A Graphical Notation for Physical Database Modelling

Abstract

In this paper we describe a graphical notation for physical database modelling. This notation provides database administrators with a means to model the physical structure of new and existing databases, thus enabling them to make more proactive and informed tuning decisions, compared to existing database monitoring tools.

1 Introduction

As with most information systems, the design and implementation of a database goes through several phases, including conceptual, logical and physical modelling [6]. These three phases are of particular interest, as they embody the progression from higher to lower levels of abstraction [16]. Conceptual models are typically highly abstract, using techniques such as entity-relationship modelling. Logical models represent the database structure in a form that is closer to the physical representation, yet still sufficiently abstract to isolate applications from the physical representation [7], and are expressed using formalisms such as the relational model. A logical model for a database can be derived by transforming the corresponding conceptual model.

Physical models represent the database structure in terms of the internal physical storage implementation of a specific database management system (DBMS) such as Oracle or DB2. A physical model for a database can be derived by transforming the corresponding logical model [4, 8]. Because of their low level of abstraction, physical level database models have tended to not be expressed using graphical notations, unlike models at higher levels of abstraction.

Physical level modelling, however, is equally as important as, if not *more* important than the higher levels, because it is the physical level that determines the performance of a database [6]. It is therefore somewhat surprising that there have been relatively few attempts to devise a graphical physical modelling notation, because such a notation can provide several advantages [8, 5, 18]:

- it can reduce complexity and thus improve understandability [17];
- it can provide a more complete and integrated display of performance tuning techniques in a database;
- database developers can be more confident about the design decisions that they make for the performance of the database;
- database performance problems are more easily visualised using a graphical notation; and
- a specific methodology is developed and used, thus enabling developers to resolve physical performance issues more systematically.

These benefits are embodied in modern database performance monitoring tools, which provide higher-level visualisations of a database's internals in order to easily identify and highlight performance problems. Such tools, however, are primarily *monitoring* tools rather than *design* tools. They may therefore unintentionally encourage database administrators (DBAs) into a *reactive* mode of continually "tweaking" the database to resolve performance issues, rather than a *proactive* mode of anticipating and designing for expected usage. It may also be difficult for a DBA using such tools to gain a clear and comprehensive overview of all the tuning techniques that are in use within a particular database [9].

In this paper we propose a graphical notation for physical database modelling. In Section 2, we provide a brief overview of commonly used physical tuning techniques. We then discuss in Section 3 two earlier approaches upon which our work is partially based. Section 4 introduces our proposed notation, and Section 5 discusses possible future work. The paper concludes in Section 6.

2 Physical tuning techniques

Database management is generally an I/O bound task, so the main performance bottleneck in most databases will be the performance of tertiary storage devices such as disk drives. Retrieving data from a hard disk is theoretically about six orders of magnitude slower than retrieving data from RAM[1]. The aim of any physical tuning strategy must therefore be to minimise the impact of slow tertiary storage, either by directly reducing the number of physical disk accesses required, or by parallelising access to disk in order to reduce contention.

These considerations have led to the development of five general physical tuning techniques, which are implemented to various degrees by most modern mainstream DBMS products:

[1] On the order of milliseconds (10^{-3}) for disk versus nanoseconds (10^{-9}) for RAM.

Indexes reduce the number of physical disk accesses required to retrieve a specific record [14], most typically by building a B+-tree [11] based on some key value within the data. Without any indexes, a DBMS often has little choice but to perform a sequential scan in order to locate a specific record. If we assume one disk access per database block, a sequential scan has average and worst case performance of $K/2b$ and $(K-1)/b$ disk accesses required, respectively, where K is the number of records and b is the number of records per database block. By comparison, a B+-tree has a worst case performance of $\log_{n/2}(K)$ [15], where n is the number of key values per index node.

Hashing is a method of quickly locating specific records by passing a key value to a *hash function*. This function ideally returns a unique physical location of a hash bucket, which contains a pointer to the associated physical record (check). Hashing schemes typically require only one or two physical disk accesses to retrieve a specific record, and perform best for exact key matches on very large tables. Hashing generally performs poorly for queries that require the retrieval of multiple records.

Clustering minimises disk access by ensuring that related records (such as an order header and its associated order lines) are physically adjacent on disk. This usually means that related records will be stored in the same database block, and can thus be retrieved with a single disk access. Clustering can, however, be expensive to maintain in a high update environment.

