data warehouse. Such happenings are method invocations on the instances of OLTP object types for which a corresponding dimension level exists in the data warehouse. Following the object-oriented perspective, an OLTP method event is modeled as a local component of the dimension level for which it may occur and is identified by the dimension level together with the method name. Thus, if we refer to an OLTP method event, we use the at-notation ⟨*OLTPMethodEvent*⟩@⟨*dimension level*⟩. Besides occurrence time and event identifier, the attributes of an OLTP method event are (1) a reference to the dimension level for which the OLTP method event occurred and the (2) parameters of the method invocation. To make OLTP method events available in data warehouses, a data warehouse designer has (1) to define the schema of OLTP method events and (2) to extend the data warehouse's extract/transform/load mechanism in order to load the occurrences of OLTP method events from OLTP systems. Since instances of OLTP method events are loaded *some time after* their occurrence in OLTP systems, analysis rules cannot be triggered immediately by OLTP method events.

Figure 5.4 presents the syntax to define OLTP method events. The FOR-clause is used to specify the dimension level for which instances of the OLTP method event will occur.

Syntax

```
OLTPMethodEventDef ::= DEFINE METHOD EVENT methodEventName
                         FOR dimLevelName
                         ON methodName ["(" ParametersDef ")" ] ";".
ParametersDef      ::= paramName":"datatype [","  ParametersDef].
```

Figure 5.4: Syntax to define OLTP method events

```
DEFINE METHOD EVENT PriceChange
  FOR Article ON changePrice (ratio: Real);

DEFINE METHOD EVENT MarketLaunch
  FOR Article ON launchOnMarket (market: Region);

DEFINE METHOD EVENT MarketWithdrawal
  FOR Article ON withdrawFromMarket (market: Region);
```

Figure 5.5: Definition of OLTP method events PriceChange, MarketLaunch, and MarketWithdrawal

Clause ON is used to specify the name of the OLTP method. The optional parameter list represents the schema of the OLTP method event. Note that attributes eventId and occtime need not be specified.

Example: Invocations of methods changePrice(ratio: Real), launchOnMarket(market: Region), and withdrawFromMarket(market: Region) on instances of OLTP object type Article are used by the active data warehouse as OLTP method events. In the data warehouse, we define the three events PriceChange(eventId: Identifier, occtime: Day, levelInstId: Article, ratio: Real), MarketLaunch(..., market: Region), and MarketWithdrawal(..., market: Region) for dimension level Article. The specification of these events is presented in Figure 5.5.

Relative Temporal Events

Since OLTP method events cannot trigger analysis rules *immediately*, relative temporal events are the means to realize *deferred* execution of analysis rules. An instance of a relative temporal event occurs a specified period after the occurrence of an OLTP method event. A relative temporal event R is defined upon an OLTP method event M and a number of "clock ticks" (Δ) to elapse between an occurrence of M and the occurrence of

Syntax

```
RelativeTmpEventDef ::= DEFINE RELATIVE EVENT relativeEventName
                            FOR dimLevelName
                            AFTER TmpDistDef
                            ON methodEventName";".
TmpDistDef          ::= numericValue (HOURS | DAYS | WEEKS | MONTHS |
                        QUARTERS | SEMESTERS | YEARS).
```

Figure 5.6: Syntax to define relative temporal events

R. R depends on M — the initiator of R — in two respects: First, the existence of R depends on the existence of M (i.e., R occurs only if M occurred). Second, R depends temporally on M (i.e., R occurs a specified period *after* M occurred). Following this definition, a relative temporal event R requires a data warehouse designer to specify (1) the OLTP method event M that will be used as initiator and (2) the delay Δ after which the relative event will occur, thus $R.occtime = M.occtime + \Delta$. A relative temporal event R is modeled as a local component of the dimension level for which the OLTP method event occurred. Besides its event identifier and its occurrence time, R consists of an attribute that refers to the initiator M. Again, we use the @-notation to provide a globally unique naming scheme for relative temporal events. If a relative temporal event is used to trigger an analysis rule, then the analysis rule will be fired for the level instance for which the relative temporal event has occurred.

Figure 5.6 presents the syntax to define relative temporal events. While the FOR-clause is used in the same way as for specifying OLTP method events, clause AFTER is used to specify the temporal distance between the occurrence of an OLTP method event (specified by clause ON) and the occurrence of a relative temporal event. The time granularities HOUR, DAY, WEEK, MONTH, QUARTER, SEMESTER, and YEAR are available for specifying temporal distances.

Example: Relative temporal event TwentyDaysPastLaunch is specified as (MarketLaunch, 20 days) and is described by schema (eventId: Identifier, occtime: Timestamp, levelInstId: Article, initiator: MarketLaunch@Article). In its event part, analysis rule ChangePriceOf-SoftDrinksAR refers to this relative temporal event. This analysis rule will be fired only for articles for which an event of type TwentyDaysPastLaunch has occurred. Figure 5.7 presents the definition of this relative temporal event.

```
DEFINE RELATIVE EVENT TwentyDaysPastLaunch
  FOR Article
  AFTER 20 DAYS
  ON MarketLaunch;
```

Figure 5.7: Definition of relative temporal event TwentyDaysPastLaunch

Calendar Events

A calendar event represents a fixed point in time, at which analysis and decision processes will be initiated. Calendar events generalize absolute temporal events and periodic temporal events. Different to OLTP method events and relative temporal events, a calendar event is globally available and may be used to trigger analysis rules of different dimension levels. Intervals of a calendar are described by at least two events, which signal the beginning and the end of the interval. We assume the existence of a calendar system [CS94, CD95, LEW96, SS92], which provides a broad set of calendar events such as BeginOfWeek, EndOfWeek, BeginOfMonth, EndOfMonth, BeginOfQuarter, EndOfQuarter, etc. If a calendar event is used to trigger an analysis rule, then the analysis rule will be fired for all level instances of the primary dimension level.

Event Management and Time Granularity

In contrast to OLTP systems, in which updates may occur at any instance of time, data warehouses are typically updated by loading data from OLTP systems on a periodic schedule. We assume that updating a data warehouse occurs at a constant interval (e.g., every 24 hours). Between two successive updates, the state of the data warehouse remains unchanged i.e., the data warehouse is in *read-only* mode. Hence, there is no difference whether an analysis rule is fired immediately after the data warehouse was updated or whether the analysis rule is fired immediately before the next update will occur. Conceptually, the finest granularity of time to detect events is the length of such a period (e.g., events occur on a per-day schedule), which is called *chronon*. In a particular implementation, the active data warehouse may choose any instance of time within such a period at which events occurrences are detected in order to fire analysis rules. These timepoints are basically determined by the system load, i.e., rules are fired when the system load is low (e.g., during night).

5.2.2 Primary Condition

Several analysis rules may share the same OLTP method as their action. These rules may be carried out at different timepoints and may utilize different multidimensional analyzes.

Thus, a certain analysis rule usually analyzes only a subset of the level instances that belong to the rule's primary dimension level. An analysis rule's *primary condition* is used to determine for a particular level instance of the primary dimension level whether multidimensional analyzes should be carried out by this analysis rule. We specify the primary condition as a boolean expression, which refers to the attributes of the primary dimension level. If omitted, the primary condition evaluates to "TRUE".

Example: In our retail example, an analysis rule that changes the price of high priced "soft drinks" analyzes data in a different way and at different timepoints than a rule that changes the price of "cereals". For this purpose, the two analysis rules ChangePriceOfHighPricedSoftDrinksAR and ChangePriceOfCerealsAR are defined. Both analysis rules are specified for primary dimension level Article and both analaysis rules refer to changePrice as their action. Expression a.category = 'Soft Drinks' is the primary condition of analysis rule ChangePriceOfHighPricedSoftDrinksAR (see Fig. 5.2), whereas expression a.category = 'Cereals' is the primary condition of analysis rule ChangePriceOfCerealsAR.

The context within which the analysis rule's primary condition is evaluated (i.e., the possible states of the data warehouse) is easy to determine since the approach taken in this thesis assumes that the data warehouse cannot be updated during rule processing. Thus, the four possible database states (taken from the classification framework of Paton and Diaz [PD99]) – DB_T (i.e., the data warehouse at start of the current transaction), DB_E (i.e., the data warehouse when the event took place), DB_C (i.e., the data warehouse when the condition is evaluated), and DB_A (i.e., the data warehouse when the action is executed) – are the same. Further, the primary condition can access the bindings associated with the event (i.e., $Bind_E$). The first row of Table 5.2 classifies the primary condition.

Although analysis rules are specified as if they were exectued for each level instance of the primary dimension level separately, an efficient implementation will execute analysis rules in a *set-oriented* manner, i.e., a single execution of an analysis rule results in decision making for a set of level instances of the primary dimension level. Since querying the data warehouse is usually costly in terms of computation time and in terms of scanning data, it is important to determine the exact subset of level instances of the primary dimension level before multidimensional analyses are carried out. We refer to this subset as the analysis rule's *analysis scope*. It will be determined (1) by the analysis rule's event (relative temporal event or calendar event) that occurs at some chronon and (2) by the rule's primary condition. Let $L_R^{prim}(c)$ denote the set of level instances that belong to the primary dimension level of analysis rule R at some chronon c. Let *Occ(E_R, c)* denote the set of event occurrences that belong to the triggering event E of analysis rule R at chronon c. Then *Affected(E_R, c)* defines the set of level instances of the primary dimension level that are affected by the occurrences of some event E at chronon c as follows:

Primary Condition	**Role** = Optional **Context** = DB_T = DB_E = DB_C = DB_A, $Bind_E$
Action	**Options** = External Behavior Invocation, Do Instead **Context** = DB_T = DB_E = DB_C = DB_A, $Bind_E$

Table 5.2: Classification of Primary Condition and Action

$$Affected(E_R, c) = \begin{cases} L_R^{prim}(c) & : \text{ if } E \text{ is a calendar event} \\ \{e.levelInstId \in L_R^{prim}(c) \mid e \in Occ(E_R,\ c)\} & : \text{ if } E \text{ is a rel. temp. event} \end{cases} \tag{5.1}$$

Let $L_R^{prim}(C^{prim}, c)$ denote the set of level instances that belong to the primary dimension level of analysis rule R, which satisfy the primary condition C^{prim} at some chronon c. Then the analysis scope of analysis rule R at chronon c, denoted as $AnalysisScope_R(c)$, is defined as follows:

$$AnalysisScope_R(c) = Affected(E_R, c) \cap L_R^{prim}(C^{prim}, c) \tag{5.2}$$

5.2.3 Action

The purpose of an analysis rule is to automatize decision making for objects that are available in OLTP systems *and* in the data warehouse. Since we follow an object-oriented approach, a "decision" means to invoke (or not to invoke) a method on a certain object in an OLTP system. In its action part, an analysis rule may refer to such a single OLTP method of the primary dimension level, which represents a transaction in an OLTP system. These transactional methods represent the "decision space" of an active data warehouse. To make the transactional behavior of an OLTP object type available in the active data warehouse, the data warehouse designer must provide (1) the specifications of the OLTP object type's methods together with required parameters, (2) the preconditions that have to be satisfied before the OLTP method can be invoked in the OLTP system, and (3) a conflict resolution mechanism, which solves contradictory decisions of different analysis rules.

Figure 5.8 presents the syntax to specify interfaces of OLTP methods to be added to the data warehouse's meta data repository.

Example: Figure 5.9 shows the definitions of method interfaces changePrice, currentPriceEquals, launchOnMarket, and withdrawFromMarket. These methods may be referenced by analysis rules that are defined for dimension level Article.

Syntax

```
OLTPMethodIFDef   ::=  DEFINE METHOD INTERFACE methodName
                         ["(" ParametersDef ")"]
                         FOR dimLevelName";".
ParametersDef     ::=  paramName":"dataType [","  ParametersDef].
```

Figure 5.8: Syntax to define OLTP method interfaces to the meta data repository

```
DEFINE METHOD INTERFACE changePrice (ratio: REAL) FOR Article;

DEFINE METHOD INTERFACE currentPriceEquals (price: REAL) FOR Article;

DEFINE METHOD INTERFACE launchOnMarket (market: Region) FOR Article;

DEFINE METHOD INTERFACE withdrawFromMarket (market: Region) FOR Article;
```

Figure 5.9: Definition of method interfaces

Table 5.2 presents how the action part of analysis rules is classified along the lines of Paton and Diaz [PD99]. No special coupling mode (other than "immediate") is needed for condition evaluation and action execution, since analysis rules cannot update the data warehouse directly and thus, in our model, no changes in the state of the data warehouse can occur during rule execution. Besides deciding to execute the rule's action for a certain level instance of the primary dimension level (denoted as "External Behavior Invocation" in Table 5.2), the conclusion of analyzing multidimensional data may be the need to make further manual analyses instead of executing a transaction in the OLTP system. This situation is denoted as "Do Instead" in Table 5.2. Since decision making in an active data warehouse is carried out without having access to OLTP systems, data in these operational systems may have substantially changed by OLTP transactions (e.g., an object has been deleted, the value of an attribute was changed, etc.). Such changes may affect the decisions generated by the active data warehouse.

A decision is considered to be *out-of-date* if (1) the object, for which the decision should be made is no longer available in the OLTP system, (2) if the precondition of the OLTP method that represents the decision no longer holds in the OLTP system, or (3) if the assumptions for decision making no longer hold in the OLTP system (e.g., the price must not have been changed). Out-of-dateness of a decision must be examined *prior* to executing an exported decision in the OLTP system, which will be carried out by the export facility of the active data warehouse. Conditions (1) and (2) are independent from a particular rule definition and need not separately be specified by a rule designer. These two conditions

need not be specified explicitly since checking whether the object for which a transaction should be carried out in an OLTP system (condition 1) must be done automatically for every exported decision and checking whether the transaction's precondition is satisfied (condition 2) is within the duties of the respective operational system. Condition (3) may vary from rule to rule and must be explicitly specified for each rule separately using clause SUBJECT TO in the rule's definition (see syntax in Figure 5.1). This clause consists of a boolean expression which is partly evaluated in the DW and partly evaluated in the OLTP system to which the decisions should be exported. We use the notation "primary dimension variable@OLTP" (see syntax in Figure 5.1) to determine that a called method must be executed in the OLTP system. If the SUBJECT TO clause is omitted, the export facility will execute the generated decision in the OLTP system if conditions (1) and (2) hold.

Example: The SUBJECT TO-clause of analysis rule ChangePriceOfSoftDrinksAR (cf. Figure 5.2) requires that an exported decision to reduce the price of an article will be executed in the OLTP system, if the current price of that article in the OLTP system equals the price of that article in the data warehouse.

In the following, we will discuss two questions that arise when describing the actions of a given rule set: (1) What happens with cyclic rule triggerings? (2) What happens if two rule executions produce contradictory decisions?

Rule Cycles

In an active OLTP database, identifying non-terminating rule triggerings (i.e., some rules trigger each other cyclically) [BCP98] is an important issue since such rule triggerings may increasingly consume system resources without arriving at a consistent final state. The occurrence of such a situation depends on the execution order of a given rule set and, once occurred, users typically cannot intervene to break up non-terminating rule execution cycles. Various tools and techniques have been developed to detect rule execution cycles at compile-time as well as at run-time [AHW95, BCP98, DPW99].

In an active data warehouse, non-terminating rule triggerings *during the same ADW cycle* will not occur since automatically generated decisions are always exported to OLTP systems first, where the corresponding transactions are invoked. Thus, no new events are generated in the data warehouse due to executing analysis rules within the same ADW cycle. Further, the order of rule execution does not impact which decisions are generated.

Cyclic rule triggering *across several ADW cycles* is an intrinsic feature of active data warehouses and is thus explicitly intended. It represents the "feedback-loop" in decision making: If the decision of an analysis rule is to execute a transaction in an OLTP system, that transaction execution may be imported into the active data warehouse as an OLTP method event at the beginning of the next ADW cycle and may again cause some analysis

rules to be executed as a response. At the end of every ADW cycle, users have the possibility to terminate the "feedback loop" by canceling a decision instead of exporting and executing the decision's transaction in an OLTP system.

Decision Conflicts

Since different analysis rules can make a decision for the same level instance of some primary dimension level during the same ADW cycle, a "decision conflict" may occur. We refer to such conflicts as *inter-rule conflicts*. To detect inter-rule conflicts, we use a "conflict table" covering the OLTP methods of the decision space. The tuples of the conflict table have the form $\langle m_1, m_2, m_3 \rangle$, where m_1 and m_2 identify two conflicting methods and m_3 specifies the conflict resolution method that will be finally executed in OLTP systems. If a conflict cannot be solved automatically (i.e., m_3 was not specified) it has to be reported to analysts for manual conflict resolution.

Example: In our example, the active data warehouse may invoke methods launchOnMarket(market: Region), withdrawFromMarket(market: Region) and changePrice(ratio: Real) for the instances of object type Article in OLTP systems. The two methods changePrice and withdrawFromMarket may cause a conflict if the price of an article should be changed *and* if the article should be withdrawn from the market at the same time. Thus, the conflict table contains the entry ⟨ changePrice@Article , withdrawFromMarket@Article , withdrawFromMarket@Article ⟩, which specifies method withdrawFromMarket as the conflict resolution method.

5.3 Multidimensional Analyzes

The multidimensional analyzes of an analysis rule are specified once by the analyst who defines cubes, cube analysis statements, and conditions for decision making. In Sections 5.3.1, 5.3.2, and 5.3.2, we describe the syntax and the semantics of these rule components in more detail.

5.3.1 Analysis Graph

The analysis graph represents all cubes that are needed for multidimensional analysis. These cubes are specified by the analyst in the same top-down fashion as in manual decision making. We distinguish between local cubes and global cubes. The cubes that compose the analysis graph of a certain analysis rule are called local cubes, since they can be used only within the analysis rule for which they have been specified. The "root" cube of the analysis graph is the most coarse grained cube, which is needed in decision making of an analysis

rule. This cube may be created bottom-up from a base cube or top-down from a globally defined cube, which represents pre-aggregated data that is available to specify the "root" cubes of several analysis rules. The specification of inner cubes of the analysis graph will be derived directly or indirectly from the "root" cube and are thus specified *incrementally*. This approach is natural to the way an analyst thinks. Nevertheless, internally the cells of a cube will be created by aggregating the cells of the base cube (cf. Chapter 4.2.1) since a cube does not provide cells at levels of detail needed by the incremental approach.

Specifying the Analysis Graph

To specify the cubes of the analysis graph, we use OLAP operators *Rollup*, *Drilldown*, *Slice*, and *Intersection* as follows:

- *Rollup* may be used to create the "root" cube of the analysis graph. To ensure that no other cube of the analysis graph represents more coarse grained data than represented by the "root" cube, inner cubes may not be defined upon OLAP operator *Rollup*. Alternatively, the root cube of an analysis graph may be created using OLAP operators *Drilldown*, *Slice*, or *Intersection*.

- *Drilldown*, *Slice*, and *Intersection* are then used to refine the "root" cube of the analysis graph incrementally. Hence, moving downwards the analysis graph, cubes increasingly represent more fine-grained data (i.e., *Drilldown*, *Intersection*) or increasingly represent data for a subset of cells (i.e., *Slice*, *Intersection*). If an analyst needs to define a more general cube than provided by a given inner cube, the analyst moves back along the specification path until he arrives at a cube, which offers the required granularity or which offers a more coarse grained granularity. This cube will be then refined by applying OLAP operators *Drilldown*, *Slice*, or *Intersection* as described above such that the newly created cube represents the required data.

Figure 5.10 presents the syntax to specify cubes of the analysis graph. In the specification of the root cube of an analysis graph, one of the dimension attributes is marked to hold the primary dimension level using clause PRIMARY. When specifying the cubes of the analysis graph incrementally, OLAP operators may be applied only to analysis dimensions. This ensures that every cube of the analysis graph describes the level instances of the primary dimension level. The non-terminal symbols to specify OLAP operators are the same as already defined in Chapter 4.2.1 and are thus not repeated here. Note, that the measure attributes of any cube are computed from the base cube. Thus, the definition of a measure attribute refers to measure attributes of the base cube.

Although the specifying global cubes is similar to specifying cubes of an analysis graph, there are three semantic differences:

Syntax

```
RootCubeDef   ::=  cubeName [CubeSchemaDef | SchemaExtDef]
                       PRIMARY dimAttrName
                       AS (RollupDef | DrilldownDef | SliceDef | IntersectionDef)";".
InnerCubeDef  ::=  cubeName AS (DrilldownDef | SliceDef | IntersectionDef)";".
```

Figure 5.10: Syntax to specify the cubes of the analysis graph

1. Only the "root" cube of the analysis graph may be specified using OLAP operator *Rollup*. Every other cube of the analysis graph is defined using OLAP operator *Drilldown*, *Slice*, or *Intersection*.

2. The dimensions of a local cube are classified into one primary dimension and several analysis dimensions. Applying an OLAP operator on a local cube to specify another local cube is restricted to analysis dimensions to ensure that every local cube describes the level instances of the primary dimension level.

3. The *Slice*-conditions of local cubes may refer to the analysis rule's event. This offers the possibility to parameterize the static definition of a cube using the attributes of the event, whose values are different for each rule execution.

Since the cubes of an analysis graph are "local components" of the analysis rule for which they were specified, a cube of the analysis graph can be used only within this analysis rule. The motivation is as follows:

First, the definition of a cube may be parameterized by the event of the analysis rule, e.g., the event may be used by a *Slice* condition. Second, the cubes of an analysis graph are closely related to the rule's decision steps. This reduces reusability of cubes by other analysis rules. Third, changes in the specification of individual cubes should affect a single analysis rule only. Since several analysis rules may be specified for a certain primary dimension level, locality helps here to eliminate unwanted side-effects with other analysis rules (e.g., the redefinition of a cube in analysis rule 1 causes a decision step of analysis rule 2 to be invalid). Although the cubes of an analysis graphs are local components of their containing analysis rule, a pool of global cubes should be available to reduce redundancies in creating root cubes.

Example: Figure 5.11 depicts analysis rule RemoveHighPricedSoftDrinksAR, which models scenario 2 of our case study. Root cube SalesUpperAustriaThisQuarter refines global cube SalesArticlesUpperAustriaQuarters using OLAP operator SLICE. The cells of this cube are selected using the "sliding window" condition t.qtrId = eoq.quarterId (i.e., current

quarter determined by the rule's event). This condition is valid for all subsequent cubes. Since the cubes are defined upon the event parameter of the analysis rule, the analysis graph is closely tied to the analysis rule that contains the graph.

"More Specific"-Relationship between Cubes

To determine the order, in which the cubes of the analysis graph will be analysed when an analysis rule is executed, we introduce a "more specific"-relationship between two local cubes $r(R)$ and $s(S)$, which is based on (1) the "more specific"-relationship between their respective cube schemas and (2) on the subset-superset relationship of the cells of the base cube, which roll up to the cells of the two cubes. The two :

1. A cube schema R is "more specific" than cube schema S, denoted as $R < S$, iff (1) R and S are comparable (as introduced in Section 4.2.1) and (2) $\forall i \in \{1..n\}$: $dom(D_i^R) \leq dom(D_i^S)$ and (3) $\exists (D_i^R, D_i^S) : dom(D_i^R) < dom(D_i^S)$[2].

   Example: Cube schemas SalesArticlesUpperAustriaQuarters(p: Article, l: Store, t: Date, ...) and SalesArticlesUACitiesQuarters(p: Article, l: City, t: Quarter, ...) are comparable, since for each dimension attribute of cube schema SalesArticlesUpperAustriaQuarters there exists exactly one distinct dimension attribute of cube schema SalesArticlesUACitiesQuarters that belongs to the same dimension path and vice versa. Further, SalesArticlesUACitiesQuarters is "more specific" than SalesArticlesUpperAustriaQuarters since dimension attribute l:City of cube schema SalesArticlesUACitiesQuarters is "more specific" than dimension attribute l:Region of cube schema SalesArticlesUpperAustriaQuarters.

2. Predicate *rolls-up*, which we introduced in Chapter 4.2.1 to determine the set of cells that belong to a cube, is used to define the subset-superset relationship between the cells of two cubes. This predicate determines the set of cells from cube $b(B)$ that roll up to cells in U, i.e., *rolls-up(U, b(B))* $= \bigcup_{u \in U}$ *rolls-up(u, b(B))*.

A cube $r(R)$ is "more specific" than another cube $s(S)$, denoted as $r(R) < s(S)$, iff (1) $R < S \wedge$ *rolls-up(r(R)^{active}, b(B))* $\subseteq$ *rolls-up(s(S), b(B))* or (2) $R = S \wedge$ *rolls-up(r(R), b(B))* $\subset$ *rolls-up(s(S), b(B))*.

Example: Cube SalesArticlesUACitiesQuarters is "more specific" than cube SalesArticlesUpperAustriaQuarters, since the cube schema of SalesArticlesUACitiesQuarters is "more specific" than the cube schema of SalesArticlesUpperAustriaQuarters (cf. previous example) *and* both cubes refer to the set of cells of the base cube that belong to region Upper Austria.

[2] D_i^R denotes dimension attribute D_i of cube schema R.

```
DEFINE ANALYSIS RULE RemoveHighPricedSoftDrinksAR FOR a:Article

ON eoq:EndOfQuarter IF (a.category = 'Soft Drinks') AND (a.pricePerUnit > 25)

USING CUBES

  SalesUpperAustriaThisQuarter PRIMARY P AS SLICE T.qtrId = eoq.quarterId
    FROM SalesArticlesUpperAustriaQuarters;

  SalesUpperAustriaThisQuarterMonths AS DRILLDOWN T TO Month
    FROM SalesUpperAustriaThisQuarter;

  SalesUACitiesThisQuarter AS DRILLDOWN L TO City
    FROM SalesUpperAustriaThisQuarter;

  SalesLinzThisQuarter AS SLICE L.name = 'Linz'
    FROM SalesUACitiesThisQuarter;

  SalesLinzThisQuarterMonths AS INTERSECTION
    FROM SalesLinzThisQuarter, SalesUpperAustriaThisQuarterMonths;

BEGIN

  ANALYZE SalesUpperAustriaThisQuarter
    TRIGGER ACTION IF SalesTotal < 100000
    DETAIL ANALYSIS IF SalesTotal < 500000;

  ANALYZE SalesUpperAustriaThisQuarterMonths
    [ v1 PRIMARY P
          IS (SalesTrend (TREND(s1.SalesTotal, s2.SalesTotal, s3.SalesTotal)))
                  FOR CELLS s1 SLICE T = eoq.1stMon AND P = a.articleId,
                            s2 SLICE T = eoq.2ndMon AND P = a.articleId,
                            s3 SLICE T = eoq.3rdMon AND P = a.articleId; ]
    TRIGGER ACTION IF (v1.SalesTrend = 'DOWN');

  ANALYZE SalesUACitiesThisQuarter
    [ v1 PRIMARY P IS (SalesLinz (SalesTotal)) SLICE L = 'Linz';
      v2 PRIMARY P IS (P P, SalesAVG (AVG(SalesTotal))
                   ROLLUP L TO ALL, T TO ALL; ]
    TRIGGER ACTION IF (v1.SalesLinz > v2.SalesAVG)
    DETAIL ANALYSIS IF (v2.SalesAVG < 30000);

  ANALYZE SalesLinzThisQuarterMonths
    [ v1 PRIMARY P
          IS (SalesRatio (((s2.SalesTotal - s1.SalesTotal)/s2.SalesTotal) * 100)
            FOR CELLS s1 SLICE T = eoq.1stMon,
                      s2 SLICE T = eoq.2ndMon; ]
    TRIGGER ACTION IF (v1.SalesRatio = -10);

TO EXECUTE (a@OLTP).withdrawFromMarket PARAMETERS market = 'Upper Austria'

END RemoveHighPricedSoftDrinksAR
```

Figure 5.11: Specification of analysis rule RemoveHighPricedSoftDrinksAR

Definition Analysis Graph

The *analysis graph* is a directed acyclic graph of cubes $AG = (V, E)$. V is a set of cubes, in which each cube has a selected dimension attribute A_i that represents the primary dimension level L^{prim} of the analysis rule. The edges $E \subseteq V \times V$ define direct "more specific" relationships between cubes. Any edge of the analysis graph has been specified by the analyst using OLAP operators *Drilldown*, *Slice*, or *Intersection*. An analysis graph must have a single "root" cube c_r.

Each cube of the analysis graph is arranged in a "more specific" hierarchy. OLAP operator *Rollup*, which may be used to create the "root" cube of the analysis graph, creates the "most general" cube of the analysis graph. OLAP operator *Drilldown* creates a "more specific" cube that represents measures in more detail. OLAP operator *Slice* creates a "more specific" cube that represents a subset of cells. OLAP operator *Intersection* creates a "more specific" cube that represents a subset of cells whose measures are represented in more detail. Although analysts specify cubes incrementally by using these OLAP operators, the complete set of direct "more specific" edges of the analysis graph may be a superset of the "more specific" edges that have been specified. To determine the order for analyzing the cubes of the analysis graph (i.e., a cube $r(R)$ will be analyzed before the sub-cubes of $r(R)$ are analyzed) when an analysis rule is executed, the set of "more specific" edges that have been explicitly specified by the user will be taken. This approach avoids rule firings that were not intended by the user since inferred "more specific" edges are ignored.

Example: The shaded area in Figure 5.12 depicts the analysis graph of analysis rule RemoveHighPricedSoftDrinksAR (cf. Figure 5.11), which consists of cubes SalesUpperAustriaThisQuarter, SalesUpperAustriaThisQuarterMonths, SalesUACitiesThisQuarter, SalesLinzThisQuarter, and SalesLinzThisQuarterMonths. Root cube SalesUpperAustriaThisQuarter is derived from the globally available cube SalesArticlesUpperAustriaQuarters. Dashed arrows depict how these cubes were specified incrementally by the analyst using the three OLAP operators. Bold arrows depict all direct "more specific" edges between these cubes. When analysis rule RemoveHighPricedSoftDrinksAR is executed, cube SalesUpperAustriaThisQuarter will be inspected first. If no definite decision could be made, either of cubes SalesUACitiesThisQuarter or SalesUpperAustriaThisQuarterMonths will be inspected next. These two cubes represent the alternative branches of the analysis graph for analyzing data and making a decision. Analysis will be continued using the remaining cubes of the analysis graph according to their "more specific"-relationships unless a definite decision was made or all cubes have been inspected. In the latter case, undecided problems will be forwarded to the analyst for further manual analysis.

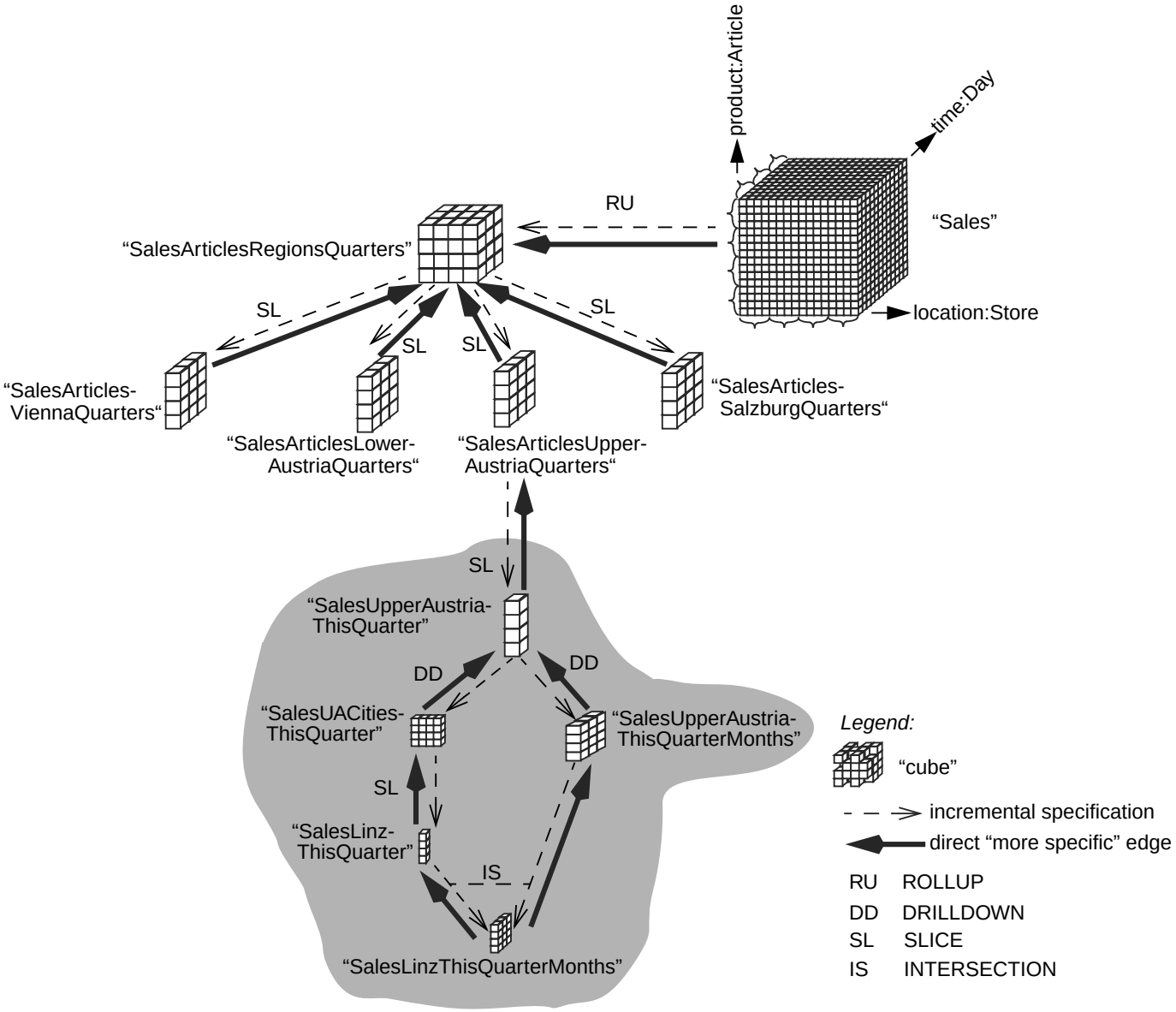

Figure 5.12: Analysis graph

5.3.2 Decision Making

A consequence of using the incremental top-down approach is that the analyst has three alternative choices when analyzing a particular cube $r(R)$ for decision making: (1) decide positive (i.e., execute the rule's action for a level instance of the primary dimension level), (2) decide negative (i.e., don't execute the rule's action for a level instance of the primary dimension level), or (3) continue analysis using a more detailed cube. In adopting this "decision-making model" for analysis rules, these choices will be represented as conditions (i.e., *trigger action condition*, *dismiss action condition*, and *detail analysis condition*) that are evaluated on the cells of $r(R)$. Since a *dismiss action condition* may be calculated as "NOT (*trigger action condition*) & NOT (*detail analysis condition*)", only the latter two conditions need to be specified as additional parts of an extended cube analysis statement, which is called *decision step*. For each cube of the analysis graph, the analyst may specify such a decision step.

Specifying Decision Steps

Decision steps will be specified using the syntax shown in Figure 5.13 after the BEGIN-clause of an analysis rule (cf. also the syntax to specify analysis rules in Figure 5.1). Since a decision step is an extended cube analysis statement (see Chapter 4.2.2 for details on specifying cube analysis statements), the analyst first specifies the name of a cube (defined in the rule's analysis graph) that will be analyzed. Next, the analyst may specify an arbitrary number of *decision variables*, which are named analytical expressions. Such a decision variable holds the virtual cells of an analytical expression. Finally, the conditions for decision making are specified. A condition may refer to the measure attributes of the cube's cells for which the decision step is specified or it may refer to the measure attributes of virtual cells identified by decision variables. Note that the virtual cells of a decision variable are derived from the cells of the main cube (i.e., the cube for which the decision step is specified) and from the cells of other cubes to which *drill across* should be performed. The dimension attributes of these virtual cells may not conform with the dimension attributes of the cubes of the analysis graph. Hence, the primary dimension attribute must be explicitly specified for these cells using clause PRIMARY. If a condition refers only to the measure attributes of the decision step's cube, the dimension attribute holing the primary dimension level is the same as for the cube that is analyzed. If a condition refers to a measure attribute of virtual cells identified by a decision variable, the analyst must specify clause PRIMARY to determine a dimension attribute of the decision variable to hold the primary dimension level. This means that each cell of a decision variable represents a particular level instance of the primary dimension level.

Example: In Figure 5.11, the decision step that analyzes cube SalesUpperAustriaThisQuarter defines predicates for the *trigger action condition* and the *detail analysis condition*.

Since this decision step defines no decision variables but refers to the measure attributes of cube SalesUpperAustriaThisQuarter instead, dimension attribute p (taken from the cube's schema) is the primary dimension attribute since it represents the primary dimension level of the analysis graph. The decision step that analyzes cube SalesUpperAustriaThisQuarterMonths refers to measure attribute SalesTrend of decision variable v1. Hence, dimension attribute P is the primary dimension attribute of v1 using clause PRIMARY P in the definition of the decision variable.

Alternatively, the *trigger action condition*, the *detail analysis condition*, or both conditions may be specified. If *trigger action condition* is omitted, its default value is FALSE, i.e., continue analysis by evaluating the *detail analysis condition*. If omitted, a *detail analysis condition* evaluates to TRUE, i.e., continue analysis using a decision step that analyzes a "more specific" cube.

Example: In Figure 5.11, the decision step that analyzes cube SalesLinzThisQuarterMonths defines *trigger action condition* as v1.SalesRatio = -10. The missing *detail analysis condition* evaluates to TRUE whenever a level instance of the primary dimension level does not satisfy the *trigger action condition*. Consequently, this decision step will never decide to dismiss the rule's action for a level instance of the primary dimension level.

Semantics of Decision Making

A level instance of the primary dimension level will be processed by an analysis rule if (1) the rule's event occurred for this level instance and if (2) this level instance satisfies the rule's primary condition. In such a situation, the level instance of the primary dimension level belongs to the rule's *analysis scope*. In decision making, an analysis rule examines for each level instance l of the analysis scope, whether the rule's specified OLTP method should be invoked in an operational system or whether l should be forwarded to further manual analysis. In making a decision, the decision steps of the analysis rule will be processed using the "more specific" order of the cubes that constitute the analysis graph as introduced above.

Before a decision step can be evaluated for a level instance of the analysis scope, incomplete or inconsistent specifications of decision steps must be made complete or consistent. Three cases can be distinguished in this respect: (1) The analyst did not define a decision step for a particular cube $r(R)$. (2) The analyst defined a decision step for $r(R)$ but did not specify one of the two conditions. (3) Both conditions have been specified but the conditions overlap. The first case is handled by introducing a default decision step. The second case is handled by adding predicate FALSE for a missing *trigger action condition* and a predicate TRUE for a missing *detail analysis condition*. The third case concerns inconsistencies between both conditions. If both conditions overlap, a *trigger action condition*

Syntax

```
DecisionStepDef       ::=  ANALYZE cubeName
                             [ "[" DecisionVarDef { "," DecisionVarDef } "]" ]
                             (TriggerActionCondDef
                              | DetailAnalysisCondDef
                              | TriggerActionCondDef DetailAnalysisCondDef) ";" .
DecisionVarDef        ::=  decisionVarName PrimDimAttrDef IS AnalyticExprDef.

TriggerActionCondDef  ::=  TRIGGER ACTION IF ConditionDef.

DetailAnalysisCondDef ::=  DETAIL ANALYSIS IF ConditionDef.

ConditionDef          ::=  CondLHSDef compOper CondRHSDef.

CondLHSDef            ::=  (decisionVarName "." measureAttr
                              | cubeMeasureAttr
                              | arithmeticExpr).
CondRHSDef            ::=  (decisionVarName "." measureAttr
                              | cubeMeasureAttr
                              | arithmeticExpr
                              | constantValue).
PrimDimAttrDef        ::=  PRIMARY dimAttrName.
```

Figure 5.13: Syntax to specify decision steps

evaluating to TRUE is given precedence over a *detail analysis condition* evaluating also to TRUE. Informally, decision making is carried out as follows:

1. The analysis rule's *analysis scope* is the set of objects in the primary dimension satisfying the primary condition. This set of objects is also the analysis scope of the "root cube".

2. The *analysis scope* of any other cube (i.e., "inner cube" or "leaf cube") consists of every level instance of the primary dimension level (1) that is in the analysis scope of some supercube and (2) that satisfies the *detail analysis condition* (but not the *trigger action condition*) of that supercube.

3. The *trigger action set* of cube $r(R)$ is the set of objects in the rule's analysis scope satisfying the *trigger action condition* of the decision step of $r(R)$.

4. The *detail analysis set* of cube $r(R)$ is the set of objects in the rule's analysis scope satisfying the *detail analysis condition* (but not satisfying the *trigger action condition*) of the decision step of $r(R)$.

5. The rule's action is executed for a level instance l of the rule's analysis scope if l belongs to any of the cubes' trigger action sets. Thus, the *trigger action set* of an analysis rule is the union of the trigger action sets of all cubes.

6. A level instance l of the rule's analysis scope must be forwarded to an analyst for further manual analysis, if (1) l belongs to any of the "leaf cubes"' detail analysis sets and if (2) l does not belong to the rule's trigger action set. Thus, the *manual analysis set* of an analysis rule is the difference between the union of the detail analysis sets of its "leaf cubes" and the rule's trigger action set.

7. Action execution is dismissed for a level instance l of the rule's analysis scope, if (1) l is not member of the rule's *trigger action set* and if (2) l is not member of the rule's *manual analysis set*. Thus, the *dismiss action set* of an analysis rule is the difference between the rule's analysis scope and the union of the *trigger action set* and the *detail analysis set*.

The rationale behind this approach is that if an object is considered to be analyzed manually due to analysis in one branch of the analysis graph but is in the *trigger action set* of another branch of the analysis graph, then manual analysis is superfluous since the analysis in the other branch has already given the answer what to do.

