Book overview, and data warehousing

Terry Halpin
Microsoft Corporation

Abstract: This article provides a brief overview of my new book on information modeling, and includes an excerpt that discusses data warehousing from an ORM perspective. The material quoted from the book remains copyrighted to Morgan Kaufmann Publishers, and is reproduced here by permission.

Introduction

This article is in response to a personal request from the editor of the *Journal of Conceptual Modeling* to write a brief article about my new book, *Information Modeling and Relational Databases*. First an overview is presented. As an example of one of the new topics, an excerpt is then provided that discusses data warehousing from an ORM perspective. The material quoted from the book remains copyrighted to Morgan Kaufmann Publishers, www.mkp.com, and is reproduced here by permission.

Book overview

The book discusses how to model information at a conceptual level where it is easily understood, map this model to a relational database, and then work with the implemented database using a relational language such as SQL or a conceptual language such as ConQuer. It is written primarily for database practitioners, as well as students of computer science or information management. To cater for both audiences, I've tried to write in a readable way, without losing precision or rigor.

The printed version of the book comprises 792 pages, with approximately 200 pages of additional material available online, making a total of about 1000 pages. To help the reader digest all this material, each of the 13 chapters includes a concise summary. In addition, glossaries are provided to summarize the notations of the main modeling techniques covered (ORM, Barker ER, IE, IDEF1X and UML).

The first two chapters introduce high level modeling approaches, then provide an historical and structural overview of conceptual architectures and development frameworks for information systems. Chapters 3 through 7 discuss in detail how to use Object Role Modeling (ORM) to develop a conceptual schema for an information system.

Chapter 8 discusses the Entity Relationship (ER) approach, starting with Chen's original notation then moving on to the three most popular notations in current use: the Barker ER notation supported by Oracle Corporation; the Information Engineering notation; and the IDEF1X notation, which is actually a hybrid of ER and relational notations. Comparisons with ORM are included along the way.

Chapter 9 examines the use of UML class diagrams for data modeling, including a detailed comparison with ORM. Business rule constructs in ORM with no graphic counter-part in UML are identified then captured in UML using user-defined constraints or notes.

Chapter 10 describes how a conceptual model may be implemented in a relational database system. Chapter 11 covers relational algebra and the essence of relational database systems, then provides a detailed treatment (90 pages) of SQL, with attention to the SQL-89, SQL-92 and SQL:1999 standards as well as some popular dialects.

Chapter 12 discusses how and when to transform one schema into another schema at the same level (conceptual or logical). Many patterns are discussed to enable conceptual schemas to be reshaped, and optimization guidelines are given to select the transformations. Normalization and denormalization are then discussed. The optimization techniques are then used to facilitate database re-engineering.

Chapter 13 examines other modeling issues, methods and trends. Topics covered include data warehousing, OLAP, conceptual query languages, schema abstraction mechanisms, process modeling (e.g. UML use cases and activity diagrams, data flow diagrams), post-relational databases (e.g. object databases and object-relational databases) and metamodeling.

Throughout, the book adopts a "cookbook" approach, with plenty of diagrams and examples. Each chapter begins with a brief overview, and ends with a chapter summary of the major points covered, with chapter notes to provide fine points and further references. There are hundreds of carefully graded exercise questions to help readers master the concepts discussed. The printed book ends with symbol glossaries, a bibliography and a comprehensive index.

Additional material is available online at the publisher's website for free download: see http://www.mkp.com/books_catalog/catalog.asp?ISBN=1-55860-672-6. This includes three appendices. Appendix A provides an overview of the evolution of computer hardware and software. Appendix B discusses two kinds of subtype matrix that can be used to determine subtype graphs from significant populations. Appendix C discusses advanced aspects of SQL, focusing on set-comparison queries and group extrema queries.