Partitioning provides parallel access paths to data by physically splitting a table into disjoint parts (either vertically or horizontally) and placing them on separate disks. This is particularly advantageous when multiple users wish to access different subsets of a set of records, because it provides a separate physical access path to each of the partitions. Partitioning can also reduce the number of disk accesses required, because there are fewer records to scan in each partition than if the table were not partitioned.

Replication provides parallel access paths to data by making multiple copies of the same records and placing them on separate disks. This is particularly advantageous when multiple users need to access the same sets of records, but is more complex to manage due to the need to keep replicas synchronised.

These techniques are normally applied to different parts of a database to achieve different effects. In order to choose an appropriate physical tuning technique, the DBA must consider various factors that may benefit only

some users of the database, or may improve the performance of the database as a whole. While most of the techniques can be combined to varying degrees, simply applying all techniques is usually not optimal, because each technique excels under different conditions. That is, what are optimal conditions for one technique may be the exact opposite for another, so the DBA needs to be able to model all the available information in order to develop an appropriate physical design.

3 Prior physical modelling techniques

To achieve an effective physical design requires a large amount of information, particularly with regard to the predicted or actual volume and usage of data within the database [6]. Incorporating this information into a graphical model can provide a more concise and clearer overview of the physical aspects of a database system. In this section we briefly discuss two previous efforts at modelling such information in a graphical manner.

3.1 Agile modelling (Ambler)

Ambler proposed a physical modelling notation based on the Unified Modelling Language (UML), as part of a larger effort to produce a "traditional" style data modelling profile for the UML [2, 3]. Ambler and others have argued the need for such a profile for some time [1, 13].

Ambler's notation focuses on the physical modelling of relational databases. The notation uses class boxes without stereotypes to represent physical tables, while indexes are represented by class boxes with the stereotype `<<index>>`, as illustrated in Figure 1. There appear to be no stereotypes for other physical tuning techniques such as partitioning, although these could be easily incorporated.

Ambler's approach suffers from two serious disadvantages. First, the notation is very limited in the types of symbol used. All physical level constructs are represented by class boxes, which in a complex diagram could make distinguishing them difficult. This limitation probably arises from the constraints on developing a new notation within the existing UML framework.

Second, his approach appears to consistently confuse the logical and physical levels of abstraction: the same notations are used to represent not only physical but also logical and conceptual elements [2]. This confusion is illustrated by the inclusion of a view (a non-physical construct) in Figure 1.

In summary, while Ambler's notation graphically models the physical level of a database, the similarity of the graphical symbols and the evident confusion between the physical and logical levels diminish its usefulness.

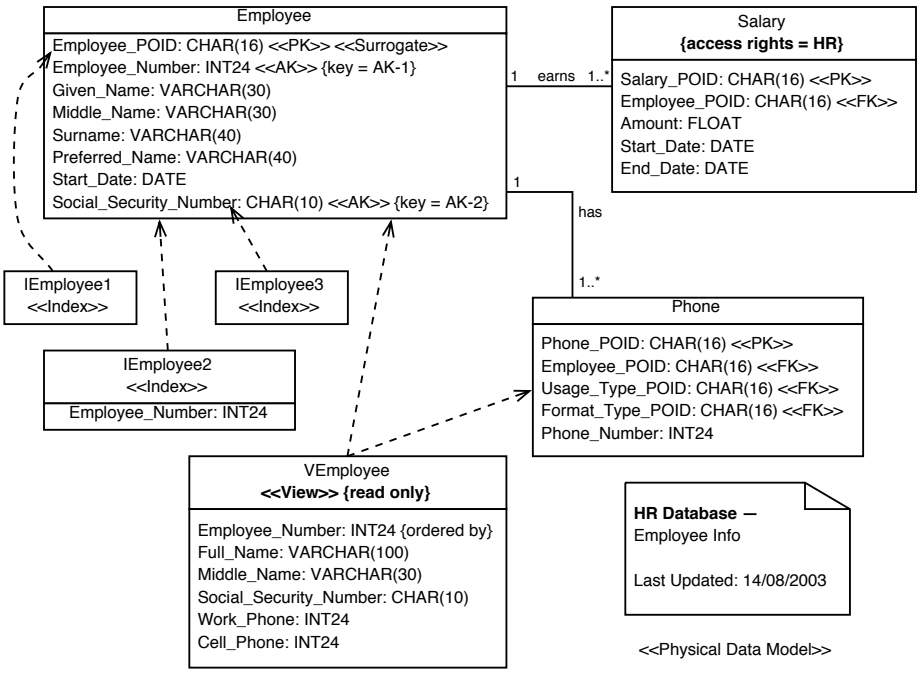


Figure 1: Ambler's physical modelling notation (adapted from [2]).