Formally, we introduce for each cube $r(R)$ of the analysis graph $AG = (V, E)$ the two conditions $TrgA_{r(R)}(l)$ and $DtlA_{r(R)}(l)$, where $TrgA_{r(R)}(l)$ represents the completed trigger action condition and $DtlA_{r(R)}(l)$ represents the completed detail analysis condition. To resolve possible inconsistencies between both conditions, the completed detail anlysis

condition is extended further by clause $\neg(TrgA_{r(R)}(l))$. $DsmA_{r(R)}(l)$ represents the decision to dismiss action execution for level instance l as the complement of $TrgA_{r(R)}(l)$ and $DtlA_{r(R)}(l)$. These conditions need to be distinguished from predicates $trgA_{r(R)}$ (representing the *trigger action condition*) and $dtlA_{r(R)}$ (representing the *detail analysis condition*), which have been specified by the analyst.

$$TrgA_{r(R)}(l) = \begin{cases} trgA_{r(R)} & : \text{ if } trgA_{r(R)} \text{ is defined for } r(R) \\ \text{FALSE} & : \text{ else} \end{cases} \tag{5.3}$$

$$DtlA_{r(R)}(l) = \begin{cases} dtlA_{r(R)} \wedge \neg(TrgA_{r(R)}(l)) & : \text{ if } dtlA_{r(R)} \text{ is defined for } r(R) \\ \neg(TrgA_{r(R)}(l)) & : \text{ else} \end{cases} \tag{5.4}$$

$$DsmA_{r(R)}(l) = \neg(TrgA_{r(R)}(l)) \wedge \neg(DtlA_{r(R)}(l)) \tag{5.5}$$

To mimic the top-down approach in analyzing cubes, we introduce $DtlA^{+}_{r(R)}(l)$ as the extended detail analysis condition of cube $r(R)$ and $TrgA^{+}_{r(R)}(l)$ as the extended trigger action condition of cube $r(R)$.

$$DtlA^{+}_{r(R)}(l) = DtlA_{r(R)}(l) \wedge \left(\bigvee_{r(R)_j \in super(r(R))} DtlA^{+}_{r(R)_j}(l) \right) \tag{5.6}$$

$$TrgA^{+}_{r(R)}(l) = TrgA_{r(R)}(l) \wedge \left(\bigvee_{r(R)_j \in super(r(R))} DtlA^{+}_{r(R)_j}(l) \right) \tag{5.7}$$

$$DsmA^{+}_{r(R)}(l) = DsmA_{r(R)}(l) \wedge \left(\bigvee_{r(R)_j \in super(r(R))} DtlA^{+}_{r(R)_j}(l) \right) \tag{5.8}$$

$DtlA^{+}_{r(R)}(l)$, $TrgA^{+}_{r(R)}(l)$, and $DsmA^{+}_{r(R)}(l)$ are defined recursively using predicate *super(r(R))*, which determines the set of direct more general cubes of some cube $r(R)$[3]. Thus, decision making starts at the most general cube of the analysis graph (i.e., the "root" cube). $DtlA^{+}_{r(R)}(l)$ is satisfied (1) if $DtlA^{+}(l)$ of one of the direct more general cubes of $r(R)$ is satisfied and (2) if the detail analysis condition of $r(R)$ (i.e., $DtlA_{r(R)}(l)$) is satisfied. $TrgA^{+}_{r(R)}(l)$ is satisfied (1) if $DtlA^{+}(l)$ of one of the direct more general cube of $r(R)$ is satisfied and (2) if the trigger action condition of $r(R)$ (i.e., $TrgA_{r(R)}(l)$) is satisfied. Finally, $DsmA^{+}_{r(R)}(l)$ is satisfied (1) if $DtlA^{+}(l)$ of one of the direct more general cube of $r(R)$ is satisfied and (2) if the dismiss action condition of $r(R)$ (i.e., $DsmA_{r(R)}(l)$) is satisfied.

[3]Note that *super(r(R))* $= \emptyset$ if $r(R)$ is the "root" cube of the analysis graph.

The semantics of decision making is defined declaratively by the boolean functions *executeOLTPMethod(l)*, *analyzeManually(l)*, and *dismissOLTPMethodExec(l)*, which determine for an entity l, whether the analysis rule's OLTP method should be executed in an OLTP system, whether the analyst should perform further manual analyzes, or or whether l should be dismissed from action execution and manual analysis.

$$executeOLTPMethod(l) = \bigvee_{r(R) \in V} TrgA^{+}_{r(R)}(l) \tag{5.9}$$

$$analyzeManually(l) = \left(\bigvee_{r(R) \in LC} DtlA^{+}_{r(R)}(l) \right) \wedge \neg(executeOLTPMethod(l)) \tag{5.10}$$

$$dismissOLTPMethodExec(l) = \neg(executeOLTPMethod(l) \vee analyzeManually(l)) \tag{5.11}$$

Function $executeOLTPMethod(l)$ returns "TRUE" if analyzing the cells of at least one cube of V results in the decision to trigger the rule's action for l. Function $analyzeManually(l)$ returns TRUE if analyzing the cells of at least one "leaf cube" (LC denotes the subset of V for which no "more specific" cubes exist) results in the decision to detail analysis *and* no other decision step decides to trigger the rule's action for l. If none of the two boolean functions are satisfied, l is considered to be decided negatively (i.e., *dismissOLTPMethodExec(l)* returns TRUE).

5.4 Procedural Evaluation of Analysis Rules

In this section, we present an approach for evaluating analysis rules procedurally. This approach complements and respects the declarative semantics of decision making as defined for analysis rules in Section 5.3.2. It can be used to implement the rule scheduler and the decision generator of an active data warehouse management system or, as shown in Chapter 9, it can be used as basis for implementing analysis rules using off-the-shelf database technology (i.e., SQL and triggers). The major principles of the procedural evaluation approach presented below are:

1. Analysis rules will be executed in a *set-oriented* manner, collecting not only all objects affected by the same event, but also objects affected by different events that belong to the same event type occurring during the same ADW cycle (i.e., the period between loading data from operational systems and exporting generated decisions back to these systems).

2. Evaluating the decision steps of an analysis rule is carried out by traversing the analysis graph in a *depth-first* manner, respecting the incremental top-down approach in decision making.

Although analysis rules are specified *instance-oriented* (i.e., analysis rules are specified as if they were fired for each level instance of the primary dimension level separately), an efficient implementation will execute an analysis rule in a *set-oriented* manner (i.e., a single execution of an analysis rule will generate decisions for a set of level instances of the primary dimension level). This avoids executing the same rule multiple times for different objects. In our approach we go even further. We also collect all objects affected by different events that belong to the same event type and that occur at the same chronon. This is the case with relative temporal events, where the initiating method events occurred in the same ADW cycle for different objects of the primary dimension level.

Example: Consider analysis rule ChangePriceOfSoftDrinksAR, which is defined to fire for every occurrence of event TwentyDaysPastLaunch that indicates the expiry of the twenty day observation period after an article was launched on a market. This event is a relative temporal event, whose instances occur for individual level instances of dimension level Article. At a particular chronon c, analysis rule ChangePriceOfSoftDrinksAR will be fired once for all instances of TwentyDaysPastLaunch that occur at c.

In the description of the procedural evaluation of analysis rules, we collect these events into event set E. For the sake of uniform presentation, we will use the event set E also for calendar events, although we know that in this case E will contain only one event instance.

Figures 5.14 and 5.15 outline the procedural semantics of analysis rule execution and decision step evaluation. They show how an analysis rule may be executed set-oriented as sketched above. Procedure ExecuteAnalysisRule fulfills two basic tasks: (1) Determine the analysis scope (i.e., the level instances of the primary dimension level, for which the analysis rule should be executed). (2) Initiate traversing the analysis rule's analysis graph at the root cube. Task (1) is accomplished by using the set of events E, which triggered the rule execution. E contains a set of event instance (of the same event type), which occurred at the same chronon. If E contains a calendar event (i.e., an absolute temporal event or a periodic temporal event), then *all* level instances of the primary dimension level will be evaluated against the primary condition (denoted by prim_cond(AR, l) in line 2). If E contains relative temporal events, only level instances of the primary dimension level for which these events occurred (denoted by occurredFor(e) = l in line 4) will be evaluated against the primary condition. Task (2) is accomplished by calling procedure EvaluateDecisionStep for the root decision step using the analysis scope (denoted by AS, holding the set of level instances to be analyzed) as the parameter. Procedure EvaluateDecisionStep returns the set of level instances, for which the rule's action should be executed (denoted by TrgA) and the set of level instances for which manual analyses should be carried out (denoted

ExecuteAnalysisRule (↓AR : Rule, ↓E : Set, ↑T : TXSpec, ↑TrgA : Set,
↑ManA : Set)

input : *analysis rule* AR *to be executed*;
set of events E *that occurred at the same chronon, each triggering rule* AR;

output : *transaction* T *to be executed in an OLTP system*;
set of objects TrgA *for which* T *must be executed*;
set of objects ManA *for which further manual analyzes should be carried out*;

Declarations

```
AS : Set   := {}        /* analysis scope */
r  : Cube               /* root cube */
objEval : ArrayOfSet    /* array indexed by cubes of the analysis graph;
                           objEval[r] holds the set of objects for which the
                           decision step of r has already been evaluated */
```

Begin

```
#0 : initialize objEval to {} for each cube of the analysis graph (not shown)
#1 : if (E contains a "calendar event") then
#2 :   AS := {l ∈ primary dimension level of AR | prim_cond(AR, l)}
#3 : elsif (E contains "relative temporal events") then
#4 :   AS := {l | (∃ e ∈ E : occurredFor(e) = l) ∧ prim_cond(AR,l)}
#5 : end if
#6 : r := "root cube" of the analysis graph of AR
#7 : EvaluateDecisionStep (r, AS, objEval, ManA, TrgA)
#8 : ManA := ManA \ TrgA
#9 : T := "action" of AR
```

End

Figure 5.14: Procedural semantics of analysis rule execution

by ManA). Evaluating the decision steps of different branches of the analysis graph may generate conflicting decisions (i.e., an object may be considered to be analyzed manually and it may be considered for executing the rule's action in the OLTP system). Remember that such "intra-rule conflicts" are resolved by giving higher priority to executing the rule's action in the OLTP system. This is achieved by reducing the set ManA by the set TrgA (line 8). Procedure ExecuteAnalysisRule finally returns the rule's action (T), the set of level instances for which this action should be executed (TrgA), and the set of level instances for which further manual analyses should be carried out (ManA).

Figure 5.15 outlines the *depth-first* traversal approach for evaluating the decision steps of the analysis graph. Parameter AS represents the analysis scope of cube r, i.e., the set of objects for which the decision step of cube r should be evaluated. Since a cube of the analysis graph may be defined as the *Intersection* of two other cubes, an object may be analyzed by a given decision step more than once. The rationale behind this is that whenever OLAP operator *Intersection* is used to define a cube r, the number of cubes from which r is derived also determines the number of alternative "more specific" edges in the analysis graph. Hence, cube r can be reached by traversing any of these "more specific" edges.

Example: Figure 5.16 depicts the analysis graph of analysis rule RemoveHighPricedSoftDrinksAR. The numbers next to the bold dashed arrows indicate the depth-first traversal order. Since cube SalesLinzThisQuarterMonths is defined as "INTERSECTION FROM SalesLinzThisQuarter, SalesUpperAustriaThisQuarterMonths", the decision step of this cube might be evaluated twice for a particular Article.

To avoid redundant evaluations, all objects that have already been analyzed by the current decision step will be withdrawn from being analyzed again using array objEval (line 1), which holds for each cube r the set of objects that have already been analyzed by r.

Evaluating the decision step of cube r means to evaluate the completed *trigger action condition* and the completed *detail analysis condition* (denoted as $TrgA_{r(R)}$ and $DtlA_{r(R)}$ in Section 5.3.2) for the objects in AS. For a given level instance l, first condition $TrgA_{r(R)}$ is evaluated (represented by predicate triggerAction(r, l)), then condition $DtlA_{r(R)}$ is evaluated for l (represented by predicate detailAnalysis(r, l)). If l satisfies condition $TrgA_{r(R)}$, it will be added to the set of level instances for which the rule's action should be executed in the OLTP system (represented by TrgA in line 2) and it will be removed from any further processing (see AS \ TrgA in line 3). If the current cube r is a "leaf cube", all objects for which further analyses should be carried out are considered to be analyzed manually (line 5). If r is an "inner cube" (here, the "root cube" is also an "inner cube"), procedure EvaluateDecisionStep is called recursively for each descendant of the current cube r with input value d representing the descendant, input value DtlA \ TrgA representing the analysis scope for subcube d, and input value objEval holding for each cube the set of objects that have been already analyzed by that cube. Procedure EvaluateDecisionStep returns array

```
EvaluateDecisionStep(↓r : Cube, ↓AS : Set, ↕objEval : ArrayOfSet, ↑ManA : Set,
                     ↑TrgA : Set)

input : cube r whose decision criteria are evaluated;
        set of objects AS which need to be analyzed by the decision criteria of r;
        array objEval holds sets of objects for which analysis is already complete;
        set of objects ManA for which manual analyzes should be carried out
          as determined by decision steps of (already visited) branches of
          the analysis graph;

output : array objEval updated for r and for the subcubes of r whose decision
           steps were evaluated;
         set ManA extended by the set of objects that need to be analyzed
           manually as determined by the decision step of r or by the decision
           steps of subcubes of r;
         set of objects TrgA for which the rule's action must be executed;

Declarations

DtlA : Set         := {} /* set of objects for which further analyzes should be
                            carried out at subcubes of r */
subcubeTrgA : Set  := {} /* trigger action set of a subcube of r */
subcubeManA : Set  := {} /* manual analysis set of a subcube of r */

Begin

  #1 :  AS := AS \ objEval[r]
  #2 :  TrgA := { l ∈ AS | triggerAction(r, l) }
  #3 :  DtlA := {l ∈ AS \ TrgA | detailAnalysis(r, l)}
  #4 :  ManA := {}
  #5 :  if (r is "leaf cube") then
  #6 :    ManA := DtlA
  #7 :  elsif (r is "inner cube" ) then
  #8 :    foreach d ∈ {direct descendants of r} do
  #9 :      EvaluateDecisionStep(d, DtlA \ TrgA, objEval, subcubeManA, subcubeTrgA)
  #10 :     TrgA := TrgA ∪ subcubeTrgA
  #11 :     ManA := ManA ∪ subcubeManA
  #12 :   end foreach
  #13 : end if
  #14 : objEval[r] := objEval[r] ∪ (DtlA \ TrgA)

End
```

Figure 5.15: Procedural semantics of decision step evaluation

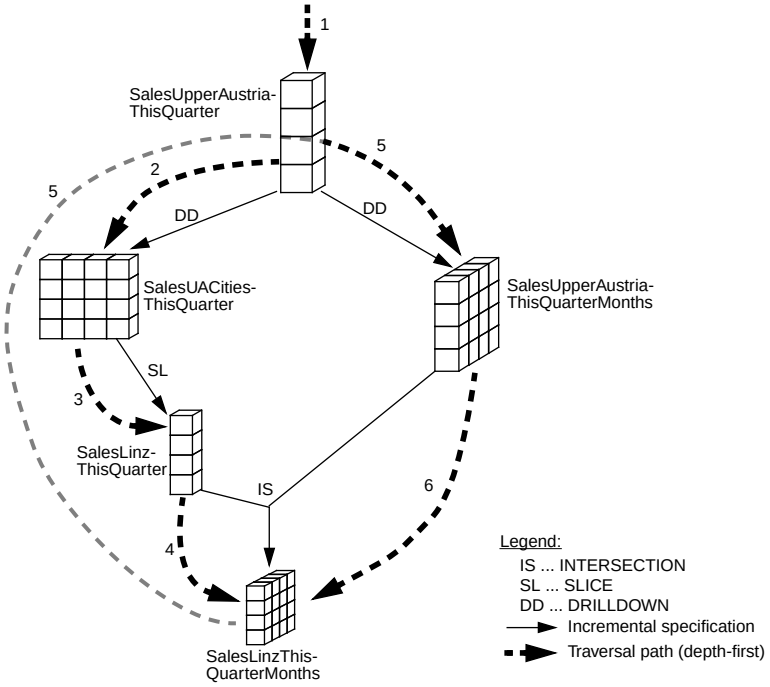

Figure 5.16: Analysis graph traversal order

objEval extended by the objects that were analyzed at the subcubes of r, the set subcubeManA holding the objects for which further manual analyses should be carried out as determined at the subcubes of r, and the set subcubeTrgA holding the objects for which condition $TrgA_{r(R)}$ of some decision step of d or of a subcube of d is satisfied. Notice, the objects in ManA consists of the set of objects to be analyzed manually according to the current evaluation stage of the analysis graph. If during later rule processing in another branch of the analysis graph, condition $TrgA_{r(R)}$ of some decision step is satisfied by some objects in ManA, these objects will be added to the set of level instances for which the rule's action must be executed in the OLTP system. Consequently, these objects are removed from ManA (line 8 in Fig. 5.14) to avoid decision conflicts. Procedure EvaluateDecisionStep finally extends the set objEval[r], which represents the set of objects that need not be analyzed again by the decision step of cube r if processed by another branch of the analysis graph.

5.5 Summary

Analysis rules extend passive data warehouses with reactive capabilities to automatize routine decision tasks (i.e., decision tasks that occur on a pre-determined schedule using standardized procedures to data analysis and decision making). These extensions are twofold:

1. From an object-oriented perspective, *events* and *actions* are added to a data warehouse. The object-oriented perspective facilitates identifying the objects for which analyzes should be carried out (i.e., the *primary dimension level*). We use a basic temporal event model to describe the timepoints at which analysis rules should be executed in the data warehouse. This event model comprises relative temporal events and calendar events (i.e., absolute temporal events and periodic temporal events). The action of an analysis rule represents the decision task to be executed in an operational system. Hence, adding actions to the data warehouse means to provide specifications of transactions that may be invoked in an OLTP system.

2. From a multidimensional perspective, we introduced an *incremental top-down approach* to specify multidimensional analyzes in a data warehouse. Following this approach, first a coarse grained cube will be inspected and analyzed with respect to execute the rule's action. If no definite decision can be made, more specific cubes will be created and analyzed. Decision making terminates for a particular object if (1) a definite decision could be made for that object (i.e., YES, execute the action in the OLTP system or NO, don't execute the action in the OLTP system) or (2) if no further cubes can be created such that the decision task is forwarded to the analyst for further manual treatment. "Decision making" is automatized (1) by specifying the cubes that need to be analyzed (i.e., *analysis graph*) and (2) by specifying the conditions that represent the decision criteria as extended cube analysis statements (i.e., *decision step*).

Analysis rules are the "glue" between the two perspectives. An analysis rule is defined for a particular primary dimension level and is "fired" at the occurrence of some temporal event. The level instances of a rule's primary dimension level are described by the cubes of the analysis graph. The decision steps are evaluated in the "more specific"-order derived from the analysis graph. If a positive decision was generated, the active data warehouse will export that decision to an OLTP system where the rule's action will be executed. If a negative decision was generated, no further actions will be carried out by the data warehouse. If the analysis rule could not generate a definite decision, further analyzes must be carried out manually by analysts.

Part III

An Extended Approach to Analysis Rules

Chapter 6

Analysis Rules Revisited

Contents

Analysis rules introduced so far provide excellent technical concepts for automatizing simple routine decision tasks and the routinizable elements of simple semi-routine decision tasks. However, more complex decision tasks pose additional requirements that go beyond what is currently supported by analysis rules. We have found that the following requirements must be satisfied by analysis rules to support complex real-world decision tasks:

1. *Support for user-defined logical events.* A user-defined logical event is raised when the database has arrived at a particular (i.e., user-defined) state, which is typically represented by a query. Such events identify situations that represent a particular logical state of the data warehouse. For instance, consider a decision task that will be carried out if the price of a particular article has been changed more than two times during the last four months. Logical events complement temporal events, since the latter only represent the periodic schedule upon which decision tasks will be carried out.

2. *Support for positive, negative, and irresolute decision criteria.* A positive decision criterion identifies s for which an analyst will definitely carry out a particular action. A negative decision criterion identifies those cases for which an analyst will definitely not carry out a particular action. Finally, an irresolute decision criterion identifies those cases for which an analyst cannot make a definite decision but must perform further analyzes instead. A system supporting these three types of decision criteria offers the convenience to specify conditions that are appropriate to a particular situation. For instance, a situation in which a price change should not be carried out if an article's sales figures remained constantly high during the last six months can be easily modeled using a negative decision criterion but is rather difficult to specify if the system does not support negative decision criteria.

3. *Support for weak and strong decision criteria.* A weak decision criterion represents the conditions that must be satisfied to consider an object as "tentatively decided". Such a tentative decision is valid unless evaluating another decision criterion produces a contradicting decision. Conversely, a strong decision criterion represents the conditions that must be satisfied to consider an object as "definitely decided", which cannot be overruled by another decision criterion. After evaluating all decision criteria, tentative decisions are turned into definite decisions. A system supporting weak and strong decision criteria allows modeling "default situations" (i.e., weak decision criteria that are applicable to a wide range of cases) but that still may be overruled by "exceptional cases" (i.e., weak or strong decision criteria that are applicable to only a small number of cases). For instance, consider a situation in which the prices of soft drinks must be changed if production costs increased by more than 3 % on average during the past two months. An exception to this default situation exists for soft drinks whose profit margins exceed a particular threshold: In such a situation, a soft drink's price will be changed only if production costs increased by more than 10 % on average during the last two months. Irrespective from changes in production costs, the price of any soft drink must be changed if sales figures dropped during the past three months by more than 3 %.

4. *Support for alternative decision-making models.* A decision-making model is an approach to compile a single global decision out of several "expert decisions" e.g., the global decision is based on the consensus of all experts. Applied to analysis rules, an "expert decision" is the result of evaluating a particular decision step, whereas the global decision is the result of executing an analysis rule as a whole. Since various real world decision tasks may require different decision-making models, a catalog of decision making models should be provided. Further, a decision-making model should be either uniformly applicable to all decision steps of an analysis rule or, alternatively, each decision step may compile its decision using an individual decision-making model. For instance, consider the situation as to whether a loan application should be granted if the amount applied for is too high such that further analyzes become necessary. In such a situation, the loan application will be granted

if the applicant's disposable income exceeds the threshold of 80 % of monthly installment and if the loan period is 2 years or less. Since both decision criteria must be satisfied in order to grant the loan application, the decision-making model underlying this decision task is based on the "positive consensus" of all participating decision criteria. Alternative decision-making models may be based on the "positive majority" of participating decision criteria, they may be based on a "single positive veto", etc. The decisions that are generated by a decision-making model must be free from conflicts between the set of positive decisions, the set of negative decisions, and the set of irresolute decisions.

5. *Support for flexible decision parameters.* The parameters that need to be specified for positive decisions (i.e., the parameters of an action execution such as a price change by +5 %) may be defined (1) independently from the analyzes that have been carried out in the data warehouse, (2) they may be defined as the results of analyzes that have been carried out in the data warehouse, or (3) they may be defined as the results of carrying out separate analyzes in the data warehouse. Complex decision tasks may use default parameters that are bound to positive decisions if more specific parameter bindings are not available. For instance, consider the decision task as to whether the price of an article should be changed: The default parameter binding (i.e., the price changing ratio) is +5 %, meaning that the article's current price should be increased by 5 %. An exception to this binding is the case in which the article's sales figures increased by less than 7 % on average during the last few months; in such a situation the price will be increased by only 2.5 %.

6. *Support for incremental specialization of decision tasks.* Decision tasks, which concern a particular set of objects, may be specialized for a subset of these objects by adding decision criteria, by refining decision criteria, by providing exceptions, or by refining the parameters of decisions. A globally observable property of a specialized decision task is that the set of undecided cases decreases and the set of (positively or negatively) decided cases increases if compared with the decided and undecided cases of the general decision task. For instance, the decision task as to whether a loan application should be granted (as described above) represents the general decision task which applies to any loan application. Specialization becomes necessary for the case if there are decision criteria that apply to personal loan applications or to mortgage loan applications only. Personal loan applications will be analyzed using the specialized decision criteria if a given loan application belongs to the set of undecided cases of the general decision task that will be applied to any loan application. Similarly, mortgage loan applications will be evaluated by applying the decision criteria of the general decision task first. If the decision as to whether a mortgage loan application should be granted remains unknown after using the general decision task, the loan application will be evaluated further using the specialized decision task for mortgage loan applications.

In the remainder of this chapter, analysis rules as introduced in Chapter 5 will be revisited

according to the above mentioned requirements. For these requirements, we will identify sub-goals and technical properties. The sub-goals are used to evaluate the expressiveness of decision support systems that use rules for decision modeling. The technical properties concern the correctness of diverse components to avoid contradictory decisions.

6.1 Logical Events

Events identify the timepoints at which analyzes should be initiated. These timepoints are either specified on the basis of a calendar or on the basis of evaluating logical conditions on the state of the data warehouse. So far, we used a basic event model that consists of OLTP method events, relative temporal events, and absolute temporal events. Absolute temporal events and relative temporal events are used to model a calendar schedule upon which analysis rules are fired whereas OLTP method events are the "reference points" to which relative temporal events may refer. Besides these temporal aspects of decision tasks, an active data warehouse should also provide more complex (i.e., logical) events that fire analysis rules when a user-defined logical condition is satisfied. Such user-defined conditions are typically expressed by means of a query language, which evaluates the state of dimensions, facts, and other events within the data warehouse.

A system supporting complex real-world decision tasks should satisfy the following sub-goal:

- **Freedom of specifying temporal and logical events.** This sub-goal requires that an analyst is free to specify events on a calendar schedule or to specify events using a query language by referring to dimensions, facts, and other events within the data warehouse.

Since logical events have been well established in the field of active database systems for several years, Section 7.1 briefly summarizes logical events and briefly sketches how logical events can empower analysis rules in active data warehouses.

6.2 Decision Criteria

Decision criteria identify the cases for which an analyst should or should not carry out a particular action in an OLTP system or for which the analyst must perform further analyzes instead. According to the proposed incremental approach for decision making within analysis rules, decision criteria are represented by the *trigger action condition* (i.e., identify the objects of the primary dimension level for which a particular action should be carried out in an OLTP system) and the *detail analysis condition* (i.e., identify the objects

of the primary dimension level for which further analyzes are necessary), which will be evaluated on the cells of a cube that represents the data to be analyzed. The triplet $\langle$ *cube*, *trigger action condition*, *detail analysis condition* $\rangle$ is called *decision step*, since the two conditions represent one step within a complex decision making procedure.

6.2.1 Criteria for Positive, Negative, and Irresolute Decisions

A positive decision criterion identifies those cases for which an analyst can decide autonomously to perform a particular action in an OLTP system. Conversely, a negative decision criterion identifies those cases for which an analyst can decide definitely to be withdrawn from action execution an from further analysis. Finally, an irresolute decision criterion identifies those cases for which an analyst will perform further analyzes before deciding positively or negatively. Although the set of unresolved cases is the complement of positive and negative cases, we allow to explicitly specify irresolute decision criteria since some analyzes require to specify positive and irresolute decision criteria or negative and irresolute decision criteria. Compared with analysis rules, a decision step's trigger action condition represents a positive decision criterion since the rule's action will be executed for all cases (i.e., the level instances of the primary dimension level) that satisfy the trigger action condition. Further, a decision step's detail analysis condition represents an irresolute decision criterion since the analysis rule will perform further analyzes for all cases that satisfy this condition using a "more specific" cube of the rule's analysis graph.

In our basic approach, negative decision criteria (i.e., the cases for which both no action should be carried out and no further analyzes should be performed) could not be specified explicitly within an analysis rule. Instead, a negative decision criterion will be computed as the complement of the positive and the irresolute decision criteria that were specified for the same decision step. The following sub-goal characterizes the analyst's freedom of specifying decision criteria in a way that is most appropriate to a particular decision situation:

- **Freedom of Specifying Criteria for Positive, Negative, and Irresolute Decisions.** This sub-goal requires that an analyst can decide freely to specify criteria for positive, negative, and/or irresolute decisions without violating the two properties mentioned above. As introduced so far, analysis rules support freedom of specifying decision criteria only partially, since an analyst has only the following choices when specifying decision criteria:

  1. *Specify no condition.* In this situation, the trigger action condition and the detail analysis condition are evaluated using their default decision values.
  2. *Specify the trigger action condition.* In this situation, the trigger action condition is evaluated using the criteria that have been specified whereas the detail analysis condition is evaluated using its default decision value.

3. *Specify the detail analysis condition.* In this situation, the trigger action condition is evaluated using its default decision value whereas the detail analysis condition is evaluated using the criteria that have been specified.
4. *Specify the trigger action condition and the detail analysis condition.* In this situation, no default truth values have to be used for condition evaluation.

An analyst cannot explicitly specify a negative decision criterion, which we call *dismiss action condition* when used in the context of a decision step. To dismiss objects of the primary dimension level from action execution (and from further analyzes) at a decision step, the analyst must adjust the trigger action condition and the detail analysis condition in order to be semantically equivalent with an alternatively specified *dismiss action condition.*

If not specified, a *trigger action condition* evaluates to *false* (i.e., don't carry out the rule's action) and a *detail analysis condition* evaluates to *true* (i.e., perform more detailed analyzes). Since conditions may overlap, there may occur conflicts between the decision to execute the rule's action in an OLTP system for a given object of the primary dimension level and the decision to continue analysis at more detailed cubes for the same object. Such conflicts will be resolved by executing the rule's action instead of performing detailed analyzes. To ensure that a decision step generates correct decisions, the following technical properties must be satisfied:

- **Completeness.** This property requires that the disjunction of a decision step's decision criteria must be true for all objects of the primary dimension level that have to be analyzed by the decision step. Completeness is satisfied by a decision step since (1) default values exist for a decision step's positive and irresolute decision criteria and (2) the decision step's negative decision criterion is computed as the complement of the positive and the irresolute decision criterion, which comprises the remaining level instances.

- **Uniqueness of Decision.** This property requires that a single decision is generated for an object of the primary dimension level that has to be analyzed by a decision step. Uniqueness of decision is satisfied by a decision step since (1) the decision criteria satisfy the completeness property and (2) possible conflicts between a decision step's positive decision and a decision step's irresolute decision are resolved in of the positive decision.

In Section 7.2, we will extend decision steps with the possibility to specify *dismiss action conditions*, which is not simply a syntactical extension but also requires the redefinition of decision criteria semantically to be compatible with the original approach.

6.2.2 Weak and Strong Decision Criteria

Weak decision criteria may be used to model default conditions, which – if satisfied – consider an object as "tentatively decided" unless the object satisfies a decision criterion that represents a definite decision. Conversely, an object satisfying a strong decision criterion is considered as "definitely decided", which cannot be overruled by a "tentative decision". A system providing decision criteria for selective decision making, as it is the case with authorization mechanisms in database systems [BJS99], must satisfy the following sub-goal:

- **Freedom of Specifying Criteria for Tentative and Definite Decisions.** This sub-goal requires that an analyst can freely specify whether some positive or negative decision criterion should be enforced weakly or strongly. So far, decision criteria within analysis rules are strongly enforced, which considers each generated decision to be "definite". The distinction between weak and strong decision criteria is not supported.

In Section 7.2, we will extend the syntax and the semantics of positive and negative decision criteria within analysis rules for weak and strong enforcement.

6.3 Decision-Making Models

A decision-making model is a prescription how to compile a single global decision out of several "expert decisions". Applied to analysis rules, each expert's decision is modeled as a separate decision step, whereas the global decision is represented by the analysis rule as a whole. A typical decision-making model is "consensus voting", where the global decision is based on the consensus of all participating experts' decisions. An alternative decision-making model, which is also used by analysis rules introduced so far, requires the positive decision of a single expert to achieve a globally positive decision but requires the negative consensus of all experts' decisions to achieve a globally negative decision. The definite positive and the definite negative decision criteria of a decision step represent the expert's decisions. For the case that the expert's decision is unknown (i.e., the irresolute decision criterion is satisfied), the decision will be compiled out of other expert's decisions by evaluating other "more specific" decision steps. To specify how an unknown decision is compiled out of several expert decisions as flexible as possible, analysis rules should support the following sub-goals:

- **Freedom of Specifying Alternative Decision-Making Models.** This sub-goal requires that alternative decision-making models should be available and that an analyst may choose a decision-making model that is most appropriate to a particular decision task. So far, analysis rules only support a single decision making model, which is used as the default decision-making model.

- **Support for Rule-Covering Decision-Making Models.** This sub-goal requires that a decision-making model should be applicable to a decision task as a whole. Each "expert" compiles a locally unknown decision out of the decisions of its sub-experts using the globally available decision-making model. The default decision-making model used by analysis rules must be obeyed by all decision steps of the analysis rule.
- **Freedom of Redefining Decision Making for Individual Decision Steps.** This sub-goal requires that an analyst is free to specify a decision-making model for a particular decision step, which will overrule the rule-covering decision making model. When redefining decision-making for a decision step, the analyst should be allowed to choose either a globally available decision-making model or to prescribe decision making individually to the decision step. Redefining decision making for a single decision step is not supported by analysis rules yet.

When a decision-making model is evaluated for an analysis rule, three sets of objects will be generated: (1) objects that represent positively decided cases, (2) objects that represent negatively decided cases, and (3) objects that represent unresolved cases. To ensure that a decision-making model generates correct decisions, the following technical properties must be satisfied:

- **Completeness.** This property requires that the set of objects for which the decision-making model will be evaluated is identical with the union of the three sets of decided objects.
- **Uniqueness of Decision.** This property requires that no object for which a decision should be made belongs to more than one set of decided objects.

These two properties are satisfied by the default decision making model used by analysis rules so far, since:

1. The decisions of each decision step are *complete* and *unique* (see Section 6.2.1).
2. An object o belongs to the set of positive decisions, iff o satisfies the trigger action condition of at least one decision step that analyzed o.
3. An object o belongs to the set of irresolute decisions, iff o satisfies the detail analysis condition of at least one "leaf" decision step (i.e., a decision step that refers to a leaf cube) and o does not belong to the set of positive decisions.
4. An object o belongs to the set of negative decisions else.

In Section 7.3, we will extend the syntax and the semantics of analysis rules to support alternative decision-making models that may be applicable to all decision steps of an analysis rule as a whole or to particular decision steps individually.

6.4 Decision Parameters

A decision parameter is a variable whose value must be specified before a decision that refers to this decision parameter can be put into action. Applied to active data warehouses, decision parameters are the parameters of an analysis rule's action, e.g., the variable holding the new price of the decision to change an article's price. It should be possible to specify the value that will be bound to a decision parameter in a flexible way as follows (sub-goals):

- **Freedom of Specifying Static or Dynamic Values for Decision Parameters.** This sub-goal requires that an analyst is free to specify a constant value for a decision parameter or to specify separate analyzes to determine the value of a decision parameter. In the first case, the parameter value is *static* since it remains the same for each decided object (e.g., e.g., change price by +3%). In the latter case, the parameter value is *dynamic* since it may differ for each decided object (e.g., change the price by the same ratio as costs changed during the past two months).
- **Support for Rule-Covering and Cube-Specific Decision Parameters.** This sub-goal requires that an analyst should be free to specify parameter bindings that will be used uniformly for action execution of all positively decided level instances (i.e., *rule-covering*) and/or to specify parameter bindings for action execution of positively decided level instances that are described by the cells of some cube of the analysis graph (i.e., *cube-specific*). Rule-covering parameter bindings are considered as the *default* bindings that will be used unless cube-specific parameter bindings are available for action execution of some level instance.

Besides specifying concrete values for decision parameters in a flexible way, decision-making systems should also offer the possibility to restrict the range of possible values of decision parameters (sub-goal):

- **Freedom to Specify Parameter Constraints for Decision Parameters.** This sub-goal requires that an analyst is free to restrict the possible values that may be bound to decision parameters (e.g., prices may change from 0% to 7.5%). This sub-goal is especially important when parameter values are determined dynamically.

The sub-goals above must be satisfied to support flexible decision parameters. So far, analysis rules only allow decision parameters to be specified statically and unconditional.

6.5 Specialization of Decision Tasks

Specialization is one of the most important concepts in computer science. It has laid the foundations of conceptual design, of modern programming languages, and of database

systems, which require the distinguished treatment of similar objects to represent the real world more adequately than without specialization. Decision tasks represent analysis processes that support management decisions at particular timepoints. In the real world, such decision tasks are often split up (1) into "general guidelines", which are valid to analyze and make a decision for a large set of objects and (2) into "specific policies" for subsets of these objects by requiring more specific timepoints, more specific decision criteria, more specific decision-making models, or more specific decision parameters in decision making. The rationale of such an incremental refinement of decision tasks can be found in the real world, where business decision making is ruled by enterprise-wide policies, which must be obeyed but which may be refined for individual cases unless contradicting the enterprise-wide policies. Another example can be found in legislation of the European Union (EU), where EU-wide laws represent the basis of legislation of its member countries.

A system supporting the autonomous execution of decision tasks, as it is the case with analysis rules in active data warehouses, should satisfy the following sub-goal:

- **Support for Level Classes and Hierarchies of Level Classes.** This sub-goal requires that the objects for which a decision should be made (i.e., the level instances of the primary dimension level) are arranged in a class hierarchy. The classes of such a class hierarchy are called *level classes*, since the extension of a particular level class is the set of objects (of the primary dimension level) that roll up to the same level instance of a superordinate dimension level. The level classes of a class hierarchy are arranged in a subset/superset relationship as provided by the hierarchies of the corresponding dimension. The top level class of such a class hierarchy is called the level class "ALL", since it covers the complete set of level instances of the dimension level for which the class hierarchy exists. The data model supporting analysis rules so far does not offer the possibility to arrange the instances of a particular dimension level according to a class hierarchy. For each dimension level, there exists only the level class "ALL", for which analysis rules may be defined.

- **Support for Class-Specific Decision Tasks.** This sub-goal requires that a decision task should be defined for the instances of a particular level class. So far, analysis rules are specified for the level class "ALL" that exists for each dimension level.

- **Freedom of Specializing Class-Specific Decision Tasks.** This sub-goal requires that an analyst is free to specialize a class-specific decision task at a subclass. Since the data model supporting analysis rules so far does not offer the possibility to define a class hierarchy, analysis rules cannot be specified at lower level classes and thus cannot be specialized.

The decisions generated by analysis rule R', which is a specialization of analysis rule R, must satisfy the following globally observable technical property to ensure the correctness of generated decisions:

- **Decision Consistency.** This property requires that the decisions generated by R' (i.e., the set of positively decided cases, the set of negatively decided cases, and the set of unresolved cases) must be compliant with the decisions that have been generated after carrying out R alternatively. Decision consistency is ensured if the following rules are obeyed:

  1. Rules concerning the event:
     (a) The *rule of event specialization* says that the number of timepoints at which rule R' can be triggered may rise at analysis rule R'.
  2. Rules concerning the primary condition:
     (a) The *rule of selective decision making* says that the analysis scope of R' may be a subset of the analysis scope of R by strengthening the primary condition of R'.
  3. Rules concerning the decision criteria:
     (a) The *rule of forbidden contradiction of definite negative decisions* requires that R' cannot decide positively for object o if R generated a definite negative decision for o.
     (b) The *rule of forbidden contradiction of definite positive decisions* requires that R' cannot decide negatively for object o if R generated a definite positive decision for o.

     Specializing decision making for tentative decisions and for unresolved cases is less strict than for definite decisions:

     (a) Since a tentative decision represents default cases for decision making, a tentative decision of R may be overruled by a tentative decision or by a definite decision of R'.
     (b) An unresolved case represents a situation in which no definite and no tentative decision was made by R. Consequently, a specialized rule R' is free (a) to make no decision, (b) to make a tentative decision, or (c) to make a definite decision for the unresolved case.
  4. Rules concerning the parameter bindings:
     (a) The *rule of providing parameter ranges* requires that analysis rule R must define a parameter range for each parameter binding.
     (b) The *rule of incremental refinement of parameter ranges* says that R' is free to restrict the parameter ranges that have been defined at analysis rule R.

Specialization concerns analysis rules as a whole as well as the different parts of analysis rules (i.e., event, primary condition, analysis graph, decision steps, action parameters, and decision-making models). Two approaches are possible to specialize the different parts of analysis rules:

- **Black Box.** This approach considers analysis rule R as a "black box" by introducing new events, conditions, cubes, etc. at analysis rule R'. The rule parts of R' are defined from scratch without referring to the corresponding parts of R.
- **White Box.** This approach considers analysis rule R as a"white box" when introducing new events, conditions, cubes, etc. at analysis rule R'. The rule parts of R' are defined on the basis of corresponding parts of R.

Although motivated above, specialization of analysis rules will be not discussed further since an in-depth discussion of this topic will be far beyond the scope of this thesis.

6.6 Summary

Analysis rules introduced so far provide excellent means to automatize simple routine decision tasks but require some extensions to support modeling of more complex decision tasks. These extensions concern (i) events, (ii) decision criteria, (iii) decision-making models, (iv) decision parameters, and (v) specialization of analysis rules. Below (cf. Table 6.1), we give an overview of the sub-goals that we have identified for these extensions and outline what is supported by analysis rules so far.

Sub-Goal	Support by analysis rules
Freedom of specifying temporal and logical events.	temporal events
Freedom of specifying criteria for positive, negative, and irresolute decisions.	positive and irresolute
Freedom of specifying criteria for tentative and definite decisions.	definite
Freedom of specifying alternative decision-making models.	no support
Support for rule-covering decision-making models.	no support
Freedom of redefining decision making for individual decision steps.	no support
Freedom of specifying static or dynamic values for decision parameters.	static
Support for rule-covering and cube-specific decision parameters.	rule-covering
Support for level classes and class hierarchies.	one implicit level class for each dimension level, no class hierarchies
Support for class-specific decision tasks.	supported by the implicit level class
Freedom of specializing class-specific decision tasks.	no support

Table 6.1: Sub-goals of requirements

Chapter 7

Analysis Rules Extended

Contents

This chapter presents the extensions to analysis rules which we motivated in Chapter 6. These extensions are (1) introduction of logical events, (2) distinguished decision criteria, (3) alternative decision-making models, and (4) flexible decision parameters. Besides these extensions, we also present a procedural semantics for extended decision making. Note that, although in Chapter 6 we motivated the need to specialize analysis rules, specialization will be not discussed further.