The answers to the exercises are contained in two files, one for the odd-numbered questions (54 pages) and one for the even-numbered questions (50 pages). The odd answers are openly accessible, but the even answers have restricted access, in order to provide classroom instructors with a range of exercises for classroom discussion. Electronic versions of the figures, as well as an Instructor's Guide with further exercises and related pedagogic material, are available to instructors.

If you are familiar with my previous book, *Conceptual Schema and Relational Database Design*, you will find that the new book has been essentially rewritten from scratch, so the changes are significant. Whole new chapters have been added (e.g. ER and UML), there are many new topics (e.g. data warehousing, OLAP, conceptual queries), several topics are covered in much greater depth (e.g. SQL) and there are many new exercises. The conceptual schema design procedure has been refined, the ORM notation has been brought more into line with that supported by Microsoft tools, and the treatment of all topics has been improved and updated. The new content has led to a much larger book, even after moving the answers to the web.

To widen the audience of the book to the industrial community, the writing style is also less academic. The following excerpt from section 13.2 illustrates one of the new topics, but assumes you are already familiar with ORM and relational databases. The bibliographical entries for the cited references, as well as the rest of section 13.2, which deals with OLAP, are omitted in this article.

Data warehousing from an ORM perspective (excerpt from section 13.2)

Most commercial information systems are built to support heavy volumes of transactions by many users on a daily basis. Examples include banking, insurance and order processing systems. These *Online Transaction Processing (OLTP)* systems typically require quick throughput for their largely predefined range of transactions, especially update transactions. To improve performance, historical data is often archived once it reaches a certain age, reducing the size of the data sets used for daily operations. A single organization may have several OLTP systems (e.g. purchasing, sales, inventory, customer records), possibly implemented using different kinds of DBMS or other software applications, and the coupling between such systems may be weak or even non-existent.

Over time, businesses became aware that the collective information contained in their various systems had great potential for analyzing market trends and improving their business processes. However their OLTP systems were unsuitable for executives to perform this task, given the poor performance and complex interface for *ad hoc* analysis queries (e.g. aggregated multi-table joins, and nested correlated subqueries). Moreover, insufficient integration or history made some queries simply impossible. Partly to address the problem of integrating data from pre-relational systems for analysis purposes, IBM proposed the notion of an "information warehouse". Although performance problems delayed acceptance of this idea for some years, a later proposal for a "data warehouse" by Bill Inmon (1993) was enthusiastically embraced by the business community, and nowadays most large companies already have, or are building, a data warehouse.

A *data warehouse* is an enterprise-wide, integrated, historical database of information extracted from individual data sources for the purpose of supporting analysis of the business by management. During analysis it is read-only. Since the patterns and trends sought from the analysis tend to evolve slowly, and some imprecision is acceptable, updates are performed in bulk according to an agreed schedule (e.g. weekly).

The construction of a data warehouse for a large enterprise can be a lengthy task. To exploit the benefits as soon as possible, the data warehouse is often built iteratively, one subject area at a time. As

subject areas are added to the data warehouse, they may be used to load data marts. A *data mart* is a smaller "departmental warehouse" focused on one subject area, often containing more summarized and less detailed data than the data warehouse. For end users who perform their analysis within one subject area, a data mart provides a simpler model adapted to their needs, and the smaller data volume often leads to better performance. This overall approach is diagrammed in Figure 1.

Many different approaches to data warehousing exist. Sometimes, data marts are built first and used to incrementally construct the data warehouse. However, if an analytical query may span multiple subject areas, it is critical that an overall enterprise architecture be in place to make the appropriate connections.

