2. State of the Art of Data Warehouse Research

We are drowning in data, but starving for knowledge!

(anonymous)

Data warehouse systems are a new technology and differ much from traditional transaction-oriented operational database systems. This chapter describes the major differences between the two concepts and their implications.

2.1. Introduction

Companies have invested much effort during the last decades in the area of information technology. Much of the work was performed to optimize the transaction-oriented systems whose main goal it is to react on customers' orders as fast and as cost-efficiently as possible. Companies have realized that valuable information is stored in their databases. The use of this information can help them to act more efficiently. Operational systems do not support the extraction of information out of databases efficiently because they were not designed for that kind of queries. Even today there is no technology available which supports both kinds of applications. Therefore, new systems are designed and implemented which support the decision making process. These systems are called *data warehouses*. The specific properties of these systems are described in this chapter. The chapter discusses approaches on how to process queries in a data warehouse efficiently.

2.2. Traditional transaction-oriented systems

Most of today's application systems follow a three-layer architecture. The upper level is the presentation layer. Typically *G*raphical *U*ser *I*nterfaces (GUI) visualize the data for the user. The application layer in the middle includes the program logic that covers the application itself, but no data is stored on this level. The data is stored in the third tier, the database layer. Applications change data by invoking operations like insert, update, and delete on the database. A sequence of operations is performed together as a transaction. These transactions have four prop-

erties: *A*tomicity, *C*onsistency, *I*solation, and *D*urability. They are abbreviated as ACID [Härder and Reuter, 1983] . Such applications are called On-Line Transaction Processing (OLTP) applications. Figure 2.1 shows an architecture with two OLTP applications accessing their databases. Graefe gives a general survey on query processing for these kinds of systems [Graefe, 1993].

Once the transaction-oriented data is stored in a database, a Decision Support System (DSS) is often built to create reports by grouping and summarizing data stored in the operational databases. There are various names for these kinds of systems; for instance, reporting tools, Management Information Systems (MIS), or executive information systems [Hannig, 1996]. In contrast to OLTP applications which read/write data from the operational databases, a DSS only reads data to get new information from the data sources. Figure 2.1 shows a DSS at the right side.

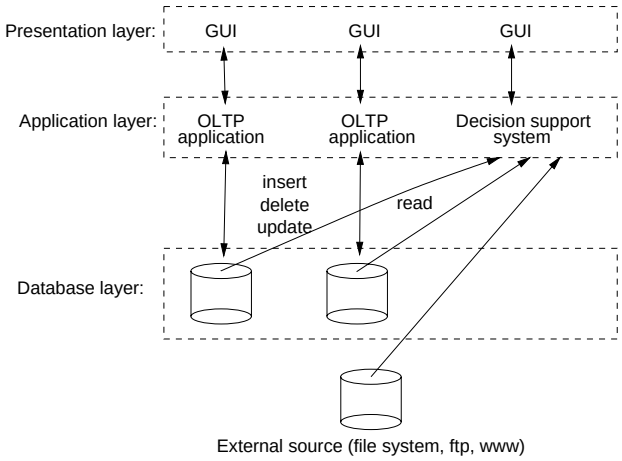

Figure 2.1.: OLTP application and information system based on operational database(s)

A benefit of this approach is that only the operational databases have to be created and maintained. A common set of metadata is used for both the operational system and the added-on DSS. The administration overhead for the DSS is rather small. However, there are significant disadvantages when the DSS and the transaction oriented application software share the same databases. The DSS can only use the actual data that is stored in the operational database. Therefore, historic analysis are usually not possible due to update or delete operations which changed the *historic* data. The operational database is optimized for transaction processes in a multi-user mode. This includes locking operations which do not support a scan of large sets of tuples well. Analytical queries often scan large amounts of tuples. These long transactions significantly decrease the performance of the operational database system. The solution which is usually applied to avoid these problems is to physically separate the transaction-oriented database from the database for the DSS. This information system is called a **data warehouse**.

2.3. Data warehouses for decision support

Inmon defined the term **data warehouse** as a *subject oriented, integrated, time variant, and non-volatile collection of data in support of management's decision making process* [Inmon, 1996]. In detail a data warehouse is:

Subject oriented. The goal of the data in the data warehouse is to improve decision making, planing, and control of the major subjects of enterprises such as customer relations, products, regions in contrast to OLTP applications that are organized around the work-flows of the company.

