

# Data Warehouse Schema Design\*

Jens Lechtenbörger  
Dept. of Information Systems  
University of Münster  
Leonardo-Campus 3  
D-48149 Münster, Germany  
lechten@wi.uni-muenster.de

## 1 Introduction

A data warehouse is an integrated database primarily used in organizational decision making. Although the deployment of data warehouses is current practice in modern information technology landscapes, the methodical schema design for such databases has only been studied cursorily. In this paper we pursue schema design for data warehouses in the spirit of classical database design, organized as a sequence of *requirement analysis and specification* to collect user requirements, *conceptual design* to model the data warehouse as a multidimensional database independently from implementation issues, *logical design* to transform the conceptual data warehouse schema into the target data model, and *physical design* to maximize performance with respect to a target database system.

In short, from a conceptual point of view a data warehouse is a multidimensional database, and *fact schemata*, such as the one shown in Figure 1, represent such databases conceptually. In Figure 1, we have a fact schema `Account` from the banking domain, where *measures* `Balance`, `BalanceClass`, and `NoOfTransactions` are shown in a two-dimensional context of *dimensions* `Time` and `Account`. Each dimension is specified by means of a lattice of *dimension levels*, whose bottom element is called *terminal* dimension level (here: `Day`, `AccID`). Importantly, domains of *optional* dimension levels may contain inapplicable null values, and *context dependencies* specify *contexts of validity* for optional dimension levels (e.g., `Age` is applicable to private customers, whereas `LegalForm` is applicable to capital companies, and annotations in the schema indicate these facts).

Given a fact schema  $F$ , we call the attributes occurring in  $F$  the *universe of  $F$* , denoted by  $U_F$ , and we note that there is a set of functional dependencies over  $U_F$ , called the *functional dependencies implied by  $F$* , denoted by  $FD_F$  (e.g., the set of terminal dimension levels functionally determines the set of measures, and each edge in a dimension lattice indicates a functional dependency).

By contrast, from a technical point of view a data warehouse for one or more (operational)

---

\*This paper is an extended abstract of [Lec01].

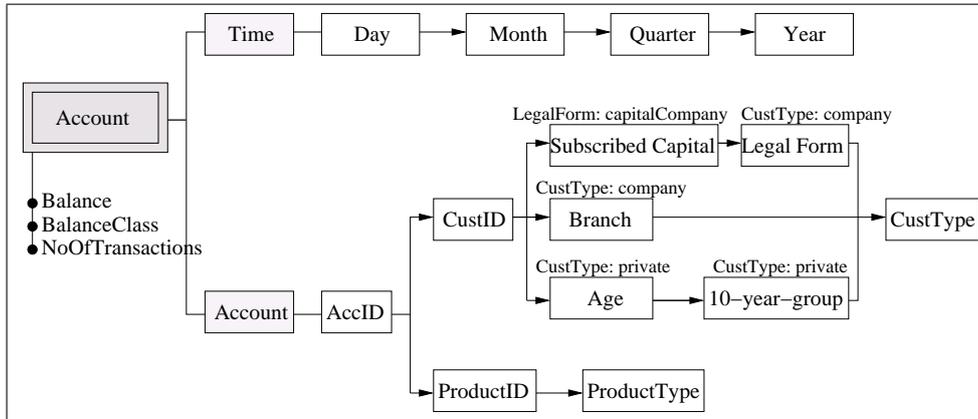


Figure 1: Sample conceptual schema.

data sources is a separate database consisting of a set of materialized views that integrate data from the sources. For our purposes, we view a data warehouse as a set of UPSJR views, i.e., relational views defined by union, projection, selection, join, and renaming.

We restrict our attention to the design process up to and including logical design, the relational data model as the target model to be used during logical design, and we emphasize that we do not address design issues that are related to, e.g., cleansing, maintenance, or meta-data management. Our aim is to define a data warehouse design process based on the *extension* of independently developed data warehouse design approaches and their *integration* into the traditional database design process. For this purpose, we strive (1) to identify and formalize desirable warehouse schema properties as design goals and (2) to set up conceptual and logical design phases in such a way that the identified design goals are guaranteed to be fulfilled.