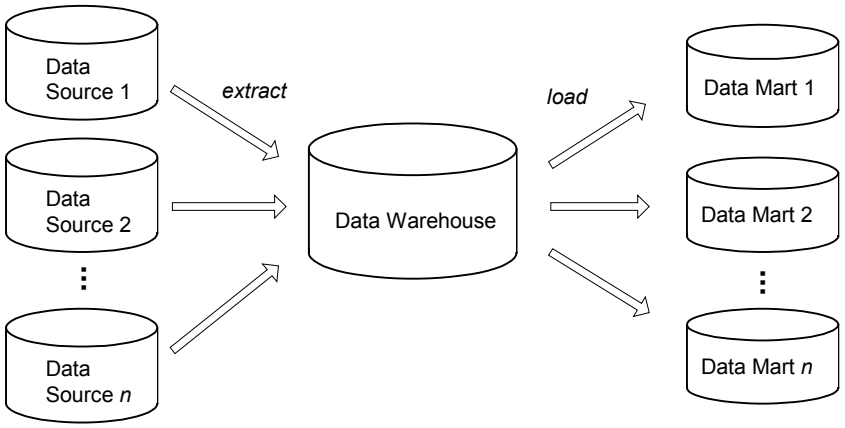

Figure 1 Data is extracted into the data warehouse then loaded into data marts

It has been argued that the data warehouse should be implemented as a fully normalized relational model, based directly on the enterprise data model, with no summary data, postponing any denormalization and aggregation to the data marts loaded from the warehouse (e.g. Moody & Kortink 2000). Current practice however usually does incorporate denormalized and summarized data in the data warehouse (Silverston et. al. 1997).

Before extracting data to the data warehouse, the business analysis needs should be determined and the relevant data sources identified. The data sources may include operational databases as well as spreadsheets and legacy systems. Any details in the data sources irrelevant to the analysis needs should be removed. For example, the phone number and street address of a store are probably not of interest, but its city probably is. The remaining data now needs to be transformed to remove errors (applying integrity rules). For example, a sexcode field with lots of "F" and "M" entries might include a few instances of "D", "G", "J" or "N". Assuming these are typographical errors, can you make a good guess as to the intended letter? (Hint: look at the keyboard).

We must also ensure that all facts of the same type are represented in the same way. As a trivial example, a customer's birthdate might be recorded as a character string in a 'DOB' field in one source, and elsewhere in a 'birthdate' column based on a date data type. Once the data is "cleansed" it is transformed into a uniform representation in the data warehouse (typically a relational database).

To facilitate the analysis of historical trends, appropriate temporal data should be included, at the desired granularity (e.g. daily, weekly or monthly). For example, suppose an operational data source stores the fact type: Employee manages Store. Over time, a store may have different managers. To retain history of these changes, when store management facts are loaded into the data warehouse the load date may be inserted into the key, to populate the historical fact type: Employee on Date managed Store. Temporal modeling in general is an interesting topic, but a full treatment of it is beyond the scope of this book.

To improve query performance, data marts (and usually the data warehouse) often contain derived data and denormalized structures. As a simple example of derived data, suppose an operational source stores the fact type Customer was born on Date. For demographical analysis, we may be interested in how product preferences are influenced by the age of customers. In this case it may be more appropriate in a data mart to store the age, or even just the age group, of the customer rather than their birthdate, using a fact type such as: Customer on Date belonged to AgeGroup. The snapshot dates are inserted when the fact type is incrementally updated.

A more typical example incorporating both derivation and denormalization is a data mart for analyzing Sales trends. An ORM conceptual model of such a mart (simplified) is shown in Figure 2. In this UoD, the company makes sales from many stores, located in various cities within the same country (e.g. the USA).