7.1 Logical Events

While OLTP method events, relative temporal events, and calendar events offer a basic approach to specify the timepoints at which analysis rules should be fired, logical events represent an arbitrary complex state of the database, which is expressed by a condition. Event detection is as follows: When the database arrives at a state in which the condition holds, an instance of the logical event is detected and its occurrence signaled. Opposed to composite events, which are defined upon event operators (e.g., and, or, sequence, times, history, etc.) and whose instances are detected using some consumption mode [CKAK94] (i.e., general, recenct, chronicle, continuous, cumulative), logical events are specified declaratively and may therefore be represented by a query. Although in the recent years several event algebras have been proposed [BL95, CKAK94, GD94, GJS92, MPC96], we believe that the declarative approach is more suitable to be used in an active data warehouse for the following two reasons:

1. Logical events may be *represented* using a logic-based query language such as SQL [MS01]. Following this approach, the SELECT-clause constitutes the event attributes, the FROM-clause represents the base relations from which data is taken, and the WHERE-clause represents the condition that must be satisfied.

2. Logical events may be *implemented* using off-the-shelf database technologies such as SQL, materialized view mechanisms, and triggers without requiring special-purpose extensions that realize the various event operators and consumption modes.

The approach to specify logical events as sketched above was inspired by *situation diagrams* [LOS96], which is a high-level graphical language to specify complex situations in active object-oriented database systems. Although situation diagrams are not suited to specify logical events in an active data warehouse, we borrowed the idea representing logical events by SQL queries. Combined with recent advancements of materialized views in commercial database technologies, logical events can be quickly realized on top of a commercial relational data warehouse. We present this approach in more detail in Section 9.1, where we describe the implementation of the knowledge model of an active data warehouse. It is up to future work to propose an extension of the graphical event specification language *situation diagrams* for specifying logical events in an active data warehouse.

7.2 Decision Criteria – Modeling Local Decisions

In Chapter 5, we provided a basic approach to specify decision criteria that analyze a particular cube of a rule's analysis graph. The decisions that are generated when evaluating the decision criteria of a particular cube are referred to as *local decisions*. In Chapter 6, we

found that the proposed basic approach of specifying decision criteria (cf. Section 5.3.2) supports decision making very coarse-grained for the following two reasons:

1. Analysts can specify only *positive* and *irresolute* decision criteria. *Negative* decision criteria cannot be specified but are computed as the complement of the two aforementioned decision criteria instead.
2. Every decision criterion is considered to be *definite*. Analysts cannot distinguish between *tentative* decision criteria, which may be overridden in a later step, and *definite* decision criteria, which cannot be overridden.

In Sections 7.2.1 and 7.2.2, we will extend our basic model to support these two requirements as follows (see also Figure 7.1):

1. For each cube of the analysis graph, either (i) *positive/negative*, (ii) *positive/irresolute*, or (iii) *negative/irresolute* decision criteria must be specified. The specified positive and negative decision criteria are *definite*. Definitely decided cases are withdrawn from being analyzed by sub-cubes (cf. Figure 7.1). In Section 7.2.1 we consider only *definite* decision criteria, (i.e., $trgA^d$ and $dsmA^d$) irresolute decision criteria (i.e., the shaded box headed with *dtlA* depicted in Figure 7.1 without considering $trgA^t$ and $dsmA^t$).
2. For irresolute decided cases, default conditions may be specified which are represented by *tentative positive* decision criteria and by *tentative negative* decision criteria. In Figure 7.1, this is depicted by a shaded box. Since tentative decisions may be overruled by definite decisions of sub-cubes tentatively decided cases are also forwarded to be analyzed in more detail by sub-cubes. In Section 7.2.2 we introduce *tentative* decision criteria (i.e., the shaded box headed with *dtlA* depicted in Figure 7.1 by further distinguishing $trgA^t$ and $dsmA^t$).

Hence, the *local decisions* of some cube $r(R)$ are represented by a 5-tuple $\langle\, TrgA^d_{r(R)}, TrgA^t_{r(R)}, DtlA_{r(R)}, DsmA^d_{r(R)}, DsmA^t_{r(R)} \,\rangle$, where $TrgA^d_{r(R)}$ represents the *definite trigger action set* of $r(R)$, $TrgA^t_{r(R)}$ represents the *tentative trigger action set* of $r(R)$, $DtlA_{r(R)}$ represents the *detail analysis set* of $r(R)$, $DsmA^d_{r(R)}$ represents the *definite dismiss action set* of $r(R)$, and $DsmA^t_{r(R)}$ represents the *tentative dismiss action set* of $r(R)$.

7.2.1 Criteria for Positive, Negative, and Irresolute Decisions

In our *basic approach*, defining local decision making at a particular cube of the analysis graph, an analyst may specify conditions for two kinds of decision criteria: (1) The *positive decision criterion* represents the conditions that must be satisfied by an instance of the

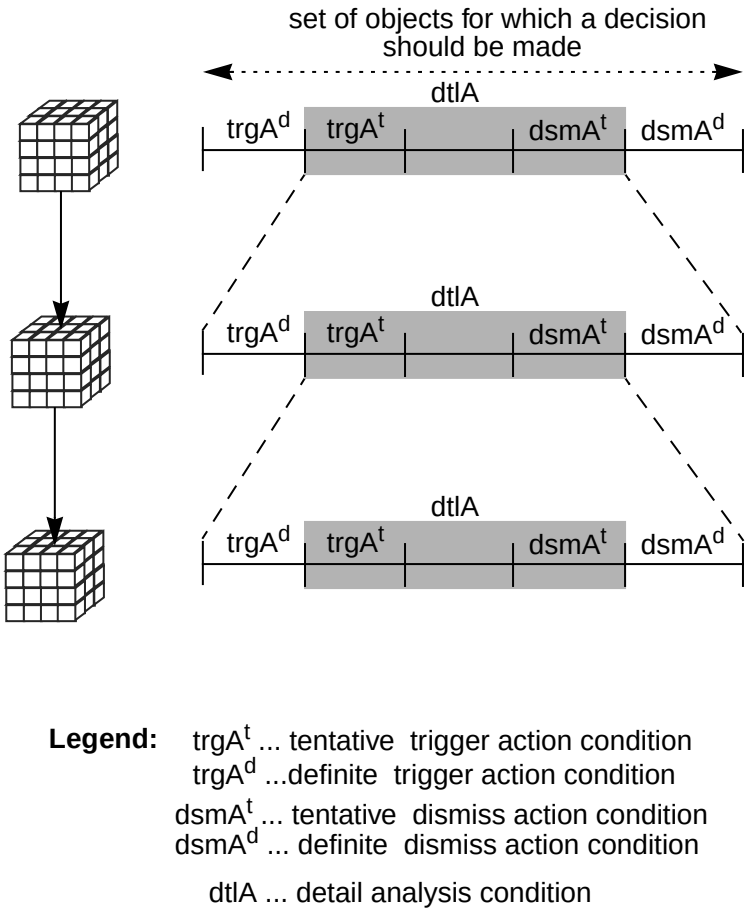

Figure 7.1: Top-down analysis using tentative and definite decision criteria

primary dimension level in order to execute the rule's action for that level instance in the OLTP system. We referred to this decision criterion as the *trigger action condition*. (2) The *irresolute decision criterion* represents the conditions that must be satisfied by an instance of the primary dimension level in order to continue analysis at a more detailed cube. We referred to this decision criterion as the *detail analysis condition*. Both kinds of decision criteria will be substituted with a default truth-value if the analyst did not specify the corresponding condition. In the case of the *trigger action condition*, this substitution value is FALSE, i.e., don't execute the rule's action in the OLTP system if not explicitly specified. In the case of the *detail analysis condition*, this substitution value is TRUE, i.e., continue analysis at a more detailed cube if not explicitly specified. If the two conditions overlap, *trigger action condition* is given precedence over *detail analysis condition*. Implicitly, the third kind of decision criterion – the *negative decision criterion* – is derived as the complement of the two aforementioned decision criteria. It identifies those cases for which action execution (and further analyzes) is *dismissed*.

In the *extended approach*, the negative decision criterion of a decision step may be specified explicitly in the same way as the positive and the irresolute decision criteria may be specified. We will refer to this decision criterion in the following as *dismiss action condition*. To specify a decision step syntactically correctly (cf. Figure 7.4), an analyst must provide one out of three possible combinations of specifying decision criteria:

1. Specify the positive and the negative decision criteria.
2. Specify the positive and the irresolute decision criteria.
3. Specify the negative and the irresolute decision criteria.

Semantically, the proposed syntax supports modeling of six alternative cases to generate local decisions and "complement decisions" (cf. Figure 7.2):

Case 1: The analyst specifies the positive (i.e., *trgA*) and the negative (i.e., *dsmA*) decision criteria. If some level instance l of the primary dimension level does not satisfy $trgA \vee dsmA$, then l will be analyzed in further detail, i.e., the "complement decision" is *dtlA*.

Case 2: The analyst specifies the positive and the irresolute (i.e., *dtlA*) decision criteria. If some level instance l of the primary dimension level does not satisfy $trgA \vee dtlA$, then the "complement decision" is not to execute the rule's action for l (i.e., *dsmA*).

Case 3: The analyst specifies the irresolute and the negative decision criteria. If some level instance l of the primary dimension level does not satisfy $dtlA \vee dsmA$, then the "complement decision" is to execute the rule's action for l (i.e., *trgA*).

Case 4: The analyst only specifies the positive decision criterion. There are two alternative complement decisions (denoted as "alternative complement 1" for *dtlA* and "alternative complement 2" for *dsmA* in Figure 7.2) if some level instance l of the primary dimension level does not satisfy the positive decision criterion. In such a situation, the analyst must decide whether l should be decided negatively or irresolute. Hence, the default condition FALSE must be specified for the unwanted alternative complement[1] (e.g., if *dtlA* should be the complement decision for l, then FALSE is specified for *dsmA*).

Case 5: The analyst specifies only the irresolute decision criterion. There are two alternative complement decisions (i.e., *trgA* and *dsmA*) that will be resolved along the lines of case 4.

Case 6: The analyst specifies only the negative decision criterion. There are again two alternative complement decisions (i.e., *trgA* and *dtlA*) that will be resolved along the lines of case 4.

Formally, to ensure the correctness of decision making of a decision step which was defined for a particular cube $r(R)$, we first complete the three decision criteria in a preparatory step with complement decisions. Note that the positive and negative decision criteria are *definite decision criteria*. Hence, $trgA^{d+}_{r(R)}$ represents the *completed definite trigger action condition* of cube $r(R)$, and $dsmA^{d+}_{r(R)}$ represents the *completed definite dismiss action condition* of cube $r(R)$. The irresolute decision criterion $dtlA^{+}_{r(R)}$ (i.e., the *completed detail analysis condition* of cube $r(R)$) will be used in a later step to determine the level instances for which tentative decision criteria will be evaluated (see Section 7.2.2).

Completion of local decisions is as follows:

1. The completed trigger action condition of $r(R)$ (i.e., $trgA^{+d}_{r(R)}$, cf. Equation 7.1) holds for some level instance l if (i) l satisfies the condition specified by the analyst or if (ii) no *definite trigger action condition* was specified and l does not satisfy either of the two remaining decision criteria (case 3 in Figure 7.2).

$$trgA^{d+}_{r(R)}(l) = \begin{cases} trgA^{d}_{r(R)}(l) & : \text{ if } trgA^{d}_{r(R)} \text{ is defined for } r(R) \\ \neg(dtlA_{r(R)}(l) \wedge dsmA^{d}_{r(R)}(l)) & : \text{ complement decision case 3} \end{cases} \quad (7.1)$$

2. The completed dismiss action condition of $r(R)$ (i.e., $dsmA^{d+}_{r(R)}$, cf. Equation 7.2) holds for some level instance l if (i) l satisfies the condition specified by the analyst

[1] To ensure semantic correctness, we forbid specifying the value "TRUE" for complement decision criteria in our syntax.

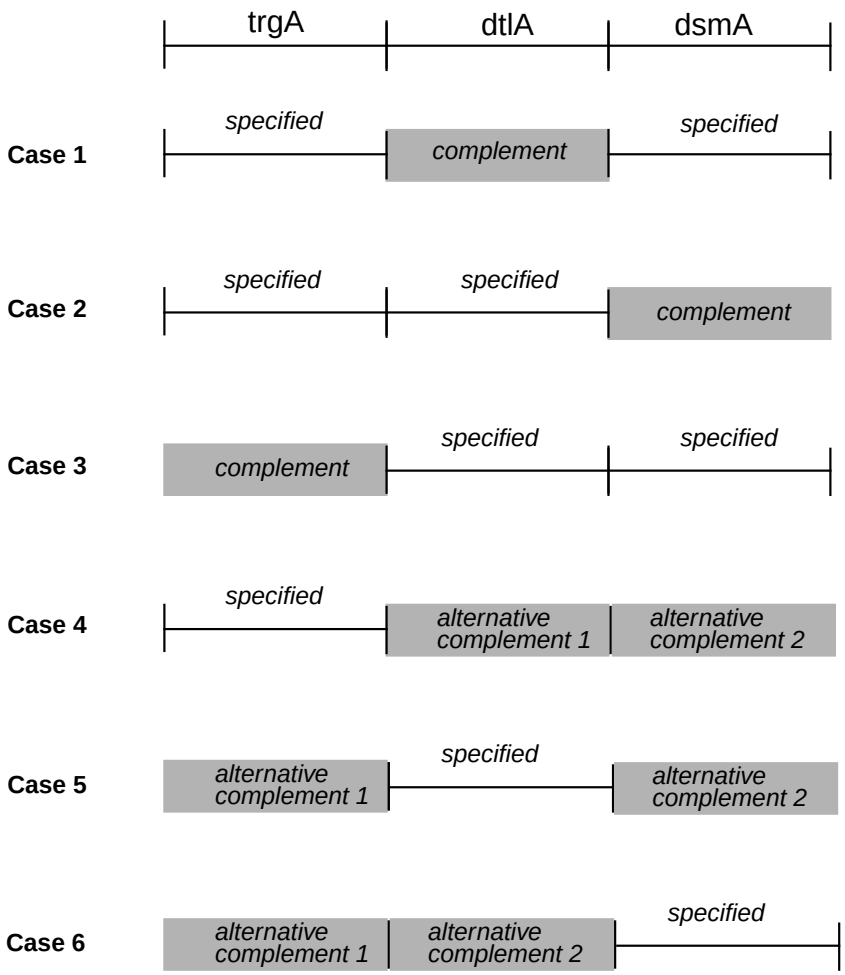

Figure 7.2: Local decision making

or if (ii) no *definite dismiss action condition* was specified and l does not satisfy either of the two remaining decision criteria (case 2 in Figure 7.2).

$$dsmA^{d+}_{r(R)}(l) = \begin{cases} dsmA^{d}_{r(R)}(l) & : \text{if } dsmA^{d}_{r(R)} \text{ is defined for } r(R) \\ \neg(dtlA_{r(R)}(l) \wedge trgA^{d}_{r(R)}(l)) & : \text{complement decision case 2} \end{cases} \tag{7.2}$$

3. The completed detail analysis condition of $r(R)$ (i.e., $dtlA^{+}_{r(R)}$, cf. Equation 7.3) holds for some level instance l if (i) l satisfies the condition specified by the analyst or if (ii) no *dismiss action condition* was specified and l does not satisfy either of the two remaining decision criteria (case 1 in Figure 7.2).

$$dtlA^{+}_{r(R)}(l) = \begin{cases} dtlA_{r(R)}(l) & : \text{if } dtlA_{r(R)} \text{ is defined for } r(R) \\ \neg(trgA^{d}_{r(R)}(l) \wedge dsmA^{d}_{r(R)}(l)) & : \text{complement decision case 1} \end{cases} \tag{7.3}$$

Informally, local decision making at a particular cube $r(R)$ using definite decision criteria as introduced above is carried out as follows:

1. The *analysis scope* of cube $r(R)$ is the set of objects in the primary dimension, for which the predecessor(s) of $r(R)$ decided to carry out further analyzes. If $r(R)$ is the "root cube", the analysis scope of $r(R)$ is identical with the analysis scope of the analysis rule.

2. The *definite detail analysis set* (i.e., $DtlA^{d}_{r(R)}$) of cube $r(R)$ is the set of objects in the analysis scope of $r(R)$ (i) satisfying the *definite detail analysis condition* (but not satisfying the *definite trigger action condition* or *definite dismiss action condition*) or (ii) satisfying the *definite trigger action condition* and the *definite dismiss action condition*. The latter case represents a conflict between the decision to execute the rule's action and the decision not to execute the rule's action. Since in such a situation both conditions are satisfied, a definite decision as to whether to execute the rule's action as to whether to dismiss action execution cannot be determined. Hence, such a conflict is resolved by carrying out further analyzes instead. Note that the *definite detail analysis set* is used later in this section for determining tentative decisions. Hence, the *final detail analysis set* of cube $r(R)$ (i.e., the set of level instances in the analysis scope of $r(R)$ for which no definite or tentative decision was made) may be only a subset of the *definite detail analysis set* of $r(R)$.

3. The *definite trigger action set* (i.e., $TrgA^{d}_{r(R)}$) of cube $r(R)$ is the set of objects in the analysis scope of $r(R)$ (that is not in the *definite detail analysis set* of $r(R)$) satisfying the *definite trigger action condition*.

4. The *definite dismiss action set* (i.e., $DsmA^{d}_{r(R)}$) of cube $r(R)$ is the set of objects in the analysis scope of $r(R)$ (that is not in the *definite detail analysis set* of $r(R)$) satisfying the *dismiss action condition.*

Note that the semantics of decision making at cube $r(R)$ in our basic approach is a special case of the semantics proposed by extended approach: In the basic approach, the decision to dismiss level instance l from action execution (denoted by $DsmA_{r(R)}(l)$) is computed as the complement of (i) the decision to trigger the rule's action for l (denoted by $TrgA_{r(R)}(l)$) and (ii) the decision to carry out more detailed analyzes for l (denoted by $DtlA_{r(R)}(l)$) (cf. Equation 5.5 in Section 5.3.2). Conflicts due to overlapping conditions between the positive and the irresolute decision criterion (only these two decision criteria could be specified) were resolved *in favor* of the decision to trigger the rule's action for l (cf. Equation 5.4 in Section 5.3.2). This is also depicted in Figure 7.3 (b). Since we allow for specifying positive, negative, and irresolute decision criteria in the extended approach, there are more alternative cases for specifying decision making at some cube $r(R)$ (cf. Figure 7.2) than in the basic approach and hence in the extended approach further conflicts between overlapping specified decision criteria (i.e., positive/negative, positive/irresolute, and irresolute/negative) may occur.

In the following, we will explain how conflicts due to overlapping conditions are resolved and how the local decisions are generated for cube $r(R)$:

1. The conflict resolution of overlapping conditions $trgA^{d+}_{r(R)}(l)$ and $dsmA^{d+}_{r(R)}(l)$ is to carry out further analyzes (i.e., $DtlA^{d}_{r(R)}(l)$, see Figure 7.3 (a)). The rationale for this conflict resolution is that such a situation cannot be resolved in favor of $trgA^{d+}_{r(R)}(l)$ nor in favor of $dsmA^{d+}_{r(R)}(l)$ since both conditions are given the same priority. Hence, $DtlA^{d}_{r(R)}(l)$ (cf. Equation 7.4 case 1) evaluates to TRUE if $trgA^{d+}_{r(R)}(l)$ and $dsmA^{d+}_{r(R)}(l)$ are both satisfied for the same level instance l. Further, $TrgA^{d}_{r(R)}(l)$ (i.e., the definite decision to trigger the rule's action for l, cf. Equation 7.5) and $DsmA^{d}_{r(R)}(l)$ (i.e., the definite decision to dismiss action execution for l, cf. Equation 7.6) are defined upon the complement of $DtlA^{d}_{r(R)}(l)$.

$$DtlA^{d}_{r(R)}(l) = \begin{cases} trgA^{d+}_{r(R)}(l) \wedge dsmA^{d+}_{r(R)}(l) & : \text{ case 1 - overlap} \\ dtlA^{+}_{r(R)}(l) \wedge \neg(trgA^{d+}_{r(R)}(l) \vee dsmA^{d+}_{r(R)}(l)) & : \text{ case 2 - else} \end{cases} \tag{7.4}$$

2. The conflict resolution of overlapping conditions $trgA^{d+}_{r(R)}(l)$ and $dtlA^{d+}_{r(R)}(l)$ is to execute the rule's action for l (i.e., $TrgA^{d}_{r(R)}(l)$, see Figure 7.3 (b)), which represents the same semantics as defined for the basic approach. Hence, $DtlA^{d}_{r(R)}(l)$ (cf. Equation 7.4 case 2) evaluates to FALSE if $dtlA^{d+}_{r(R)}(l)$ is satisfied and if $trgA^{d+}_{r(R)}(l)$ is satisfied since $DtlA^{d}_{r(R)}(l)$ is defined upon the complement of $trgA^{d+}_{r(R)}(l)$.

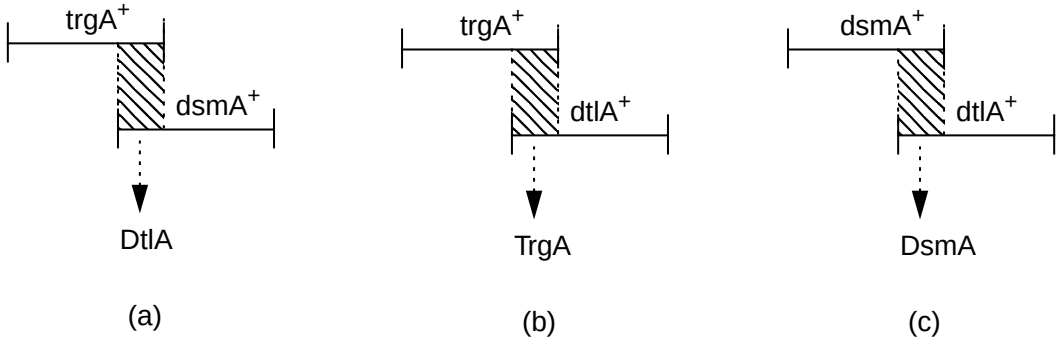

Figure 7.3: Resolving Conflicts between Decision Criteria

$$TrgA^{d}_{r(R)}(l) = trgA^{d+}_{r(R)}(l) \wedge \neg DtlA^{d}_{r(R)}(l) \tag{7.5}$$

3. The conflict resolution of overlapping conditions $dsmA^{d+}_{r(R)}(l)$ and $dtlA^{d+}_{r(R)}(l)$ is to dismiss l from action execution (i.e., $DsmA^{d}_{r(R)}(l)$, see Figure 7.3 (c)). The rationale for this conflict resolution is that whenever a level instance is decided negatively, no further analyzes need to be carried out. Hence, $DtlA^{d}_{r(R)}(l)$ (cf. Equation 7.4 case 2) evaluates to FALSE if $dtlA^{d+}_{r(R)}(l)$ is satisfied and if $dsmA^{d+}_{r(R)}(l)$ is satisfied since $DtlA^{d}_{r(R)}(l)$ is defined upon the complement of $dsmA^{d+}_{r(R)}$.

$$DsmA^{d}_{r(R)}(l) = dsmA^{d+}_{r(R)}(l) \wedge \neg DtlA_{r(R)}(l) \tag{7.6}$$

7.2.2 Tentative and Definite Decisions

So far, we only considered decision making at some cube $r(R)$ using decision criteria that support *definite* decisions. Once an object is decided definitely, it is withdrawn from further analyzes. Hence, a later – possibly contradictory – decision cannot invalidate the original decision. Further, definite decisions do not support modeling of default situations that represent pre-decisions for a great number of objects but still offer the possibility to be overruled later. In this section we introduce *tentative* decisions, which differ from definite decisions in that these decisions may be overruled by later definite decisions. An object that satisfies a tentative decision criterion will be analyzed further unless a definite decision is made for this object. Since irresolute decisions are never *definite*, we introduce the distinction between tentative and definite decisions only for positive and negative decisions. Hence, the following priorities among decisions exist:

1. Definite decisions overrule tentative and irresolute decisions.

2. Tentative decisions overrule irresolute decisions.

Figure 7.1 illustrates the top-down approach in decision making using tentative and definite decision criteria. Syntactically, tentative and definite decisions may be specified for the *trigger action condition* and the *dismiss action condition* in the definition of a decision step using clauses Tentative-IF and Definite-IF (see syntax in Figure 7.4).

In a preparatory step, missing tentative decision criteria are completed and conflicts due to overlapping specified tentative decision criteria are resolved as follows:

1. A missing tentative decision criterion is complemented using the default value FALSE.
2. If the tentative positive and the tentative negative decision criteria are specified, both are "reduced" by any "overlapping" part.

The so-completed decision criteria are denoted as $trgA^{t+}_{r(R)}$ and $dsmA^{t+}_{r(R)}$, respectively.

$$trgA^{t+}_{r(R)}(l) = \begin{cases} trgA^{t}_{r(R)}(l) & : \text{if } trgA^{t}_{r(R)} \text{ is defined but } dsmA^{t}_{r(R)} \text{ is not defined for } r(R) \\ trgA^{t}_{r(R)}(l) \wedge \neg dsmA^{t}_{r(R)}(l) & : \text{if } trgA^{t}_{r(R)} \text{ and } dsmA^{t}_{r(R)} \text{ are defined for } r(R) \\ \mathsf{FALSE} & : \text{if } trgA^{t}_{r(R)} \text{ is not defined for } r(R) \end{cases} \tag{7.7}$$

$$dsmA^{t+}_{r(R)}(l) = \begin{cases} dsmA^{t}_{r(R)}(l) & : \text{if } dsmA^{t}_{r(R)} \text{ is defined but } trgA^{t}_{r(R)} \text{ is not defined for } r(R) \\ dsmA^{t}_{r(R)}(l) \wedge \neg trgA^{t}_{r(R)}(l) & : \text{if } dsmA^{t}_{r(R)} \text{ and } trgA^{t}_{r(R)} \text{ are defined for } r(R) \\ \mathsf{FALSE} & : \text{if } dsmA^{t}_{r(R)} \text{ is not defined for } r(R) \end{cases} \tag{7.8}$$

Informally, decision making using tentative and definite decision criteria at a particular cube $r(R)$ is carried out as follows:

1. Definite decisions are generated using the approach described in the previous section. Note that the *trigger action set* of cube $r(R)$ described in the previous section is now referred to as the *definite trigger action set* of $r(R)$ (denoted as $TrgA^{d}_{r(R)}$); the *dismiss action set* of cube $r(R)$ described in the previous section is now referred to as the *definite dismiss action set* of $r(R)$ (denoted as $DsmA^{d}_{r(R)}$). Further note that $DtlA^{d}_{r(R)}$ represents the *definite detail analysis set* of $r(R)$. Tentative decision criteria will be evaluated for this set of objects.

Syntax

```
DecisionStepDef        ::=  ANALYZE cubeName
                              ["[" DecisionVarDef { "," DecisionVarDef } "]" ]
                              (TriggerActionCondDef DismissActionCondDef
                              | TriggerActionCondDef DetailAnalysisCondDef
                              | DetailAnalysisCondDef DismissActionCondDef).
TriggerActionCondDef   ::=  TRIGGER ACTION
                              (DefiniteCondDef [TentativeCondDef]).
DetailAnalysisCondDef  ::=  DETAIL ANALYSIS IF ConditionDef.
DismissActionCondDef   ::=  DISMISS ACTION
                              (DefiniteCondDef [TentativeCondDef]).
TentativeCondDef       ::=  Tentative-IF ConditionDef.
DefiniteCondDef        ::=  Definite-IF ConditionDef.
```

Figure 7.4: Extended syntax to specify decision steps

2. A level instance l of the analysis scope of $r(R)$ is in the *tentative trigger action set* of $r(R)$ (denoted as $TrgA^t_{r(R)}$) if (i) l is in the *definite detail analysis set* of $r(R)$ (i.e., no local definite decision was generated for l), (ii) l satisfies the *tentative trigger action condition*, (iii) l does not satisfy the *tentative dismiss action condition* of $r(R)$.

3. A level instance l of the analysis scope of $r(R)$ is in the *tentative dismiss action set* of $r(R)$ (denoted as $DsmA^t_{r(R)}$) if (i) l is in the *definite detail analysis set* of $r(R)$ (i.e., no local definite decision was generated for l), (ii) l satisfies the *tentative dismiss action condition*, (iii) l does not satisfy the *tentative trigger action condition* of $r(R)$.

4. The *final detail analysis set* of $r(R)$ (denoted as $DtlA_{r(R)}$) is the set of objects for which no definite or tentative decision could be generated. A level instance l is in $DtlA_{r(R)}$ if (i) l satisfies the *detail analysis condition* of $r(R)$ and no other definite/tentative decision criterion holds for l or if (ii) l is in the *definite detail analysis set* of $r(R)$ and l satisfies both the *tentative trigger action condition* and the *tentative dismiss action condition* of $r(R)$.

Conflicts due to overlapping conditions are resolved in the same way as described in the previous section, i.e., overlapping conditions $trgA^t_{r(R)}$ and $dsmA^t_{r(R)}$ are resolved in favor of $DtlA_{r(R)}$, overlapping conditions $trgA^t_{r(R)}$ and $dtlA_{r(R)}$ are resolved in favor of $TrgA^t_{r(R)}$, and overlapping conditions $dsmA^t_{r(R)}$ and $dtlA_{r(R)}$ are resolved in favor of $DsmA^t_{r(R)}$. Since

tentative decisions are overruled by definite decisions, Equations 7.10 and 7.11 are defined upon $DtlA^{d}_{r(R)}(l)$, which identifies those cases for which no definite decision was made.

$$DtlA_{r(R)}(l) = DtlA^{d}_{r(R)}(l) \wedge \neg(trgA^{t+}_{r(R)}(l) \vee dsmA^{t+}_{r(R)}(l)) \tag{7.9}$$

$$TrgA^{t}_{r(R)}(l) = DtlA^{d}_{r(R)}(l) \wedge trgA^{t+}_{r(R)}(l) \tag{7.10}$$

$$DsmA^{t}_{r(R)}(l) = DtlA^{d}_{r(R)}(l) \wedge dsmA^{t+}_{r(R)}(l) \tag{7.11}$$

As identified in Section 6.2, decision making at a particular cube $r(R)$ should be *complete* (i.e., the disjunction of a decision step's decision criteria must be true for all objects of the analysis scope of $r(R)$) and *unique* (i.e., a decision step should generate a single decision for a level instance of the analysis scope of $r(R)$). Decision making using the extended approach satisfies these two properties as follows:

1. Decision making is *complete* since for each level instance l of the analysis scope of $r(R)$ definite and tentative decision criteria are evaluated. If l does not satisfy either of these decision criteria, it is considered to be analyzed further. Syntactically, analysts may specify only two out of three possible decision criteria. Hence, the third decision criterion is computed as the complement of what has been specified.

2. Decision making is *unique* since conflicts between positive and negative decision criteria are resolved in favor of the irresolute decision. Further, definite decisions have a higher priority than tentative or irresolute decisions.

7.3 Decision-Making Models – Modeling Global Decisions

While *local decisions* are generated by evaluating the decision criteria of a particular cube, the *global decisions* of an analysis rule are generated by combining the local decisions taken from the cubes of the analysis graph. In the following we give two recursive definitions of the term *global decision*, which are also illustrated in Figure 7.5.

Definition 1. The *global decisions of some cube* $r(R)$ (denoted as $globalDec_{r(R)}$) are represented by a 5-tuple $\langle\ TrgA^{gd}_{r(R)}, TrgA^{gt}_{r(R)}, DtlA^{g}_{r(R)}, DsmA^{gd}_{r(R)}, DsmA^{gt}_{r(R)}\ \rangle$, where $TrgA^{gd}_{r(R)}$ represents the *global definite trigger action set*, $TrgA^{gt}_{r(R)}$ represents the *global tentative trigger action set*, $DtlA^{g}_{r(R)}$ represents the *global detail analysis set*, $DsmA^{gd}_{r(R)}$

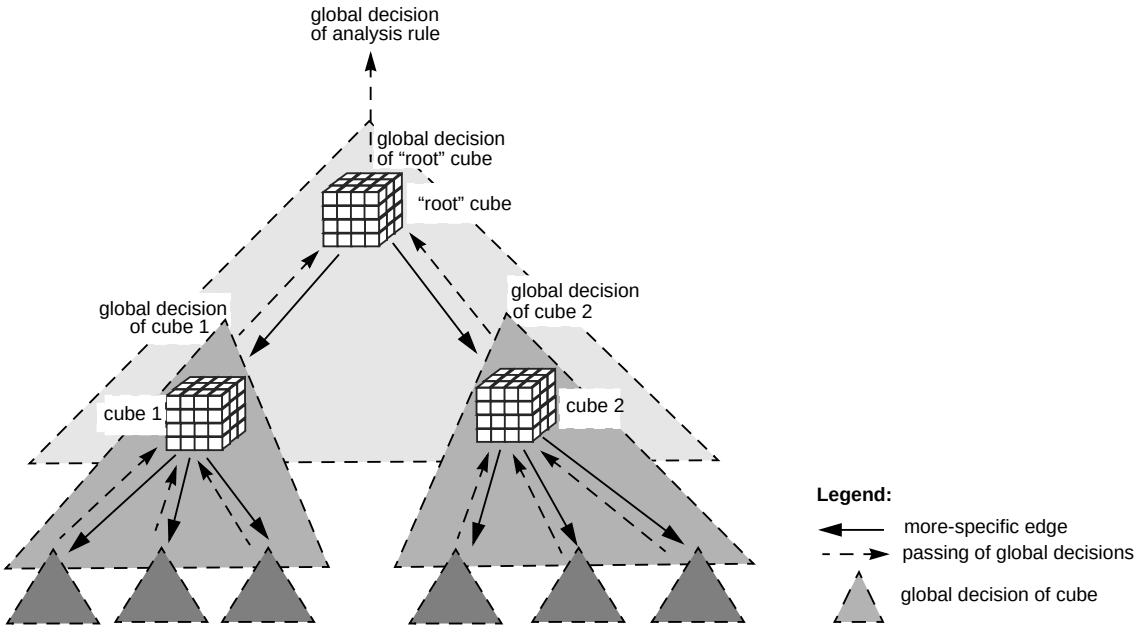

Figure 7.5: Principal idea of global decision making

represents the *global definite dismiss action set*, and $DsmA^{gt}_{r(R)}$ represents the *global tentative dismiss action set* of $r(R)$. This 5-tuple is derived from the global decisions of sub-cubes of $r(R)$ and from the local decisions of $r(R)$ (denoted as $localDec_{r(R)}$), hence[2]

$$globalDec_{r(R)} = \begin{cases} (\biguplus_{d \in desc(r(R))} globalDec_d) \oplus localDec_{r(R)} & : \text{ if } r(R) \text{ is an "inner" cube} \\ localDec_{r(R)} & : \text{ if } r(R) \text{ is a "leaf" cube} \end{cases} \quad (7.12)$$

For the case that $r(R)$ is an "inner" cube, the global decisions of sub-cubes of $r(R)$ need to be integrated, i.e., $\biguplus_{d \in desc(r(R))} globalDesc_{r(R)}$. The resulting 5-tuple $\langle\, TrgA^{id}_{r(R)}, TrgA^{it}_{r(R)}, DtlA^{i}_{r(R)}, DsmA^{id}_{r(R)}, DsmA^{it}_{r(R)} \,\rangle$ represents the *integrated* global decisions of the sub-cubes of $r(R)$. For each cube, the semantics of operator $\uplus$, the *decision-making model* of $r(R)$, may be different. Section 7.3.2 introduces introduces an approach to specify the semantics of $\uplus$. In a next step, the integrated global decisions are *merged* with the local decisions of $r(R)$ using operator $\oplus$. The semantics of this operator is defined in Section 7.3.1. For the case that $r(R)$ is a "leaf" cube, the global decision of $r(R)$ is a copy of the local decisions of $r(R)$.

Since the global decision of an analysis rule is derived from the global decision of the "root" cube, all decision criteria have already been evaluated at this stage. Hence, any remaining

[2]Predicate *desc(r(R))* represents the direct descendants (i.e., sub-cubes) of cube $r(R)$.

tentative decision will be turned into a definite decision, i.e., the two sets $TrgA^g_{AR}$ and $DsmA^g_{AR}$ represent definite decisions. A level instance l of the rule's analysis scope is in the *global trigger action set* of AR if the l is in the *global definite trigger action set* of the "root" cube or if l is in the *global tentative trigger action set* of the "root" cube (cf. Equation 7.13). The rule's negative decision for l is derived accordingly from the global decision of the "root" cube (cf. Equation 7.15). Further manual analyzes must be carried out for l if the global decision of the "root" cube is detail analysis for l (cf. Equation 7.14).

Definition 2. The *global decision of an analysis rule* (denoted as $globalDec_{AR}$) is a 3-tuple $\langle\ TrgA^g_{AR}, DtlA^g_{AR}, DsmA^g_{AR}\ \rangle$, where $TrgA^g_{AR}$ represents the *global trigger action set*, $DtlA^g_{AR}$ represents the *global detail analysis set*, and $DsmA^g_{AR}$ represents the *global dismiss action set* of the analysis rule. This 3-tuple is derived from the global decision of the "root" cube of the analysis rule.

$$TrgA^g_{AR}(l) = TrgA^{gd}_{root}(l) \vee TrgA^{gt}_{root}(l) \tag{7.13}$$

$$DtlA^g_{AR}(l) = DtlA^g_{root}(l) \tag{7.14}$$

$$DsmA^g_{AR}(l) = DsmA^{gd}_{root}(l) \vee DsmA^{gt}_{root}(l) \tag{7.15}$$

In the remainder of this section, we define the semantics of operator $\oplus$ and we propose several semantics of operator $\uplus$. We will further extend our syntax to specify decision-making models and we will present an approach to evaluate decision-making models procedurally.

7.3.1 Semantics of Operator $\oplus$

Decision making at sub-cubes of cube $r(R)$ is carried out for those level instances of the analysis scope of $r(R)$ for which $r(R)$ could not generate a definite decision (see Figure 7.1). Hence, the analysis scope of a sub-cube of $r(R)$ is the union of irresolute and tentative decided cases of $r(R)$. The following principles need to be considered when merging the local decisions of $r(R)$ (represented by the 5-tuple $\langle\ TrgA^d_{r(R)}, TrgA^t_{r(R)}, DtlA_{r(R)}, DsmA^d_{r(R)}, DsmA^t_{r(R)}\ \rangle$) with the integrated global decisions of sub-cubes of $r(R)$ (represented by the 5-tuple $\langle\ TrgA^{id}_{r(R)}, TrgA^{it}_{r(R)}, DtlA^i_{r(R)}, DsmA^{id}_{r(R)}, DsmA^{it}_{r(R)}\ \rangle$):

1. Definite decisions overrule tentative and irresolute decisions.

2. Tentative decisions overrule irresolute decisions.

3. A "more specific" tentative decision overrules a "more general" tentative decision.

Note that the integrated local decisions of $r(R)$ represent the level instances in the definite detail analysis set of $r(R)$, which is denoted as $DtlA^{d}_{r(R)}$ in Section 7.2. This set consists of (i) the local irresolute decisions of $r(R)$ (i.e., $DtlA_{r(R)}$) and (ii) the local tentative decisions of $r(R)$ (i.e., $TrgA^{t}_{r(R)}$ and $DsmA^{t}_{r(R)}$). Further note that integrating the global decisions of subcubes will be treated later in Section 7.3.2. Integrated decisions satisfy *completeness* and *uniqueness* as we will explain in Section 7.3.2.

For a level instance l of the analysis scope of $r(R)$, the following global decisions are made:

$$TrgA^{gd}_{r(R)}(l) = TrgA^{d}_{r(R)}(l) \vee TrgA^{id}_{r(R)}(l) \tag{7.16}$$

$$DsmA^{gd}_{r(R)}(l) = DsmA^{d}_{r(R)}(l) \vee DsmA^{id}_{r(R)}(l) \tag{7.17}$$

$$TrgA^{gt}_{r(R)}(l) = (TrgA^{t}_{r(R)}(l) \wedge DtlA^{i}_{r(R)}(l)) \vee TrgA^{it}_{r(R)}(l) \tag{7.18}$$

$$DsmA^{gt}_{r(R)}(l) = (DsmA^{t}_{r(R)}(l) \wedge DtlA^{i}_{r(R)}(l)) \vee DsmA^{it}_{r(R)}(l) \tag{7.19}$$

$$DtlA^{g}_{r(R)}(l) = DtlA^{d}_{r(R)}(l) \wedge \neg(TrgA^{gd}_{r(R)}(l) \vee DsmA^{gd}_{r(R)}(l) \vee TrgA^{gt}_{r(R)}(l) \vee DsmA^{gt}_{r(R)}(l)) \tag{7.20}$$

Figure 7.6 illustrates merging of global and local decisions by considering the three decisions that constitute the set of level instances, for which sub-cubes of $r(R)$ generated global decisions (i.e., $DtlA^{d}_{r(R)} = \mathrm{TrgA}^{t}_{r(R)} \cup \mathrm{DtlA}_{r(R)} \cup \mathrm{DsmA}^{t}_{r(R)}$). As depicted graphically, merged decisions satisfy *uniqueness* (i.e., the merged decision sets are non-overlapping) since integrated decisions of sub-cubes satisfy *uniqueness* (shown later) and since the rules to derive merged decisions ensure *uniqueness*. Further, since integrated decisions of sub-cubes satisfy completeness (shown later), merged decisions also satisfy *completeness* (i.e., decisions are generated for *all* level instances to be analyzed).