3.2 Physical design using an entity model (Beynon-Davies)

Beynon-Davies proposed a method for analysing and modelling the physical usage patterns of a database [5]. In his method, various aspects of the physical performance of a database are measured, such as the size and expected growth rates of tables (volume analysis), the volatility of tables, and the frequency of transactions (usage analysis). The data obtained from these analyses are then used to annotate a logical level entity-relationship diagram (ERD) of the database, producing what is known as a *composite usage map* (see Figure 4 on page 9 for an example).

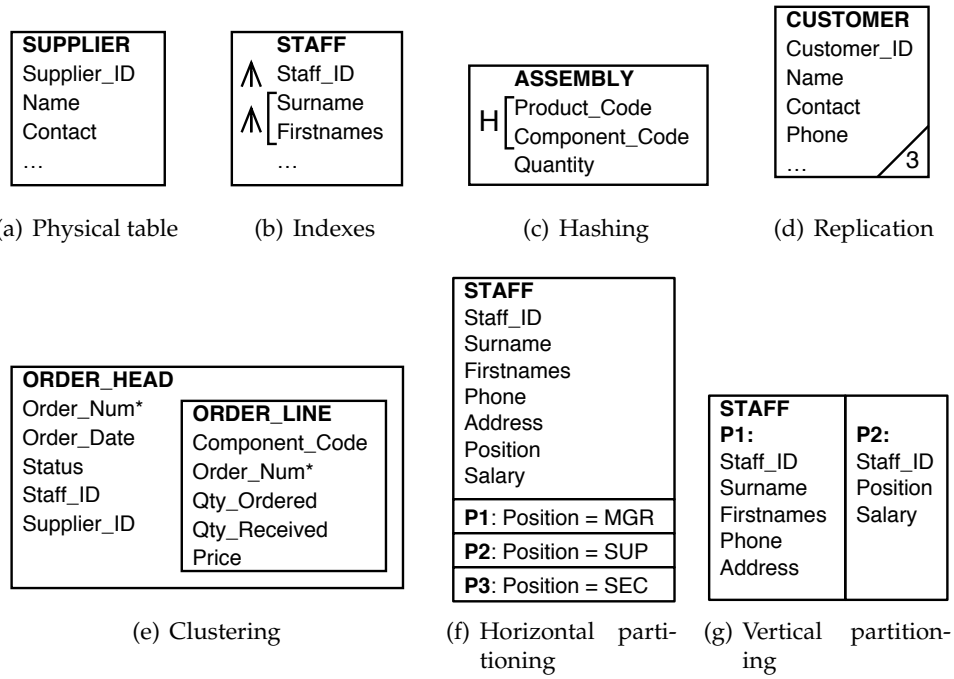
Beynon-Davies' method provides a very good mechanism for representing the usage statistics of a database in a coherent manner, but is rather complex and time-consuming to undertake without some form of automation. Our experience with teaching this method at undergraduate level shows that even with a relatively small database, the designer can quickly become overwhelmed by the sheer volume of usage data involved.

In addition, Beynon-Davies' method does not produce any conclusions as to which physical tuning methods should be implemented—rather it summarises the information required to enable these decisions to be made. Beynon-Davies' method is thus more a notation for summarising the physical usage patterns of a database, rather than a notation for physical modelling per se.

4 A new physical notation

Both of the notations discussed in the previous sections are limited in their ability to graphically model the physical level of a database. Ambler's notation lacks clarity and is thus potentially confusing, while Beynon-Davies' notation only summarises the physical usage patterns of a database rather than providing an actual physical level data model. We have therefore adopted aspects from both approaches to devise a graphical notation that enables database designers to graphically model the common physical database tuning techniques discussed in Section 2.