The remainder of this paper is organized as follows. In Section 2, we present the design goals of multidimensional normal forms and independence, which are addressed by a design process outlined in Section 3. In Sections 4, 5, and 6, we summarize requirement analysis, conceptual design, and logical design as phases of this design process, and in Section 7 we sketch a method to enforce update independence in the course of logical design. We conclude the paper in Section 8.

## 2 Design Goals

While normal forms have a long tradition in the area of relational databases as guidelines for good schema design, research on quality factors for data warehouses or multidimensional databases has started only recently.

## 2.1 Summarizability and Multidimensional Normal Forms

The work presented by Lenz and Shoshani [LS97], which provides necessary conditions for summarizability, can be regarded as a first step to define quality factors for conceptual data warehouse schemata. The notion of *summarizability* refers to the possibility to compute aggregate values with a lower level of detail from aggregate values with a higher level of detail. Roughly speaking, summarizability is given if an analyst is guaranteed to obtain consistent results when performing sequences of roll-up operations along a path in a dimension lattice. Lenz and Shoshani [LS97] argue that summarizability is of most importance for queries concerning multidimensional data. Hence, any multidimensional schema should be set up in such a way that summarizability is obtained to the highest possible degree, and violations of summarizability should be expressed in the schema. Following and extending the summarizability results presented in [LS97], Lehner et al. [LAW98] define the *generalized multidimensional normal form* (GMNF), which ensures summarizability in the presence of optional dimension levels in a context-sensitive manner and supports an efficient physical database design.

We extend the framework of [LAW98] by showing that there is a natural correspondence between optional dimension levels and attributes occurring in sub-classes of generalization hierarchies in the sense of Smith and Smith [SS77], and we define three increasingly restrictive multidimensional normal forms to address this correspondence (see also [LV02b]).

We briefly summarize the impact of those three multidimensional normal forms. Let  $F$  be a fact schema with universe  $U_F$ , and let  $FD_U$  be a set of functional dependencies over  $U_F$  that can be observed in the application domain. Associated with  $F$  there is a second set of functional dependencies over  $U_F$ , namely the functional dependencies implied by  $F$ ,  $FD_F$ . We argue that  $F$  can only be regarded as reasonable schema for the application domain if certain relationships among  $FD_U$  and  $FD_F$  hold, and these relationships are made formally precise in terms of first multidimensional normal form (1MNF). Roughly, 1MNF demands that (1) the functional dependencies  $FD_F$  do occur in the application domain, (2) the potential for roll-up and drill-down operations available in the application domain is preserved in the fact schema, and (3) the fact schema embodies the multidimensional contexts of measures.

While the first of these requirements is a natural *semantic faithfulness* condition that should be satisfied in any database, the following one is specific to the context of multidimensional data, where it formalizes a *completeness* aspect with respect to coverage of the underlying application domain. Finally, it can be shown that the last of these requirements *avoids* certain kinds of *redundancies* that arise due to transitive functional dependencies; hence, this last requirement sets up a loose connection between traditional normal forms and multidimensional ones.

Next, 2MNF strengthens 1MNF by taking care of optional dimension levels in order to guarantee *summarizability* in a context-sensitive manner, just as GMNF does. Thus, if a schema is in 2MNF then an analyst (or an analysis tool for that matter) can identify optional dimension levels based on schema information, which has two consequences.

(1) If an analyst considers measures in conjunction with optional dimension levels, then she is aware of the fact that she is only looking at partial information. (2) Contradictory queries (e.g., group company customers according to their jobs) can be avoided.

Finally, 3MNF places further restrictions on schemata in 2MNF, and these restrictions are sufficient to *construct* a class hierarchy concerning dimension levels, which is implicitly contained in a fact schema, in terms of relation schemata that *avoid null values*. This normal form will be our formal guideline to measure the quality of *conceptual* data warehouse schemata. In the next subsection, we turn towards properties of *logical* data warehouse schemata.

## 2.2 Independence Properties

From a technical point of view we regard a data warehouse as a relational database that stores a set of materialized relational views. These materialized views have to be maintained in order to keep track of updates in the operational databases, and self-maintainability has been identified as desirable property that can be exploited during view maintenance to avoid maintenance queries and update anomalies.