Equations 7.16, 7.17, 7.18, 7.19, and 7.20 define the semantics of operator $\oplus$ declaratively as follows:

Global Definite Decisions. A global definite decision is generated for level instance l of the analysis scope of $r(R)$ if (i) a local definite decision was generated for l or (ii) if a global integrated definite decision was generated for l. In the first case, l is in the *definite trigger action set* of $r(R)$ (i.e., $TrgA^{d}_{r(R)}(l)$ evaluates to TRUE in Equation 7.16) or in the *definite*

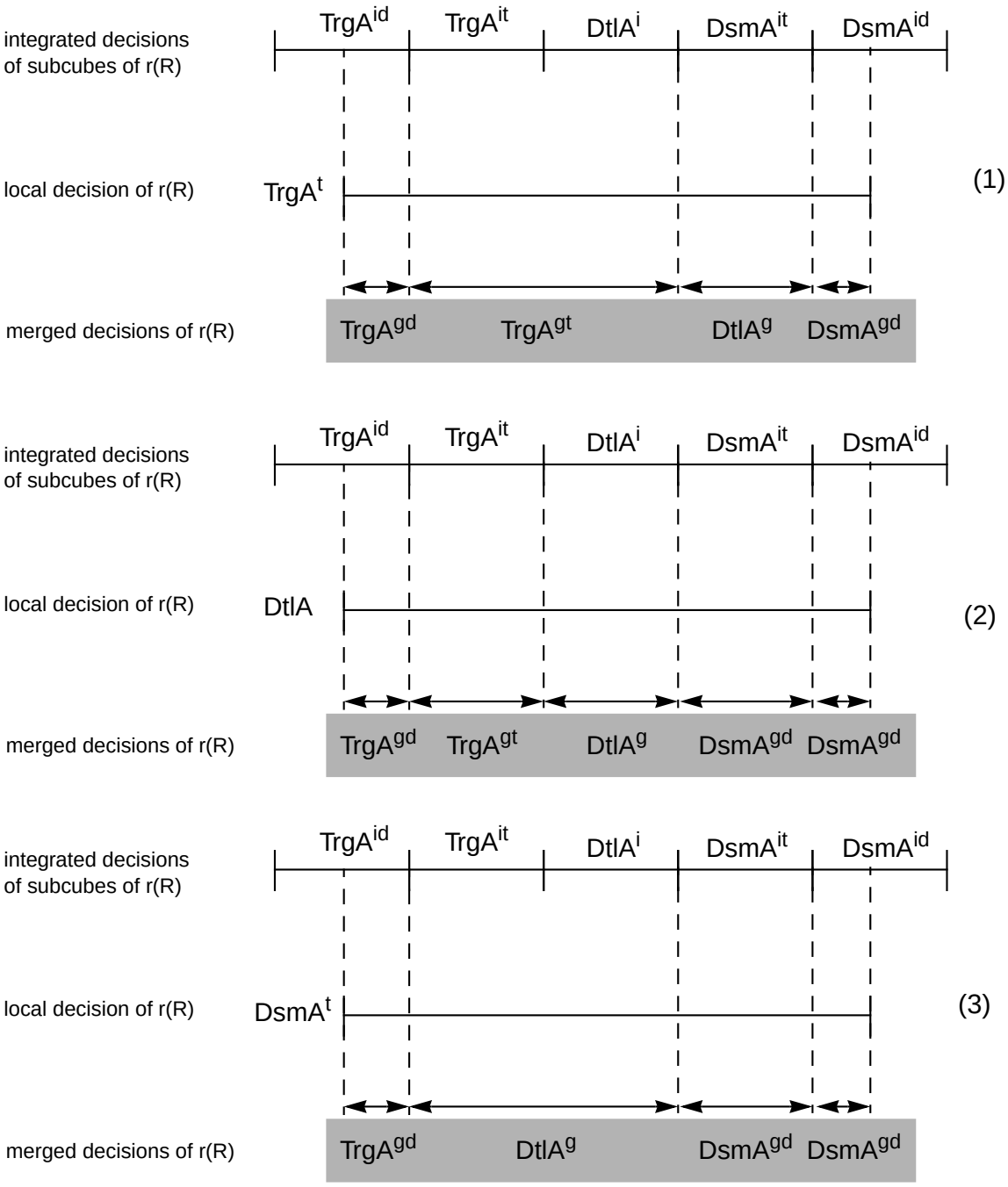

Figure 7.6: Merging global and local decisions

dismiss action set of $r(R)$ (i.e., $DsmA^{d}_{r(R)}(l)$ evaluates to TRUE in Equation 7.17). In the latter case, l is in the *definite trigger action set* of sub-cubes of $r(R)$ after the decision-making model of $r(R)$ was evaluated (i.e., $TrgA^{id}_{r(R)}(l)$ evaluates to TRUE in Equation 7.16) or l is in the *definite dismiss action set* of sub-cubes of $r(R)$ after the decision-making model of $r(R)$ was evaluated (i.e., $DsmA^{id}_{r(R)}(l)$ evaluates to TRUE in Equation 7.17). Note that although a local tentative decision may exist for l, a "more specific" definite decision will overrule the local decision.

Global Tentative Decisions. A global tentative decision is generated for level instance l of the analysis scope of $r(R)$ if (i) a local tentative decision was generated for l or (ii) if a global integrated tentative decision was generated for l. If both (i) and (ii) are valid for l, the global integrated tentative decision is given precedence over the local tentative decision since (ii) is "more specific" than (i). A positive global tentative decision (i.e., $TrgA^{gt}_{r(R)}$) is generated for l if l is in the *tentative trigger action set* of $r(R)$ and in the *detail analysis set* of sub-cubes of $r(R)$ after the decision-making model of $r(R)$ was evaluated (i.e., $TrgA^{t}_{r(R)}(l) \wedge DtlA^{i}_{r(R)}(l)$ evaluates to TRUE in Equation 7.18) or if l is in the *trigger action set* of sub-cubes of $r(R)$ after the decision-making model of $r(R)$ was evaluated (i.e., $TrgA^{it}(l)$ evaluates to TRUE in Equation 7.18). A negative global tentative decision (i.e., $DsmA^{gt}_{r(R)}$) is generated for l if l is in the *tentative dismiss action set* of $r(R)$ and in the *detail analysis set* of sub-cubes of $r(R)$ after the decision-making model of $r(R)$ was evaluated (i.e., $DsmA^{t}_{r(R)}(l) \wedge DtlA^{i}_{r(R)}(l)$ evaluates to TRUE in Equation 7.18) or if l is in the *dismiss action set* of sub-cubes of $r(R)$ after the decision-making model of $r(R)$ was evaluated (i.e., $DsmA^{it}(l)$ evaluates to TRUE in Equation 7.19).

Global Irresolute Decisions. A global irresolute decision (i.e., $DtlA^{g}_{r(R)}(l)$ is generated for level instance l of the analysis scope of $r(R)$ if l is in the *definite detail analysis set* of $r(R)$ and if no global definite or tentative decision was generated for l after the decision-making model of $r(R)$ was evaluated (i.e., $DtlA^{d}_{r(R)}(l) \wedge \neg\ (...)$ evaluates to TRUE in Equation 7.20). In any other situation a global definite or a global tentative decision was generated.

7.3.2 Decision-Making Models

A *decision-making model* is a particular way of combining the global decisions of sub-cubes of $r(R)$ such that a single global integrated decision can be generated by $r(R)$ for the level instances in the analysis scope of these sub-cubes. Hence, a decision-making model defines the semantics of operator $\uplus$ (cf. Equation 7.12). The motivation to allow for alternative decision-making models is as follows:

1. Various decision tasks may require different decision-making models to compose a single global decision out of several "expert opinions". An "expert opinion" is rep-

resented as the global decision of a sub-cube of some cube $r(R)$. One such decision-making model may be based on the *consensus* of all "experts". Another decision-making model may be based on the *single positive vote* of an expert (i.e., a positive decision is made if at least one sub-cube generates a positive decision).

2. Each sub-cube may autonomously generate its global definite, tentative, and irresolute decisions. Since a cube $r(R)$ may have several alternative sub-cubes, the global decisions of these sub-cubes may be contradictory for a particular level instance l. The decision-making model is responsible for resolving such conflicts and for generating a new global decision. The way of solving conflicts is not the same for every business decision problem. Hence, alternative decision-making models are necessary.

3. Every cube may generate its global decisions using its own decision-making model (i.e., cube-specific) or there may be a rule-covering decision making model that is valid for all cubes of the analysis graph. Rule-covering and cube-specific decision making models may be also combined (e.g., the rule-covering decision-making model is used as default for all cubes, a cube-specific decision-making model individually defines decision-making for a particular cube).

Note that the analysis scope of the sub-cubes of $r(R)$ is the set of tentative and irresolute decided cases of $r(R)$. The rationale for considering also tentative decided cases is that a tentative decision criterion represents a "default" situation for making a decision that may be overruled by a "more specific" decision criterion in a later step (i.e., a decision criterion that is evaluated on a "more specific" cube than $r(R)$).

Evaluating a decision-making model for some cube $r(R)$ generates a 5-tuple $\langle\ TrgA^{id}_{r(R)}, TrgA^{it}_{r(R)}, DtlA^{i}_{r(R)}, DsmA^{id}_{r(R)}, DsmA^{it}_{r(R)}\ \rangle$ which represents the *integrated global decisions* of sub-cubes of $r(R)$ respecting the semantics of $\uplus$. As identified in Section 6.3, a decision-making model must generate *complete* and *unique* decisions. *Completeness* requires that for *each* level instance of the cube's analysis scope a decision is generated. *Uniqueness* requires that exactly *one* decision is made for a particular level instance of the cube's analysis scope (i.e., any two different decision sets are disjoint: $TrgA^{id}_{r(R)} \cap TrgA^{it}_{r(R)} = \{\}, TrgA^{id}_{r(R)} \cap DtlA^{i}_{r(R)} = \{\}, TrgA^{id}_{r(R)} \cap DsmA^{it}_{r(R)} = \{\}, TrgA^{id}_{r(R)} \cap DsmA^{id}_{r(R)} = \{\}, TrgA^{it}_{r(R)} \cap DtlA^{i}_{r(R)} = \{\}, TrgA^{it}_{r(R)} \cap DsmA^{it}_{r(R)} = \{\}, TrgA^{it}_{r(R)} \cap DsmA^{id}_{r(R)} = \{\}, DtlA^{i}_{r(R)} \cap DsmA^{i}_{r(R)} = \{\}, DtlA^{i}_{r(R)} \cap DsmA^{id}_{r(R)} = \{\}, DsmA^{it}_{r(R)} \cap DsmA^{id}_{r(R)} = \{\})$.

Example: The decision-making model of our basic approach (cf. Section 5.3.2) generates the following integrated decisions for level instance l of some cube's *detail analysis set* (note that our basic approach only generates definite decisions):

1. Level instance l is in the *integrated definite trigger action set* if l is in the *global definite trigger action set* of any sub-cube of $r(R)$.

2. Level instance l is in the *integrated detail analysis set* if l is in the *global detail analysis set* of any sub-cube of $r(R)$ and if l is not in the *global definite trigger action set* of any sub-cube of $r(R)$.
3. Level instance l is in the *integrated definite dismiss action set* in any other situation, i.e., if l is not in the *global definite trigger action set* of any sub-cube of $r(R)$ and if l is not in the *global definite detail analysis set* of any sub-cube.

Semantics of Decision-Making Models

We define the semantics of a decision-making model using an n-dimensional table, which we call *decision table*. Each dimension of a decision table represents the global decisions of a cube $r(R)$ (i.e., $TrgA^{gd}_{r(R)}$, $TrgA^{gt}_{r(R)}$, $DtlA^{g}_{r(R)}$, $DsmA^{gd}_{r(R)}$, $DsmA^{gt}_{r(R)}$), which need to be integrated with the global decisions of other cubes (represented by the remaining dimensions of the decision table). A decision-making model is specified by an analyst by assigning an *integrated decision* (i.e., $TrgA^{id}_{r(R)}$, $TrgA^{it}_{r(R)}$, $DtlA^{i}_{r(R)}$, $DsmA^{id}_{r(R)}$, $DsmA^{it}_{r(R)}$) to each cell of a decision table. Hence, an integrated decision defines how the global (possibly conflicting) decisions of sub-cubes will be integrated. A decision table defines the semantics of operator $\uplus$. A decision table is specified correctly if there is an entry for every cell (i.e., there is an integrated decision for every combination of global decisions of sub-cubes). Correctness means that decision making is *complete* (i.e., there is an integrated decision for every possible combination of global decisions) and *unique* (i.e., every cell of the decision table contains exactly one entry).

Example: Table 7.1 presents the semantics of the decision-making model of our basic approach. Assume that these two cubes are cube1 and cube2 from Figure 7.5. Column headers and row headers represent the global decisions of two cubes, respectively. Each cell of this decision table defines how the "root" cube (cf. Figure 7.5) integrates the global decisions of cube 1 and cube 2. If *any* of the two cubes generated a positive decision (i.e., $TrgA^{gd}$) for some level instance l, the resulting integrated decision will be also positive (i.e., $TrgA^{id}$) for l. In Table 7.1, this is indicated by value $TrgA^{id}$ of the cells of the first row and the first column, respectively. If *both* cubes generated a negative decision (i.e., $DsmA^{gd}$) for l, the integrated decision is also negative (i.e., $DsmA^{id}$) for l. In Table 7.1, this is indicated by value $DsmA^{id}$ in the bottom right cell (i.e., at the intersection of row and column headed $DsmA^{gd}$). In any other situation (i.e., $DtlA^{g}$ vs. $DtlA^{g}$ and $DsmA^{gd}$ vs. $DtlA^{g}$) the integrated decision is irresolute (i.e., $DtlA^{i}$) for l. Since our basic approach only generates definite decisions, the tentative decisions $TrgA^{gt}$ and $DsmA^{gt}$ are not considered in Table 7.1.

We distinguish two classes of decision-making models:

1. In an *anonymous decision-making model*, generating integrated decisions is defined upon the global decisions of a set of sub-cubes as a whole. The dimensions of a

$\uplus^l$	$TrgA^{gd}$	$DtlA^{g}$	$DsmA^{gd}$
$TrgA^{gd}$	$TrgA^{id}$	$TrgA^{id}$	$TrgA^{id}$
$DtlA^{g}$	$TrgA^{id}$	$DtlA^{i}$	$DtlA^{i}$
$DsmA^{gd}$	$TrgA^{id}$	$DtlA^{i}$	$DsmA^{id}$

Table 7.1: Decision-making model of the basic approach

corresponding decision table may represent the global decisions of any of these cubes, i.e., the dimensions are not bound to particular cubes. Such a decision table may be used to integrate the global decisions incrementally if integrated decisions are defined upon *all* or *any* (or a combination of both) global decisions of sub-cubes. In such a situation, the number of dimensions that are needed to model the semantics of the decision-making model is reduced to 2. Decision making for n cubes is carried out using a *ladder strategy* (illustrated in Figure 7.7) by first integrating the global decisions of any two sub-cubes and then integrating the intermediately integrated decisions with the global decisions of the remaining cubes recursively. We denote a recursive decision table with $\uplus^l$. Note that such a recursive decision table must be *symmetrical*. Otherwise, the ladder strategy cannot not be applied.

Example: The decision-making model of our basic approach is an anonymous decision-making model since a particular dimension of that table is not bound to the global decisions of a particular cube but may represent the decisions of any cube. The decision-table (cf. Table 7.1) may be used to integrate the global decisions of n sub-cubes recursively since $TrgA^{id}$ is satisfied if $TrgA^{gd}$ of *any* sub-cube is satisfied and $DsmA^{gd}$ is satisfied if *all* sub-cubes agree upon their global decisions. Assume that the "root" cube in Figure 7.5 has three sub-cubes. Table 7.1 can be used to integrate the global decisions of any pair of cubes $\langle$cube1, cube2$\rangle$, $\langle$cube1, cube3$\rangle$, or $\langle$cube2, cube3$\rangle$. The intermediately integrated decisions are represented by any dimension of the 2-dimensional table and are then recursively integrated with the global decisions of the remaining (third) cube.

2. In a *named decision-making model*, generating integrated decisions is defined upon the global decisions of particular cubes, i.e., each dimension of a decision table represents the global decisions of a particular cube. When modeling integrated decisions by a decision table, each cube must be represented by a separate dimension. Since the dimensions of such a decision table are bound to particular cubes (i.e., dimensions cannot be interchanged between cubes), the ladder strategy for integrating the global decisions of sub-cubes is not applicable.

   Example: The decision-making model presented in Table 7.2 is a *named decision-making model* since the two dimensions of the table are bound to particular

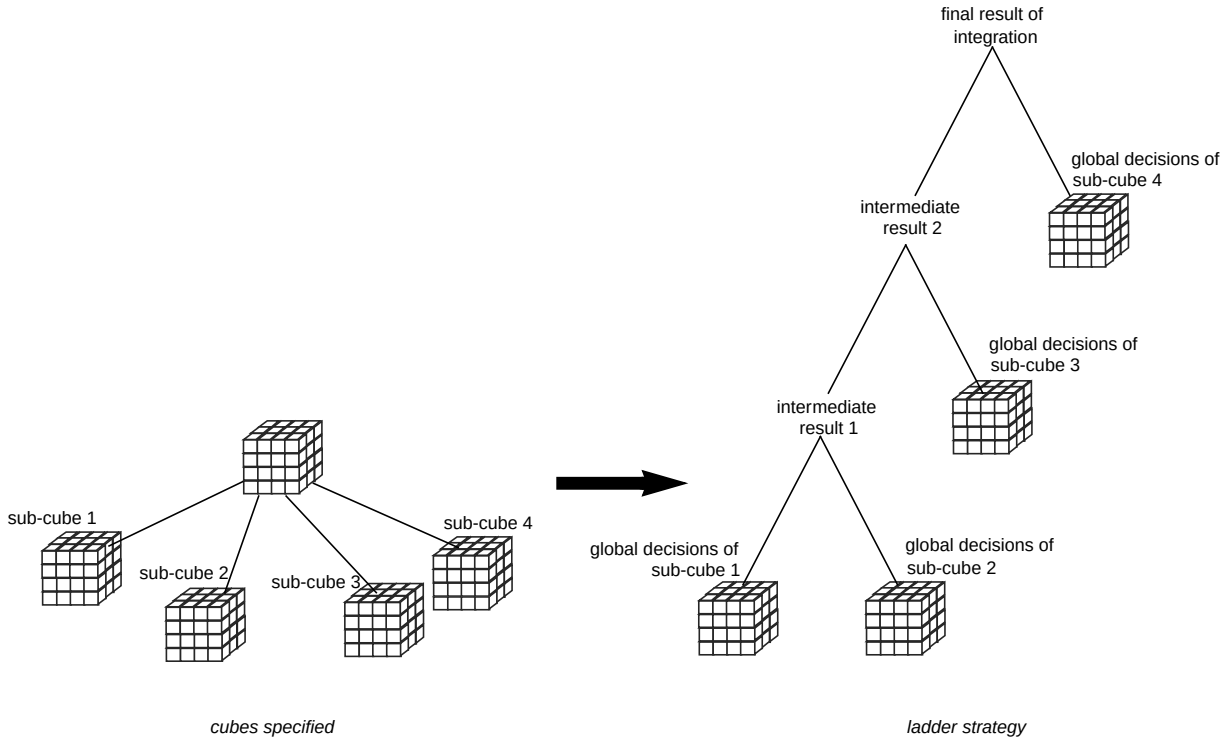

Figure 7.7: Ladder strategy to evaluate a 2-dimensional decision table for the global decisions of n cubes

$\uplus$		global decisions of cube 2		
		$TrgA^{gd}$	$DtlA^{g}$	$DsmA^{gd}$
global decisions of cube 1	$TrgA^{gd}$	$TrgA^{id}$	$TrgA^{id}$	$DsmA^{id}$
	$DtlA^{g}$	$TrgA^{id}$	$DtlA^{i}$	$DtlA^{i}$
	$DsmA^{gd}$	$TrgA^{id}$	$DtlA^{i}$	$DsmA^{id}$

Table 7.2: Named decision-making model

cubes and the table is not symmetrical. Note that for simplicity of presentation, we only consider definite and irresolute decisions here. If cube 1 decides for a particular level instance l to execute the rule's action but cube 2 decides to dismiss action execution for l, the integrated decision will be to dismiss action execution for l, whereas if cube 1 decides to dismiss action execution for l and cube 2 decides to trigger the rule's action for l, the integrated decision will be to trigger the rule's action for l.

An anonymous decision-making model may be specified once for all cubes of a particular rule (i.e., rule-covering decision-making model) or it may be used alternatively to integrate the global decisions of sub-cubes of a particular cube (cube-specific decision-making model). Since a named decision making model is bound to the global decisions of particular cubes, it cannot be used as a rule-covering decision-making model. Further, extending or modifying an analysis graph does not affect an anonymous decision-making model whereas a named decision-making model needs to be modified to consider the changes of the analysis graph for decision making.

Correctness. Integrating global decisions using n-dimensional tables satisfies *completeness* and *uniqueness*, which we identified in Section 6.3, as follows: *Completeness* is satisfied since (i) a cube's global decisions that need to be integrated are *complete*, (ii) each possible global decision is represented as a value of the dimensions of the n-dimensional table (i.e., no decision is missed), and (iii) the cells of the n-dimensional table are non-empty. *Uniqueness* is satisfied since from every cell of the n-dimensional table *exactly one* integrated decision can be derived.

Specifying Decision-Making Models

To specify decision-making models, we use the syntax presented in Figure 7.8. Due to the increased complexity of named decision-making models, we advocate to specify such decision-making models using the proposed syntax, while anonymous decision-making models can be specified using a 2-dimensional decision table alternatively. In the following, we will explain this syntax to specify decision-making models and define the semantics of the

proposed anonymous *decision quantifiers* ALL, ANY, and MAJ. In the examples above, we only considered integrating definite and irresolute global decisions. Extending these decision tables syntactically to support also tentative decisions is a seamless extension, which needs not be discussed separately. From now on we will consider definite, tentative, and irresolute global decisions to be integrated.

Basics. A correctly specified decision-making model consists of at most five rules to integrate global decisions of sub-cubes, one for each integrated decision (i.e., $TrgA^id$, $TrgA^{it}$, $DtlA^i$, $DsmA^{it}$, $TrgA^{id}$). In Figure 7.8, such a rule is represented by the non-terminal DecisionMakingRuleDef. A decision-making rule may refer either to the global decisions of a set of anonymous cubes (i.e., non-terminal AnonymousDecTermDef) or it may refer to the decision of a particular cube (i.e., non-terminal NamedDecTermDef). To ensure the *completeness* of decision making, one of these rules must be defined upon keyword ELSE, which identifies those cases to which no other decision-making rule is applicable. Such a rule is used to fill up the values of all empty cells in the corresponding decision table with the specified integrated decision. *Uniqueness* is achieved by transforming the syntactical specification to an n-dimensional decision table. Constructing an n-dimensional decision table from these rules is as follows: The IF/ELSE-branch of DecisionMakingRuleDef identifies those rows and columns whose cell values need to be replaced with a particular integrated decision that was specified in clause DecisionExprDef. *Completeness* is satisfied if there is at least one integrated decision for each cell of the n-dimensional table. *Uniqueness* is satisfied if from each cell of the generated decision table exactly one integrated decision can be derived.

Example: Consider a decision-making model that generates the same integrated decision (e.g., $TrgA^{id}$) for any combination of global decisions of sub-cubes by simply ignoring the global decisions of these sub-cubes. Such a decision-making model is specified using the following decision-making rule: DECIDE TrgA_D ELSE. Each cell value of the 2-dimensional table that is constructed from this specification will be replaced with the integrated decision $TrgA^{id}$. Table 7.3 presents the corresponding decision table. If we wish to specify a decision-making rule that generates a definite negative decision (i.e., $DsmA^{id}$) whenever the global decision of cube1 is also definite negative, the corresponding rule is: DECIDE DsmA_D IF DsmA_D@cube1. Table 7.4 presents the corresponding decision table.

Quantifying Decisions. Specifying anonymous decision predicates such as "decide positive if *any* sub-cube decided positive", "decide negative if *all* sub-cubes decided negative", or "decide positive if at least 3 out of 10 sub-cubes decided positive" etc. requires the

[3]Note that we introduced TrgA and DsmA to simplify specifying decision-making rules. The semantics of TrgA is TrgA_D ∨ TrgA_T. The semantics of DsmA is DsmA_D ∨ DsmA_T.

Syntax

```
DecisionMakingDef      ::= DEFINE DECISION-MAKING MODEL dmName
                             [IGNORE ListOfDecisionsDef] AS
                             DECIDE DecisionMakingRuleDef
                             { DECIDE DecisionMakingRuleDef }";".
DecisionMakingRuleDef  ::= IntegrDecisionDef
                             (IF DecisionExprDef
                             | ELSE).
DecisionExprDef        ::= DecisionTermDef
                             { (AND | OR) [NOT] DecisionTermDef }.
DecisionTermDef        ::= AnonymousDecTermDef | NamedDecTermDef.

AnonymousDecTermDef    ::= "(" DecisionQuantDef GlobalDecisionDef ")"
                           | "(" AnonymousDecTermDef
                             (AND | OR) [NOT] AnonymousDecTermDef ")".
NamedDecTermDef        ::= GlobalDecisionDef "@" cubeName
                           | GlobalDecisionDef "@" "(" cubeName {"," cube-
                           Name} ")".
DecisionQuantDef       ::= (ALL | ANY | MAJ).

IntegrDecisionDef      ::= ( TrgA_D
                             | TrgA_T
                             | DtlA
                             | DsmA_D
                             | DsmA_T).
GlobalDecisionDef[3]   ::= ( TrgA_D
                             | TrgA_T
                             | TrgA
                             | DtlA
                             | DsmA_D
                             | DsmA_T
                             | DsmA).
ListOfDecisionsDef     ::= GlobalDecisionDef {"," GlobalDecisionDef}.
```

Figure 7.8: Syntax to specify decision-making models

DECIDE DtlA ELSE

$\uplus$	$TrgA^{gd}$	$TrgA^{gt}$	$DtlA^{g}$	$DsmA^{gt}$	$DsmA^{gd}$
$TrgA^{gd}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$
$TrgA^{gt}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$
$DtlA^{g}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$
$DsmA^{gt}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$
$DsmA^{gd}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$	$\boldsymbol{DtlA}^{i}$

Table 7.3: Completing an empty decision table

DECIDE DsmA_D IF DsmA_D@cube1

$\uplus$		global decisions of cube 2				
		$TrgA^{gd}$	$TrgA^{gt}$	$DtlA^{g}$	$DsmA^{gt}$	$DsmA^{gd}$
global decisions of cube 1	$TrgA^{gd}$					
	$TrgA^{gt}$					
	$DtlA^{g}$					
	$DsmA^{gt}$					
	$\boldsymbol{DsmA}^{gd}$	$\boldsymbol{DsmA}^{id}$	$\boldsymbol{DsmA}^{id}$	$\boldsymbol{DsmA}^{id}$	$\boldsymbol{DsmA}^{id}$	$\boldsymbol{DsmA}^{id}$

Table 7.4: Deriving decision table entries from a named decision making rule

quantification of global decisions of sub-cubes. For this purpose, we introduce anonymous *decision quantifiers* ALL, ANY, and MAJ (cf. non-terminal DecisionQuantDef in Figure 7.8).

1. Decision quantifier ALL X (X represents a particular global decision of a sub-cube) requires that all sub-cubes generated global decision X for some level instance l. Constructing decision table entries from a decision making rule DECIDE Y IF ALL X that is defined upon this quantifier is as follows:

   (a) Identify that cell of the decision table whose dimension values are X (i.e., decision_table[X, X, X, ..., X]).

   (b) Replace the entry of this cell with the specified integrated decision (i.e., decision_table[X, X, X, ..., X] = Y). *Uniqueness* is violated if an entry already exists for that cell.

   Example: Table 7.5 presents how decision making rule DECIDE DtlA IF ALL DtlA will be translated into corresponding entries of a decision table. For presentation purposes, the decision table in Table 7.5 consists of only two dimensions.

2. Decision quantifier ANY X requires that at least one sub-cube generated global decision X for some level instance l. Constructing decision table entries from a decision making rule DECIDE Y IF ANY X that is defined upon this quantifier is as follows:

DECIDE DtlA IF ALL DtlA

$\uplus^l$	$TrgA^{gd}$	$TrgA^{gt}$	$\boldsymbol{DtlA}^g$	$DsmA^{gt}$	$DsmA^{gd}$
$TrgA^{gd}$					
$TrgA^{gt}$					
$\boldsymbol{DtlA}^g$			$\boldsymbol{DtlA}^i$		
$DsmA^{gt}$					
$DsmA^{gd}$					

Table 7.5: Deriving decision table entries from decision quantifier ALL

DECIDE DtlA IF ANY DtlA

$\uplus^l$	$TrgA^{gd}$	$TrgA^{gt}$	$\boldsymbol{DtlA}^g$	$DsmA^{gt}$	$DsmA^{gd}$
$TrgA^{gd}$			$\boldsymbol{DtlA}^i$		
$TrgA^{gt}$			$\boldsymbol{DtlA}^i$		
$\boldsymbol{DtlA}^g$	$\boldsymbol{DtlA}^i$	$\boldsymbol{DtlA}^i$	$\boldsymbol{DtlA}^i$	$\boldsymbol{DtlA}^i$	$\boldsymbol{DtlA}^i$
$DsmA^{gt}$			$\boldsymbol{DtlA}^i$		
$DsmA^{gd}$			$\boldsymbol{DtlA}^i$		

Table 7.6: Deriving decision table entries from decision quantifier ANY

(a) Identify those cells of the decision table having at least one dimension value X (i.e., decision_table[X, ...], decision_table[...,X, ...], decision_table[...,X], decision_table[X,X,...], etc.).

(b) Replace the entries of these cells with the specified integrated decision (i.e., decision_table[X, ...] = Y, decision_table[...,X, ...] = Y, decision_table[...,X] = Y, decision_table[X,X,...] = Y, etc.). *Uniqueness* is violated if for any of these cells there exists already an entry that represents a different integrated decision.

Example: Table 7.6 presents how decision making rule DECIDE DtlA IF ANY DsmA will be translated into corresponding entries of a decision table.

3. Decision quantifier MAJ X requires that the "bare majority" (i.e., $\geq$ 51 %) of subcubes generated global decision X for some level instance l. Constructing decision table entries from a decision making rule DECIDE Y IF MAJ X that is defined upon this quantifier is as follows:

   (a) Identify those cells of the decision table having value X for at least 51 % of their dimensions[4] (i.e., decision_table[X, ...]$^{count(X)/n \geq 0.51}$, decision_table[...,X, ...]$^{count(X)/n \geq 0.51}$, decision_table[...,X]$^{count(X)/n \geq 0.51}$, decision_table[X,X,...]$^{count(X)/n \geq 0.51}$, etc.).

[4] decision_table[X, ...]$^{count(X)/n \geq 0.51}$ means that cell decision_table[X, ...] has at least 51 % X-dimension values.

DECIDE TrgA_D IF MAJ TrgA_D

Dimension 3: $TrgA^{gd}$

$\uplus$	$TrgA^{gd}$	$TrgA^{gt}$	$\boldsymbol{DtlA^{g}}$	$DsmA^{gt}$	$DsmA^{gd}$
$TrgA^{gd}$	$\boldsymbol{TrgA^{id}}$	$\boldsymbol{TrgA^{id}}$	$\boldsymbol{TrgA^{id}}$	$\boldsymbol{TrgA^{id}}$	$\boldsymbol{TrgA^{id}}$
$TrgA^{gt}$	$\boldsymbol{TrgA^{id}}$				
$\boldsymbol{DtlA^{g}}$	$\boldsymbol{TrgA^{id}}$				
$DsmA^{gt}$	$\boldsymbol{TrgA^{id}}$				
$DsmA^{gd}$	$\boldsymbol{TrgA^{id}}$				

Table 7.7: Deriving decision table entries from decision quantifier MAJ

(b) Replace the entries of these cells with the specified integrated decision (i.e., decision_table[X, ...]$^{count(X)/n \geq 0.51}$ = Y, ...). *Uniqueness* is violated if for any of these cells there exists already an entry that represents a different integrated decision.

Note that decision quantifier MAJ cannot be described by a decision table that may be evaluated using the ladder strategy.

Example: Table 7.7 presents how decision making rule DECIDE TrgA_D IF MAJ TrgA_D will be translated into corresponding entries of a decision table. For presentation purposes, this 3-dimensional decision table presents the cells with dimension values GlobalDecisonDef, GlobalDecisionDef, TrgA_D.

Such decision terms (cf. syntax in Figure 7.8) may be combined within a decision making rule using conjunctors AND and OR (cf. syntax in Figure 7.8) to construct a more complex decision making rule. Further, a decision term may also be negated using the unary operator NOT. Identifying the cells is as follows:

$term_1$ AND $term_2$: Identify those cells of the decision table that satisfy the predicate specified in $term_1$. Identify those cells of the decision table that satisfy the predicate specified in $term_2$. The resulting set of cells is determined upon the *intersection* ($\cap$) of the set of cells that satisfy $term_1$ and the set of cells that satisfy $term_2$.

$term_1$ OR $term_2$: Identify those cells of the decision table that satisfy the predicate specified in $term_1$. Identify those cells of the decision table that satisfy the predicate specified in $term_2$. The resulting set of cells is determined upon the *union* ($\cup$) of the set of cells that satisfy $term_1$ and the set of cells that satisfy $term_2$.

NOT $term_1$: Identify those cells of the decision table that satisfy the predicate specified in $term_1$. The resulting set of cells that satisfy NOT $term_1$ is determined as the *difference* ($\setminus$) between the complete set of cells of the decision table and the set of cells that satisfy $term_1$.

```
DEFINE DECISION-MAKING MODEL basic AS
  DECIDE TrgA_D IF ANY TrgA_D
  DECIDE DsmA_D IF ALL DsmA_D
  DECIDE DtlA ELSE;
```

Figure 7.9: Decision-making model of the basic approach

Several decision making rules may combined arbitrarily to constitute a complete decision-making model. Note that the decision-making model must satisfy *completeness* and *uniqueness* as discussed above.

Example: The specification of the decision-making model of our basic approach is presented in Figure 7.9. Since this decision-making model generates (i) definite positive decisions, (ii) definite negative decisions, and (iii) irresolute decisions, we specified three rules for decision making, one for each integrated decision. A definite positive decision is generated if ANY sub-cube generated a global definite positive decision for a particular level instance. A definite negative decision is generated if ALL sub-cubes generated a definite negative decision for a particular level instance. In any other situation, the integrated decision is irresolute.

Patterns for decision-making

While named decision-making models introduce priorities among the decisions of individual decision-makers (i.e., cubes), anonymous decision-making models treat decisions uniformly as it is the case with "voting procedures" in real world, from which we borrow three basic patterns: *consensus*, *majority*, and *veto* (or *minority*). In the following, we will explain how these patterns can be adopted by anonymous decision-making models using positive/negative, definite/tentative, and irresolute decision criteria. Since positive and negative decisions are either definite or tentative, we will explain some varieties of decision-making models that represent these patterns.

Consensus. A consensus based decision-making model requires all sub-cubes to agree upon their decision concerning a particular decision task (i.e., execute/don't execute the action of an analysis rule). In an anonymous decision-making model, decision quantifier ALL is used to enforce consensus among the global decisions of sub-cubes. In a named decision-making model, the global decision upon which all sub-cubes must agree must be explicitly conjuncted among these sub-cubes (e.g., TrgA_D@sub-cube$_1$ AND ... AND TrgA_@sub-cube$_n$). Note that the priorities among definite, tentative, and irresolute decisions have been established in Section 7.2 to generate *local decisions* of one cube, while here we integrate *global decisions* of several cubes.

$\uplus^l$	$TrgA^{gd}$	$TrgA^{gt}$	$DtlA^{g}$	$DsmA^{gt}$	$DsmA^{gd}$
$TrgA^{gd}$	$TrgA^{id}$	$TrgA^{it}$	$DtlA^{i}$	$DtlA^{i}$	$DtlA^{i}$
$TrgA^{gt}$	$TrgA^{it}$	$TrgA^{it}$	$DtlA^{i}$	$DtlA^{i}$	$DtlA^{i}$
$DtlA^{g}$	$DtlA^{i}$	$DtlA^{i}$	$DtlA^{i}$	$DtlA^{i}$	$DtlA^{i}$
$DsmA^{gt}$	$DtlA^{i}$	$DtlA^{i}$	$DtlA^{i}$	$DsmA^{it}$	$DsmA^{it}$
$DsmA^{gd}$	$DtlA^{i}$	$DtlA^{i}$	$DtlA^{i}$	$DsmA^{id}$	$DsmA^{it}$

Table 7.8: Decision-making model requiring the "definite consensus" among sub-cubes

```
DEFINE DECISION-MAKING MODEL DefiniteConsensus AS
  DECIDE TrgA_D IF ALL TrgA_D
  DECIDE TrgA_T IF ALL TrgA AND ANY TrgA_T
  DECIDE DtlA   ELSE
  DECIDE DsmA_T IF ALL DsmA AND ANY DsmA_T
  DECIDE DsmA_D IF ALL DsmA_D;
```

Figure 7.10: Specification of decision-making model DefiniteConsensus

Example: Table 7.8 presents a decision-making model, which requires the *definite consensus* among the global decisions of sub-cubes to generate an integrated decision. *Definite consensus* means that this decision-making model generates definite decisions only if all cubes agree upon their definite decisions. Otherwise tentative or irresolute decisions are generated. Decision-making is as follows (cf. the syntax in Figure 7.10): A definite decision (i.e., $TrgA^{id}$ or $DsmA^{id}$) is generated only if all cubes generated definite decisions (i.e., $TrgA^{gd}$ or $DsmA^{gd}$, respectively). A tentative decision (i.e., $TrgA^{it}$ or $DsmA^{it}$) is generated if (i) all cubes generated tentative decisions (i.e., $TrgA^{gt}$ or $DsmA^{gt}$, respectively) or if (ii) some cubes generated tentative decisions while the remaining cubes generated definite decisions (i.e., $TrgA^{gd}$ or $DsmA^{gd}$, respectively). In any other situation, the integrated decision is irresolute (i.e., $DtlA^{i}$). Integrated decision $TrgA^{it}$ is defined upon $TrgA$, which evaluates to TRUE for a given level instance if a cube's global decision is $TrgA^{id}$ or $TrgA^{it}$. Integrated decision $DsmA^{it}$ is defined upon $DsmA$, which evaluates to TRUE for a given level instance if a cube's global decision is $DsmA^{gd}$ or $DsmA^{gt}$ (i.e., irrespective whether the cube's global decision is tentative or definite).

Since our approach supports the integration of five alternative global decisions of sub-cubes, the analyst is free to specify alternative decision-making models, which all require the consensus among decision-makers (i.e., sub-cubes) but that differ in which integrated decision is generated finally.

$\uplus^l$	$TrgA^{gd}$	$TrgA^{gt}$	$DtlA^{g}$	$DsmA^{gt}$	$DsmA^{gd}$
$TrgA^{gd}$	$TrgA^{id}$	$\boldsymbol{TrgA}^{id}$	$DtlA^{i}$	$DtlA^{i}$	$DtlA^{i}$
$TrgA^{gt}$	$\boldsymbol{TrgA}^{id}$	$TrgA^{it}$	$DtlA^{i}$	$DtlA^{i}$	$DtlA^{i}$
$DtlA^{g}$	$DtlA^{i}$	$DtlA^{i}$	$DtlA^{i}$	$DtlA^{i}$	$DtlA^{i}$
$DsmA^{gt}$	$DtlA^{i}$	$DtlA^{i}$	$DtlA^{i}$	$DsmA^{it}$	$\boldsymbol{DsmA}^{id}$
$DsmA^{gd}$	$DtlA^{i}$	$DtlA^{i}$	$DtlA^{i}$	$\boldsymbol{DsmA}^{id}$	$DsmA^{id}$

Table 7.9: Decision-making model requiring the "tentative consensus" of sub-cubes

```
DEFINE DECISION-MAKING MODEL TentativeConsensus AS
  DECIDE TrgA_D IF ALL TrgA AND ANY TrgA_D
  DECIDE TrgA_T IF ALL TrgA_T
  DECIDE DtlA   ELSE
  DECIDE DsmA_T IF ALL DsmA_T
  DECIDE DsmA_D IF ALL DsmA AND ANY DsmA_D;
```

Figure 7.11: Specification of decision-making model TentativeConsensus

Example: Table 7.9 presents an alternative decision-making model, which requires the *tentative consensus* of all cubes to generate an integrated decision. *Tentative consensus* means that an integrated tentative decision is generated only if all cubes agree upon their tentative decisions. More specifically, an integrated definite decision is generated if (i) all cubes agree definitely upon their global decisions or if (ii) some cubes generated global definite decisions while the remaining cubes generated global tentative decisions. Otherwise, irresolute decisions are generated. This decision-making model respects the priorities between definite, tentative, and irresolute decisions, which we established in Section 7.2, since conflicts between definite and tentative decisions are resolved in favor of the definite decision (i.e., definite decisions overrule tentative decisions). Figure 7.11 presents the syntactical specification of this decision-making model. If we compare this decision-making model with *definite consensus*, we can observe that the number of integrated tentative decisions decreases while the number of integrated definite decisions increases (compare the bold typed integrated decisions in Table 7.9 with the corresponding integrated decisions in Table 7.8).

So far, we considered all possible global decisions of sub-cubes (i.e., $TrgA^{gd}$, $TrgA^{gt}$, $DtlA^{g}$, $DsmA^{gt}$, $DsmA^{gd}$) to be relevant for generating a particular integrated decision. Excluding a particular global decision of sub-cubes from decision making can be achieved by formulating rather complex decision predicates (e.g., consider a decision-making model that requires the consensus among positive decisions of sub-cubes without considering those sub-cubes that generated irresolute decisions). To simplify the specification of such predicates, we

$\uplus^l$	$TrgA^{gd}$	$TrgA^{gt}$	$DtlA^{g}$	$DsmA^{gt}$	$DsmA^{gd}$
$TrgA^{gd}$	$TrgA^{id}$	$TrgA^{it}$	$\boldsymbol{TrgA}^{id}$	$DtlA^{i}$	$DtlA^{i}$
$TrgA^{gt}$	$TrgA^{it}$	$TrgA^{it}$	$\boldsymbol{TrgA}^{it}$	$DtlA^{i}$	$DtlA^{i}$
$DtlA^{g}$	$\boldsymbol{TrgA}^{id}$	$\boldsymbol{TrgA}^{it}$	$DtlA^{i}$	$\boldsymbol{DsmA}^{it}$	$\boldsymbol{DsmA}^{id}$
$DsmA^{gt}$	$DtlA^{i}$	$DtlA^{i}$	$\boldsymbol{DsmA}^{it}$	$DsmA^{it}$	$DsmA^{it}$
$DsmA^{gd}$	$DtlA^{i}$	$DtlA^{i}$	$\boldsymbol{DsmA}^{id}$	$DsmA^{id}$	$DsmA^{it}$

Table 7.10: Decision-making model requiring the "definite consensus" among sub-cubes without considering irresolute decisions

introduced clause IGNORE in the definition of a decision-making model (cf. non-terminal DecisionMakingDef in Figure 7.8). When clause IGNORE is specified to exclude some global decision of sub-cubes from being considered in integrating decisions, this global decision is uniformly excluded from all decision-making rules of the decision-making model.