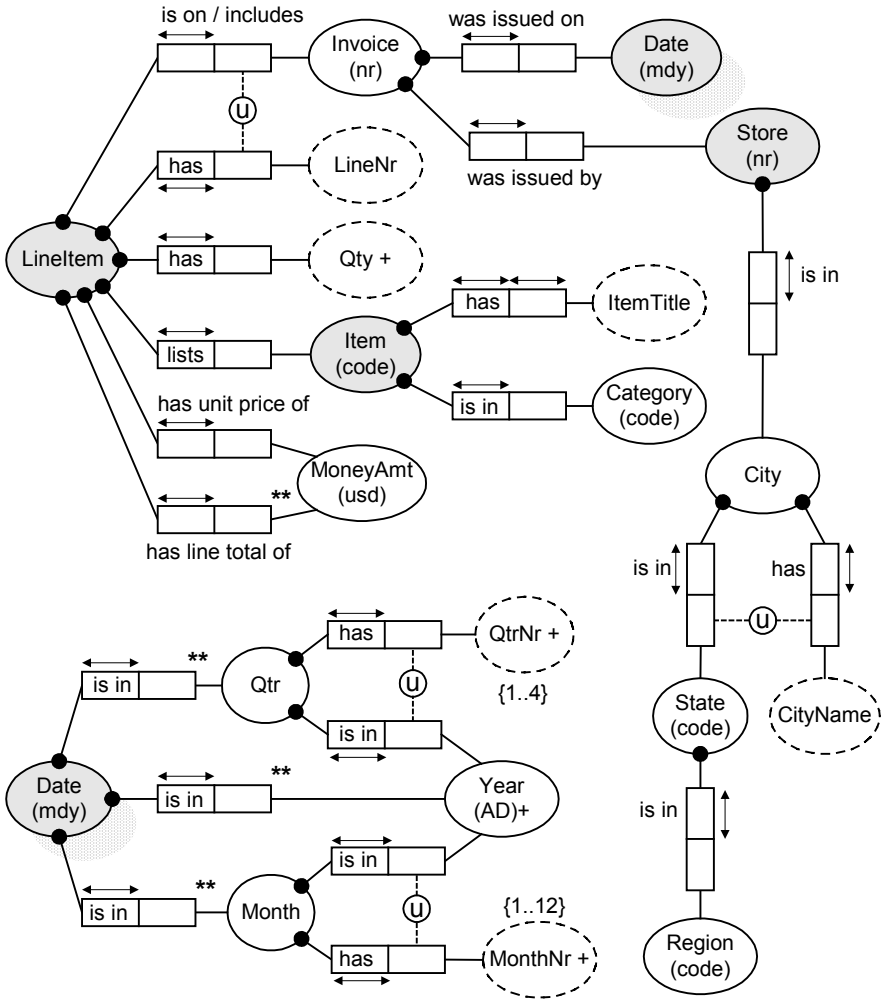

Figure 2 A conceptual schema for the Sales data mart (derivation rules omitted)

A city is identified by combining its name and state (e.g. Portland, Oregon differs from Portland, Maine). States are grouped into regions (e.g. Oregon and Washington belong to the NorthWest region). Items sold have a unique code and title, and belong to a category (e.g. Developer Tools). A line item is identified by its line number on a given invoice, and records the sale of an item, as well as the quantity and unit price for that item. The line total for each line item is derived by multiplying the quantity by the unit price, and then stored. To support sales analysis over time, the month number, quarter number and year for each date is derived and stored. Although calendar years are used here, we could use fiscal years instead or as well.

In Figure 2, fact types that are derived and stored are marked "**", but for simplicity the derivation rules are omitted. Using the default Rmap algorithm, this conceptual schema would map to the following six table schemes: *LineItem*(invoiceNr, lineNr, itemCode, qty, unitPrice, lineTotal); *Invoice*(invoiceNr, saleDate, storeNr); *Item*(itemCode, itemTitle, category); *Store*(storeNr, stateCode, cityName); *StateLocation*(stateCode, region); *TimeDimension*(saleDate, saleYear, QtrNr, MonthNr).

However, to improve query performance it is decided to denormalize the relational schema to four table schemes, as shown in Figure 3. The Item and TimeDimension tables are normalized, but the Sale and

Store tables are not. The Sale table is denormalized to 1NF, since saleDate and storeNr are functionally dependent on invoiceNr, which is just part of the primary key. The Store table is denormalized to 2NF since region is functionally dependent on stateCode, a non-key attribute.