Roughly, a set of materialized views is *self-maintainable* if it is possible to determine the new view instance after every database update only based on the old view instance and the update information itself [GJM96]. Since a logical data warehouse schema is defined in terms of a set of materialized views, the property of self-maintainability can immediately be perceived as property of logical data warehouse schemata. In the data warehousing context this property appears particularly attractive for the following reasons: A data warehouse typically integrates information from a variety of heterogeneous information sources. In such a scenario, it might already be difficult to extract deltas representing source changes, but querying the sources is almost impossible [GJM96]. Indeed, information sources may not provide query interfaces or may not permit ad-hoc queries. Even if querying of sources is possible then the associated maintenance algorithms can incur processing delays with undesirable loads on operational systems, and such queries can cause maintenance anomalies. Clearly, these problems do not occur if the data warehouse is self-maintainable.

So far, we have argued that a data warehouse should be self-maintainable. According to the terminology introduced by Laurent et al. [LLSV01], self-maintainability is also called update independence, because a self-maintainable data warehouse is independent from the information sources as far as the consistent integration of updates is concerned. Moreover, the concept of update independence can be extended to queries as well. Intuitively, a data warehouse is query-independent, if every query to the sources can be answered using the data warehouse views *only*. The motivation for enabling data warehouses to answer queries that could also be posed directly to the sources is similar to the motivation for update independence, i.e., direct querying of sources may be impossible or too expensive. Therefore, a data warehouse should not only be update-independent but also query-independent with respect to the queries that are important to the warehouse users.

### 3 Data Warehouse Design

Traditional database design proceeds in a sequence of conceptual, logical, and physical design steps, where each step results in a corresponding database schema. Importantly, this separation of design phases allows to reason about different aspects of a database system at varying levels of abstraction. There is a growing consensus that this separation of design phases known from traditional database design is also advantageous in the context of data warehouse design. Nevertheless, existing data warehouse design processes lack clearly defined design goals. In particular, previous design processes are unaware of desirable schema properties such as multidimensional normal forms and independence. Indeed, we argue that conceptual data warehouse schemata should satisfy 3MNF, whereas logical data warehouse schemata should be at least update-independent.

In the following sections we outline a data warehouse design process that comprises the phases of traditional database design and addresses the design goals of normal forms and independence. We assume that data warehouse design starts with the conceptual design, which is divided into requirement analysis and design of the conceptual schema; afterwards, logical and physical design are carried out in separate phases. As even a cursory treatment of the physical design phase is beyond the scope of this work, we refer the reader to [BG01], where an initial set of pointers to relevant research on this subject can be found.

### 4 Requirement Analysis and Specification

In the first phase of data warehouse design the data requirements to be met by the forthcoming data warehouse have to be analyzed and specified. As opposed to the requirement analysis performed during traditional database design, data warehouse design aims at the integration of a number of pre-existing operational data sources. Hence, the schemata describing these sources form a major input to the requirement analysis.

In the course of the requirement analysis, data warehouse designers, warehouse end users, and business domain experts select relevant data warehouse attributes and define initial OLAP queries based on the information found in operational database schemata. We neglect a detailed description of *how* the requirement analysis can be accomplished; instead, we focus on *what* the requirement specification should deliver in order to support the schema design process. To this end, we propose to structure the requirement specification in terms of a set of initial multidimensional or OLAP queries and a derivation and usage table, which describes the identified relevant data warehouse attributes.

The multidimensional queries can be seen as yardstick for the functionality of the data warehouse under design, while the derivation and usage table contains an informal description for each identified warehouse attribute, specifies how its values are derived from the operational databases, and indicates whether the attribute may be used as measure or dimensional attribute. Afterwards, each identified warehouse attribute has to be classified either as measure or as dimension level or as property attribute. Then, to prepare the design goal of multidimensional normal forms, optional dimension levels must be distinguished

from mandatory ones, and a context of validity has to be identified for each optional level. In this respect we propose to determine a *basis* for the functional dependencies among warehouse attributes, and we show how such a basis helps to identify (a) measures, (b) certain problems related to dimension levels, and (c) contexts of validity.