Example: Consider the decision-making model DefiniteConsensus, which we defined syntactically in Figure 7.10. Using this decision making-model, a definite positive decision is generated if all sub-cubes agree upon their global decisions (i.e., DECIDE TrgA_D IF ALL TrgA_D). If we wish to exclude global irresolute decisions of sub-cubes (i.e., $DtlA^{g}$) from being considered in generating definite positive decisions, we must extend the corresponding decision making rules to DECIDE TrgA_D IF (ALL (TrgA_D OR DtlA) AND ANY TrgA_D) etc. or we alternatively use clause IGNORE DtlA in the definition of the decision-making model. Table 7.10 presents the modified decision-making model (called DefiniteConsensus2), in which positive and negative integrated decisions are derived from the global decision of sub-cubes without considering global irresolute decisions of sub-cubes. Figure 7.12 presents the corresponding syntactical specification.

Note that IGNORE reduces the number of global decisions to be counted when decision quantifier MAJ is used in a decision making rule.

Example: Assume that clause IGNORE DtlA was specified for a decision-making model that integrates the global decisions of three sub-cubes. A decision making rule of this decision-making model is as follows: DECIDE TrgA_D IF MAJ TrgA_D. Further assume that the dimension values of a particular cell are decision_table[TrgA_D, DtlA, DtlA]. Since global decision DtlA is ignored, this cell satisfies quantifier MAJ.

Majority. A majority based decision-making model requires the majority of sub-cubes to agree in their decisions concerning a particular decision task. Decision quantifier MAJ

```
DEFINE DECISION-MAKING MODEL DefiniteConsensus2 IGNORE DtlA AS
  DECIDE TrgA_D IF ALL TrgA_D
  DECIDE TrgA_T IF ALL TrgA ANY TrgA_T
  DECIDE DtlA   ELSE
  DECIDE DsmA_T IF ALL DsmA AND ANY DsmA_T
  DECIDE DsmA_D IF ALL DsmA_D;
```

Figure 7.12: Specification of decision-making model DefiniteConsensus2

```
DEFINE DECISION-MAKING MODEL DefiniteMajority AS
  DECIDE TrgA_D IF MAJ TrgA_D AND (NOT ANY TrgA_T)
  DECIDE TrgA_T IF MAJ TrgA AND ANY TrgA_T
  DECIDE DtlA   ELSE
  DECIDE DsmA_T IF MAJ DsmA AND ANY DsmA_T
  DECIDE DsmA_D IF MAJ DsmA_D AND (NOT ANY DsmA_T);
```

Figure 7.13: Specification of decision-making model DefiniteMajority

(i.e., more than 50% of sub-cubes must agree upon a particular global decision) is used to enforce the "bare" majority of global decisions of sub-cubes.

Example: Figure 7.13 presents the majority based decision-making model DefiniteMajority, which requires the "bare" majority among the global decisions of sub-cubes to generate a corresponding integrated decision. *Definite majority* means that an integrated definite decision is generated only if the majority of cubes agree upon their definite decisions. This decision-making model is "weaker" than decision-making model DefiniteConsensus since instead of requiring the consensus among cubes, only the "bare" majority is required for decision-making. Note that a 2-dimensional table is inappropriate to model the semantics of decision quantifier MAJ. Hence, we only present the syntactical specification of this decision-making model.

Consensus based and majority based decision-making models may be combined unless completeness and uniqueness are not violated by the resulting decision-making model.

Example: Figure 7.14 presents the specification of decision-making model PosTentConsNegDefMaj which combines the decision-making model TentativeConsensus (for positive decisions) with decision-making model DefiniteMajority (for negative decisions). Hence, an integrated positive decision requires the *tentative consensus* among the global decisions of sub-cubes while an integrated negative decision requires the *definite majority* among the global decisions of sub-cubes. An integrated irresolute decision will be generated in any other situation.

```
DEFINE DECISION-MAKING MODEL PosTentConsNegDefMaj AS
  DECIDE TrgA_D IF ALL TrgA AND ANY TrgA_D
  DECIDE TrgA_T IF ALL TrgA_T
  DECIDE DtlA   ELSE
  DECIDE DsmA_T IF MAJ DsmA AND ANY DsmA_T
  DECIDE DsmA_D IF MAJ DsmA_D;
```

Figure 7.14: Specification of decision-making model PosTentConsNegDefMaj

```
DEFINE DECISION-MAKING MODEL PosDefMajNegVeto AS
  DECIDE TrgA_D IF MAJ TrgA_D AND (NOT ANY DsmA) AND (NOT ANY TrgA_T)
  DECIDE TrgA_T IF MAJ TrgA AND ANY TrgA_T AND (NOT ANY DsmA)
  DECIDE DtlA   ELSE
  DECIDE DsmA_T IF (ANY DsmA_T) AND (NOT ANY DsmA_D)
  DECIDE DsmA_D IF ANY DsmA_D;
```

Figure 7.15: Specification of decision-making model PosDefMajNegVeto

Veto (Minority). Many decision-making models in real world incorporate *veto mechanisms*, i.e., the possibility of a single decision-maker or of a minority of decision-makers to prevent from a decision that was generated by an authorized majority of decision-makers due to the vote. While consensus based decision making models have an implicit veto mechanism, i.e., consensus is not satisfied if at least a single decision-maker decides contrary to the decisions of the remaining decision-makers, veto mechanisms need to be specified explicitly for all other decision-making models. Note that in a named decision-making model, the veto depends on the decision of a particular decision-maker. In an anonymous decision-making model, the veto depends on the decision of any decision-maker. Hence, we use decision quantor ANY for specifying veto mechanisms. Note that this decision quantifier represents an arbitrary number (≥ 1) of decision-makers (i.e., sub-cubes) that may prevent some other integrated decision.

Example: Figure 7.15 presents the specification of decision-making model PosDefMajNegVeto, which requires the *definite majority* among the global decisions of sub-cubes to generate an integrated positive decision. This decision-making model incorporates a *single negative veto* against positive decisions as follows: An integrated definite negative decision is generated if the global decision of any sub-cube is definite negative. An integrated tentative negative decision is generated if the global decision of any sub-cube is tentative negative but no other sub-cube generated a global definite negative decision.

7.4 Decision Parameters

Besides having the possibility to specify individual semantics to generate global decisions as required by particular decision tasks, analysts should also be free to specify the bindings of decision parameters in a flexible way. By *decision parameters*, we mean the parameters of an analysis rule's action, which will be invoked in an OLTP system for each positively decided object. In our basic approach, we allowed for *static* parameter bindings (i.e., constant values will be used as the bindings of action parameters) that are defined once for all level instances for which the analysis rule's action should be executed (i.e., *rule-covering*). In Section 6.4, we identified that (i) the bindings of decision parameters should be determined more dynamically (i.e., as the result of multidimensional analyzes) and that (ii) analysts should have the possibility to specify these bindings once for all level instances of the primary dimension level (i.e., rule-covering) or for each cube of the analysis graph (i.e., cube-specific). Figure 7.16 illustrates these "dimensions" with which the support of parameter bindings is classified. While our basic approach only supported static parameter bindings that were specified for all decided level instances of an analysis rule, the extended approach allows for static and dynamic parameter bindings that may be specified for the local decisions of separate cubes of an analysis graph or for all final decisions of an analysis rule. When using dynamic decision parameters, we further identified that analysts should be free to restrict the possible bindings for each decision parameter by specifying constraints that must be satisfied by these bindings. To keep the model simple, we restrict these constraints to parameter "ranges" (e.g., if transaction changePrice should be invoked for some level instance in an OLTP system, the price-changing ratio of this transaction may range from 2 % – 7 %).

Note that rule-covering/cube-specific decision parameters and static/dynamic decision parameters are orthogonal. While rule-covering/cube-specific decision parameters differ from each other in the set of level instances for which these bindings are determined, static/dynamic decision parameters differ from each other in how parameter bindings are determined (i.e., statically using constant values or dynamically by evaluating a query). Hence, we will first introduce rule-covering/cube-specific decision parameters and will afterwards provide the syntax to specify these parameters using static/dynamic parameter bindings and to specify ranges to restrict the possible bindings of a decision parameter.

7.4.1 Rule-Covering/Cube-Specific Bindings

Rule-Covering Bindings. A rule-covering parameter binding is defined once to be used uniformly when executing the rule's action for the positively decided level instances of that rule. Hence for each positively decided object, there exists a 1:1 relationship between the decision parameters and their bindings at any stage of generating decisions, which is illustrated in the left part of Figure 7.17. Rule-covering parameter bindings are *default bindings* since these bindings will be used for action execution when no "more specific" bindings

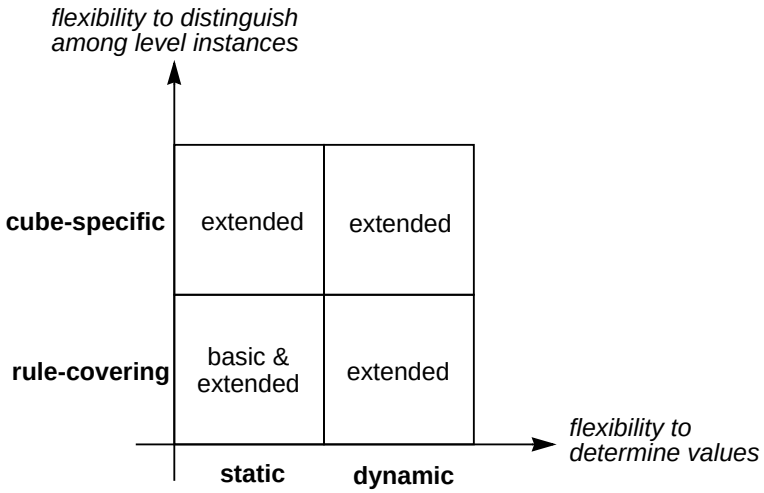

Figure 7.16: Dimensions to describe parameter bindings

(i.e., cube-specific bindings) exist. Syntactically, rule-covering bindings are specified in clause TO EXECUTE ... PARAMETERS of an analysis rule.

Cube-Specific Bindings. A cube-specific parameter binding is defined for the positively decided level instances taken from the local decisions of a particular cube of the rule's analysis graph. Such a binding will be determined only for the level instances for which a positive decision (tentative and definite) was generated after evaluating the decision criteria of some cube. Since a particular level instance l may be analyzed by more than one cube of the rule's analysis graph, there may exist a 1:n relationship between a decision parameter and its bindings for l intermittently. After decisions have been generated by the analysis rule, there is again a 1:1 relationship between decision parameters and their bindings, which is represented in the right part of Figure 7.17. If the analysis rule finally generated a positive decision for l, one of these alternative bindings may be used as action parameters when the rule's action is executed in an OLTP system. Hence, an analyst must define for each decision parameter *param* the binding to be used for action execution if several alternative bindings exist for *param*. We introduce function $f(ab) \rightarrow fb$ to determine a single final binding fb using a set of alternative bindings ab as input. If the alternative bindings are numeric, the analyst may also derive the final binding using one of the predefined functions COMMON_VAL, SUM, AVG, MAX, or MIN or may provide a user-defined function instead. These functions are defined as follows:

COMMON_VAL requires that alternative cube-specific bindings must be identical in order to be used for action execution. If these cube-specific bindings are not identical, a corresponding rule-covering (default) binding must be used instead.

SUM summarizes the alternative cube-specific bindings.

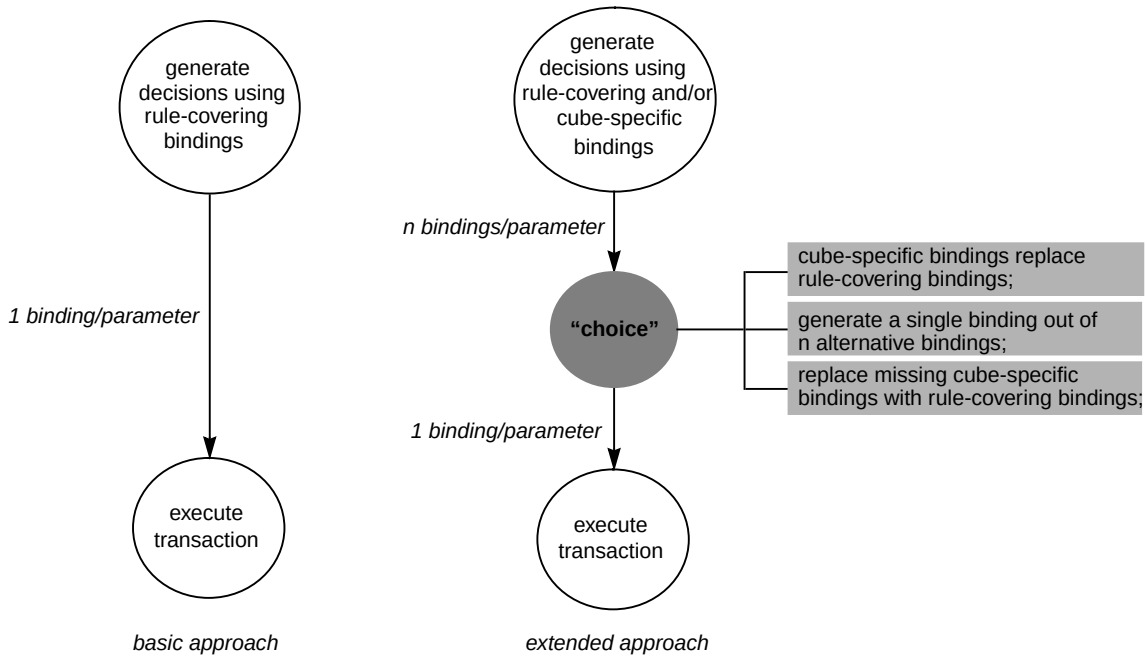

Figure 7.17: Parameter bindings – basic approach vs. extended approach

AVG computes the average of the set of alternative cube-specific bindings.

MAX retrieves the maximum binding out of the set of alternative cube-specific bindings.

MIN retrieves the minimum binding out of the set of alternative cube-specific bindings.

Note that functions SUM, AVG, MAX, and MIN are only applicable to numeric bindings. If no cube-specific bindings could be determined, the corresponding rule-covering default binding will be used for action execution. Cube-specific parameter bindings should be used together with to rule-covering parameter bindings. Syntactically, cube-specific bindings are specified in clause TRIGGER ACTION of a decision step.

Parameter Ranges. Each decision parameter *param* has an associated range of values $\psi = (val_1, val_2, ..., val_n)$, which defines the permitted values to which *param* may be bound when the action is executed. This range of values is particularly important for checking the validity of dynamically determined bindings. Further, if a binding that was derived from cube-specific bindings exceeds this range of values, the corresponding rule-covering binding will be used as default instead. Syntactically, ranges of values may be specified by clause RESTRICT for each action parameter in clause TO EXECUTE ... PARAMETERS of an analysis rule. If clause RESTRICT is omitted, the associated range of values ψ is unrestricted. If a decision parameter has a numeric domain, an analyst may specify an upper and a lower bound of the range which is represented by a predicate. If a decision

Syntax

```
ParameterDef      ::=  PARAMETERS [DynBindingDef]
                         parameterName "=" RuleParamValDef [RangeDef]
                         { "," parameterName "=" RuleParamValDef [RangeDef] }.
DynBindingDef     ::=  ANALYZE cubeName
                         "[" DecisionVarDef { "," DecisionVarDef } "]".
RuleParamValDef   ::=  (COMMON_VAL | SUM | AVG | MAX | MIN | userDefFunct)
                         ELSE ParamValDef.
ParamValDef       ::=  (constant | PrimDimVarAttr | EventVarAttr
                         | DecisionVarName "." measureAttrName
                         | DecisionVarName "." dimAttrName).
RangeDef          ::=  RESTRICT TO "(" (ValEnumDef | TopBotDef) ")".

ValEnumDef        ::=  value { "," value }.

TopBotDef         ::=  topValue "/" bottomValue.
```

Figure 7.18: Syntax to specify rule-covering parameter bindings

parameter has a non-numeric domain, an analyst may specify an enumeration of possible values that define the domain.

7.4.2 Static Bindings/Dynamic Bindings

A static binding is a constant value that will be used as action parameter for all positive decisions. Dynamic bindings are determined by evaluating a query for each positive decision separately. Figure 7.18 presents the syntax to specify rule-covering static and dynamic parameter bindings. Non-terminal ParameterDef is an extension to the original syntax for specifying rule-covering parameter bindings in Section 5.1. Non-terminal DynBindingDef allows analysts to specify an analytic expression whose virtual cells may be used to determine dynamic parameter bindings as follows: The dynamic binding of an action parameter may be the projected value of a dimension attribute or the projected value of a measure attribute of a virtual cell. Non-terminal RangeDef allows analysts to specify parameter ranges by explicitly enumerating the allowed values or by defining an upper and a lower bound for numeric parameters alternatively.

Example: Figure 7.19 presents a modified version of analysis rule ChangePriceOfSoftDrinksAR, which we introduced in Section 5.1. Clause TO EXECUTE refers to OLTP

method changePrice, which requires a float value as parameter indicating the price changing ratio. The default value that will be bound to this action parameter is 0.85. If cube-specific parameter bindings have been generated for some level instance l, the common value of these bindings (i.e., COMMON_VAL) will be used for action execution. Otherwise constant value 0.85 will be used as static binding. The range of possible values that may be bound to this parameter is defined in clause RESTRICT TO with an upper bound (0.95) and a lower bound 0.85).

Figure 7.20 presents the syntax to specify cube-specific static and dynamic parameter bindings. Non-terminal TriggerActionCondDef is an extension to the syntax for specifying positive decision criteria of a decision step. Since these parameter bindings are defined within a decision step, clause PARAMETERS does not provide for dynamic bindings separately. Instead, dimension attributes and measure attributes of the decision step's decision variables may be used as dynamic parameter bindings.

Example: Analysis rule ChangePriceOfSoftDrinksAR in Figure 7.19 defines a cube-specific parameter binding with the decision step that was defined for cube QuantSalesAllLocationsDays. This binding is only determined for level instances that will be analyzed by this decision step. Decision variable change represents the changing ratio of sold quantities for two ten day-periods. If the changing ratio is below the threshold of 0.95, the rule's action should be executed for the analyzed level instance. In such a situation, this changing ratio will be used as binding instead of the default binding defined in clause TO EXECUTE ... PARAMETERS. Note that whenever this dynamically determined value exceeds the range of values that may be used for action execution, the default binding will be used instead.

7.5 Procedural Evaluation of Decision Making

In Section 5.4, we presented a procedural semantics to evaluate analysis rules of our basic approach by providing the two procedures ExecuteAnalysisRule and EvaluateDecisionStep. Procedure ExecuteAnalysisRule initiates top-down evaluation of decision making by determining the analysis scope for a given analysis rule and by invoking procedure EvaluateDecisionStep for the root cube of the analysis graph. Procedure EvaluateDecisionStep realizes two tasks: (i) evaluate the decision criteria defined for cube $r(R)$ (i.e., generate the local decisions of $r(R)$) and (ii) try to generate a "global decision" as soon as possible. These two procedures were designed to support our basic decision-making model only. Since we allow for alternative decision-making models in the extended approach, evaluating decision criteria and making global decisions must be treated in separate steps. In this section we introduce a procedural semantics for evaluating decision criteria as defined in Section 7.2 (i.e., considering positive/negative/irresolute decision criteria and tentative/definite decision criteria) and we will introduce a procedural semantics for generating global decisions as defined in

```
DEFINE ANALYSIS RULE ChangePriceOfSoftDrinksAR FOR a:Article

ON 20pl:TwentyDaysPastLaunch BIND 20pl.levelId TO a
IF (a.category = 'Soft Drinks')

USING CUBES

  QuantitiesAllLocationsAllTimes
    (product P, location L, time T, SUM(quantity) TotalQty) PRIMARY P AS
    ROLLUP L TO ALL-Locations, T TO ALL-Times
    FROM Sales;

  QuantitiesAllLocationsDays
    DRILLDOWN T TO Day
    FROM QuantitiesAllLocationsAllTimes;

BEGIN

  ANALYZE QuantSalesAllLocationsAllTimes
     TRIGGER ACTION Definite-IF TotalQty < 10000
     DETAIL ANALYSIS IF TotalQty < 20000;

  ANALYZE QuantSalesAllLocationsDays [
       change PRIMARY P
              IS s1.P P, AVG(2nd10Days.TotalQty)/AVG(1st10Days.TotalQty) ratio
                 FOR CELLS 1st10Days SLICE RANK(T) <= 10 AND P = a.articleId,
                           2nd10Days SLICE RANK(T) > 10 AND P = a.articleId ]
     TRIGGER ACTION Definite-IF change.ratio <= 0.95
                    PARAMETERS ratio = change.ratio
     DISMISS ACTION Definite-IF FALSE;

TO EXECUTE (a@OLTP).changePrice
   PARAMETERS ratio = (COMMON_VAL ELSE 0.85) RESTRICT TO (0.95/0.85)

SUBJECT TO ((a@OLTP).currentPriceEquals PARAMETERS price = a.pricePerUnit)

END ChangePriceOfSoftDrinksAR
```

Figure 7.19: Modified Analysis Rule ChangePriceOfSoftDrinksAR

Section 7.3 (i.e., integrating global decisions of sub-cubes and merging these decisions with the local decisions of a cube).

The procedural evaluation of local and global decision making is split up in a top-down manner into the following sub-tasks:

1. Generate the global decisions of an analysis rule. In this step, (i) the analysis scope of some analysis rule is determined, (ii) decision making is initiated, and (iii) parameter

Syntax

TriggerActionCondDef	::=	TRIGGER ACTION DefiniteCondDef [TentativeCondDef] [ParameterDef].
ParameterDef	::=	PARAMETERS parameterName "=" ParamValDef { "," parameterName "=" ParamValDef }.

Figure 7.20: Syntax to specify cube-specific parameter bindings

bindings are determined for the positive decisions of the analysis rule. Procedure ExecuteAnalysisRule (cf. Figure 7.21) represents the procedural semantics of this step.

2. Generate the global decisions of a particular cube $r(R)$ by merging the local decisions of $r(R)$ with the integrated global decisions of sub-cubes of $r(R)$. The declarative semantics of this step was defined by operator $\oplus$ in Section 7.3.1. The procedural semantics of this step is represented by procedure GenerateGlobalDecisions (cf. Figure 7.22 and Figure 7.23).

3. Generate the local decisions of a cube. The declarative semantics of generating local decisions using positive/negative, definite/tentative, and irresolute decision criteria was defined in Section 7.2. The procedural semantics of this step is represented by procedure GenerateLocalDecisions (cf. Figure 7.24).

4. Generate global decisions of sub-cubes of $r(R)$ and integrate these global decisions. The procedural semantics to generate global decisions of sub-cubes of $r(R)$ is represented by procedure GenerateGlobalDecisions. Since the semantics of integrating global decisions is typically closely related to a particular decision task, we provided an approach to specify alternative decision-making models ($\uplus$) with which an analyst may define the semantics for integration. Procedure EvaluateDecisionMakingModel (cf. Figure 7.25) represents the procedural semantics to integrate the global decisions of sub-cubes of $r(R)$.

5. Determine the parameter bindings (i.e., cube-specific and rule-covering) to execute the rule's action for the positive decisions of an analysis rule, as defined in Section 7.4. The procedural semantics of this step is represented by procedure DetermineParameterBindings (cf. Figure 7.26).

7.5.1 Analysis Rule Execution

Executing an analysis rule AR that respects the semantics of the extended approach of decision-making consists of the following steps:

1. Determine the analysis scope of AR (i.e., the set of level instances in the primary dimension for which the set of event occurrences E occurred).
2. Generate positive/negative, definite/tentative, and irresolute decisions as specified by the decision steps of AR using the decision-making models defined for the cubes of the analysis scope.
3. Determine the parameter bindings for the set of positively decided level instances of AR.

Figure 7.21 represents procedure ExecuteAnalysisRule, which we originally introduced in Section 5.4 to realize rule execution of our basic approach. We adopted this procedure to be used by the extended approach as follows:

- The analysis scope of analysis rule AR may be determined either by *global events* (i.e., calendar events and logical events, which did not occur for particular level instances) or by *local events* (i.e., relative temporal events and logical events, which occurred for particular level instances). For the case that a global event triggers rule AR (lines 1 and 2), the analysis scope is determined as the set of level instances in the rule's primary dimension level that satisfy the rule's primary condition. For the case that a local event triggers rule AR (lines 3 and 4), the analysis scope is determined as the set of level instances in the rule's primary dimension level (i) for which the event occurred and (ii) which satisfy the rule's primary condition.
- Since the global decisions of an analysis rule are derived from the global decisions of the "root" cube, procedure GenerateGlobalDecisions is invoked for the "root" cube (cf. line 7 in Figure 7.21), which also initiates decision making at the sub-cubes of the "root" cube. Note that the global decisions of an analysis rule are definite. Hence, any tentative decision of the "root" cube will be "upgraded" (cf. Equations 7.13 and 7.15) to a definite decision. This is realized by unifying the definite and the tentative positive global decisions of the "root" cube and by unifying the definite and the tentative negative global decisions of the "root" cube (cf. lines 8 and 10 in Figure 7.21). Irresolute decisions require further manual analysis (cf. line 9 in Figure 7.21) and are therefore neither considered as *positive* nor as *negative*.
- Parameter bindings that are needed to execute the rule's action in an OLTP system for the positively decided level instances (i.e., TrgA) are determined as the final step in processing an analysis rule (cf. line 12 in Figure 7.21).

7.5.2 Global Decisions of a Cube

The global decisions of some cube r taken from the analysis graph of AR are generated by merging the *local decisions* of r with the *integrated global decisions* of sub-cubes of r.

```
ExecuteAnalysisRule (↓AR : Rule, ↓E : SetOfEventInstances, ↑T : TXSpec,
                     ↑TrgA : SetOfLevelInstances, ↑B : ArrayOfArray,
                     ↑ManA : SetOfLevelInstances, ↑DsmA : SetOfLevelInstances)

input : analysis rule AR to be executed;
        set of event instances E that occurred at the same chronon,
          each triggering rule AR;

output : transaction T to be executed in an OLTP system;
         set of objects TrgA for which T must be executed;
         2 dimensional array B holding the final parameter bindings to execute T,
          indexed (i) by the level instances in TrgA and (ii) by the parameters
          that are needed to execute T for the level instances in TrgA;
         set of objects ManA for which further manual analyzes should be carried out;
         set of objects DsmA for which T should not be executed;

Declarations

AS : SetOfLevelInstances  := {}        /* analysis scope */
root : Cube
posCubeDecs : ArrayOfSetOfLevelInstances /* array (indexed by the cube names of AR)
                         representing the objects positively decided by each cube
                         (needed to determine cube − specific parameter bindings) */
TrgA^rd : SetOfLevelInstances := {} /* def. trigger action set of the "root" cube; */
TrgA^rt : SetOfLevelInstances := {} /* tent. trigger action set of the "root" cube; */
DtlA^r : SetOfLevelInstances := {} /* detail analysis set of the "root" cube; */
DsmA^rt : SetOfLevelInstances := {} /* tent. dismiss action set of the "root" cube; */
DsmA^rd : SetOfLevelInstances := {} /* def. dismiss action set of the "root" cube; */

Begin

  #0 : /* initialize posCubeDecs to {} for each cube of the analysis graph */

  /* Determine the analysis scope of AR */
  #1 : if (E contains "global events") then
  #2 :   AS := {l ∈ primary dimension level of AR | prim_cond(AR, l)}
  #3 : elsif (E contains "local events") then
  #4 :   AS := {l | (∃ e ∈ E : occurredFor(e) = l) ∧ prim_cond(AR,l)}
  #5 : end if

  /* Generate positive, negative, and irresolute decisions */
  #6 : r := "root cube" of the analysis graph of AR
  #7 : GenerateGlobalDecisions (root, AS, posCubeDecs,
                                TrgA^rd, TrgA^rt, DtlA^r, DsmA^rt, DsmA^rd);
  #8 : TrgA := TrgA^rd ∪ TrgA^rt;
  #9 : ManA := DtlA^r;
  #10 : DsmA := DsmA^rd ∪ DsmA^rt;

  /* Determine the parameter bindings for TrgA */
  #11 : T := "action" of AR
  #12 : DetermineParameterBindings(AR, TrgA, posCubeDecs, B);

End
```

Figure 7.21: Procedural semantics of analysis rule execution

Hence, procedure GenerateGlobalDecisions (cf. Figure 7.22 and Figure 7.23) is split up into three steps to realize this task:

1. Generate the local decisions of cube r by invoking procedure EvaluateDecisionCriteria in line 1. This procedure is explained later.

2. Generate the global decisions of sub-cubes of r by invoking procedure GenerateGlobalDecisions recursively for each sub-cube d. To respect the semantics of decision making as defined in Section 7.3, the analysis scope of each sub-cube d (i.e., subAS) consists of the level instances in the analysis scope of r reduced by the definite local decisions of r (cf. line 3 in Figure 7.23). Note that subAS could be also defined alternatively upon the union of local tentative and local irresolute decisions of r (i.e., $\mathsf{TrgA}^t \cup \mathsf{DtlA} \cup \mathsf{DsmA}^t$). The global decisions of some sub-cube d are collected in variable globalDecisions, which is a 2-dimensional array that is indexed by the names of the cubes and the names of global decisions. A particular cell with index $[r, d]$ of this array holds the set of objects that represent the global decision d of cube r. After the global decisions of all sub-cubes of r have been collected in variable globalDecisions, these decisions are integrated (i.e., $\uplus$) by invoking procedure EvaluateDecisionMakingModel for these decisions using the decision-making model associated with r (represented by variable decTable). Procedure EvaluateDecisionMakingModel is explained in more detail later.

3. Merge (i.e., $\oplus$) the local decisions of r with the integrated global decisions of sub-cubes of r as defined in Section 7.3.1. Lines 14 - 18 represent Equations 7.16, 7.17, 7.18, 7.19, and 7.20, which define the semantics of merging local decisions with integrated global decisions (i.e., operator $\oplus$ in Section 7.3.1). Note that if r is a "leaf" cube, procedure IntegrateGlobalSubDecisions will generate empty sets of integrated decisions for the sub-cubes of r. In such a situation, lines 14 - 18 copy the members of local decision sets (i.e., TrgA^d, TrgA^t, DtlA, DsmA^t, DsmA^d) to the corresponding global decision sets (i.e., TrgA^{gd}, TrgA^{gt}, DtlA^g, DsmA^{gt}, DsmA^{gd}), which respects the semantics of Equation 7.12.

7.5.3 Local Decisions of a Cube

Evaluating the decision criteria of some cube $r(R)$ to generate the *local decisions* of $r(R)$ consists of three steps:

1. Determine the definite decisions of $r(R)$ for the level instances in the analysis scope of $r(R)$ by evaluating the *definite trigger action condition* and the *definite dismiss action condition*. A level instance l belongs to one of these two definite decision sets, if l satisfies one of the two conditions but not both.

GenerateGlobalDecisions(↓ r : Cube, ↓ AS : SetOfLevelInstances,
↕ posCubeDecs : ArrayOfSetOfLevelInstances,
↑ TrgA^{gd} : SetOfLevelInstances, ↑ TrgA^{gt} : SetOfLevelInstances,
↑ DtlA^{g} : SetOfLevelInstances, ↑ DsmA^{gt} : SetOfLevelInstances,
↑ DsmA^{gd} : SetOfLevelInstances)

input : *cube* r *whose global decisions should be generated*;
set AS *represents the analysis scope of* r;
array posCubeDecs *represents the positive (tentative and definite) decisions for each cube of the analysis graph (indexed by cube names); this array will be updated with the positive decisions of* r;

output : TrgA^{gd}, TrgA^{gt}, DsmA^{gd}, DsmA^{gt}, DtlA^{g} *represent the global decisions of* r;

Declarations

subAS : SetOfLevelInstances /* *analysis scope of sub − cubes*; */
TrgA^{d} : SetOfLevelInstances := {} /* *local def. trigger act. set of cube* r; */
TrgA^{t} : SetOfLevelInstances := {} /* *local tent. trigger act. set of cube* r; */
DtlA : SetOfLevelInstances := {} /* *local detail analysis set of cube* r; */
DsmA^{t} : SetOfLevelInstances := {} /* *local tent. dismiss act. set of cube* r; */
DsmA^{d} : SetOfLevelInstances := {} /* *local def. dismiss act. set of cube* r; */

TrgA^{id} : SetOfLevelInstances := {} /* *integr. def. trigger act. set of sub−cubes*; */
TrgA^{it} : SetOfLevelInstances := {} /* *integr. tent. trigger act. set of sub−cubes*; */
DtlA^{i} : SetOfLevelInstances := {} /* *integr. detail analysis set of sub−cubes*; */
DsmA^{it} : SetOfLevelInstances := {} /* *integr. tent. dismiss act. set of sub−cubes*; */
DsmA^{id} : SetOfLevelInstances := {} /* *integr. def. dismiss act. set of sub−cubes*; */

globalDecisions : ArrayOfArrayOfSet /* *to hold the global decisions of cubes; first index identifies a particular cube, second index identifies a particular set of decisions* */
decTable : DecisionTable /* *n − dimensional decision table to integrate the global decisions of sub − cubes of* r */

Figure 7.22: Declaration of procedure GenerateGlobalDecisions

```
Begin

  /* Generate local decisions of r */
  #1 : EvaluateDecisionCriteria (r, AS, posCubeDecs,
                                 TrgA^d, TrgA^t, DtlA, DsmA^t, DsmA^d);

  /* Generate global decisions of sub-cubes of r and integrate these decisions */
  #2 : if r has sub-cubes then
  #3 :   subAS := AS \ (TrgA^d ∪ DsmA^d);
  #4 :   foreach d ∈ {direct descendants of r} do
  #5 :     GenerateGlobalDecisions (d, AS, posCubeDecs,
                                    TrgA^gd, TrgA^gt, DtlA^g, DsmA^gt, DsmA^gd);
  #6 :     globalDecisions[d,"TrgA^gd"] := TrgA^gd;
  #7 :     globalDecisions[d,"TrgA^gt"] := TrgA^gt;
  #8 :     globalDecisions[d,"DtlA^g"] := DtlA^gd;
  #9 :     globalDecisions[d,"DsmA^gt"] := DsmA^gt;
  #10 :    globalDecisions[d,"DsmA^gd"] := DsmA^gd;
  #11 :  end foreach
  #12 :  decTable := decision table associated with cube r;
  #13 :  EvaluateDecisionMakingModel (decTable, globalDecisions, TrgA^id, TrgA^it,
                                      DtlA^i, DsmA^it, DsmA^id);
  #14 : end if

  /* Merge local decisions of r with the integrated global decisions of sub-cubes */
  #14 : TrgA^gd := TrgA^d ∪ TrgA^id;
  #15 : DsmA^gd := DsmA^d ∪ DsmA^id;
  #16 : TrgA^gt := (TrgA^t ∩ DtlA^i) ∪ TrgA^it;
  #17 : DsmA^gt := (DsmA^t ∩ DtlA^i) ∪ DsmA^it;
  #18 : DtlA^g := subAS \ (TrgA^gd ∪ DsmA^gd ∪ TrgA^gt ∪ DsmA^gt);

End
```

Figure 7.23: Body of procedure GenerateGlobalDecisions

2. Determine the tentative decisions of $r(R)$ for the level instances for which no definite decision was generated by evaluating the *tentative trigger action condition* and the *tentative dismiss action condition*. A level instance l belongs to one of these two tentative decision sets, if l satisfies one of the two conditions but not both. In any other case, l is decided *irresolute* (i.e., continue analysis for l).

3. Register the positive (tentative/definite) decisions of $r(R)$ in order to determine cube-specific parameter bindings in a later step. The rationale to determine parameter bindings in a later step is as follows: For efficiency of evaluating queries in the data warehouse (i.e., dynamic parameter bindings), bindings are not determined for every positive local decision, but only for the set of positive global decisions of the analysis rule AR, which is typically as subset of the positive local decisions of the cubes of AR. Hence, less queries are evaluated which yields higher performance of decision making in active data warehouses.

Figure 7.24 presents procedure EvaluateDecisionCriteria, which evaluates the decision criteria of some cube $r(R)$ for the set of objects in the analysis scope (denoted as AS) of $r(R)$. In the first part of this procedure, definite decision criteria are evaluated for the level instances in the cube's analysis scope:

1. Functions triggerAction^{d+}(r, l) and dismissAction^{d+}(r, l) determine for a given level instance l of the primary dimension level of $r(R)$ whether l satisfies the *definite trigger action condition* (line 1) or whether l satisfies the *definite dismiss action condition* (line 2), respectively. Function triggerAction^{d+}(r, l) evaluates the completed trigger action condition of cube $r(R)$ (i.e., $trgA^{d+}_{r(R)}$) as defined by Equation 7.1 (cf. Section 7.2.1). Function dismissAction^{d+}(r, l) evaluates the completed dismiss action condition of cube $r(R)$ (i.e., $dsmA^{d+}_{r(R)}$) as defined by Equation 7.2 (cf. Section 7.2.1). Note that the completed detail analysis condition of $r(R)$ is not evaluated since it will be determined as the complement of the completed trigger action condition and the completed dismiss action condition.

2. Since conditions that represent definite decision criteria may overlap, we need to determine the set of objects for which no definite decision can be made (denoted as DtlA^d), cf. line 3). This set is determined as (i) the set of objects for which both the *definite trigger action condition* and the *definite dismiss action condition* are satisfied or (ii) as the set of objects that satisfy the *detail analysis condition* but that do not satisfy the definite decision criteria. Note that overlappings between definite and tentative decision criteria are not treated here since definite decisions overrule tentative decisions.

3. The *definite trigger action set* (denoted as TrgA^d, cf. line 4) is determined as the set of objects that satisfy the *definite trigger action condition* but for which no conflicts exist.

4. The *definite dismiss action set* (denoted as DsmA^d, cf. line 5) is determined as the set of objects that satisfy the definite dismiss action condition but for which no conflicts exist.

In the second part of procedure EvaluateDecisionCriteria (i.e., lines 6 - 10), tentative decision criteria are evaluated using the set of objects for which no definite decision was made (denoted as DtlA^d) as the reduced analysis scope.

1. Functions $\mathsf{triggerAction}^{t+}(\mathsf{r}, \mathsf{l})$ and $\mathsf{dismissAction}^{t+}(\mathsf{r}, \mathsf{l})$ determine for a given level instance l of the primary dimension level of $r(R)$ whether l satisfies the *tentative trigger action condition* (line 6) and whether l satisfies the *tentative dismiss action condition* (line 7), respectively. Note that these functions are evaluated only for those level instances for which no definite decision could be generated (i.e., level instances in DtlA^d). Function $\mathsf{triggerAction}^{t+}(\mathsf{r}, \mathsf{l})$ evaluates the completed tentative trigger action condition of cube $r(R)$ (i.e., $trgA^{t+}_{r(R)}$) as defined by Equation 7.7 (cf. Section 7.2.2). Function $\mathsf{dismissAction}^{t+}(\mathsf{r}, \mathsf{l})$ evaluates the completed tentative dismiss action condition of cube $r(R)$ (i.e., $dsmA^{t+}_{r(R)}$) as defined by Equation 7.8 (cf. Section 7.2.2).

2. The set of irresolute decided objects (i.e., the *final detail analysis set*, denoted as DtlA, cf. line 8 and Equation 7.9 in Section 7.2.2) is determined as (i) the set of objects that satisfy the *tentative trigger action condition* and the *tentative dismiss action condition* (i.e., overlappings between tentative decision criteria) or (ii) the set of objects for which no tentative decision criterion holds (i.e., those objects that implicitly satisfy the *detail analysis condition* but no other decision criterion).

3. The *tentative trigger action set* (denoted as TrgA^t, cf. line 9) is determined as the set of objects that satisfy the *tentative trigger action condition* but for which no conflicts exist (cf. Equation 7.10 in Section 7.2.2).

4. The *tentative dismiss action set* (denoted as DsmA^t, cf. line 10) is determined as the set of objects that satisfy the *tentative dismiss action condition* but for which no conflicts exist (cf. Equation 7.11 in Section 7.2.2).

The third part of procedure EvaluateDecisionCriteria "registers" the local positive decisions (i.e., $\mathsf{TrgA}^d \cup \mathsf{TrgA}^t$ in line 11), for which cube specific parameter bindings should be determined in a later step (see Section 7.5.5). For this purpose, we use array posCubeDecs (indexed by the cube names of the analysis graph) that holds the positive decisions of each cube.

Note that procedure EvaluateDecisionCriteria generates only local decisions while procedure EvaluateDecisionStep of our basic approach (cf. Figure 5.15) also realizes global decision making by integrating the global decisions of sub-cubes of $r(R)$. Variables TrgA, DtlA, and ManA in procedure EvaluateDecisionStep represent *final decisions*, whereas variables TrgA^d, TrgA^t, DsmA^d, DsmA^t, and DtlA in procedure EvaluateDecisionCriteria represent the *local decisions* of some cube r.