The symbols that we have adopted for this notation are shown in Figure 2. Some of these are adapted from other notations, while some we have created ourselves. The symbols have been chosen to be intuitive and simple to draw, so as to produce diagrams that are as clear and uncluttered as possible. Physical models may be developed using this notation either with or without a prior Beynon-Davies style analysis.



(a) Physical table (b) Indexes (c) Hashing (d) Replication

(e) Clustering (f) Horizontal partitioning (g) Vertical partitioning

Figure 2: Proposed symbols for physical tuning techniques.

A physical table is represented by a simple box, as shown in Figure 2(a). This is similar to most logical and conceptual level ERD notations. The fields of the physical table may be included, with or without physical data types, as appropriate. It could be argued that a different symbol should be used to avoid confusion between, for example, physical tables and concep-

tual entities. However, because these constructs belong to different levels of abstraction, they should not both appear on the same diagram, and so there is no real potential for confusion.

B-tree indexes are represented by a small tree-like symbol within a table, as shown in Figure 2(b). The index key is listed next to this symbol. Composite keys are indicated by a grouping symbol. Hashing is represented in a similar way, but uses an "H" symbol instead of a tree symbol (see Figure 2(c)). These notations could easily be extended to cater for other types of index, such as bitmap or R-tree indexes.

Clustering is represented by nesting one table inside another, as shown in Figure 2(e) (adapted from [5]). The cluster key is indicated by an asterisk (*) attached to the appropriate field(s). Tables may be nested to as many levels as required in order to represent more complex clustering schemes. This notation is intuitive, and clearly indicates the field(s) on which the records are clustered.

Partitioning is represented by splitting a table into either vertical or horizontal partitions according to the style of partitioning, as shown in Figure 2(f) and 2(g) (adapted from [15]). Once again, the notation is intuitive, and allows the partition definitions to be easily indicated.

Replication is indicated by placing a diagonal bar across the bottom-right corner of the table to be replicated, along with the total number of replicas, as shown in Figure 2(d). This is adapted from a similar notation used in data flow diagrams [10]. This notation could also be used to indicate replication of individual table partitions, for DBMSs that permit this combination.

Consider the entity-relationship diagram shown in Figure 3, which uses Martin notation [12] to depict a database for a consumer electronics manufacturer. A corresponding Beynon-Davies' composite usage map based on the fourteen most significant transactions (see the appendix on page 12) is shown in Figure 4 on page 9. The arrows represent physical access paths, while the number attached to each access path indicates the number of disk accesses per hour along that path. The diagram clearly highlights some potential performance problem areas in the database, for example:

- There are many disk accesses along the access paths between the Sale_head/Sale_line and the Order_head/Order_line tables. Since each pair of tables will normally be accessed together, both pairs could perhaps be candidates for clustering (depending on the mix of update versus read operations).

- There appear to be multiple transactions accessing the Staff table. This could imply a need for partitioning.

- There is an extremely high access rate on the Customer table. Further examination, however, reveals that this rate only occurs for a short