## 5 Conceptual Design

The conceptual design phase performs a transformation of the semi-formal requirement specification into a conceptual schema using a novel, formalized multidimensional data model. The formalization results in a multidimensional diagram such as the one shown in Figure 1, which comprises fact schemata with their related measures and dimension lattices in an intuitive graphical notation. We propose an algorithmic design process to derive fact schemata starting from the requirement specification and functional dependencies of the operational schemata, and we prove that the resulting schemata satisfy 3MNF.

As shown in Figure 2, we structure the process of conceptual data warehouse design into three sequential phases: design of initial fact schemata, dimensional lattice design, and definition of summarizability constraints (see also [HLV00]).

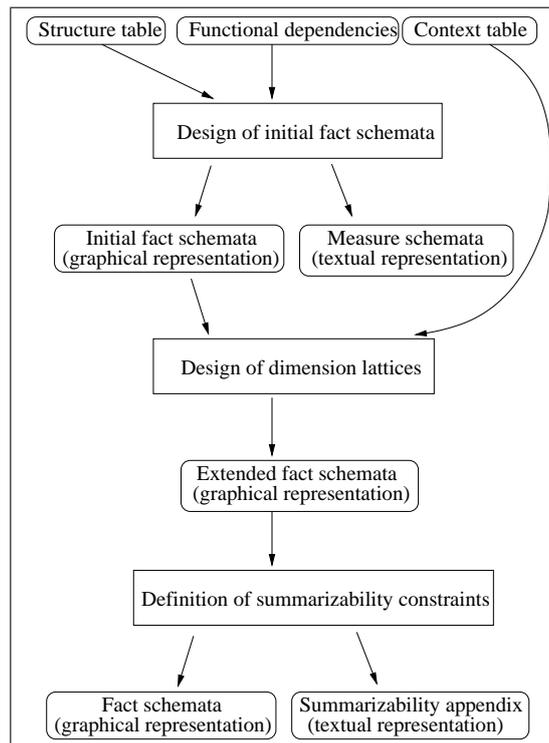


Figure 2: Conceptual schema design process.

The definition of initial fact schemata starts from the requirement specification, and the goal is to generate initial fact schemata that contain only terminal dimension levels and measures, thereby specifying the multidimensional context of measures. We just remark that this phase rests upon an analysis of functional dependencies among measures and candidate dimension levels. In the next phase, we gradually develop the dimension lattice for each terminal dimension level. To this end, we augment each initial schema with additional dimension levels by “chasing” functional dependencies starting from the terminal dimension level until no further changes arise. The final phase is about the definition of summarizability constraints. Roughly, all meaningful combinations of measures, dimension levels, and aggregate functions are enumerated.

Let  $F$  be an output schema according to the above procedure. Then we have:

**Theorem 1.**  *$F$  is in third multidimensional normal form.*

## 6 Logical Design

The conceptual schema as derived according to the previous section is now going to be transformed into a logical data warehouse schema. For this purpose, we present a transformation process that starts from a set of fact schemata in 3MNF and produces a set of view definitions which constitute an update-independent data warehouse.

The overall transformation process is sketched in Figure 3. In a first step, a relational database schema  $\mathbf{D}_W$  is generated, which expresses the information modeled by the conceptual fact schemata on a logical level in terms of relation schemata and foreign key constraints. Taking advantage of input fact schemata in 3MNF, null values are avoided in relation schemata representing dimension levels. Afterwards, the resulting database schema is linked to the operational databases. To this end, a set of UPSJR views  $W$  over the operational databases is derived such that the materialization of  $W$  implements the database schema  $\mathbf{D}_W$ . Finally, the views  $W$  thus obtained are rendered update-independent by adding a suitable amount of auxiliary information to  $W$ , which ends the logical schema design process. Roughly, we use a so-called view complement [BS81] as key tool to derive a suitable amount of auxiliary information that renders the data warehouse update-independent. For this purpose, expressions to compute complements for UPSJR views are given. Importantly, these expressions are applicable to a larger class of views than those offered by earlier approaches and lead at the same time to uniformly smaller complements; furthermore, the complexity of constructing these expressions is shown to be polynomial in the size of schema information, which is in striking contrast to previous approaches, which are NP-complete (see also [LV02a]).