EvaluateDecisionCriteria(↓ r : Cube, ↓ AS : SetOfLevelInstances,
↕ posCubeDecs : ArrayOfSetOfLevelInstances,
↑ TrgA^{d} : SetOfLevelInstances, ↑ TrgA^{t} : SetOfLevelInstances,
↑ DtlA : SetOfLevelInstances, ↑ DsmA^{d} : SetOfLevelInstances,
↑ DsmA^{t} : SetOfLevelInstances)

input : *cube* r *whose decision step is evaluated*;
set of objects AS *which need to be analyzed by the decision step of* r;
array posCubeDecs *represents the positive (tentative and definite) decisions for each cube of the analysis graph (indexed by cube names); this array will be updated with the positive decisions of* r;

output : *set of objects* TrgA^{d} *that satisfy the definite trigger action condition of* r;
set of objects TrgA^{t} *that satisfy the tentative trigger action condition of* r;
set of objects DtlA *that should be analyzed in further detail*;
set of objects DsmA^{d} *that satisfy the definite dismiss action condition of* r;
set of objects DsmA^{t} *that satisfy the tentative dismiss action condition of* r;

Declarations

DtlA^{d} : SetOfLevelInstances /* *objects for which no def. decision was made*; */
$\text{trgA}^{\text{d+}}$: SetOfLevelInstances /* *objects satisfy the def. trigger action cond. of* r; */
$\text{trgA}^{\text{t+}}$: SetOfLevelInstances /* *objects satisfy the tent. trigger act. cond. of* r; */
$\text{dsmA}^{\text{d+}}$: SetOfLevelInstances /* *objects satisfy the def. dismiss act. cond. of* r; */
$\text{dsmA}^{\text{t+}}$: SetOfLevelInstances /* *objects satisfy the tent. dismiss act. cond. of* r; */
dtlA^{+} : SetOfLevelInstances /* *objects satisfy the detail anal. cond. of* r; */

Begin

/* *Evaluate definite decision criteria of* r */
#1 : $\text{trgA}^{\text{d+}} := \{ \text{l} \in \text{AS} \mid \text{triggerAction}^{\text{d+}}(\text{r}, \text{l}) \}$;
#2 : $\text{dsmA}^{\text{d+}} := \{ \text{l} \in \text{AS} \mid \text{dismissAction}^{\text{d+}}(\text{r}, \text{l}) \}$;
#3 : $\text{DtlA}^{\text{d}} := ((\text{trgA}^{\text{d+}} \cap \text{dsmA}^{\text{d+}}) \cup (\text{AS} \setminus (\text{trgA}^{\text{d+}} \cup \text{dsmA}^{\text{d+}}))$;
#4 : $\text{TrgA}^{\text{d}} := \text{trgA}^{\text{d+}} \setminus \text{DtlA}^{\text{d}}$;
#5 : $\text{DsmA}^{\text{d}} := \text{dsmA}^{\text{d+}} \setminus \text{DtlA}^{\text{d}}$;

/* *Evaluate tentative decision criteria of* r */
#6 : $\text{trgA}^{\text{t+}} := \{ \text{l} \in \text{DtlA}^{\text{d}} \mid \text{triggerAction}^{\text{t+}}(\text{r}, \text{l}) \}$;
#7 : $\text{dsmA}^{\text{t+}} := \{ \text{l} \in \text{DtlA}^{\text{d}} \mid \text{dismissAction}^{\text{t+}}(\text{r}, \text{l}) \}$;
#8 : $\text{DtlA} := \text{DtlA}^{\text{d}} \setminus (\text{trgA}^{\text{t+}} \cup \text{dsmA}^{\text{t+}})$;
#9 : $\text{TrgA}^{\text{t}} := \text{DtlA}^{\text{d}} \cap \text{trgA}^{\text{t+}}$;
#10 : $\text{DsmA}^{\text{t}} := \text{DtlA}^{\text{d}} \cap \text{dsmA}^{\text{t+}}$;

/* *Needed to determine cube – specific parameter bindings in a later step* */
#11 : $\text{posCubeDecs}[\text{r}] := \text{TrgA}^{\text{d}} \cup \text{TrgA}^{\text{t}}$;

End

Figure 7.24: Procedural semantics of decision criteria evaluation

EvaluateDecisionMakingModel(↓ decTable : DecisionTable,
↓ globalDecs : ArrayOfArrayOfSetOfLevelInstances,
↑ TrgA^{id} : SetOfLevelInstances, ↑ TrgA^{it} : SetOfLevelInstances,
↑ DtlA^{i} : SetOfLevelInstances, ↑ DsmA^{it} : SetOfLevelInstances,
↑ DsmA^{id} : SetOfLevelInstances)

input : *decision table* decTable *represents the decision – making model*;
array globalDecs *represents the global decisions that need to be integrated*;

output : TrgA^{id}, TrgA^{it}, DsmA^{id}, DsmA^{it}, DtlA^{i} *represent the integrated decisions*;

Declarations

objects : SetOfLevelInstances /* *set of objects in the primary dimension level*; */

Begin

#1 : **foreach** entry ∈ entries(decTable) **do**
#2 : objects := globalDecs[cube_1, entry.d_1] ∩ ... ∩ globalDecs[cube_n, entry.d_n]
#3 : **case** entry.integrDec **of**
#4 : "TrgA^{id}" : TrgA^{id} := TrgA^{id} ∪ objects
#5 : "TrgA^{it}" : TrgA^{it} := TrgA^{it} ∪ objects
#6 : "DtlA^{i}" : DtlA^{i} := DtlA^{i} ∪ objects
#7 : "DsmA^{it}" : DsmA^{it} := DsmA^{it} ∪ objects
#8 : "DsmA^{id}" : DsmA^{id} := DsmA^{id} ∪ objects
#9 : **end case**
#10 : **end foreach**

End

Figure 7.25: Procedural semantics for evaluating a decision-making model

7.5.4 Integrating Global Decisions

Integrating the global decisions that have been generated by the sub-cubes of cube $r(R)$ is based upon the following principle:

- Evaluate the decision-making model associated with $r(R)$ using the global decisions of sub-cubes of $r(R)$ as input. A decision-making model is represented by a n-dimensional decision table, whereby each dimension represents one sub-cubes and the values of these dimensions represent the possible global decisions of these sub-cubes. Note that although anonymous decision-making models may be specified using a 2-dimensional decision-table (if the table is *symmetrical*, cf. Section 7.3), we use n-dimensional decision-tables for evaluation.

While generating the global decisions of sub-cubes of r is already realized by procedure GenerateGlobalDecisions, integrating these global decisions is realized by procedure EvaluateDecisionMakingModel (cf. Figure 7.25), which evaluates a particular decision-making model (represented by the n-dimensional decision table decTable) for the sub-cubes of r. The global decisions of sub-cubes that need to be integrated are represented by the 2-dimensional array gobalDecs. Function entries(decTable) retrieves the set of entries that represent the integrated decisions of decision table decTable. One such entry is identified by the values of this decision table's dimensions, i.e., entry.$d_i \in \{\text{TrgA}^{gd}$", TrgA^{gt}", "DtlA^{g}", "DsmA^{gt}", "DsmA^{gd}"$\}$ where d_i represents one dimension. The value of such a decision table entry (denoted as entry.integrDec) is a string that represents one of the 5 possible integrated decisions TrgA^{id}, TrgA^{it}, "DtlA^{i}, "DsmA^{it}, and "DsmA^{id}. Decision making is as follows: In line 2, the set of objects that represent a particular constellation of global decisions as required by the current decision table entry is determined as the intersection of the corresponding decision sets taken from the global decisions of each sub-cube. In line 3, the integrated decision for this entry is determined. Finally, in lines 4 - 8 the set of objects that represent this particular decision constellation will be added to the integrated decisions sets that correspond with the integrated decision of the decision table entry.

Example: Assume that a particular entry of a 3-dimensional decision table (the three dimensions represent the global decisions of three sub-cubes, respectively) is described by the dimension values "TrgA^{gd}", "TrgA^{gd}", "TrgA^{gd}" and provides "TrgA^{id}" as the integrated decision for such a constellation. This means that if the three sub-cubes agree upon their global decision "TrgA^{gd}" for some level instance l, then the integrated global decision is "TrgA^{id}" for this level instance.

7.5.5 Parameter Bindings

Determining parameter bindings is based upon the following principles:

```
DetermineParameterBindings(↓AR : Rule, ↓posRuleDecs : SetOfLevelInstances,
                           ↓posCubeDecs : ArrayOfSetOfLevelInstances,
                           ↑bindings : ArrayOfArrayOfBindings)

input : analysis rule AR;
        set posRuleDecs represents the final positive decisions of AR;
        array posCubeDecs represents the positive (tentative and definite) decisions
          for each cube of the analysis graph (indexed by cube names);

output : 2 dimensional array bindings represents the final bindings for each positively
           decided level instance/action parameter (indexed by level instance and
           parameter name);

Declarations

cubeBindings : ArrayOfArrayOfArray /* data structure to hold cube − specific bindings
                                      indexed by level instances, cubes, and
                                      parameters; */
parameters : Set /* represents pairs ⟨parameter name, function to derive a final binding
                    from possible alternative bindings⟩) to determine the bindings for
                    action execution */

Begin

  #0 : parameters := set of parameter/function pairs of AR;

       /* determine cube − specific bindings */
  #1 : foreach r ∈ set of cubes that define cube − specific bindings do
  #2 :   foreach l ∈ posCubeDecs[r] do
  #3 :     if l ∈ posRuleDecs then
  #4 :       foreach ⟨p, f⟩ ∈ parameters do
  #5 :         cubeBindings[l, r, p] := cubeBindingValue(l, r, p);
  #6 :       end foreach
  #7 :     else /* ignore level inst. l, no positive decision generated by rule AR */
  #8 :     end if
  #9 :   end foreach
  #10 : end foreach

       /* determine final bindings */
  #11 : foreach l ∈ posRuleDecs do
  #12 :   foreach ⟨p, f⟩ ∈ parameters do
  #13 :     bindings[l, p] := applyFunction(f, cubeBindings[l, _, p]);
  #14 :     if (bindings[l, p] IS NULL) OR (isInvalid(AR, p, bindings[l, p])) then
  #15 :       bindings[l, p] := ruleBindingValue(AR, p);
  #16 :       if isInvalid(AR, p, bindings[l, p]) then
  #17 :         bindings[l, p] := NULL;
  #18 :       end if
  #19 :     end if
  #20 :   end foreach
  #21 : end foreach

End
```

Figure 7.26: Procedural semantics to determine parameter bindings

1. Determine cube-specific parameter bindings first.

2. If multiple alternative cube-specific parameter bindings exist for the same level instance, apply function $f(ab) \rightarrow fb$ (e.g., COMMON_VAL, SUM, ...) to derive the final binding fb from the set of alternative bindings ab.

3. If there are no alternative bindings available or if the final binding fb violates the allowed parameter range, use the rule-covering parameter binding for action execution.

4. Let the analyst specify the binding if a rule-covering parameter binding also violates the allowed parameter range.

Figure 7.26 presents procedure DetermineParameterBindings, with which cube-specific and rule-covering bindings are determined for the action parameters of analysis rule AR. Parameter bindings are determined only for the positive (definite and tentative) decisions of an analysis rule, which are represented by input parameter posRuleDecs. To determine cube-specific parameter bindings, input parameter posCubeDecs provides for each cube of the analysis graph the set of positively decided level instances. The final bindings for executing the rule's action are returned by parameter bindings, which is a 2-dimensional array that is indexed (i) by the positively decided level instances and (ii) by the action's parameters. Determining parameter bindings is as follows:

- In a first step (lines 1 - 10), cube-specific parameter bindings are determined for the level instances that represent the positive decisions of the analysis rule. If such a level instance l belongs to the positive decisions of a cube that defines cube-specific parameter bindings, these bindings will be determined for l (cf. lines 3 - 6 in Figure 7.26). We use function cubeBindingValue(l, r, p) to determine for some level instance l the static or the dynamic binding of parameter p that was defined at cube r. If the parameter binding is static, this function retrieves a copy of the corresponding value; if the parameter binding is dynamic, this function evaluates the underlying query for l. Every cube-specific binding is stored in the 3-dimensional array cubeBindings, which is indexed by level instances, cubes, and parameters.

- In a second step (lines 11 - 21), the final bindings will be determined by deriving a single binding from alternative cube-specific bindings (line 13) using function applyFunction(f, cubeBindings[l, _, p]), whereby cubeBindings[l, _, p] identifies the alternative cube-specific bindings (note that the index identifying the cube is not specified; hence the bindings for level instance l of parameter p across all cubes will be taken) and f represents a particular function that derives the final binding from these values (i.e., COMMON_VAL, SUM, AVG, MIN, MAX or user-defined). If no cube-specific bindings exist for l or if the value determined by f violates the allowed parameter range of p (cf. line 14 - function isInvalid examines for some parameter p of rule AR whether the parameter range is violated by binding bindings[l, p]), the rule-covering default

binding of p is determined instead using function ruleBindingValue(AR, p) (cf. line 15). If the default binding also violates the allowed parameter range, the correct binding must be determined manually before the rule's action can be executed for l. Hence, bindings[l, p] is set to NULL in line 17.

7.6 Summary

In this chapter, we provided several extensions to analysis rules with which an analyst may specify very complex analysis rules that meet the needs of real-world decision making. These extensions are:

1. *Logical Events.* Logical events offer the possibility to fire an analysis rule when the database arrives at a particular state (described by logical and temporal predicates of a query).

2. *Decision Criteria.* We extended our basic approach of modeling decision criteria (i.e., positive and irresolute) with the possibility to specify positive, negative, and irresolute decision criteria and with the possibility to define default situations in which positive and negative decision criteria apply (i.e., tentative decision criteria).

3. *Decision-Making Models.* We introduced an approach to specify alternative semantics for integrating the global decisions generated by the cubes of an analysis graph (i.e., alternative "decision-making models"). Decision-making models are employed to define the semantics for integrating the global decisions of the sub-cubes of a single cube and define the semantics for integrating the global decisions of the sub-cubes of all cubes of an analysis graph.

4. *Decision Parameters.* We introduced an approach to specify the bindings of decision parameters flexibly by providing parameter bindings uniformly once for all positive decisions of an analysis rule (i.e., rule-covering parameter bindings) or for the positive decisions of each cube of the analysis graph separately (i.e., cube-specific parameter bindings). While rule-covering parameter bindings represent default bindings that will be used unless cube-specific parameter bindings exist. These parameter bindings may be either determined statically (i.e., using constant values) or dynamically (i.e., as the result of executing a query).

5. *Procedural Semantics.* We described a procedural semantics to generate decisions respecting the extended approach.

Part IV

Putting Active Data Warehouses into Work

Chapter 8

Passive Warehouse Data

Contents

Since many data warehouses are built upon relational database technology, this section explains how the specifications of multidimensional data, as introduced in Chapter 4, are translated to the relational data model using the *Structured Query Language* (SQL) [SQL99] as data definition language. Section 8.1 shows how dimension levels and base cubes are mapped to a relational representation that supports efficient processing of queries. Section 8.2 shows how incrementally specified cubes and analytical expressions are mapped to view relations and queries in SQL. Finally, Section 8.2.3 discusses related work on selecting and maintaining materialized views that may be used to speed up SQL queries.

8.1 Multidimensional Data Structures

Multidimensional data as described by base cubes and dimension levels are realized in the relational data model as relations, which are called *fact tables* and *dimension tables* [KS95], respectively. The relation schema of a relation representing a fact table consists of (1) a foreign key attribute referencing a dimension table for each dimension attribute that represents the corresponding dimension level and (2) an attribute for each measure. All foreign key attributes together constitute the compound primary key of the fact table. Dimension tables can be implemented using one of three alternative implementation approaches – *star schema*, *snowflake schema*, or *starflake schema*. The three approaches have in common

that dimension levels are mapped to relation schemas but they differ in the way in which dimension tables are arranged around a fact table and in the degree of normalization of dimension tables.

Star Schema. A *star schema* [Inm96, Sys95] is the simplest approach to implement multidimensional data upon relational database technology. Using this approach, all level schemas that belong to the hierarchies of a particular dimension are mapped to a single relation schema – the *star table*, which is fully de-normalized and thus causes redundancies among the instances of represented dimension levels. The term *star* characterizes the topology of the schema, in which the dimension tables "surround" the fact table like the tines of a star. Figure 8.1 (a) depicts this topology. The shaded box in the center represents the fact table, while the white boxes labeled D_1, D_2, and D_3 represent dimension tables. Further, each dimension table represents several dimension levels L_{ij}, which are depicted within the white boxes. Arrows represent foreign key attributes. Mapping the level schemas to the relation schema of a star table is as follows: The identifying attributes of the level schemas are represented as relational attributes which constitute the compound primary key of the star table; rollup attributes are not mapped to attributes of the star table but are added to the data warehouse's meta-data repository instead[1]; describing attributes are mapped to non-key attributes of the star table. The rationale to use de-normalized star tables is twofold:

1. The number of tuples in fact tables is continually growing while the number of tuples in star tables typically remains constant or changes only slowly. Further, updates and deletes from star tables occur only infrequently such that only little efforts are needed to maintain the redundancies imposed by star tables.

2. To roll measures from fact tables up to a particular dimension level, only a single join (with the star table that represents this dimension level) is needed. This speeds up OLAP queries significantly.

Snowflake Schema. A *snowflake schema* [CBS99] avoids the redundancies imposed by a star schema since for each level schema a separate relation schema – called *snowflake table* – is defined. The attributes of a level schema are mapped to attributes of the corresponding snowflake table as follows: The identifying attribute is mapped to a primary key attribute; rollup attributes are mapped to foreign key attributes referencing the snowflake tables that represent the corresponding superordinate dimension levels; describing attributes are mapped to non-key attributes. Figure 8.1 (b) depicts the topology of a snowflake schema.

[1] This meta-data repository consists of several tables that describe dimension paths, i.e., which attributes are used to establish the arcs that constitute dimension paths in the multidimensional data model in order to establish a mapping between the multidimensional data model and the corresponding relational implementation.

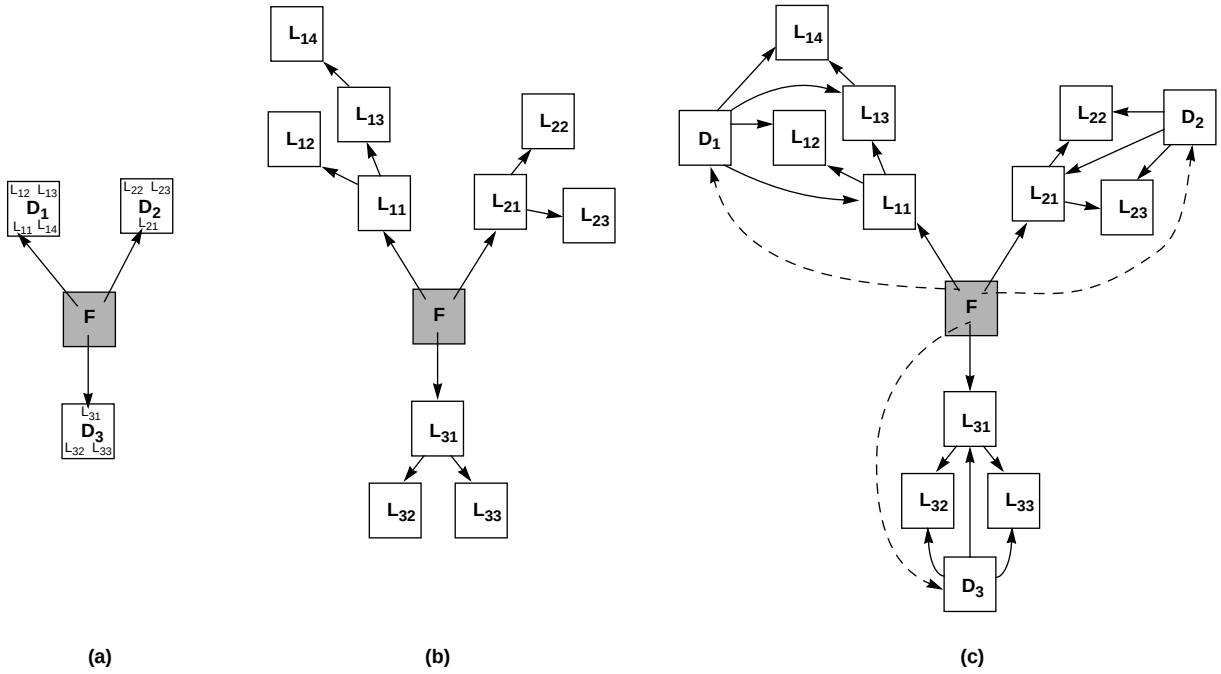

Figure 8.1: Topology of (a) star schema, (b) snowflake schema, and (c) starflake schema

To rollup measure values to a particular dimension level L_{ij}, the fact table must be joined with all snowflake tables that represent subordinate dimension levels of L_{ij} along one dimension path. E.g., to rollup a measure attribute from fact table F to dimension level L_{14}, F must be joined with L_{11}, then with L_{13}, and finally with L_{14}. Since joining tables will slow down OLAP queries significantly, pure snowflake schemas are not used in practice to represent multidimensional data.

Starflake Schema. A *starflake schema* [AM97], which is our preferred implementation data model, combines star and snowflake schemas as follows: Each level schema is mapped to a separate relation schema as described above for snowflake schemas (i.e., snowflake tables). But since joining tables for rolling up measures is inefficient, an additional star table is provided whose relation schema consists of the primary key attributes of that dimension's snowflake tables. This star table represents a "pre-join" among the snowflake tables. Rollup attributes and describing attributes are available only in the snowflake tables. Figure 8.1 (c) depicts the topology of the starflake schema. Dashed arrows indicate that fact table F can alternatively be joined with any of the dimension's star tables. The distinction between star tables and snowflake tables offers the following advantages:

1. *Rollup queries* can be executed efficiently using the denormalized star tables.

Dimension	Relational Representation	Star	Snowfl.
Product	**Article**(articleId, *category*, producer, name, pricePerUnit)		×
	Category(name, descr)		×
	Producer(name)		×
	Product(*articleId, categoryName*)	×	
Location	**Store**(storeId, *city*, owner)		×
	City(cityId, *region*, name, zip, noOfInhab)		×
	Region(name, size, noOfInhab)		×
	Location(*storeId, cityId, regionName*)	×	
Time	**Day**(date, *month, week*, weekday)		×
	Month(monthId, *quarter*, name)		×
	Quarter(qtrId, *year*)		×
	Week(weekId, *year*, noInYear)		×
	Year(year)		×
	Time(*date, monthId, quarterId, weekId, year*)	×	
Base Cube	**Sales**(*product, location, time*, sales, quantity)		
	Offerings(*product, location, timeFrom, timeTo*)		

Figure 8.2: Representation of multidimensional data using the starflake schema

2. Redundancies in storing data are reduced by using the normalized snowflake tables. Further, snowflake tables support processing of *slice queries*.

E.g., to process a typical OLAP query, which rolls up measure values from F to dimension level L_{14} if the corresponding measure values satisfy the slice condition $L_{14}.I = X$, D_1 will be first joined with $\sigma_{I=X}(L_{14})$ and the result will be afterwards joined with F. This technique will be used in the next section to map incrementally specified cubes to SQL view relations.

Example: Figure 8.2 presents the relation schemas that represent our retail dimensional model using the starflake schema. Primary key attributes are depicted underlined, foreign key attributes are depicted in italics. Figure 8.2 further indicates whether a relation schema represents a *star* table or whether a relation schema represents a *snowflake* table.

Note that in the following we will denote a relation representing the *star table* of dimension dim_i as star_{dim_i} and a relation representing the *snowflake table* of dimension level L_i as snow_{L_i}.

8.2 Multidimensional Analyzes

This section describes how incrementally specified cubes and analytical expressions are mapped to SQL view relations and to SQL queries using the tables of the starflake schema, which we explained above.

8.2.1 Mapping Cubes

Incrementally specified cubes are mapped to SQL view relations. A cube's extension consists of those cells, to which cells from the base cube can roll up. Although cubes are specified incrementally by applying OLAP operators *Rollup*, *Drilldown*, *Slice*, or *Intersection* on existing cubes, the cells of these cubes are created by rolling up the cells of the base cube. Correspondingly, the tuples of the view relation that represents the cube's extension will be created bottom up from the fact table that represents the cube's base cube using clause GROUP BY. Note that an SQL view relation that realizes a particular cube is derived from the cube's definition, not the textual specification of that cube. Hence, the conceptual operators ROLLUP, DRILLDOWN, SLICE, and INTERSECTION are not relevant here.

Figure 8.3 presents the definition of some cube $r(R)$ and the skeleton of a generic SQL view relation that represents $r(R)$. The different parts of this SQL view relation are determined as follows:

1. The view relation's schema is defined by the cube's name followed by the names of dimension attributes (denoted by D_i) and the names of measure attributes (denoted by M_i) taken from the cube schema R.

2. The SELECT-clause consists of a list of attributes one for each dimension attribute and each measure attribute from the cube schema. Dimension attributes are projections of star tables that represent the dimension levels of a particular dimension star_{dim_i}. One projected attribute represents a particular dimension level, which is denoted by $\mathsf{star}_{dim_i}.\mathsf{L}_{D_i}$. The values of a measure attribute M_i are determined by aggregation function f_i taken from the cube's aggregation functions AGG. The number and order of these projections must conform with the number and order of attribute names in the view relation's schema.

3. The FROM-clause consists of the *fact table* that represents the cube's base cube $\mathsf{b(B)}$, *star tables* representing dimensions dim_i $(i \in (1..m))$ (these star tables are denoted as star_{dim_i}), and *snowflake tables* representing dimension levels L_{D_i} $(i \in (1..n))$ that are needed to evaluate the cube's slice conditions cond_i (these snowflake tables are denoted as $\mathsf{snow}^{cond_i}_{L_{D_i}}$).

(a) For each dimension attribute D_i of the base cube $\mathsf{b(B)}$ the corresponding star table star_{dim_i} – representing the identifiers of the dimension levels of dim_i and the dimension paths among these dimension levels – is needed,

i. if the domain of the cube's dimension attribute D_i represents more coarse grained level instances than provided by the base cube tuples from the fact table, or

ii. if a slice condition specified over dimension attribute D_i refers to a more coarse grained dimension level L_{D_i} than provided by the corresponding dimension attribute D_i of fact table $\mathsf{b(B)}$ (i.e., $\mathsf{b(B).D}_i < \mathsf{L}_{D_i}$).

(b) For each slice condition cond_i, which refers to some dimension level L_{D_i} of dimension attribute D_i, the corresponding snowflake table $\mathsf{snow}_{L_{D_i}}^{cond_i}$ is needed if cond_i refers to a rollup attribute or to a describing attribute of L_{D_i}. Note that if cond_i refers to the identifying attribute of L_{D_i}, evaluation of cond_i is realized using dimension attribute D_i of $\mathsf{b(B)}$ if $\mathsf{b(B).D}_i = \mathsf{L}_{D_i}$, or the dimension attribute representing the identifiers in L_{D_i} in the corresponding star table star_{dim_i} if $\mathsf{b(B).D}_i < \mathsf{L}_{D_i}$.

4. The WHERE-clause consists of three kinds of predicates:

(a) *Join predicates* between the fact table $\mathsf{b(B)}$ and the star tables star_{dim_i} to "rollup" the tuples of $\mathsf{b(B)}$.

(b) *Join predicates* between the star tables and their associated snowflake tables that are needed to evaluate slice conditions.

(c) *Selection predicates* that represent the slice conditions, which are taken from the cube's definition. If a slice condition refers to an identifying attribute of dimension level L_{Di}, not the primary key attribute I of snowflake table $\mathsf{snow}_{L_{D_i}}^{cond_i}$ is used, but – for efficiency of query evaluation – the corresponding foreign key attribute L_{D_i} of star table star_{dim_i} will be used instead.

5. The GROUP BY-clause is used to define the levels of granularities as specified in the cube schema by grouping the tuples created in the WHERE-clause and by aggregating their measure values in the FROM-clause. The GROUP BY-clause refers to foreign key attributes of star tables that represent the domains of corresponding dimension attributes in the cube schema. If the domain of a particular dimension attribute was specified to represent level instance $\mathsf{all\text{-}L}_{inf}$ of the top dimension level, the star table that represents this dimension attribute will be removed from the GROUP BY-clause and the corresponding projection in the SELECT-clause will be replaced with constant '$\mathsf{all\text{-}L}_{inf}$' (not shown in Figure 8.3).

Example: The upper part of Figure 8.4 presents the definition of cube SalesArticlesUpperAustriaQuarters2001. The lower part Figure 8.4 presents the view relation that

cube definition of $r(R)$

- *schema* $R = (D_1 : dom(D_1), ..., D_m : dom(D_m),\ M_1 : dom(M_1), ..., M_n : dom(M_n))$ *with* $dom(D_i)$ = *set of identifiers of dimension level* L_i
- *base cube* $b(B)$
- *conditions* C
- *aggregation functions* $AGG = \{\langle\ M_1,\ f_1\ \rangle,\ ...,\ \langle\ M_n,\ f_n\ \rangle\}$

```
CREATE VIEW  r(R) (D_1, ..., D_m, M_1, ..., M_n) AS

   SELECT    star_dim_1.L_D_1, ..., star_dim_m.L_D_k, f_1, ..., f_n

   FROM      b(B), star_dim_1, ..., star_dim_m, snow_L_D_1^cond_1, ..., snow_L_D_m^cond_m

   WHERE  /* 4a */  b(B).D_1 = star_dim_1.L_D_1 AND ... AND b(B).D_m = star_dim_m.L_D_m AND

          /* 4b */  star_dim_1.L_D_1 = snow_L_D_1^cond_1.I AND ... AND star_dim_m.L_D_m = snow_L_D_m^cond_m.I

          /* 4c */  AND cond_1 AND ... AND cond_m

   GROUP BY  star_dim_1.L_D_1, ..., star_dim_m.L_D_m
```

Figure 8.3: Mapping the definition of some cube $r(R)$ to an SQL view relation

implements this cube. Since the slice conditions of this cube are defined upon the identifying attributes of dimension levels Region and Year, star tables Location and Time can be used for evaluating these conditions (i.e., Location.region = 'Upper Austria' and Time.year = '2001' in place of Region.name = 'Upper Austria' and Year.year = 2001). Therefore, snowflake tables Region and Year need not be considered in the FROM-clause.

8.2.2 Mapping Cube Analysis Statements

Cube analysis statements analyze a particular cube using one or several analytical expressions. Since cube analysis statements define the context (i.e., the cells of the cube) of analytical expressions, only analytical expressions are mapped to SQL. The principal idea of this mapping is as follows: Analytical expressions are mapped to SQL queries in a similar way as cubes are mapped to SQL view relations. While the SQL view relation that represents a particular cube refers in its FROM-clause to the relation that represents the cube's base cube $b(B)$, the SQL query that represents a particular analytical expression

cube definition of SalesArticlesUpperAustriaQuarters2001

- *schema* R = (P : Article, L : Region, T : Quarter, SalesTotal : ATS)
- *base cube* Sales
- *conditions* C = {L : Region.name = 'Upper Austria', T : Year.year = '2001'}
- *aggregation functions* AGG = {⟨ SalesTotal, SUM(sales) ⟩}

```
CREATE VIEW SalesArticlesUpperAustriaQuarters2001 (P, L, T, SalesTotal) AS

   SELECT   Product.article, Location.region, Time.quarter, SUM(Sales.sales)

   FROM     Sales, Product, Location, Time, Year

   WHERE  /* 4a */ Sales.article = Product.article AND
                   Sales.location = Location.store AND
                   Sales.time = Time.date AND
          /* 4b */ Location.region = Region.name AND Time.year = Year.year AND
          /* 4c */ Region.name = 'Upper Austria' AND Year.year = '2001'

   GROUP BY  Product.article, Location.region, Time.quarter;
```

alternative WHERE–*clause* :

```
   WHERE  /* 4a */ Sales.article = Product.article AND
                   Sales.location = Location.store AND
                   Sales.time = Time.date AND
          /* 4c */ Location.region = 'Upper Austria' AND Time.year = '2001'
```

Figure 8.4: Definition of cube SalesArticlesUpperAustriaQuarters2001 and corresponding view relation

```
ANALYZE r(R) [
      f1 M1, ..., fn Mn IS
         SLICE cond1 CONJ ... CONJ condm
         ROLLUP D1 TO L_D1, ...,  Di TO L_Di ]
```

```
SELECT    star_dim1.L_D1 D1,...,star_dim1.L_Di Di, r(R).Dj, ...,r(R).Dm, f1 M1,...,fn Mn

FROM      r(R), star_dim1,...,star_dimm, snow^(L_D1^cond1),...,snow^(L_Dm^condm)

WHERE   /* 3a */ r(R).D1 = star_dim1.L_D1 AND ... AND r(R).Dm = star_dimm.L_Dm AND

        /* 3b */ star_dim1.L_D1^cond1 = snow^(L_D1^cond1).I AND ... AND star_dimm.L_Dm^condm = snow^(L_Dm^condm).I

        /* 3c */ AND cond1 CONJ ... CONJ condm

GROUP BY  star_dim1.L_D1,...,star_dim1.L_Di, r(R).Dj, ..., r(R).Dm, r(R).M1
```

Figure 8.5: SQL statement representing the mapping for simple analytical expressions

refers in its FROM-clause to the SQL view relation that represents the cube $r(R)$ for which the cube analysis statement was specified. If an analytical expression is used as a decision variable for specifying decision criteria in an analysis rule, the SQL query that represents the analytical expression is realized as an SQL view relation (discussed in Section 9.1.3).

Simple Analytical Expressions

A simple analytical expression, which refers only to the cells of a single cube without using cell identifiers, is mapped to a simple SELECT - FROM - WHERE - GROUP BY query. The upper part of Figure 8.5 presents the generic structure of a simple analytical expression; the lower part of Figure 8.5 presents the corresponding generic SQL query that represents such a simple analytical expression. The different parts of this SQL query are determined as follows:

1. The SELECT-clause refers to the cube's dimension attributes or – if ROLLUP was specified – to the attributes of star tables that are needed for ROLLUP. These projections represent the dimension attributes (denoted by D_i) of the virtual cells. Further, the SELECT-clause consists of the formulas (denoted by f_i) upon which the values of measure attributes (denoted by M_i) are defined.

2. The FROM-clause refers to (i) the view relation that represents cube r(R) for which the cube analysis statement is defined, (ii) the star tables that represent the dimensions of r(R), and (iii) the snowflake tables that represent the dimension levels for which slice conditions have been specified.

(a) For each dimension attribute D_i of cube $\mathsf{r(R)}$ the corresponding star table star_{dim_i} – representing the identifiers of the dimension levels of dim_i and the dimension paths among these dimension levels – is needed,

i. if the domain of a virtual cell's dimension attribute D_i requires more coarse grained level instances than provided by the base cube tuples from the fact table (i.e., a ROLLUP-clause was specified for D_i), or

ii. if in the SLICE-clause, a condition – specified over dimension attribute D_i – refers to a more coarse or to a more fine grained dimension level L_{D_i} than provided by the corresponding dimension attribute D_i of cube $\mathsf{r(R)}$ (i.e., $\mathsf{r(R).D}_i < \mathsf{L}_{D_i}$ or $\mathsf{r(R).D}_i > \mathsf{L}_{D_i}$).

(b) For each slice condition cond_i taken from the SLICE-clause, which refers to some dimension level L_{D_i} of dimension attribute D_i, the corresponding snowflake table $\mathsf{snow}_{L_{D_i}}^{cond_i}$ is needed if cond_i refers to a rollup attribute or to a describing attribute of L_{D_i}. Note that if cond_i refers to the identifying attribute of L_{D_i}, evaluation of cond_i is realized using dimension attribute D_i of $\mathsf{b(B)}$ if $\mathsf{b(B).D}_i = \mathsf{L}_{D_i}$, or the dimension attribute representing the identifiers in L_{D_i} in the corresponding star table star_{dim_i} if $\mathsf{b(B).D}_i < \mathsf{L}_{D_i}$ or $\mathsf{b(B).D}_i > \mathsf{L}_{D_i}$.

3. The WHERE-clause consists of three kinds of predicates:

(a) *Join predicates* between the view relation representing cube $\mathsf{r(R)}$ and the star tables as required by the FROM-clause.

(b) *Join predicates* between the star tables and the snowflake tables.

(c) *Selection predicates* that represent slice conditions as specified in the SLICE-clause of the analytical expression. Note that *Slice* conditions may be conjuncted with AND or OR – Figure 8.5 denotes this as CONJ in the WHERE-clause – and may be optionally negated using NOT.

4. The GROUP BY-clause is needed to group the tuples that have been previously generated in the WHERE-clause.

(a) If ROLLUP was specified in the analytical expression, grouping is performed using those attributes of star tables that represent the dimension levels L_{D_1}, ..., L_{D_m} as specified in the ROLLUP-clause (denoted as $\mathsf{star}_{dim_1}.\mathsf{L}_{D_1}$,...,$\mathsf{star}_{dim_i}.\mathsf{L}_{D_i}$ in Figure 8.5).

(b) Additionally, grouping is performed using the dimension attributes of the view relation that represents $r(R)$ (denoted as $\mathsf{r(R).D}_j$, ..., $\mathsf{r(R).D}_m$ in Figure 8.5) for the dimension attributes for which no ROLLUP was specified.

(c) If a dimension attribute D_k is rolled up to the top-level dimension level ALL-L_{inf}, value 'all-L_{inf} defines the domain of D_k in the SELECT-clause. Grouping is omitted for such a dimension attribute.

```
ANALYZE SalesArticlesCitiesQuarters2001 [

   SUM(SalesTotal) SalesTerm1-2001 IS
     SLICE T = 'Q1-2001' OR T = 'Q2-2001'
     ROLLUP L TO Region, T TO ALL-Days;
]
```

```
SELECT s.P P, locations.regions L, 'all-Days' T, SUM(SalesTotal) SalesTerm1-2001
FROM SalesArticlesCitiesQuarters2001 s, location
WHERE /* 3a */ s.location = location.cityId
      /* 3c */ s.T = 'Q1-2001' OR s.T = 'Q2-2001'
GROUP BY s.P, location.regionName;
```

Figure 8.6: Simple analytical expression

(d) Although the values of dimension attributes of a virtual cell, which constitute the GROUP BY-clause, identify the measure values of this cell, the corresponding SQL query must include those measure attributes in the GROUP BY-clause that are projected in the SELECT-clause, i.e., the formula f_l projects the values of measure attribute M_l without applying any aggregation function on that measure attribute (denoted as $r(R).M_l$ in Figure 8.5).

Example: Figure 8.6 presents the specification of a simple analytical expression together with the corresponding SQL query. This analytical expression selects those cells from cube SalesArticlesCitiesQuarters2001 that represent sales figures of the first two quarters of year 2001 for each city. With clause ROLLUP, the so-selected cells will be rolled up to cells of granularity Article × Region × ALL-Days. Aggregation function SUM(SalesTotal summarizes the values of these cells. The corresponding SQL query is presented in the lower part of Figure 8.6. Star table locations is used to rollup dimension attribute L from dimension level City to dimension level Region. Dimension attribute P is not rolled up. Hence the GROUP BY-clause refers to this dimension attribute as provided by cube SalesArticlesCitiesQuarters2001. Since dimension attribute T is rolled up to the top level dimension level ALL-Days, the SELECT-clause defines 'all-Days' as the value of T for all tuples generated by this query.

Complex Analytical Expressions

A complex analytical expression, which is defined upon several cell identifiers, is mapped to an SQL query (presented in Figure 8.7) that extends the SQL query pattern provided for simple analytical expressions. The increased complexity of this SQL query stems from the need to map each cell identifier (and its associated SLICE and ROLLUP clauses) to a

separate SQL query that follows the pattern which we explained above for simple analytical expressions. These queries are nested in the FROM-clause of the outer analytical expression. A nested query may either refer to the cube for which the analytical expression is defined (denoted by cube_1 in the FROM-clause of the nested query), or to a cube to which a *drill-across* should be performed (denoted by cube_n in the FROM-clause of the nested query). Tuples of these nested queries are identified by their respective cell identifiers defined in the analytical expression. The outer SQL query that represents a complex analytical expression is composed upon the following pattern:

1. The SELECT-clause refers to the dimension attributes of cells as specified in the analytical expression or – if an outer ROLLUP-clause was specified – to the attributes of star tables that are needed for ROLLUP as described for simple analytical expressions above. These dimension attributes are referred to as D_i. Further, the SELECT-clause consists of the formulas f_i to determine the values of measure attributes as described for mapping simple analytical expressions. Note that the different ways to specify aggregation functions are mapped differently to SQL. While an aggregation function specified as AGG(cellId.measAttr) is evaluated and projected in the outer SELECT clause, an aggregation function specified as cellId.AGG(measAttr) is evaluated and projected in the inner SELECT-clause of cellId and the outer SELECT-clause projects the values aggregated by the inner query. Recall the discussion on evaluating so-specified aggregation functions in Section 4.2.2.

2. The FROM-clause consists of (1) the nested queries that represent cell identifiers, (2) the star tables that are needed for ROLLUP and SLICE as described for simple analytical expressions above, and (3) the snowflake tables that are needed for evaluating slice conditions as described for simple analytical expressions above.

3. The WHERE-clause, which is needed if an outer SLICE-clause or an outer ROLLUP-clause have been specified for the analytical expression, consists of the following predicates:

   (a) *Join predicates* between star tables and those cell identifiers to which the outer ROLLUP-clause or the outer SLICE-clause refer.

   (b) *Join predicates* between star tables and snowflake tables if a slice condition refers to a dimension level's rollup attribute or describing attribute.

   (c) *Selection predicates* that represent slice conditions as specified in the outer SLICE statement of the analytical expression (e.g., join predicates among the cell identifiers).