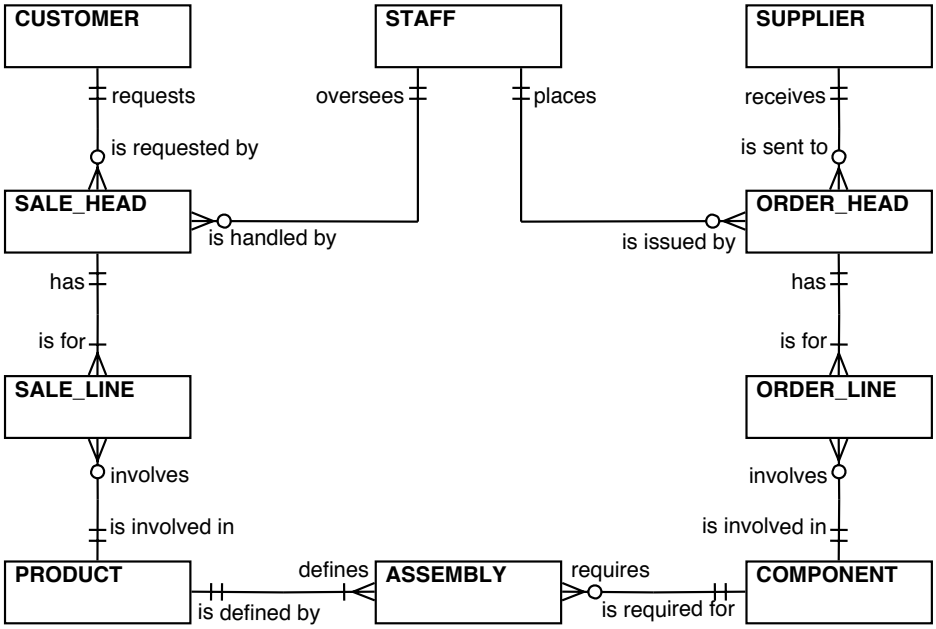


Figure 3: ERD of the example database.

period once per month, and that the transaction in question only requires read access. Replication of the Customer table could therefore be a suitable solution to ensure that this short, intense and intermittent transaction does not interfere with normal day-to-day transaction processing.

The suggestions above can be represented as a physical model using our proposed modelling notation, as shown in Figure 5 (some details have been omitted to save space). Note that we have placed indexes on all primary keys as a matter of course.

5 Future research

The current notation, while it covers the major aspects of physical modelling, is not complete and could be extended in various ways, for example:

- The current notation only caters for B-tree indexes and hashing. An obvious extension is to define symbols for other types of index, such as bitmap, reverse-key, R-trees, etc.
- There is currently no way to specify physical placement information, for example, which devices different partitions should be placed on.

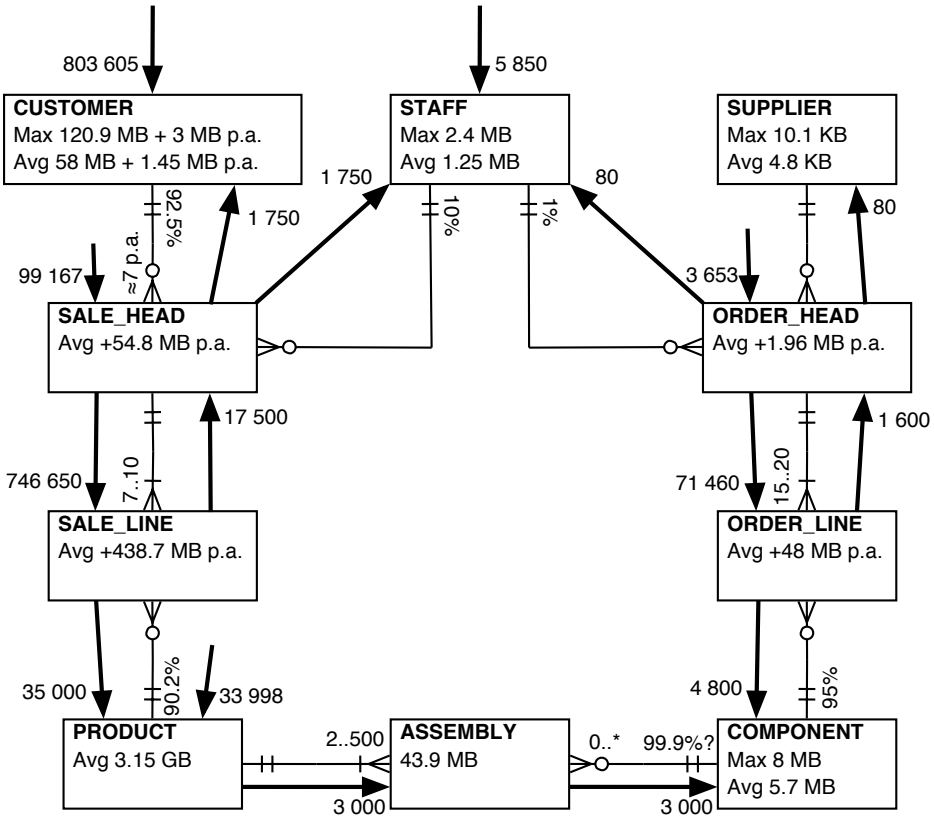


Figure 4: Beynon-Davies composite usage map for the example database.