Besides, we propose a complement-based method to guarantee independence of a set of views with respect to an arbitrary set of operations, which can be applied to enforce independence properties of pre-existing warehouses or to support warehouse evolution (see also [LLSV01]).

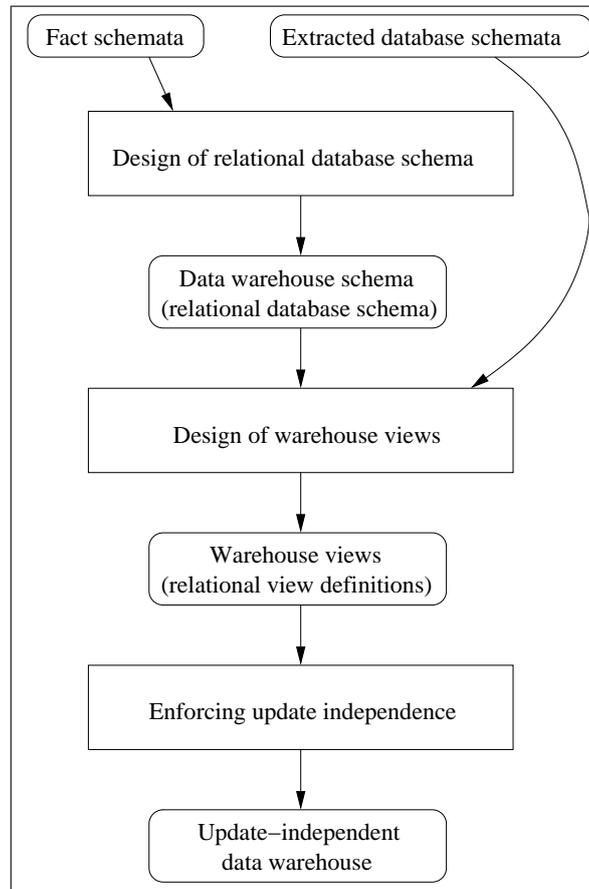


Figure 3: Logical schema design process.

## 7 Update Independence

We are now going to demonstrate how update independence of the data warehouse under design can be enforced in the course of the logical design phase based on the notion of view complement.

While it is well-known that single views involving projections or joins are almost never update-independent [GJM96], we now state the only known necessary condition concerning update independence:

**Proposition 2.** *Let  $D$  be a set of relation schemata without foreign keys, and let  $V$  be a set of views over  $D$ .*

1. *Let  $V_0 =_{df} \pi_X(R) \in V$  be a view over  $D$ . If  $V$  is update-independent then it is possible to derive a duplicate counter for all tuples in instances of projection  $V_0$ .*

2. Let  $V_0 =_{df} R_1 \bowtie R_2 \in V$  be a view over  $D$ . If the domains of all key attributes of  $R_1$  and  $R_2$  are infinite then the following holds: If  $V$  is update-independent then the information of  $R_1$  and  $R_2$  is completely contained in  $V$ .

Based on Proposition 2 we pursue the following approach to achieve the design goal of update independence by means of (a) modifying view definitions and (b) augmenting the warehouse with auxiliary views. Let  $D$  denote a set of (exported) relation schemata, and let  $V^0$  denote a set of UPSJR views over  $D$  defining the warehouse schema.

In view of statement (1) of Proposition 2, we check for each view in  $V^0$  whether it preserves the keys of all accessed operational relation schemata or not. If a view preserves all keys (possibly in renamed form) then it remains unchanged. Otherwise, we augment the projection with a duplicate counter. Let  $V^1$  denote the set of views obtained from  $V^0$  using this transformation.