4. The tuples generated in the WHERE-clause are grouped upon the same principles as described for simple analytical expressions above.

```
ANALYZE r(R) [
      cellId1.D1 D1, ..., cellIdk.Dm Dm, f1 M1, ..., fn Mn IS
         SLICE cond1 CONJ ... CONJ condn
         ROLLUP cellId1.D1 TO L_D1, ..., cellIdm.Dm TO L_Dm
         FOR CELLS cellId1 SLICE cond1 CONJ ... CONJ condn
                           ROLLUP D1 TO L_D1, ..., Dm TO L_Dm
                     ⋮
                  cellIdi SLICE ... ROLLUP ... ,
                     ⋮
                  cellIdk : cubek SLICE ... ROLLUP ... ]
```

```
SELECT   star_dim1.L_D1 D1,...,star_dimi.L_Di Di, r(R).Dj, ...,r(R).Dm, f1 M1,...,fn Mn

FROM   (SELECT  star_dim1.L_D1,...,star_dimn.L_Dk, f1,...,fn
        FROM   cube1, star_dim1,...,star_dimn, snow^(L_D1^cond1),...,snow^(L_Dn^condn)
        WHERE  cube1.D1 = star_dim1.L_D1 AND ... AND cube1.Dn = star_dimn.L_Dn AND
               star_dim1.L_D1^cond1 = snow^(L_D1^cond1).I AND ... AND star_dimn.L_Dn^condn = snow^(L_Dn^condn).I AND
               cond1 CONJ ... CONJ condn
        GROUP BY  star_dim1.L_D1,...,star_dimn.L_Dk) cellId1
         ⋮
       (SELECT ...
        FROM     cube1, ...
        WHERE    ...
        GROUP BY ...)  cellIdi,
         ⋮
       (SELECT  ...
        FROM     cubek, ...
        WHERE    ...
        GROUP BY ...)  cellIdk,
        star_dim1,...,star_dimn, snow^(L_D1^cond1),...,snow^(L_Dn^condn)

WHERE   cellId1.D1 = star_dim1.L_D1 AND ... AND cellIdm.Dm = star_dimm.L_Dm AND
        star_dim1.L_D1^cond1 = snow^(L_D1^cond1) AND ... AND star_dimm.L_Dm^condm = snow^(L_Dm^condm) AND
        cond1 CONJ ... CONJ condm

GROUP BY  star_dim1.L_D1,...,star_dimi.L_Di, r(R).Dj, ..., r(R).Dm, r(R).M1
```

Figure 8.7: SQL statement representing the mapping for complex analytical expressions

```
ANALYZE SalesArticlesRegionsDays2001 [
  (P s1.P, L s1.L, T s1.T, SalesTotal (s1.SalesTotal))
                SLICE s1.P = o1.P AND s1.L = o1.L AND
                      (s1.T <= o1.timeFrom OR s1.T >= o1.timeTo)
                FOR CELLS s1 SLICE P = 'AJD',
                          o1:Offerings SLICE timeFrom:Year.year = '2001' OR
                                             timeTo:Year.year = '2001';   ]
```

```
SELECT s1.P P, s1.L L, s1.T T, s1.SalesTotal SalesTotal

FROM (SELECT P, L, T, SalesTotal
      FROM SalesArticlesRegionsDays2001 s
      WHERE s.P = 'ADJ'
      GROUP BY P, L, T, SalesTotal) s1,

     (SELECT o.P, o.L, o.T
      FROM Offerings o, time t1, time t2
      WHERE o.timeFrom = t1.date AND o.timeTo = t2.date AND
            t1.year = '2001' AND t2.year = '2001'
      GROUP BY o.P, o.L, o.T) o1

WHERE s1.P = o1.P AND s1.L = o1.L AND
      (s1.T <= o1.timeFrom OR s1.T >= o1.timeTo)

GROUP BY s1.P, s1.L, s1.T, s1.SalesTotal
```

Figure 8.8: Complex analytical expression

Example: The upper part of Figure 8.8 presents the specification of a complex analytical expression that retrieves those sales of article AppleJuiceDeluxe for which no official offering existed in year 2001. The lower part of Figure 8.8 presents the corresponding SQL query of this analytical expression. The two nested queries in the FROM-clause determine the tuples of cell identifiers s1 and o1. The outer WHERE clause consists of the *Slice*-predicates defined in the outer SLICE-clause of the analytical expression. Note that star tables are not needed in the outer SQL query since (i) no outer ROLLUP clause was specified and since (ii) the outer SLICE conditions can be evaluated using dimension attribute T of cell identifier s1 and dimension attributes timeFrom and timeTo of cell identifier o1.

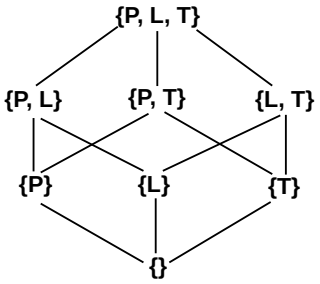

Figure 8.9: 3-dimensional lattice representing dimensions Product, Location, and Time

8.2.3 Related Work on View Materialization and View Maintenance

In Section 8.2.1, we described an approach to map cubes to the relational data model using view relations that refer to fact tables in the FROM-clause of their constituting SQL query. In a typical data warehouse environment, fact tables store huge amounts of data which cause queries to take very long to complete. Such delays are usually unacceptable such that queries are not issued against original fact tables but are rewritten to *materialized views* (or *summary tables*) [GM99] that store pre-aggregated data from fact tables. Since the number of materialized views may be large in a data warehouse environment with limited space, only a subset of the theoretically possible set of views (given N dimensions, the number of possible subviews is 2^N) can be provided. The complete set of views is usually represented as a multidimensional lattice, where each node represents a particular grouping of dimensions that may be a candidate for materialization. Figure 8.9 depicts a lattice that represents the possible subviews for materialization using the three dimensions of fact table Sales. Node { P, L, T } represents fact table Sales whereas node {} represents the "coarsest" level of granularities. While this lattice only represents the subviews for three dimensions, the number of subviews further increases when considering the various dimension levels for each dimension (not depicted in Figure 8.9).

In the following, we give an overview of approaches (1) to determine *which* views of the multidimensional lattice should be selected for materialization such that a large number of queries can be answered efficiently and (2) to propagate updates from fact tables to existing materialized views without requiring the tuples from the materialized view to be recomputed from scratch. Approaches to determine which views of the multidimensional lattice should be selected for materialization are the following:

1. Harinarayan et al [HRU96] propose a *greedy algorithm* (cf. Figure 8.10) for selecting a set of k (i.e., limit of views imposed by the database system) subviews of the multidimensional lattice that are most beneficial to materialize in order to reduce the time needed to answer the queries given some storage space. The benefit $B(v_i, S)$ of a view v_i is defined as the number of tuples that need not be selected from the smallest

```
S = {top view};
for i=1 to k do begin
   select that view v not in S such that B(v,S) is maximized;
   S = S union v;
end;
resulting S is the greedy selection;
```

Figure 8.10: Greedy Algorithm for view selection as proposed by Harinarayan et al

view v_j in S to answer queries posed on descendant views of v_i. That view v, whose benefit $B(v, S)$ is maximized will be added to S. The greedy algorithm starts with the top view (i.e., the fact table) and is evaluated for all views of a multidimensional lattice. The resulting set of views S is the "greedy selection" for materialization. Gupta et al [GHRU97] extended the greedy algorithm to determine also the optimal set of indexes defined on these views.

2. Baralis et al [BPT97] propose a heuristic approach to determine the set of views for materialization. Using this approach, view v_i of the multidimensional lattice is considered to be a *candidate view* for answering a given query q_i, if v_i is "associated with" q_i (i.e., the attributes in the GROUP BY-clause of v_i – called the *grouping attributes* – represent exactly the granularities as needed by q_i) or if there exist at least two other candidate views v_j and v_k such that v_i is the "least upper bound" of these two views (i.e., v_i can be used to answer v_j and v_k). Further, materializing v_i must be "beneficial", i.e., the costs for updating v_i must be less than the costs for answering the query using another (already materialized) view v_j. The set of candidate views will be then reduced heuristically by incrementally replacing grouping attributes of a view v_i using functional dependencies defined over the grouping attributes to obtain a more coarse grained view v_i' that may also be used to answer the queries but whose number of tuples is smaller than the number of tuples of view v_i.

3. Ezeife [Eze99] proposes another approach to select a view from the multidimensional lattice which is then fragmented horizontally using selection predicates from sample queries in order to minimize the number of tuples to be scanned when a query is posed. The proposed view selection algorithm is a modified version of the greedy algorithm introduced in [GHRU97]. After a view has been selected using the greedy approach, it will be horizontally fragmented using predicates taken from representative user queries. As a next step, the size of the fragmented view is recomputed as the average number of rows accessed by all queries using the fragments of the view. Then the greedy benefit $B(v_i, S)$ is computed and the finally returned set of fragments of views will be materialized. To determine those fragments of a view that produce fastest response time for a given warehouse query, a *fragment-advisor algorithm* is proposed. This algorithm performs runtime analysis that makes maximal use of

already statically defined and materialized view fragments.

After views from the multidimensional lattice have been materialized in order to speed up warehouse queries, such as the queries to determine the cells of a cube, these materialized views must be constantly kept up to date when new tuples are inserted into the fact table or when existing tuples are updated or deleted from the fact table. In the following we sketch a prominent approach for view maintenance selected from literature, we sketch two approaches implemented in commercial database systems, and we explain the differences between view maintenance approaches in a centralized data warehouse environment and approaches in distributed data warehouses:

1. Mumick et al [MQM97] propose the *summary-delta table* approach to maintain materialized views such that the batch update window needed for view maintenance is kept minimal. A summary-delta table represents the net changes to a materialized view due to the changes to the fact tables. View maintenance using summary-delta tables is divided into two phases: During the *propagate* phase, the tuples of the summary-delta table are computed. Since materialized views are not accessed, this does not affect availability of the data warehouse. During the *refresh*-phase, in which materialized views are locked, net changes represented in summary-delta tables are applied to the corresponding materialized views.

2. Various commercial database systems such as Oracle [BDD+98] and IBM-DB2 [LSPC00] provide mechanisms to define materialized views and to maintain the tuples of these views automatically when source data change. Both systems provide mechanisms for incremental view maintenance using delta tables in a similar way as proposed in [MQM97]. Alternatively, both systems allow materialized views to be refreshed fully, i.e., to recompute the complete set of tuples of the materialized view from its base table.

3. Various other approaches such as [ZGMHW95, AEASY97, HZ96, LMSS95, MK00] are designed to support a data warehouse architecture in which views are defined on relations from OLTP systems. The proposed algorithms not only consider maintenance of views, but are also designed to integrate data from different OLTP systems. The difference to the centralized data warehouse architecture, considered in this thesis, is that these algorithms are used to extract data from OLTP systems in order keep fact tables up to date whereas materialized views as considered in this thesis are derived from fact tables. Therefore, the approach proposed by Mumick et al [MQM97], which is also supported by commercial database systems, is considered to be superior for view maintenance in a centralized data warehouse architecture.

Chapter 9

Realizing Analysis Rules

Contents

Since active data warehousing is a young discipline, *active data warehouse management* systems (ADWMS) are commercially not available yet. An ADWMS typically should provide (1) the knowledge model (event, condition, action), (2) a declarative language to specify analysis rules, and (3) an execution model to generate decisions when an analysis rule fires. In Chapter 5, we described such a knowledge model and an approach to specify multidimensional analyzes and we introduced a syntax to specify analysis rules. This approach may be used to represent the "kernel" of an ADWMS. In this chapter, we describe the realization the knowledge model and the execution model of this approach using off-the-shelf database technology (i.e., triggers and SQL). We have chosen to use *off-the-shelf database technology* since realizing a proprietary ADWMS is laborious and bears the risk that future commercial data warehouse systems let prototypical implementation efforts become obsolete before the whole system can be put into work. The rationale to use triggers and SQL is that most of today's data warehouses are built upon relational database technology, which offer basic active functionality in the form of *triggers* [SQL99]. This active functionality is well suited to realize higher level concepts such as analysis rules. Extending existing passive data warehouses with the capabilities to automatize decision

making for routine decision tasks is thus of primary interest. Hence, a formerly passive data warehouse can be quickly turned into an active data warehouse.

This chapter is organized as follows: Section 9.1 first describes how the knowledge model (events, actions, and analysis rules as a whole) of analysis rules can be realized on top of a passive data warehouse schema, whose implementation was described in Chapter 8. Section 9.2 then describes the implementation of our basic approach of decision making using triggers and SQL. Finally, Section 9.3 summarizes the contributions of this chapter.

Note, that since we have tested the proposed implementation using Oracle DBMS, all subsequent SQL statements follow the syntax of Oracle/SQL [Ora99]. Further note that the proposed realization does not touch topics related to managing events in a highly efficient manner. Refer to [CKAK93, CM94, Cha95] for algorithms on event management.

9.1 Knowledge Model

Realizing the knowledge model of an active data warehouse requires (1) to provide mechanisms for event detection, event signaling, and action execution and (2) to provide a basic structure for decision making.

9.1.1 Events

Information about events needs to be represented as part of the meta knowledge of an active data warehouse to support processing of OLAP queries (e.g., when was the last price change of a certain article, etc.). The structure of an event type (i.e., event identifier, occurrence time, describing attributes) can be represented by a relation schema, which is called *event table*. An instance of an event type E is then represented by a tuple in the event table of E. In the following, we present the implementation of basic events (i.e., OLTP method events, relative temporal events, and calendar events), we sketch an approach to implement logical events (introduced in Section 7.1), and we outline the architecture of an event manager.

Basic Events

Different mechanisms are applied to create instances of the three categories of basic event types (i.e., OLTP method event, relative temporal event, calendar event). Note the difference between the creation of an event instance in the data warehouse (i.e., by inserting a tuple into an event table) and the occurrence of that event instance to fire analysis rules.

- Instances of *calendar events* (i.e., *absolute temporal events* and *periodic temporal events*) can be created in advance, i.e., before they occur, with an occurrence time

in the future. This may be compared with making an entry (represented by the event) in one's calendar. Instances of calendar events are created and inserted into the respective event tables using a calendar tool [CS94, CD95, LEW96, SS92], using timer-driven database triggers [HN99], or simply by issuing the necessary insert statements manually.

- Instances of *OLTP method events* occur in OLTP systems. They need to be "recorded" there and are then loaded into the active data warehouse together with conventional data. In the data warehouse, OLTP method events are used as the initiators of relative temporal events.

- Instances of *relative temporal events* are created in the active data warehouse as soon as their initiators are imported from OLTP systems. Creating a relative temporal event is realized by an AFTER INSERT trigger, which fires after the initiating OLTP method event was loaded.

Example: Figures 9.4 (a), (b), and (c) depict how instances of OLTP method event MarketLaunch, calendar event EndOfQuarter, and relative temporal event TwentyDaysPastLaunch are created. Part (d) of Figure 9.4, which deals with detecting occurrences of logical events, and part (e) of Figure 9.4, which deals with detecting occurrences of basic events, are explained below.

Logical Events

Logical events can be represented by SQL queries that range over the relations that represent other events, dimension levels, or cubes. As identified in [LOS96], the conditions to be met by a logical event may be (i) logical conditions or (ii) temporal conditions. While logical conditions may be specified for the tuples of any relation, temporal conditions can be specified only for tuples of relations that represent events. Note that the term *event* is used here in a very broad sense. An event may be (i) an event in the narrower sense such as an OLTP method event, a relative temporal event, a logical event, or a calendar event or (ii) any further kind of "event data" (i.e., data having a timestamp) such as the cells of a cube having a time dimension. Although this distinction is meaningless in the relational data model, it helps in specifying logical events. Another important distinction is as to whether the logical event only *selects* a subset of events from a given set of events or as to whether the logical event *creates* new event instances. In the first case, a new event identifier is assigned to the tuple that represents the event instance while occurrence time and all other event parameters are taken from the selected event instance that satisfies the logical and temporal conditions. In the latter case, the system creates a new event instance having a different schema and semantics than existing events. In this case, every newly created event instance is given a new identifier and a new occurrence time. Hence, the SQL query that represents a logical event is either *event-selecting* or *event-generating*.

```
SELECT NewEventId(), CurrentChronon(), articleId
FROM priceChange
WHERE TO_DAY(occtime) >= (TO_DAY(CurrentChronon()) - 60)
GROUP BY articleId
HAVING COUNT(*) > 2;
```

Figure 9.1: Event-generating SQL query representing logical event FrequentPriceChange

The basic structure of an SQL query representing a logical event is as follows:

- The SELECT-clause represents the parameters (i.e., attributes) of the logical event. In the case of an event-selecting SQL query, the SELECT-clause assigns each selected tuple a new event identifier using function NewEventId() and projects all other parameters (including occurrence time) of the event instance that satisfies the temporal and logical conditions. In the case of an event-generating SQL query, a new event identifier and occurrence time are assigned to every newly created tuple using functions NewEventId() and CurrentChronon() respectively. The remaining event parameters are taken from several other event instances that constitute the newly created event instance.
- The FROM-clause consists of the relations that are needed to specify the logical and/or temporal conditions. These relations represent (i) other events, (ii) dimension levels (i.e., snowflake tables), or (iii) cubes.
- The WHERE-clause consists of (i) join predicates among event tables, snowflake tables, and views representing cubes, (ii) predicates representing the logical conditions, and (iii) predicates representing the temporal conditions.
- The optional GROUP BY-clause consists of the attributes that determine the grouping of tuples. In the case of an event-selecting SQL query, the set of attributes that constitutes the GROUP BY-clause must be a superset of the attributes that describe the selected event.

Example: Assume a situation in which an analysis rule should be fired when the price of an article was changed more than two times during the last two months (i.e., 60 days). To detect an instance of this logical event, the (temporal) condition requires that during the last 60 days, more than two price changes occurred for the *same* article. Figure 9.1 shows the event-generating SQL query that represents this logical event. The temporal condition is mapped to the WHERE-clause and to the GROUP BY-clause. Function TO_DAY() truncates the precision of an attribute of type TIMESTAMP (i.e., "YYYY-MM-DD hh:mm:sec") to the precision of type DATE (i.e., "YYYY-MM-DD"). Function CurrentChronon() retrieves the timestamp of the current ADW cycle.

Hence, condition TO_DAY(occtime) >= (TO_DAY(CurrentChronon()) - 60) selects only price changes that occurred within the past 60 days. Clause GROUP BY articleId ensures that these price changes occurred for the *same* article. Finally, the HAVING-clause selects only groups of price changes with more than two nested tuples.

While logical conditions and simple temporal conditions can be expressed as boolean predicates in the WHERE-clause of the SQL query, two fundamental concepts of logical events – *nested event parameters* and *temporal adjacency* – go beyond the above query structure and need to be treated in further detail.

Nested Event Parameters. In the case of an event-selecting SQL query, all parameters of the logical event are taken from exactly one other event. In the case of an event-generating SQL query, the newly generated event is composed of several more primitive events. Hence, there is a 1:N mapping between a newly created logical event instance and instances of primitive events whereby the number of instances of primitive events may be different for each instance of the logical event. This is the case when an event-generating SQL query defines a GROUP BY-clause. A consequence of grouping tuples is that each attribute that is not part of the GROUP BY-clause is only accessible in the SELECT-clause using an aggregation function such as COUNT(), SUM(), MAX(), etc. Nevertheless, it is often necessary to refer directly to the attributes of "nested events" – called "nested event parameters" – in the definition of an analysis rule. For this purpose, for each logical event nested event parameters and the eventId of the corresponding logical event are stored in a separate table – the *nested events table*. Nested event parameters are determined by a separate SQL query that is basically a copy of the original query representing the logical event, except that (i) the SELECT-clause represents all nested event parameters without assigning a new event identifier and occurrence time and (ii) the GROUP BY-clause is omitted. This query is evaluated by an AFTER INSERT-trigger, which fires immediately after an instance of the logical event is inserted into the logical event table. Each tuple representing a nested event will be inserted in the *nested events table* together with the eventId of the logical event instance. To be used correctly, the nested events table must be joined with the event table holding logical event instances (join is effected upon the eventId of the logical event instance).

Example: Consider the event-generating SQL query of Figure 9.1, which represents logical event FrequentPriceChange. Every tuple generated by this query indicates that more than two price changes occurred for a particular article during the last two months. Further information concerning the individual price changes such as event identifier, occurrence time, or ratio of price change are not accessible. For this purpose, the trigger NestedEventParametersFrequentPriceChange in Figure 9.2 is defined. Variable :new.eventId holds the event identifier of the newly generated logical event instance. The remaining attributes articleId, eventId, occtime, and ratio in the SELECT-clause of this query represent the projection of nested parameters. If the definition

```
CREATE TRIGGER NestedEventParametersFrequentPriceChange
AFTER INSERT ON FrequentPriceChangeOcc
FOR EACH ROW
BEGIN
   INSERT INTO NestedEventsFrequentPriceChange
               (logicalEventId, eventId, occtime, articleId, ratio)
         (SELECT :new.eventId, eventId, occtime, articleId, ratio
          FROM   priceChange
          WHERE  TO_DAY(occtime) >= (TO_DAY(CurrentChronon()) - 60));
END; /* NestedEventParametersFrequentPriceChange */
```

Figure 9.2: Trigger to determine the nested event parameters of logical event FrequentPriceChange

```
SELECT   NewEventId(), occtime, noInMonth
FROM     (SELECT eventId, occtime,
                 RANK() OVER (PARTITION BY month ORDER BY occtime) as noInMonth
          FROM   mondays)
WHERE    noInMonth = 2;
```

Figure 9.3: Event-selecting SQL query representing logical event 2ndMondayInMonth

of an analysis rule refers to these attributes, those tuples from table NestedEventsFrequentPriceChange are selected, whose logicalEventId is identical with the event identifier of the logical event instance.

Temporal Adjacency. Formulating temporal conditions like "the second monday of a month" or "the first price change after market launch" requires an explicit ordering of events according to their occurrence time, which is called *temporal adjacency* of events. Such an explicit ordering of events is then used to express the temporal condition as a boolean predicate in the WHERE-clause. Hence, we advocate two steps to map such conditions to an SQL query:

Step 1: Determine the ordering of events as needed by the temporal condition using a separate SQL query. Ordering is realized by defining a separate attribute in the SELECT-clause, whose values are determined using function RANK().

Step 2: Embed the query of Step 1 in the FROM-clause of the SQL query that represents the logical event. Add a boolean predicate to the WHERE-clause that refers to this ordering attribute to express temporal adjacency.

Example: Consider a situation, in which an analysis rule should be fired periodically on every second monday of a month (i.e., 2ndMondayInMonth). We assume that the underlying calendar system provides the relation mondays (eventId, occtime, month), whose tuples represent only mondays. The event-selecting SQL query that represents logical event 2ndMondayInMonth (shown in Figure 9.3) uses this relation to establish the ordering as follows: The nested query in the FROM-clause determines for each month (specified by clause PARTITION BY month) the ordering of mondays according to their occurrence time (specified by clause ORDER BY occtime) using function RANK(). This ordering is represented by attribute noInMonth in the SELECT-clause of the nested query. The outer query then selects only those mondays whose number in month (i.e., noInMonth) is 2.

Realizing Logical Events with Off-the-Shelf Database Technology. As already mentioned in Section 8.1, various commercial database systems such as Oracle or IBM-DB2 offer the possibility to define materialized views and to maintain (i.e., refresh) these views automatically when base data change. We use these technologies to realize the detection of logical events in a straight forward manner as follows:

1. SQL queries that represent logical events are implemented as materialized views. Detecting changes in the base relations and propagating these changes to the materialized view (i.e., inserting new tuples into materialized views) will be realized using any of the mechanisms provided by the underlying DBMS.

2. In order to determine the exact timepoint within an ADW cycle, at which event detection should be initiated, we advocate to specify these materialized views to be refreshed *on demand*. Typically, a DBMS supporting materialized views provides a stored procedure that realizes on demand-refreshment when called. Once per ADW cycle, this procedure will be called from procedure SignalEvents, which we defined in our event manager (see Section 9.1).

3. As described above, an AFTER INSERT-trigger is needed for event-generating logical events to determine the nested events parameters associated with the logical event instances.

4. Since the tuples of a materialized view are stored implicitly in a separate table, this table is used as the *signaled events table*. If an analysis rule fires upon the occurrence of a logical event, the trigger that initiates rule processing will fire AFTER INSERT on the signaled events table of the logical event.

Example: Figure 9.4 illustrates how instances of type 2ndMondayInMonth are detected using materialized views. Every tuple that is generated by the view maintenance mechanism provided by the underlying database system represents a new instance

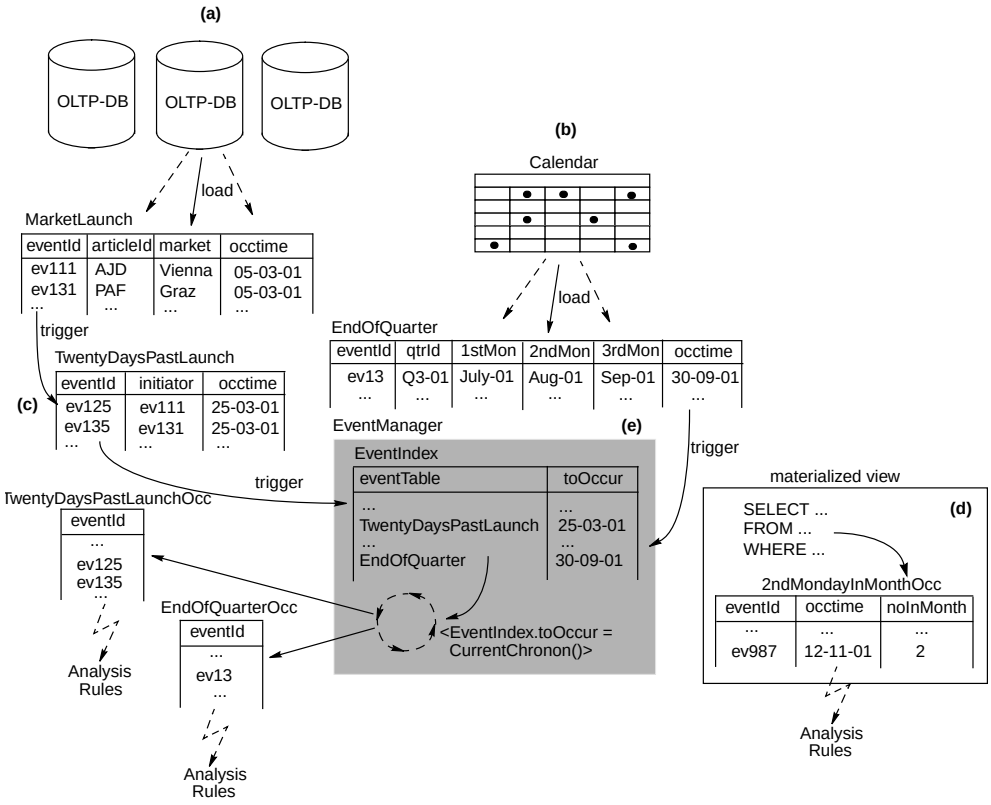

Figure 9.4: Event management and analysis rule invocation

of this logical event type. An analysis rule that is defined upon this event type will be fired whenever the view maintenance mechanism inserts a tuple into the signaled events table 2ndMondayInMonthOcc.

Event Manager

The interval at which data is loaded from operational systems has an impact on the granularity of time at which it is meaningful to consider occurrence times of events since the state of the data warehouse remains unchanged during an ADW cycle. If this interval is constant, as we assume for simplicity, we can identify it with the finest granularity of time at which events occur in the active data warehouse. Thus, in our case one chronon corresponds exactly with the duration of one ADW cycle. In a future extension, the boundaries of an ADW cycle may be determined more dynamically by the timepoints at which events are scheduled to occur, which will lead to ADW cycles of varying durations.

Example: In our running example, we assume that one ADW cycle constantly lasts one day. Thus, events will be scheduled to occur once per day without requiring to consider hours, minutes, or seconds.

Since event occurrences are needed to fire and process analysis rules, they are held in separate tables, called *signaled events tables*. Signaling an event occurrence is realized by inserting the primary key value of the tuple representing the event instance into the signaled events table. For basic events (i.e., relative temporal events and calendar events) this task is carried out by the active data warehouse's *event manager*. For logical events, we use the detection mechanisms of materialized views as described above. To avoid that the event manager must query every event table for detecting new event occurrences, an *event index* is used, which stores the name of an event table together with the chronon, at which one or more event instances (represented as tuples of this event table) will occur. Each pair $\langle eventtable, chronon \rangle$ exists only once in the event index. This approach avoids that every instance of an event needs to be stored in the event index. The event manager of our demonstration prototype is realized as an Oracle PL/SQL package providing a table that serves as the *event index* and the two stored procedures IndexEvent and SignalEvents. Indexing a calendar event or indexing a relative temporal event instance is realized by triggers, which call procedure IndexEvent after an instance of the respective event type was created. At the beginning of each ADW cycle (after operational data was loaded into the data warehouse), procedure SignalEvents is called to select the identifiers of the event instances that occur at this chronon. The so-selected event identifiers are inserted into the respective signaled events tables and are then considered to have *occurred*. This insert operation fires the execution of analysis rules. At the end of every ADW cycle, the signaled event tables will be cleared to be prepared for rule processing in some future ADW cycle.

Example: Figure 9.4 (e) illustrates the principal idea of the event manager, which is depicted in the shaded box. The event index holds the names of event tables TwentyDaysPastLaunch and EndOfQuarter and the timepoints (chronons), at which instances of these event types will occur. On March 25, 2001, the primary key values identifying event instances of type TwentyDaysPastLaunchOcc are inserted into signaled events table TwentyDaysPastLaunchOcc if the occtime of these event instances is 25-03-01. Corresponding actions will be carried out for instances of event type EndOfQuarter on September 30, 2001.

The event manager as well as analysis rules need to refer to the full event information about event occurrences of a certain temporal event type E. For this, we define a view relation $current_E$ for each E, which holds all data describing the event instances of E that occur at the current chronon. We use function CurrentChronon() to determine the timestamp of the current chronon (i.e., current ADW cycle).

Example: Figure 9.5 shows the definition of view CurrentTwentyDaysPastLaunchOcc, which selects all event occurrences of event TwentyDaysPastLaunch that occur at the current chronon.

```
CREATE VIEW CurrentTwentyDaysPastLaunchOcc
         (eventId, occtime, initiator, init_occtime, articleId, market) AS

  SELECT TwentyDaysPastLaunch.eventId, TwentyDaysPastLaunch.occtime,
         TwentyDaysPastLaunch.initiator, MarketLaunch.occtime,
         MarketLaunch.articleId, MarketLaunch.market
  FROM   TwentyDaysPastLaunch, TwentyDaysPastLaunchOcc, MarketLaunch
  WHERE  TwentyDaysPastLaunch.eventId = TwentyDaysPastLaunchOcc.eventId AND
         TwentyDaysPastLaunch.occtime = CurrentChronon() AND
         TwentyDaysPastLaunch.initiator = MarketLaunch.eventId;
```

Figure 9.5: Definition of event set CurrentTwentyDaysPastLaunchOcc

In the case of logical events, view relation $current_E$ is defined upon the following query patterns[1]:

1. If the logical event is event selecting, $current_E$ is defined upon query SELECT * FROM "signaled events table" WHERE occtime = CurrentChronon()).
2. If the logical event is event generating, $current_E$ is defined upon query SELECT * FROM "signaled events table" s, "nested events table" n WHERE s.eventId = n.logicalEventId AND s.occtime = CurrentChronon()).

9.1.2 Actions

Data about the action-part of analysis rules are part of the meta-data (i.e., which actions are available, formal parameters, etc.) and are part of the operational data describing concrete action executions that have to be carried out in OLTP systems when "positive decisions" are exported. In the following, we only describe the meta-data part and elaborate on the operational part later in the paper. From a meta-data point of view, data about actions are represented as tuples of the relation schemas given in Figure 9.6. Relation schema Actions stores the name and the precondition of an action together with the information for which dimension level the action may be carried out. The meta-data repository of a relational data warehouse will also contain meta-data relations that describe conventional multidimensional data (e.g., Levels(levelId, levelName...), LevelAttributes(attributeId, attributeName, ...), Hierarchies(rollupFromLevel, rollupToLevel), Cubes(cubeId, cubeName, ...), etc.), which are trivial and thus not depicted in Figure 9.6. Relation schema Parameters provides information about the formal parameters of actions. Relation schema Conflicts represents the "conflict resolution table", which is needed to detect and to resolve inter-rule conflicts as discussed in Section 5.2.3.

[1]Note that view relation $current_E$ provides access only to *current* event occurrences, i.e., event occurrences whose occurrence time coincides with the current ADW cycle.

Actions (*actionId*, levelId, actionName, precondition)
Parameters (*parameterName*, *actionId*, dataType, position)
Conflicts (*actionId*1, *actionId*2, conflictResolution)

Figure 9.6: Relational representation of action meta data

9.1.3 Analysis Rules

To enable decision making of an analysis rule AR, three basic implementation tasks are required: (1) Define the cubes of the analysis graph. (2) Define the decision variables that have been specified in the decision steps. (3) Enable decision making by defining "containers" that hold (a) the set of "positively decided" level instances (i.e., the level instances for which the rule's action must be executed in the OLTP system) and (b) the set of "undecided" level instances (i.e., the level instances which need to be analyzed manually by a knowledge worker). The set of level instances for which definitely no action is to be taken (i.e., "decide negative") need not be maintained. In the following, we use AR as a subscript to denote that a particular component is specific to rule AR. Many relational DBMS offer the possibility to modularize schema objects using packages, modules, or db-schemas in order to avoid naming conflicts and to encapsulate schema objects and program code that implement AR. Assume that the subscript AR represents such a modularization technique. In our prototype, encapsulating the implementation of AR is realized by a db-schema in Oracle.

Analysis Graph. Incrementally specified cubes are implemented as view relations using the mapping approach described in Section 8.2. If an analysis rule's event variable is used to specify a *Slice*-condition, the view relation representing the current event set $current_E$ of the analysis rule's triggering event is added to the FROM-clause of the view relation representing the cube. The predicate that represents the *Slice*-condition is then added to the WHERE-clause of the view relation representing the cube.

Decision Variables. Since decision variables extend analytical expressions (cf. Section 4.2.2), we extend the relational mapping of analytical expressions as follows:

1. Decision variables are mapped to view relations. The analytical expression of a decision variable is mapped to an SQL query as described in Section 8.2.2. To provied a unique naming among the decision variables defined for AR, the name of such a view relation is composed of the name of the cube that is analyzed together with the name of the decision variable defined in the decision step.

2. Since the analysis rule's event variable and the analysis rule's primary dimension variable may be used in the specification of decision variables, the view relation representing $current_E$ and the snowflake table that represents the primary dimension level are added to the FROM, WHERE, and GROUP BY-clauses of the view relation representing the decision variable.

Decision Containers. Executing an analysis rule generates two sets of tuples: The first set represents the level instances of the analysis scope for which the rule's action must be carried out (i.e., the rule's "positive decisions"). The second set represents the level instances of the analysis scope for which further manual analyses must be carried out (i.e., the rule's "undecided" cases). For a particular analysis rule AR, the set of objects representing the "positive decisions" are held in table ExecuteOLTPMethod$_{AR}$(entityId, param1, param2, ...) whereas all "undecided" cases are held in the single-attribute table AnalyzeManually$_{AR}$(entityId). An entry in table ExecuteOLTPMethod$_{AR}$ identifies a certain level instance of the analysis scope with attribute entityId together with the parameter values that are required for executing the action in the OLTP system. An entry in table AnalyzeManually$_{AR}$ identifies a level instance of the analysis scope, which must be analyzed manually in more detail.

9.2 Execution Model

In this section, we describe an implementation of the procedural evaluation of analysis rules using cascading triggers. Although also other approaches to implement decision making are possible (e.g., realizing decision making as a single monolithic stored procedure), cascading triggers offer a more open architecture, which is easy to extend and modify if the specification of an analysis rule changes.

Executing an analysis rule requires (1) to determine the set of level instances of the primary dimension level that constitute the analysis scope and (2) to process the decision steps using cascading triggers.

9.2.1 Analysis Scope

The analysis scope of AR comprises the level instances of the primary dimension level that are "affected" by the analysis rule's event and that satisfy the primary condition. Determining the analysis scope will be carried out after occurrences of the rule's event have been signaled by the event manager. Although a rule is specified as if it were fired for each level instance of the analysis scope separately, implementing an analysis rule follows the *set-oriented* rule processing approach.

Determining the analysis scope of AR is realized by trigger InitiateAnalysis$_{AR}$, which fires after the occurrences of the analysis rule's event have been signaled by the event manager. For a calendar event, all level instances of the primary dimension level are "affected". But only the subset of affected level instances that satisfy the analysis rule's primary condition will be inserted into table AnalysisScope$_{AR}$. In the procedural evaluation of analysis rules (cf. Section 5.4), this set is determined as AS := { I ∈ primary dim level of AR | prim_cond(AR,I). The corresponding SQL statement is as follows[2]:

```
INSERT INTO AnalysisScope_AR ( SELECT snow^L^prim.I
                               FROM snow^L^prim
                               WHERE C^prim );
```

For a relative temporal event, also set $current_E$ must be taken into consideration to determine the set of level instances affected by the relative temporal event. In the procedural evaluation of analysis rules, this set is determined as AS := { I | ∃ e ∈ E: occurredFor(e) = I ∧ prim_cond(AR,I). The corresponding SQL statement is as follows:

```
INSERT INTO AnalysisScope_AR ( SELECT snow^L^prim.I
                               FROM snow^L^prim, current_E
                               WHERE snow^L^prim.I = current_E.L^prim AND C^prim );
```

To follow a set-oriented approach for executing analysis rules, all triggers representing a decision step are defined to fire only once using clause FOR EACH STATEMENT in the trigger definition. Since most trigger systems employ *immediate coupling* for executing triggers, the implementation principle as described above realizes the proposed *depth-first* approach.

Example: Figure 9.7 depicts the definition of trigger InitiateAnalysis$_{AR}$ for analysis rule RemoveHighPricedSoftDrinksAR. This trigger fires after an occurrence of calendar event EndOfQuarter has been inserted into event occurrence table EndOfQuarterOcc. The first two DELETE-statements are needed for preprocessing to ensure that the rule's decision containers are empty at the beginning of rule execution. The analysis scope of rule RemoveHighPricedSoftDrinksAR is then determined by the INSERT-statement. Since this analysis rule is defined upon calendar event EndOfQuarter, the analysis scope comprises the set of objects of the primary dimension level, that satisfy the primary condition a.category = SSoft DrinksÄND a.pricePerUnit > 25. The remaining SQL statements of this trigger will be explained later.

[2] snow$^{L^{prim}}$ denotes the snowflake table representing the primary dimension level L^{prim}. C^{prim} denotes the predicate representing the primary condition taken from the definition of analysis rule AR.

```
CREATE TRIGGER InitiateAnalysis_AR
AFTER INSERT ON EndOfQuarterOcc FOR EACH STATEMENT
BEGIN
  /* preprocessing → make sure that no decisions from past rule executions
      still exist */
  DELETE FROM ExecuteOLTPMethod_AR;
  DELETE FROM AnalyzeManually_AR;

  /* determine analysis scope → causes depth − first evaluation of decisions steps */
  INSERT INTO AnalysisScope_AR ( SELECT a.articleId
                                 FROM  article a
                                 WHERE a.category = 'Soft Drinks' AND
                                       a.pricePerUnit > 25);

  /* postprocessing → solve intra rule conflicts */
  DELETE FROM AnalyzeManually_AR
         WHERE entityId IN (SELECT  entityId
                            FROM ExecuteOLTPMethod_AR);

  /* postprocessing → cleanup */
  DELETE FROM AnalysisScope_AR;
  DELETE FROM DetailAnalysis_SalesUpperAustriaThisQuarter_AR;
  DELETE FROM DetailAnalysis_SalesUpperAustriaThisQuarterMonths_AR;
  ...
END;
```

Figure 9.7: Definition of trigger InitiateAnalysis$_{AR}$

9.2.2 Decision Steps

Decision steps are realized by triggers (called *decision step triggers*), whereby each trigger realizes a particular recursive call of procedure EvaluateDecisionStep (cf. Figure 5.15) for decision step d of a particular cube $r(R)$. The major task of a decision step trigger is to evaluate conditions *triggerAction* and *detailAnalysis* of d for some objects of the analysis scope. The two conditions are represented by SQL queries that determine the set of objects satisfying these conditions respectively. Mapping the condition predicate specified in clause TRIGGER ACTION IF or in clause DETAIL ANALYSIS IF is as follows:

```
SELECT primDimAttr
FROM cube, decVar1, ..., decVarn
WHERE conditionPredicate;
```

- The projected dimension attribute primDimAttr in the SELECT-clause of this query holds the level instances of the primary dimension level. This attribute was specified by clause PRIMARY in the definition of "root cube" or in the definition of the condition.
- Depending on which measure attributes are referenced by the conditionPredicate (representing $TrgA_{r(R)}$ or $DtlA_{r(R)}$) in the WHERE-clause, the FROM-clause may contain the view relation representing the cube for which the decision step was specified and/or the view relaltions representing the decision variables decVar_1,...,decVar_n that were specified to provide additional measure attributes.

The set of objects in the analysis scope of $r(R)$ (explained below) satisfying condition $TrgA_{r(R)}$ is inserted into table $\mathsf{ExecuteOLTPMethod}_{AR}$ and is thus withdrawn from further rule processing. The following SQL statement presents the mapping of the procedural statement TrgA := {I ∈ AS | triggerAction(r(R),I)}, which determines the level instances of the analysis scope satisfying the *trigger action condition*:

```
INSERT INTO ExecuteOLTPMethod_AR ( SELECT entityId
                                   FROM  AnalysisScope_AR^r(R)
                                   WHERE entityId IN (TrgACondSubquery_r(R))
```

- Predicate $\mathsf{TrgACondSubquery}_{r(R)}$ represents the SQL query to evaluate condition $TrgA_{r(R)}$ as described above.
- If the analyst did not specify a *trigger action condition*, $TrgA_{r(R)}$ evaluates to FALSE. In such a situation, the INSERT-statement is omitted in the definition of the corresponding decision step trigger.

Example: Figure 9.8 depicts the definition of the "root" decision step trigger of analysis rule RemoveHighPricedSoftDrinksAR. The first INSERT-statement determines the set of objects satisfying condition $TrgA_{r(R)}$, which is represented by predicate SalesTotal ¡ 100000. Since in this predicate, a measure attribute from cube $\text{SalesUpperAustriaThisQuarter}_{AR}$ is referenced, dimension attribute p is projected in the SQL query and the view relation representing cube $\text{SalesUpperAustriaThisQuarter}_{AR}$ constitutes the FROM-clause. The second INSERT-statement is explained later. Figure 9.9 illustrates the cascading trigger model, which realizes decision making of analysis rule RemoveHighPricedSoftDrinksAR. A decision step trigger is depicted by an oval. Insert statements and trigger firings are depicted by dashed arrows. A dashed termination symbol ($\dashv$) indicates that no level instances are selected by the respective condition of a decision step and thus no further processing in that branch is necessary. Condition $TrgA_{AR}^{r(R)}$ of decision step trigger $\text{DecisionStep\_SalesUpperAustriaThisQuarter}_{AR}$ selects article AJD as "decided positively" by inserting the identifier of the tuple representing that level instance into table $\text{ExecuteOLTPMethod}_{AR}$.

The set of objects satisfying condition $DtlA_{r(R)}$ represents the analysis scope for the decision steps of direct descendants of $r(R)$. This set is determined by the procedural statement DtlA := {l ∈ AS \ TrgA | detailAnalysis(r(R), l)}, which is mapped to SQL as follows:

```
INSERT INTO DetailAnalysis_AR^r(R) ( SELECT entityId
                                     FROM (( SELECT entityId
                                             FROM AnalysisScope_AR^r(R))
                                           MINUS
                                           ( SELECT entityId
                                             FROM ExecuteOLTPMethod_AR))
                                     WHERE entityId IN (DtlACondSubquery_r(R))
```

- The FROM-clause of the query determining the tuples to be inserted into table $\text{DetailAnalysis}_{AR}^{r(R)}$ represents the set difference between level instances of the analysis scope and the level instances that are already decided "positively", i.e., AS \ TrgA.
- Predicate $\text{DtlACondSubquery}_{r(R)}$ represents the SQL query to evaluate condition $DtlA_{r(R)}$ as described above.
- If the analyst did not specify a *detail analysis condition*, $DtlA_{r(R)}$ evaluates to TRUE. Hence, all level instances for which no decision was generated yet are inserted into table $\text{DetailAnalysis}_{AR}^{r(R)}$. In such a situation, the WHERE-clause of the query determining the tuples to be inserted into table $\text{DetailAnalysis}_{AR}^{r(R)}$ is omitted.