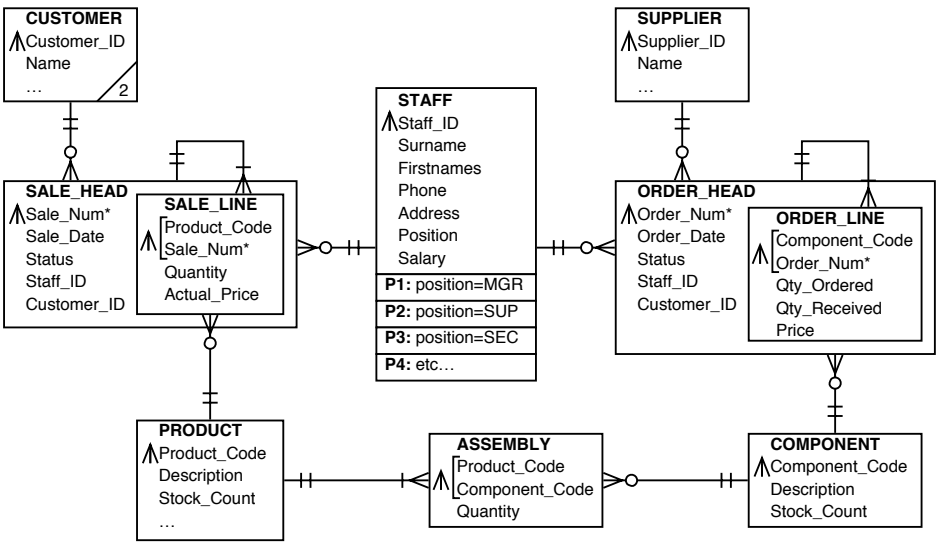


Figure 5: Physical database model for the example database.

- It may be useful to identify which replica or partition a particular (physical) relationship applies to.

We plan to evaluate the efficacy of the proposed notation by using the notation with undergraduate students in an advanced database course. We will then compare this with using Beynon-Davies' method alone.

6 Conclusion

In this paper we have described a graphical notation for physical database modelling, which enables database administrators (DBAs) to model the physical structure of new and existing databases in a more abstract manner. This will enable them to make more proactive and informed tuning decisions, compared to existing database monitoring tools, which tend to encourage a more reactive approach to database tuning. The notation uses simple and intuitive symbols to represent major physical database structures, and can easily represent complex physical schemas.

The notation is to be evaluated with undergraduate students in an advanced database course; the results of this evaluation will be compared with other physical modelling methods.

References

[1] Scott W. Ambler. *Building Object Applications That Work: Your Step-By-Step Handbook for Developing Robust Systems with Object Technology*. Cambridge University Press, New York, 1998.

[2] Scott W. Ambler. *Agile Database Techniques: Effective Strategies for the Agile Software Developer*. Wiley Application Development Series. John Wiley & Sons, New York, 2003.

[3] Scott W. Ambler. *The Object Primer: Agile Model-Driven Development with UML 2.0*. Cambridge University Press, New York, third edition, 2004.

[4] Don S. Batory. Modeling the storage architectures of commercial database systems. *ACM Transactions on Database Systems*, 10(4):463–528, December 1985.

[5] Paul Beynon-Davies. Using an entity model to drive physical database design. *Information and Software Technology*, 34(12):804–812, December 1992.

[6] Paul Beynon-Davies. *Database Systems*. Macmillan, third edition, 2003.

[7] Edgar F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, June 1970.

[8] Thomas M. Connolly and Carolyn E. Begg. *Database Systems: A Practical Approach to Design, Implementation and Management*. International Computer Science Series. Addison-Wesley, Harlow, Essex, third edition, 2002.

[9] Michael J. Corey and Michael Abbey. *Oracle Data Warehousing: A Practical Guide to Successful Data Warehouse Analysis, Build, and Roll-Out*. The Authorized Oracle Press™ Editions. Oracle Press/Osborne McGraw-Hill, Berkeley, California, 1997.

[10] Chris Gane and Trish Sarson. *Structured Systems Analysis: Tools and Techniques*. Prentice-Hall Software Series. Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

[11] Donald Ervin Knuth. *The Art of Computer Programming*, volume 3: Sorting and Searching. Addison-Wesley, Reading, Massachusetts, third edition, 1997.

[12] James Martin. *Information Engineering, Book II: Planning and Analysis*. Prentice-Hall, Englewood Cliffs, New Jersey, revised edition, 1990.