The decision to denormalize in this way is indicated by annotating the conceptual schema, as shown in Figure 2. Here the *key object types* are *shaded*. An object type is a key object type if and only if its primary identification scheme is used as the primary key of a table. Graphically, each key object type forms the root of a tree, where each node is an object type and each edge is a functional (*n*:1 or 1:1) predicate. For example, from the Store object type we run down functional chains to the leaf object types CityName and Region, gathering all the fact types on the way to group them into a single table based on the identifier for Store.

A functional chain stops if it runs into a key object type (or a leaf or a non-functional predicate). For example, starting at LineItem we gather up all its functional fact types, as well as those for Invoice, but we cannot proceed past Date, Store or Item, since these are key object types. This leads to the Sale table in Figure 3.

The denormalized Sale and Star tables contain embedded functional dependencies (e.g. stateCode → region), but there is no need to enforce these because they have already been enforced in the operational tables from which the data mart is derived. Since the operational tables are used for base updates, and not the data mart, it is acceptable to denormalize the data mart in this way. Reducing the number of tables eliminates the need for many joins, leading to faster queries.

The schema in Figure 3 is composed of a central table (Sale) linked by foreign key connections to outer tables (Item, Store and TimeDimension). This is called a *star schema*, since the central table may be viewed as the center of a star pattern, with its outer tables becoming "points of the star". In data warehousing terminology, the *central table* is called a "*fact table*" and the outer tables are *dimension tables*. Since all tables contain facts, the term "fact table" is rather inappropriate here. Moreover, ORM uses the term "fact table" to mean an elementary fact table. To avoid confusion, I'll use the more descriptive "central table" instead of the more popular "fact table" in this context.

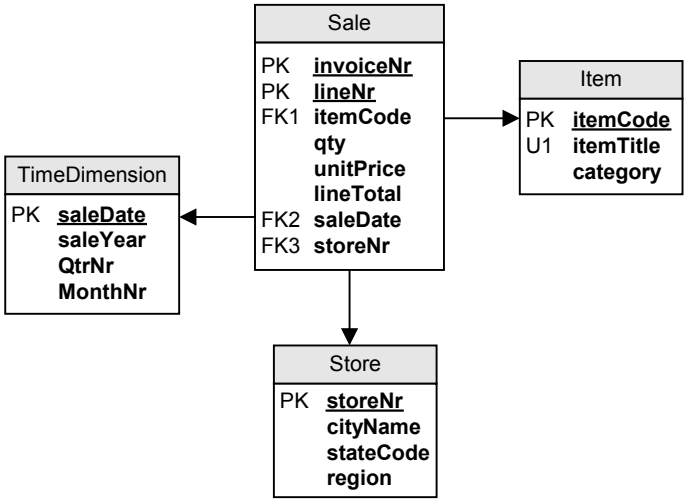

Figure 3 A denormalized, relational star schema for the model in Figure 2

Some approaches require the primary key of the central table to include all the keys of its dimension tables. This would require the (invoiceNr, lineNr) key of the Sale table in Figure 3 to be expanded to the superkey (invoiceNr, lineNr, itemCode, saleDate, storeNr). From a purely logical standpoint this is not required, since joins can be made on non-key attributes.

If some dimension tables are themselves used as central tables for other stars, the overall schema is called a "snowflake schema". A set of star schemas with shared dimension tables is sometimes called a "galaxy". … (end of excerpt)

Conclusion

I hope that the overview and brief excerpt gave you a reasonable idea of the book's contents. John Zachman, Prof. Robert Meersman and Prof. Gordon Evererst were kind enough to write glowing forewords, a bunch of reviewers and editors helped to whip my manuscript into shape, and our dedicated team at the Enterprise Frameworks and Tools Unit at Microsoft have worked their butts off to ensure that a conceptual modeling method as good as ORM has a first class modeling tool to match. This tool, called Visio For Enterprise Architects, will be included in Microsoft Visual Studio .NET Enterprise Architect, planned for release in late 2001. If you have any constructive comments on the book, please e-mail me at my private e-mail address: WytLytPub@att.net. For technical questions about Microsoft's ORM technology, please use my work e-mail address: TerryHa@microsoft.com .