We distinguish three different cases to determine the analysis scope of some cube $r(R)$ in order to fire subsequent decision step triggers. The motivation for this distinction, which

```
CREATE TRIGGER DecisionStep_SalesUpperAustriaThisQuarter_AR
AFTER INSERT ON AnalysisScope_AR FOR EACH STATEMENT
BEGIN
  /* Step 1: determine set TrgA satisfying condition TrgA_r(R) */
  INSERT INTO ExecuteOLTPMethod_AR (
        SELECT  entityId
        FROM    AnalysisScope_AR
        WHERE   entityId IN (SELECT  p
                             FROM  SalesUpperAustriaThisQuarter_AR
                             WHERE SalesTotal  <  100000));

  /* Step 2: determine set DtlA satisfying condition DtlA_r(R)
             → causes depth − first processing of subsequent decision step triggers */
  INSERT INTO DetailAnalysis_SalesUpperAustriaThisQuarter_AR (
        SELECT  entityId
        FROM    ((SELECT entityId FROM AnalysisScope_AR)
                  MINUS
                 (SELECT entityId FROM ExecuteOLTPMethod_AR))
        WHERE  entityId IN (SELECT p
                            FROM  SalesUpperAustriaThisQuarter_AR
                            WHERE SalesTotal  <  500000));
END;
```

Figure 9.8: Implementation of the "root" decision step trigger

we will introduce below, is (1) to minimize the number of tables and triggers needed in order to determine the exact set of tuples representing the analysis scope of some cube $r(R)$ (and thus to avoid unnecessary trigger firings) and (2) to process an object of the rule's analysis scope only once by each decision step trigger. This goal requires distinct treatments (1) for the "root cube", which has no predecessor, (2) for an "intersection cube" whose analysis scope is determined by two or more direct predecessors, and (3) for any other cube whose analysis scope is determined by exactly one direct predecessor. The different cases are handled as follows:

1. If cube $r(R)$ represents the "root cube" of the analysis graph, its analysis scope is identical with the analysis scope of AR. The decision step trigger that implements the decision step of $r(R)$ fires after table AnalysisScope$_{AR}$ was populated.

   Example: Trigger DecisionStep_SalesUpperAustriaThisQuarter$_{AR}$ (cf. Figure 9.8), which is the "root" decision step trigger of analysis rule RemoveHighPricedSoftDrinksAR, fires AFTER INSERT ON AnalysisScope$_{AR}$ FOR EACH STATEMENT.

2. If cube $r(R)$ was specified by OLAP operators SLICE or DRILLDOWN, $r(R)$ has a single direct predecessor, $pre_{AR}^{r(R)}$, and the analysis scope of $r(R)$ is the set of objects that satisfy condition *detailAnalysis* of the decision step of cube $pre_{AR}^{r(R)}$. This set of objects is kept in table DetailAnalysis_$pre_{AR}^{r(R)}$, which is called *detail analysis table* of cube $pre_{AR}^{r(R)}$. All successor cubes of $pre_{AR}^{r(R)}$ that have $pre_{AR}^{r(R)}$ as their single direct predecessor can share this detail analysis table as common analysis scope such that unnecessary duplications of analysis tables are avoided.

   Example: The second INSERT-statement of trigger DecisionStep_SalesUpperAustriaThisQuarter$_{AR}$ in Figure 9.8 evaluates condition $DtlA_{r(R)}$ of the root decision step trigger and populates table DetailAnalysis_SalesUpperAustriaThisQuarter$_{AR}$, which is the detail analysis table of this cube. Since cubes SalesUpperAustriaThisQuarterMonths$_{AR}$ and SalesUACitiesThisQuarter$_{AR}$ are both defined as "DRILLDOWN FROM SalesUpperAustriaThisQuarter$_{AR}$", table DetailAnalysis_SalesUpperAustriaThisQuarter$_{AR}$ represents the analysis scope for decision step triggers DecisionStep_SalesUpperAustriaThisQuarterMonths$_{AR}$ and DecisionStep_SalesUACitiesThisQuarter$_{AR}$ (not shown). Both triggers fire AFTER INSERT ON DetailAnalysis_SalesUpperAustriaThisQuarter$_{AR}$ FOR EACH STATEMENT.

3. If cube $r(R)$ was specified upon OLAP operator INTERSECTION, the analysis scope of $r(R)$ is the union of tuples in the detail analysis tables of all direct predecessors of $r(R)$. Thus, the decision step trigger implementing the decision step of $r(R)$ must fire if any of the predecessors of $r(R)$ populates its detail analysis table. Such a

situation would be a typical case for "OR"-events (i.e., AFTER INSERT ON (Table_1 OR Table_2), which are not supported by SQL in this form. Thus, we introduce an *analysis scope table* for $r(R)$, which will be populated every time a predecessor of $r(R)$ populates its detail analysis table. The decision step trigger implementing the decision step of $r(R)$ fires AFTER INSERT ON $\text{AnalysisScopeFor}_{AR}^{r(R)}$. Forwarding the tuples of the *detail analysis table* of a predecessor of $r(R)$ to the *analysis scope table* of $r(R)$ is realized by a trigger that inserts a tuple from the *detail analysis table* into the *analysis scope table* of $r(R)$ only if that tuple is *not* already contained in the analysis scope table of $r(R)$. This ensures that an object will not be analyzed by a particular decision step more than once. This is depicted in Figure 9.9 by an oval labeled forward. The algorithm in Fig. 5.15 achieves this in line 1 by removing the set of objects that were already analyzed by the decision step of the current cube from the set of objects to be currently analyzed.

Example: Since cube $\text{SalesLinzThisQuarterMonths}_{AR}$ is specified as "INTERSECTION FROM $\text{SalesUpperAustriaThisQuarterMonths}_{AR}$, $\text{SalesUACitiesThisQuarterMonths}_{AR}$", the analysis scope table AnalysisScopeFor_$\text{SalesLinzThisQuarterMonths}_{AR}$ is created. As depicted in Figure 9.9, decision step trigger $\text{DecisionStep\_SalesUpperAustriaThisQuarterMonths}_{AR}$ first inserts articles PAF and LES into its detail analysis table, which are forwarded to table $\text{AnalysisScopeFor\_SalesLinzThisQuarterMonths}_{AR}$. This insert statement fires decision step trigger $\text{DecisionStep\_SalesLinzThisQuarterMonths}_{AR}$, which selects articles PAF and AMP to be analyzed manually. Then decision step trigger $\text{DecisionStep\_SalesUACitiesThisQuarter}_{AR}$ inserts articles LES and PAF into table $\text{DetailAnalysis\_SalesUACitiesThisQuarter}_{AR}$, but only article LES will be forwarded to the analysis scope table since AMP was already processed before.

During rule processing, some level instances may be considered both, as "decided positively" *and* as "undecided" (i.e., analyze manually) because decision steps that belong to a certain "branch" of the analysis graph realize decision making isolated from all other branches. A basic assumption of our decision-making model is that a level instance belongs to the "positive decisions" of an analysis rule if at least one decision step "decides" that the rule's action must be executed for that level instance in the OLTP system. Thus, resolving *intra-rule conflicts* as described above is accomplished by removing the set of "positively decided" level instances from the set of "undecided" level instances at the end of rule processing. Since the first task of trigger $\text{InitiateAnalysis}_{AR}$ initiates processing of decision steps *immediately*, solving intra-rule conflicts represents the final task of rule execution. Note that *immediate* trigger execution realizes executing the decision steps of an analysis rule in a *depth-first* traversal of the analysis graph. After all decision step triggers have been executed, intra-rule conflicts must be solved (i.e., "positively decided" level instances are removed from "undecided" level instances) and the tables that are needed to pass over entities to subsequent decision steps (i.e., table $\text{AnalysisScope}_{AR}$, the *detail*

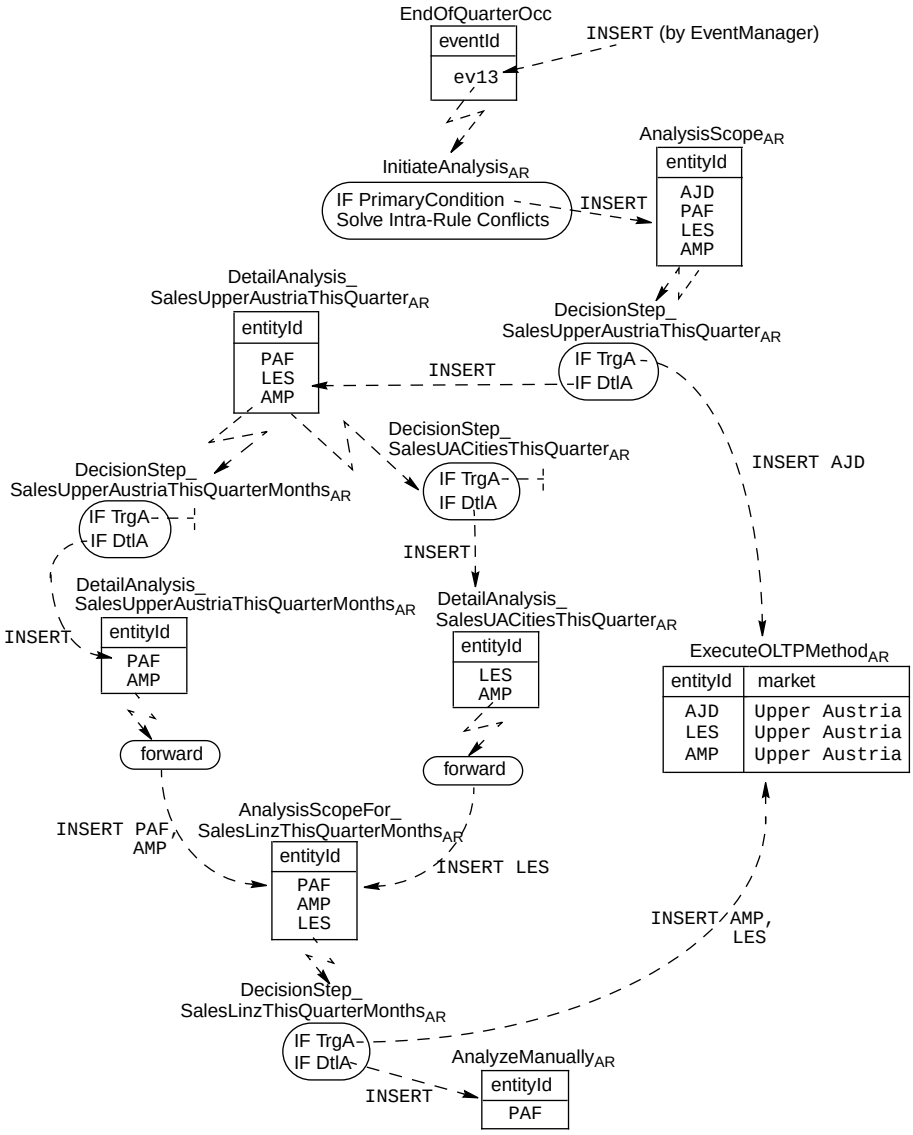

Figure 9.9: Analysis rule execution

analysis tables, and the *intersection tables*) must be cleaned for future rule executions. The proposed procedural semantics solves intra-rule conflicts in exactly the same way (see line 8 in Figure 5.14). This intra-rule conflict resolution strategy realizes the declarative semantics of analysis rule execution as given in Section 5.3.2. The tasks just described are the final tasks of rule execution and will be carried out by trigger InitiateAnalysis$_{AR}$.

Example: In Figure 9.7 the DELETE statements of trigger InitiateAnalysis$_{AR}$ realize the task of solving intra-rule conflicts and the task of cleaning up containers that were populated during rule processing.

9.3 Summary

In this chapter, we introduced an approach to implement the knowledge model and the execution model of analysis rules using off-the-shelf database technology (i.e., SQL and triggers). The presented implementation respects the declarative and the procedural semantics defined in Section 5.3.2. Since almost every relational DBMS supports SQL and triggers, the proposed solution is widely applicable. The major benefits are:

1. The presented approach allows practitioners to transform their currently passive data warehouse quickly into an active data warehouse without requiring special-purpose active data warehouse *management* technology which is not yet commercially available.

2. The techniques presented in this chapter may be utilized by system developers to build an active data warehouse management system (ADWMS) (1) on top of an existing relational DWMS, (2) by extending the "kernel" of an existing multi-dimensional database management system (MDDB) [Col96] with active data warehouse functionality, or (3) by constructing a new system from scratch.

Part V

Conclusion

Chapter 10

Summary and Future Work

Contents

10.1 Summary

In this thesis, we described an approach to automatize decision-support activities of knowledge workers (i.e., inspecting multi-dimensional data cubes, making decisions) that are typically carried out on a pre-determined schedule. The motivation for the approach taken is as follows: Decision-support activities are supported by separate information systems – called data warehouse (DW) and on-line analytical processing (OLAP) systems – that are decoupled from operational systems (on-line transaction processing – OLTP – systems) supporting day-to-day business activities. During the last years, it has been a general practice in OLTP system design to let these systems react autonomously (i.e., without explicit user intervention) to happenings/events that are of interest to the application domain (e.g., when the special offerings period ends, increase the prices of articles offered by 10 % etc.). Event-Condition-Action (ECA) rules are the means to represent such reactive mechanisms. If a database system supports the detection of events and the firing of ECA rules, it is called *active database system*. Although data warehouse and OLAP systems are supported by database systems too, the ECA paradigm has not yet been extended towards

supporting decision-support activities. This thesis described the necessary extensions to make data warehouses *active*.

10.1.1 Architecture

We described the architecture of *active data warehouses (ADW)*. Besides the conventional passive data warehouse architecture, which only supports loading data from operational systems and storing these data according to the multidimensional paradigm required by front-end OLAP tools, the architecture of an active data warehouse provides the following components:

1. A module with which analysis rules may be defined, stored, and executed.
2. A module in which events may be defined, stored, and signaled to analysis rules.
3. A module that provides information about actions that may be executed in OLTP systems as the "decisions" generated by analysis rules.
4. A module with which decisions (i.e., actions to be executed and parameters of these actions) may be exported to OLTP systems.
5. A front-end tool supporting the design/modification/evolution of analysis rules.

Active data warehouses realize "closed feedback-loops" since these systems support autonomous loading of data from OLTP systems, making decisions within the active data warehouse, and exporting these decisions back to OLTP systems. We coined the term "active data warehouse cycle", which represents one such loop.

10.1.2 Data Model

We introduced a novel multidimensional data model and an associated query language (*cube analysis statements/analytical expressions*), which may be used to describe multidimensional data on a conceptual/logical level. The strength of this data model is that it supports specifying cubes incrementally (i.e., cubes are derived from other, existing cubes), which is common practice by graphical front-end OLAP tools but is not supported by multidimensional data models. The query language supports analyzing such cubes by creating "virtual cells", that are derived from the cells of one or several other cubes. This data model and the query language are used as the basis to specify multidimensional analyzes to be carried out automatically by analysis rules.

10.1.3 Analysis Rules

We introduced *analysis rules*, which are an extension to ECA rules in order to automatize decision-support activities in data warehouses. Analysis rules are specified for some *primary dimension level*, which contains the objects (e.g., articles) for which decisions should be generated.

Event: Analysis rules may fire upon the occurrence of (i) relative temporal events (e.g., 20 days after an article's price has been changed), (ii) calendar events (e.g., end of quarter), (iii) logical events (e.g., 2nd monday of the month in which an article's price was changed).

Condition: Analysis rules evaluate several conditions on the objects in the primary dimension level using the data in the data warehouse. This "analytical task" is carried out hierarchically by incrementally refining the granularity of the data cubes inspected (e.g., sales figures of articles per year $\rightarrow$ sales figures of articles per month $\rightarrow$ sales figures of articles per day). The resulting graph of cubes that will be inspected when an analysis rule is executed is referred to as the rule's *analysis graph*.

Action: If a condition holds for some object of the primary dimension level, the rule's "decision" is to initiate the execution of a particular transaction in an OLTP system.

Beyond this principal structure, we have proposed a framework to specify flexible decision-making in active data warehouses:

Decision Criteria: Analysts are free to specify any combination of positive/negative/irresolute decision criteria at each cube of the analysis graph. If a *positive* decision criterion is satisfied, the decision to execute the rule's action for the analyzed object is generated. If a *negative* decision criterion is satisfied, the decision not to execute the rule's action for the analyzed object is generated. If an *irresolute* decision criterion is satisfied, the decision to analyze more detailed cubes is generated. For positive/negative decision criteria, we introduced the distinction between definite decision criteria and tentative decision criteria. While a *definite* decision criterion cannot be overruled by other decision criteria, a *tentative* decision criterion represent a "default" decision that may be overruled by other more specific decision criteria.

Decision-Making Models: Since each cube may generate its own "local decisions" as specified by decision criteria, the various decisions generated by the cubes of the analysis graph must be "coordinated" by a *decision-making model*. We presented an approach to specify various alternative decision-making models supporting the integration of positive definite, positive tentative, irresolute, negative tentative, and negative definite decisions generated by the sub-cubes of some cube $r(R)$.

Decision Parameters: Analysts are free to specify the bindings of decision parameters (i.e., the parameters of the analysis rule's action to be executed in an OLTP system) (i) as the result of analyzing some cubes of the analysis graph (i.e., individual parameter bindings), (ii) once for all positively decided objects of the primary dimension level (i.e., default parameter bindings), (iii) or as a combination of the two aforementioned approaches. These bindings may be determined either statically (i.e., using constant values) or dynamically (i.e., as the result of evaluating an analytical expression).

We described decision-making as presented above declaratively and procedurally. With the declarative semantics, we showed that decision-making is *complete* (i.e., a decision is generated for all objects in the primary dimension level) and *unique* (i.e., exactly one decision is generated for each object in the primary dimension level). The procedural semantics provides some generic procedures that may be used to implement our approach of decision-making free from any particular implementation data model. E.g., the procedures may be used to extend a multidimensional DBMS with the capabilities to execute analysis rules.

10.1.4 Implementation

Since many data warehouses are built upon relational database technology, we presented a mapping (i) of our multidimensional data model and the associated query language and (ii) of a basic model of analysis rules to the relational data model using *off-the-shelf database technology* (i.e., SQL, triggers, and materialized views). This mapping may be used as "kernel" implementation of decision-making within an active data warehouse.

10.2 Future Work

Since active data warehouses is a young field of research, there are several interesting questions that need to be investigated in future work. The most challenging questions are outlined below:

Flexible Loading: A basic assumption of this thesis is that the loading of data from operational systems is carried out at a constant interval (i.e., the duration of an ADW cycle cannot be changed). Analysis rules are fired after loading is finished. To improve the quality of decision-making, loading data from OLTP systems may be initiated by analysis rules, which require some "up-to-dateness" of the data warehouse before being executed (e.g., firing an analysis rule that analyzes the sales data of the last two weeks is reasonable only if the data warehouse provides these sales data).

Hence, the timepoints of loading data may be determined flexibly by the scheduled events that trigger analysis rules.

Incomplete Data: Another basic assumption of this thesis is that the data warehouse provides complete data, i.e., there are no missing facts and no null values. The quality of decision making will decrease if we assume that an active data warehouse may also provide some level of "incompleteness". Completing missing data using default values or using more complex inferences is necessary to guarantee that no "wrong decisions" are generated when an analysis rule is executed.

Rule Specialization: The primary dimension level may be compared with a base class from an object-oriented perspective providing all objects for which a decision should be generated. This base class may be specialized whereby each sub-class may represent a particular sub-set of objects for which a decision should be generated. Correspondingly, analysis rules could be specialized too, respecting the decisions generated by the analysis rule defined for the instances of the base class ("decision consistency"). We already motivated this extension in Section 6.5 but did not elaborate further since it is beyond the scope of this thesis.

Meta-Decision Rules: So far, analysis rules may defined without further coordination of decision-making processes. E.g., to accept and to reject a paper for a conference, two analysis rules may be specified. If a paper has been accepted after the first analysis rule (specified for activity accept) fired, the second analysis rule (specified for activity reject) need not fire for that paper. Coordinating such decision processes requires the introduction of *meta-decision rules.*

Process Integration: Active data warehouses generate decisions that are exported to operational systems where the corresponding transactions will be executed. These operational systems realize (parts of) business processes that impose several restrictions on the data maintained. Since an active data warehouse may initiate the execution of transactions in several systems, there is no possibility to coordinate these executions (e.g., in the OLTP system supporting stock-keeping, transaction changePrice may be executed whereas in the OLTP system supporting marketing activities, transaction changePrice is forbidden since the article is currently "on sale"). Executing actions in OLTP systems should be coordinated by a separate module that provides complete knowledge on the status of objects for which decisions have been generated.

Tool Support: Modeling analysis rules is carried out by end-users (i.e., analysts) not database designers. Hence, there is the need to provide a high-level graphical representation of analysis rules using various tools providing flexible and intuitive interaction with the rule base of an active data warehouse. Important requirements on these tools are: (i) interactive graphical top-down design of cubes to constitute the analysis graph, (ii) design of decision steps (specification of conditions, checking the correctness of conditions, etc.), (iii) high-level design of events (relative temporal events, calendar events, logical events), and (iv) browsing of generated decisions.

List of Figures

List of Tables

Bibliography

[AEASY97] D. Agrawal, A. El Abbadi, A. Singh, and T. Yurek. Efficient view maintenance in data warehouses. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 417–427, 1997.

[AGS97] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling Multidimensional Databases. In A. Gray and P. Larson, editors, *Proc. of the 13th Intl. Conf. on Data Engineering, April 7-11, 1997 Birmingham U.K*, pages 232–243. IEEE Computer Society Press, 1997.

[AHW95] A. Aiken, J.M. Hellerstein, and J. Widom. Static Analysis Techniques for Predicting the Behavior of Active Database Rules. *ACM Transactions on Database Systems*, 20(1):3–41, 1995.

[AM97] S. Anahory and D. Murray. *Data Warehousing in the Real World: A practical Guide for Building Decision Support Systems.* Addison-Wesley, 1997.

[BCP98] E. Baralis, S. Ceri, and S. Paraboschi. Compile-Time and Runtime Analysis of Active Behaviors. *TKDE*, 10(3):353–370, 1998.

[BDD+98] R. G. Bello, K. Dias, A. Downing, J. Feenan Jr., W. D. Norcott, H. Sun, A. Witkowski, and M. Ziauddin. Materialized Views in Oracle. In A. Gupta, O. Shmueli, and J. Widom, editors, *Proceedings of 24rd Int'l Conf. on Very Large Data Bases, August 24-27, 1998, New York City, USA*, pages 659–664. Morgan Kaufmann, 1998.

[BJS99] E. Bertino, S. Jajodia, and P. Samarati. A Flexible Authorization Mechanism for Relational Data Management Systems. *ACM Transactions on Information Systems*, 17:101–140, 1999.

[BL95] Mikael Berndtsson and Brian Lings. Logical Events and ECA Rules. Technical Report HS-IDA-TR-95-004, University of Sövde, 1995.

[BL97] A. Bauer and W. Lehner. The Cube-Query-Language (CQL) For Multidimensional Statistical and Scientific Database Systems. In *DASFAA'97,*

Proceedings of 5th International Conference on Database Systems for Advanded Applications, April 1-4, 1997, Melbourne, Australia, pages 263–272. World Scientific Press, 1997.

[BPT97] E. Baralis, S. Paraboschi, and E. Teniente. Materialized Views Selection in a Multidimensional Database. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 156–165. Morgan Kaufmann, 1997.

[CBS99] T. Connolly, C. Begg, and A. Strachan. *Database Systems: A Practical Approach to Design, Implementation, and Management*, chapter 25, pages 912–947. Addison-Wesley, 2nd edition, 1999.

[CCS93] E.F. Codd, S.B. Codd, and C.T. Sally. *Providing OLAP (On-Line Analytical Processing) To User Analysts: An IT Mandate*. Arbor Software Corporation, White Paper, 1993.

[CCS94] C. Collet, T. Coupaye, and T. Svensen. NAOS: Efficient and modular reactive capabilities in an Object-Oriented Database System. In *Proc. of the 20th Intl. Conf. on Very Large Databases, Santiago, Chile*, 1994.

[CD95] D. Cukierman and J. Delgrande. A Language to Express Time Intervals and Repetition. In *Int. Workshop on Temporal Representation and Reasoning TIME, Melbourne Beach, Florida*, April 1995.

[CD97] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *ACM SIGMOD Record*, 26(1):65–74, March 1997.

[CGMR95] S. Castangia, G. Guerrini, D. Montesi, and G. Rodriguez. Design and Implementiation for the Active Rule Language of Chimera. In N. Revell and A M. Tjoa, editors, *Proc. of the Workshop on Database and Expert Systems Applications (DEXA)*. OMNIPRESS, 1995.

[Cha95] S. Chakravarthy. Architectures and monitoring techniques for active databases: An evaluation. *Data & Knowledge Engineering*, 16(1):1–26, July 1995.

[Che75] Peter P. Chen. The enity-relationship model: Toward a unified view of data. In Douglas S. Kerr, editor, *Proceedings of the International Conference on Very Large Data Bases, September 22-24, 1975, Framingham, Massachusetts, USA*, page 173. ACM, 1975.

[CKAK93] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Anatomy of a Composite Event Detector. Technical Report UF-CIS-TR-93-039, University of Florida, Computer and Information Sciences, December 1993.

[CKAK94] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite Events for Active Databases: Semantics, Contexts and Detection. In *Proc. of the 20th Intl. Conf. On Very Large Databases (VLDB)*, 1994.

[CM94] S. Chakravarthy and D. Mishra. Snoop: An Expressive Event Specification Language for Active Databases. *Data & Knowledge Engineering*, 14(1):1–26, November 1994.

[Col96] G. Colliat. OLAP, Relational, and Multidimensional Database Systems. *SIGMOD Record*, 25(3):64–69, 1996.

[Cou97] OLAP Council. The OLAP Council MDAPI. http://www.olapcouncil.org/research/apily.htm, 1997.

[Cou00] OLAP Council. http://www.olapcouncil.org, 2000.

[CS94] R. Chandra and A. Segev. Implementing Calendars and Temporal Rules in Next Generation Databases. In *Proceedings of the 10th Int. Conference of Data Engineering (ICDE)*, February 1994.

[CT98a] L. Cabibbo and R. Torlone. A Logical Approach to Multidimensional Databases. In *EDBT'98, Proc. of 6th Intl. Conf. on Extending Database Technology, March 23-27, 1998, Valencia, Spain*, pages 183–197. Springer LNCS 1377, 1998.

[CT98b] L. Cabibbo and R. Torlone. From a Procedural to a Visual Query Language for OLAP. In *SSDBM'98, Proceedings of 10th International Conference on Scientific and Statistical Database Management, July 1-3, 1998, Capri, Italy*, pages 74–83. IEEE Computer Society Press, 1998.

[CT98c] L. Cabibbo and R. Torlone. Querying Multidimensional Databases. In S. Cluet and R. Hull, editors, *6th International Workshop on Database Programming Lanugages (DBPL-6)*, volume 1369 of *LNCS*, pages 319 – 335. Springer Verlag, August 1998.

[CW96a] S. Ceri and J. Widom. *Applications of Active Databases*, chapter 10, pages 259–291. In Widom and Ceri [WC96], 1996.

[CW96b] S. Ceri and J. Widom. *Standards and Commercial Systems*, chapter 9, pages 232–258. In Widom and Ceri [WC96], 1996.

[DHL90] U. Dayal, U. Hsu, and R. Ladin. Organizing Long-running Activities with Triggers and Transactions. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, 1990.

[DPW98] A. Dinn, N.W. Paton, and M.H. Williams. *RAP: The ROCK & ROLL Active Programming System*, chapter 18, pages 323–336. In Paton [Pat98], 1998.

[DPW99] A. Dinn, N.W. Paton, and M.H. Williams. Active Rule analysis and Optimisation in the ROCK & ROLL Deductive Object-Oriented Database. *Information Systems*, 24(4):327–353, 1999.

[DT99] A. Datta and H. Thomas. The cube data model: a conceptual model and algebra for on-line analytical processing in data warehouses. *Decision Support Systems*, 27(3):289–301, 1999.

[EG98] S.M. Embury and P.M.D. Gray. *Database Internal Applications*, chapter 19, pages 339–366. In Paton [Pat98], 1998.

[EN00] R. Elmasri and S.B̃. Navathe. *Fundamentals of Database Systems.* Addison Wesley, 3rd edition, 2000.

[Eze99] C.I. Ezeife. Selecting and materializing horizontally partitioned warehouse views. *Data & Knowledge Engineering*, 36(2):185–210, January 1999.

[FP98] P. Fraternali and S. Paraboschi. *Chimera: A Language for Designing Rule Applications*, chapter 17, pages 309–322. In Paton [Pat98], 1998.

[GCB+97] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, and M. Venkatrao. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *Data Mining and Knowledge Discovery*, 1(1):29 – 53, March 1997.

[GD93] S. Gatziu and K.R. Dittrich. Events in an Active Object-Oriented Database System. In N. Paton and M. Williams, editors, *Proc. of the 1st Intl. Workshop Rules in Database Systems (RIDS)*, pages 23 – 39. Springer Verlag, August 1993.

[GD94] S. Gatziu and K. R. Dittrich. Detecting Composite Events in Active Database Systems Using Petri Nets. In J. Widom and S. Chakravarthy, editors, *Proc. of the 4th Intl. Workshop on Research Issues in Data Engineering (RIDE)*, Active Databse Systems, pages 2–9. IEEE Computer Society, February 1994.

[GHRU97] H. Gupta, V. Harinarayan, A. Rajaraman, and Jeffrey D. Ullman. Index selection for olap. In Alex Gray and Per-Åke Larson, editors, *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997 Birmingham U.K*, pages 208–219. IEEE Computer Society, 1997.

[GJS92] Narain H. Gehani, H. V. Jagadish, and Oded Shmueli. Composite event specification in active databases: Model & implementation. In Li-Yan Yuan, editor, *18th International Conference on Very Large Data Bases, August 23-27, 1992, Vancouver, Canada, Proceedings*, pages 327–338. Morgan Kaufmann, 1992.

[GL97] M. Gyssens and Laks V. S. Lakshmanan. A Foundation for Multi-dimensional Databases. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 106–115. Morgan Kaufmann, 1997.

[GL98] F. Gingras and L. V. S. Lakshmanan. *nD − SQL*: A Multi-Dimensional Language for Interoperability and OLAP. In A. Gupta, O. Shmueli, and J. Widom, editors, *VLDB'98, Proc. of 24rd Intl. Conf. on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 134–145. Morgan Kaufmann, 1998.

[GM99] A. Gupta and I.S. Mumick, editors. *Materialized Views: Techniques, Implementations, and Applications.* MIT Press, 1999.

[GMR98] M. Golfarelli, D. Maio, and S. Rizzi. Conceptual Design of Data Warehouses from E/R Schemes. In H. El-Rewini, editor, *HICSS'98, Proc. of the 31st Hawaii Intl. Conf. on System Sciences, Volume VII, 1998*, pages 334–343. IEEE Computer Society Press, 1998.

[Han96] Eric N. Hanson. The Design and Implementation of the Ariel Active Database Rule System. *TKDE*, 8(1):157–172, 1996.

[HK00] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*, chapter 2, pages 39–104. Morgan Kaufmann Publishers, 1st edition, 2000.

[HN99] Eric N. Hanson and Lloyd X. Noronha. Timer-Driven Database Triggers and Alerters: Semantics and a Challenge. *ACM SIGMOD Record*, 28(4):11–16, December 1999.

[HRU96] V. Harinarayan, A. Rajaraman, and Jeffrey D. Ullman. Implementing data cubes efficiently. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, pages 205–216. ACM Press, 1996.

[HZ96] R. Hull and G. Zhou. A framework for supporting data integration using the materialized and virtual approaches. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, pages 481–492. ACM Press, 1996.

[Inm96] W.H. Inmon. *Building the Data Warehouse, Second Edition.* John Wiley, 1996.

[JLS99] H. V. Jagadish, L. V. S. Lakshmanan, and D. Srivastava. What can Hierarchies do for Data Warehouses? In *VLDB'99, Proceedings of 25th Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 530–541. Morgan Kaufmann Publishers, 1999.

[KCGS93] W. Kim, I. Choi, S. Gala, and M. Scheevel. On Resolving Schematic Heterogeneity in Multidatabase Systems. *Distributed and Parallel Databases*, 1(3):251–279, July 1993.

[KLRSR95] G. Kappel, P. Lang, S. Rausch-Schott, and W. Retschitzegger. Workflow management based on objects, rules, and roles. *Data Engineering Bulletin*, 18(1):11–18, 1995.

[KS91] W. Kim and J. Seo. Classifying Schematic and Data Heterogeneity in Multidatabase Systems. *IEEE Computer*, 24(12):12–18, December 1991.

[KS95] R. Kimball and K. Strethlo. Why Decision Support Fails and How To Fix It. *SIGMOD Record*, 24(3):91–97, September 1995.

[Leh98] W. Lehner. Modeling Large Scale OLAP Scenarios. In *EDBT'98, Proc. of 6th Intl. Conf. on Extending Database Technology, March 23-27, 1998, Valencia, Spain*, pages 153–167. Springer LNCS 1377, 1998.

[LEW96] Jae Y. Lee, R. Elmasri, and J. Won. Specification of calendars and time series for temporal databases. In Bernhard Thalheim, editor, *Conceptual Modeling - ER'96, 15th International Conference on Conceptual Modeling, Cottbus, Germany, October 7-10, 1996, Proceedings*, volume 1157 of *Lecture Notes in Computer Science*, pages 341–356. Springer, 1996.

[LMSS95] James J. Lu, G. Moerkotte, J. Schü, and V.S. Subrahmanian. Efficient maintenance of materialized mediated views. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, pages 340–351. ACM Press, 1995.

[LOS96] P. Lang, W. Obermair, and M. Schrefl. Situation Diagrams. In R.R. Wagner and H. Thoma, editors, *Proc. of the 7th Intl. Conf. on Database and Expert Systems Applications*, LNCS 1134, pages 400–421. Springer, September 1996.

[LOS97] P. Lang, W. Obermair, and M. Schrefl. Modeling Business Rules with Situation/Activation Diagrams. In A. Gray and P. Larson, editors, *Proc. of the 13th Intl. Conf. on Data Engineering (ICDE)*, pages 455–464. IEEE Computer Society Press, April 1997.

[LSPC00] W. Lehner, R. Sidle, H. Pirahesh, and R. Cochrane. Maintenance of automatic summary tables. In Weidong Chen, Jeffrey F. Naughton, and Philip A.

Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, volume 29, pages 512–513. ACM, 2000.

[LW96] C. Li and X. S. Wang. A Data Model for Supporting On-Line Analytical Processing. In *CIKM'96, Proceedings of 5th International Conference on Information And Knowledge Management, 1996, Rockville, Maryland*, pages 81–88. ACM Press, 1996.

[MD89] D. R. McCarthy and U. Dayal. The Architecture of an Active Data Base Management System. volume 18 of *SIGMOD Record*, pages 215–224. ACM Press, May 1989.

[Mic01] Microsoft. OLE-DB for OLAP. http://www.microsoft.com/data/oledb/olap/spec/, 2001.

[MK00] M. Mohania and Y. Kambayashi. Making aggregate views self-maintainable. *Data & Knowledge Engineering*, 32:87 – 109, 2000.

[MPC96] R. Meo, G. Psaila, and S. Ceri. Composite Events in Chimera. In *Proc. of the Intl. Conf. on Extending Database Technology (EDBT)*, LNCS 1057. Springer, March 1996.

[MQM97] Inderpal S. Mumick, D. Quass, and Barinderpal S. Mumick. Maintenance of data cubes and summary tables in a warehouse. In Joan Peckham, editor, *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 100–111. ACM Press, 1997.

[MS01] J. Melton and A.R̃. Simon. *SQL:1999 – Understanding Relational Language Components*. Morgan Kaufmann Publishers, 2001.

[Nol99] C. Nolan. Manipulate and Query OLAP Data Using ADOMD and Multidimensional Expressions. Microsoft Systems Journal, aug 1999.

[Obe98] W. Obermair. *Active Object-Oriented Databases: From Conceptual Design to Logical Design*. PhD thesis, Universität Linz, Institut für Wirtschaftsinformatik, 1998.

[Ora99] Oracle 8i Online Documentation, Release 8.1.5, 1999.

[Pat98] N. Paton, editor. *Active Rules in Database Systems*. Springer Verlag, 1998.

[PC95] N. Pendse and R. Creeth. *The OLAP-Report: Succeeding with On-Line Analytical Processing*, volume 1. Business Intelligence, 1995.

[PD99] N. Paton and O. Diaz. Active database systems. *ACM Computing Surveys*, 31(1):63 – 103, March 1999.

[PJ99] T. B. Pedersen and C. S. Jensen. Multidimensional Data Modeling for Complex Data. In *ICDE'99, Proceedings of 15th International Conference on Data Engineering, March 23-26, 1999, Sydney, Australia*, pages 336–345. IEEE Computer Society Press, 1999.

[Red97] RedBrick. Decision-Makers, Business Data, and RISQL. White Paper, 1997.

[RJB99] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual.* Addison Wesley, 1999.

[RPS98] S. Reddi, A. Poulovassilis, and C. Small. *PFL: An Active Functional DBPL*, chapter 16, pages 297–308. In Paton [Pat98], 1998.

[RSC98] K.A. Ross, D. Srivastava, and D. Chatiziantoniou. Complex Aggregation at Multiple Granularities. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT), March 23-27, 1998, Valencia, Spain*, pages 263 – 277. Springer LNCS 1377, 1998.

[SBH99] C. Sapia, M. Blaschka, and G. Höfling. An Overview of Multidimensional Data Models for OLAP. Technical Report FR-1999-001, Bayerisches Forschungszentrum für Wissensbasierte Systeme, Erlangen, München, Passau, feb 1999.

[SBHD98] C. Sapia, M. Blaschka, G. Höfling, and B. Dinter. Extending the e/r model for the multidimensional paradigm. In Dik Lun Lee, Ee-Peng Lim, Mukesh K. Mohania, and Yoshifumi Masunaga, editors, *Advances in Database Technologies, ER '98 Workshops on Data Warehousing and Data Mining, Mobile Data Access, and Collaborative Work Support and Spatio-Temporal Data Management, Singapore, November 19-20, 1998, Proceedings*, volume 1552 of *Lecture Notes in Computer Science*, pages 105–116. Springer, 1998.

[SQL99] ANSI/ISO/IEC 9075-2-1999, Database Languages – SQL – Part 2: Foundation (SQL/Foundation), 1999.

[SS92] M. Soo and R. Snodgrass. Multiple Calendar Support for Conventional Database Management. Technical Report TR 92-07, Department of Computer Science, Univ. of Arizona, Tuscon, AZ 85721, February 1992.

[ST00] M. Schrefl and T. Thalhammer. On Making Data Warehouses Active. In M. Mohania and A. Min Tjoa, editors, *Proc. 2nd Intl. Conf. on Data Warehousing and Knowledge Discovery (DaWaK), Greenwhich, London (UK), September 4-6, 2000.* Springer LNCS, 2000.

[Sys95] Red Brick Systems. Star Schemas and STARjoin Technology. Technical report, Red Brick Systems, Los Gatos, CA, 1995.

[TBC99] N. Tryfona, F. Busborg, and J. G. B. Christiansen. starER: A Conceptual Model for Data Warehouse Design. In *Proceedings ACM Second International Workshop on Data Warehousing and OLAP (DOLAP), November 2-6, 1999, Kansas City, Missouri, USA*, pages 3–8. ACM Press, 1999.

[TP98] J. Trujillo and M. Palomar. An Object Oriented Approach to Multidimensional Database Conceptual Modeling (OOMD). In *Proceedings of the First International Workshop on Data Warehousing and OLAP (DOLAP), November 7, 1998, Washington DC, USA*. ACM Press, 1998.

[TSM01] T. Thalhammer, M. Schrefl, and M. Mohania. Active Data Warehouses: Complementing OLAP with Analysis Rules. *To appear in Journal Data & Knowledge Engineering*, 2001.

[Vas98] P. Vassiliadis. Modeling Multidimensional Databases, Cubes and Cube Operations. In M. Rafanelli and M. Jarke, editors, *10th Intl. Conf. on Scientific and Statistical Database Management, Proceedings, Capri, Italy, July 1-3, 1998*, pages 53–62. IEEE Computer Society Press, 1998.

[VS99] P. Vassiliadis and T. Sellis. A Survey of Logical Models for OLAP Databases. *ACM SIGMOD Record*, 28(4):64–69, December 1999.

[WC96] J. Widom and S. Ceri, editors. *Active Database Systems—Triggers and Rules For Advanced Database Processing.* Morgan Kaufmann Publishers, 1996.

[ZB98] J. Zimmermann and A.P. Buchmann. *REACH*, chapter 14, pages 263–277. In Paton [Pat98], 1998.

[ZGMHW95] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View maintenance in a warehousing environment. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, pages 316–327. ACM Press, 1995.

Curriculum Vitae

Personal Record

Name:	Mag. Thomas Thalhammer
Date of Birth:	08 Mai 1973
Place of Birth:	Salzburg
Citizenship:	Austria
Home Address:	A-5202 Neumarkt, Hauptstraße 66

University Education

1992–1997: Study of Business Information Systems at the Johannes Kepler University of Linz, Austria

1998–2001: Ph.D. study at the Johannes Kepler University of Linz, Austria

Professional Experience

1993–1996:	Tutor for courses in algorithms, data modeling, and object-oriented database systems at the Johannes Kepler University of Linz
1998–2000:	Teaching in data modeling, database systems, and internet programming at the Fachhochschule Hagenberg
1998–onwards:	Teaching in data modeling and database systems at the Johannes Kepler University, Linz
1998 onwards:	Teaching and research associate ("Universitätsassistent") at the Department of Business Information Systems, Data & Knowledge Engineering, at the Johannes Kepler University of Linz, headed by Prof. Michael Schrefl