[13] Eric J. Naiburg and Robert A. Maksimchuk. *UML for Database Design*. Addison-Wesley Object Technology Series. Addison-Wesley, Reading, Massachusetts, 2001.

[14] Steve Roti. Indexing and access mechanisms. *DBMS Magazine*, 9(5):65–68, May 1996.

[15] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, Boston, fourth edition, 2002.

[16] D. Tsichritzis and A. Klug, editors. *The ANSI/SPARC/X3 DBMS framework*. AFIPS Press, 1978.

[17] Edward R. Tufte. *Visual Explanations: Images and Quantities, Evidence and Narrative*. Graphics Press, Cheshire, Connecticut, 1997.

[18] John Willits. *Database Design and Construction: An Open Learning Course for Students and Information Managers*. Library Association Publishing, London, 1992.

Appendix: Significant transactions in the example database

The following transactions represent the fourteen most significant in the example database, where "most significant" is determined by the impact the transaction has on the database. Transactions that have negligible impact on the database (e.g., they occur less than once per day and do not involve a large number of accesses) are excluded from the list, as are transactions whose frequency is unknown. The transactions are listed in no particular order. Transaction volume calculations are based on an assumption of 250 working days per year and eight working hours per day. Note that the term "reference" in this context represents a potential physical disk access.

Add or change a customer: About 10 000 customers are added per year, which implies about $10\,000/250/8 = 5$ added per hour. About 500 updates per working day implies 62.5 per hour. There are no deletes. This gives a total of 67.5 transactions per hour. Each transaction involves two table references, giving a total of 135 references per hour.

Enter new sale: On average there are about about 7 000 new sales per day, but the peak rate is 14 000 per day, which implies 1 750 transactions per hour. Each transaction involves 43 table references, giving a total of 75 250 references per hour.

Enter new order: On average there are about 320 new orders per day, but the peak rate is 640 per day, which implies 80 transactions per hour. Each transaction involves 63 table references, giving a total of 5 040 references per hour.

Change sale status: Assuming the same as for entering new sales implies a peak rate of 1 750 transactions per hour. Each transaction involves two table references, giving a total of 3 500 references per hour.

Change order status: Assuming the same as for entering new orders implies a peak rate of 80 transactions per hour. Each transaction involves two table references, giving a total of 160 references per hour.

Generate product catalogue: One hour once per month for all 16 492 products implies a peak access rate of 16 492 references per hour. (If we average it across a whole month we get a rate of about 0.006 transactions or 103 references per hour, which does not really reflect the true impact that this transaction has on the database.)

Receive order shipment: Shipments arrive at the same rate as outgoing orders, which implies a peak rate of 80 transactions per hour. Each transaction involves 81 table references, giving a total of 6 480 references per hour.

Download assembly details: Frequency varies, but probably no more than fifty per day (each transaction represents one product). This implies a peak rate of about 6 transactions per hour. Each product has between two and a few hundred components. For argument's sake, we will say that no product has more than 500 components. Each transaction thus involves 1 001 table references, giving a total of 6 006 references per hour.

Generate salaries: 2.5 hours once per month for all 9 000 staff implies a peak access rate of 3 600 references per hour. (If we average it across a whole month we get a rate of about 0.016 transactions or 22.5 references per hour.)

Special deals mail merge: Fifteen minutes once per week for 100 000 regular customers implies an effective peak access rate of 400 000 references per hour. (If we average it across a whole month we get a rate of about 0.002 transactions or 2 500 references per hour.)

Product catalogue mail merge: One hour once per month for all 400 000 customers implies a peak access rate of 400 000 references per hour. (If we average it across a whole month we get a rate of about 0.006 transactions or 2 500 references per hour.)

Look up product details: Assuming the same as for entering new sales implies a peak rate of 1 750 transactions per hour. Each transaction involves at most table references, giving a total of at most 17 500 references per hour.

Generate monthly accounts: Four hours once per month for 9 000 staff (assuming that salaries are included in the outgoings), maximum 291 667 sales, and maximum 13 333 orders. Assuming that accesses are spread evenly across the four hours for all tables (not very realistic, but it enables us to make a reasonable calculation in the absence of further detail) implies peak access rates of 2 250, 72 917 and 3 333 references per hour, respectively. (If we average it across a whole month we get a rate of about 0.025 transactions or 491 references per hour.)