To address statement (2) of Proposition 2, we add complementary views to the warehouse to ensure that relation schemata involved in joins can be computed from warehouse views. To this end, we perform the following transformation on each view  $U_j \in V^1$ . If  $U_j$  involves the union operation, i.e., if  $U_j$  is of the form  $\bigcup_{i=1}^{u_j} V_{j_i}$ , where each  $V_{j_i}$  is a PSJR view, then we change the definition of  $U_j$  into a “source-preserving union”. Let  $V^2$  denote the set of source-preserving UPSJR views obtained from  $V^1$  using the previous transformation. We now derive a set  $A_{ui}$  of auxiliary views to render  $V^2$  update-independent. For this purpose, we compute a complement  $C$  of  $V^2$  with respect to  $D$ . For each relation schema  $R$  that is involved in a join in some view in  $V^2$  we add the complementary views for  $R$  to  $A_{ui}$ .

**Theorem 3.** *Using the above notation, let  $W = V^2 \cup A_{ui}$  be a set of views over  $D$ . Then  $W$  is update-independent.*

## 8 Concluding Remarks

This work advances data warehouse schema design beyond a process based on experience and rules of thumb towards a formal framework where rigorous methods are available to achieve clearly defined design goals. However, the present approach could be refined and extended in a number of aspects. First, we have neglected physical schema design, which is a mandatory part of any database implementation to optimize the logical schema for performance reasons. Second, meta-data management is largely ignored, and a systematic study concerning the co-design of warehouse schemata and meta-data repository still needs to be performed. Next, we have not paid attention to special issues that arise when temporal data is stored or analyzed, and the design of data warehouses using temporal data models does not seem to have been studied yet. Finally, we have shown how a data warehouse can be rendered update-independent based on rewritings that add duplicate counters to warehouse views and the additional storage of complementary views to deal with updates concerning join views. Additionally, Proposition 2 suggests that this additional amount of information is even necessary, if join views do not involve projections. In general,

however, it remains an open problem to determine a minimal amount of information that is necessary to render a set of views update-independent.

## References

- [BS81] F. Bancilhon, N. Spyrtos, "Update Semantics of Relational Views," *ACM TODS* 6, 1981, 557–575.
- [BG01] A. Bauer, H. Günzel, eds., *Data Warehouse Systeme — Architektur, Entwicklung, Anwendung*, dpunkt.verlag, 2001.
- [BLT86] J.A. Blakeley, P. Larson, F.W. Tompa, "Efficiently Updating Materialized Views," *Proc. ACM SIGMOD* 1986, 61–71.
- [GJM96] A. Gupta, H.V. Jagadish, I.S. Mumick, "Data Integration using Self-Maintainable Views," *Proc. 5th EDBT* 1996, LNCS 1057, 140–144.
- [HLV00] B. Hüsemann, J. Lechtenbörger, G. Vossen, "Conceptual data warehouse modeling," *Proc. DMDW* 2000: 6.
- [LLSV01] D. Laurent, J. Lechtenbörger, N. Spyrtos, G. Vossen, "Monotonic Complements for Independent Data Warehouses," *VLDB Journal* 10 (4), Springer-Verlag, 2001, 295–315.
- [Lec01] J. Lechtenbörger, *Data Warehouse Schema Design*, Inaugural-Dissertation zur Erlangung des Doktorgrades der Naturwissenschaften im Fachbereich Mathematik und Informatik der Mathematisch-Naturwissenschaftlichen Fakultät der Westfälischen Wilhelms-Universität Münster; available as volume 79 in "Dissertationen zu Datenbanken und Informationssystemen," Akademische Verlagsgesellschaft Aka GmbH, Berlin, 2001.
- [LV02a] J. Lechtenbörger, G. Vossen, "On the Computation of Relational View Complements," *Proc. 21st PODS* 2002, 142–149.
- [LV02b] J. Lechtenbörger, G. Vossen, "Multidimensional Normal Forms for Data Warehouse Design," 2002, to appear in *Information Systems*, Elsevier Science.
- [LAW98] W. Lehner, J. Albrecht, H. Wedekind, "Normal Forms for Multidimensional Databases," *Proc. 10th SSDBM* 1998, 63–72.
- [LS97] H. Lenz, A. Shoshani, "Summarizability in OLAP and statistical databases," *Proc. 9th SSDBM* 1997, 132–143.
- [SS77] J.M. Smith, D.C.P. Smith, "Database Abstractions: Aggregation and Generalization," *ACM TODS* 2, 1977, 105–133.