Integrated. The data in the data warehouse is loaded from different sources that store the data in different formats. The data has to be checked, cleansed and transformed into a unified format to allow easy and fast access.

Time variant. In operational systems, data is valid as of the moment of access, whereas in data warehouse systems the data is valid as of a defined moment of time.

Non-volatile. After the data is inserted in the data warehouse it is neither changed nor removed. The only exceptions are that *false* data is inserted or the capacity of the data warehouse is exceeded and archiving becomes necessary.

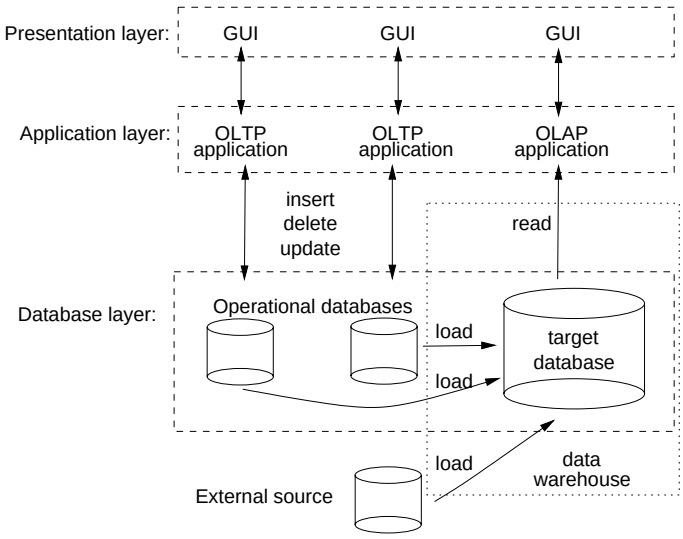

Figure 2.2.: Distinct databases for the transaction-oriented applications and the TDB the for information system/OLAP application

Data warehouses are physically separated from the operational DBMS. Such an architecture is shown in Figure 2.2. The data warehouse integrates data from multiple heterogeneous sources to support the need for structured and/or ad-hoc

Table 2.1.: Twelve golden OLAP rules as defined by Codd

Transparency	Multidimensional conceptual view
Accessibility	Consistent reporting performance
Client-server architecture	Dynamic sparse matrix handling
Generic dimensionality	Unrestricted cross-dimensional operations
Multi-user support	Intuitive data manipulation
Flexible reporting	Unlimited dimensions and aggregation levels

queries, analytical reporting, and decision support. This kind of application is called *O*n *L*ine *A*nalytical *P*rocessing (OLAP) . OLAP allows the transformation of data into strategic information. Codd defined twelve *golden OLAP rules* [Codd, 1994]. Table 2.1 presents the name of these rules. The size of a data warehouse can be in the range of TB (10^{12} Bytes). According to the MetaGroup study *1999 Data Warehouse Marketing Trends/Opportunities* more than 30 % of all data warehouse installations store more than 1 TB, and 14 % of all data warehouse installations have more than 1,000 users.

A data warehouse system DW can formally be defined as a tuple consisting of one target database TDB, $metadata$, and a set OP of operations:

$$DW = (TDB, metadata, OP)$$

The TDB is the target database for the loaded and transformed data. On this database analytical queries are performed. The set OP of operations is partitioned into four different groups:

Extraction. Extraction operations filter data from different internal or external data sources into temporary data bases. The sources are databases, flat files, web sites etc.

Transformation. These operations transform extracted data into an uniform format. Model-, schema-, and data- conflicts are resolved during the transformation phase.

Load. The load operations load the transformed data into the target database TDB.

Aggregating and grouping. The target database of the data warehouse system does not only store operational data, but also aggregated data. Aggregate and group operations calculate summarized data from base data.

Metadata such as descriptions of data structures stored at different sources, supports each of the above groups of operations. Metadata in the data warehouse system is even more important than in operational systems [Anahory and Murray, 1997]. Metadata is used to check the consistency of data and to ensure that only *safe operations* [Lenz and Shoshani, 1997] are performed.

Table 2.2.: Differences between OLTP and OLAP

aspect	**OLTP**	**OLAP**
level of data	detailed	aggregated
amount of data per transaction	small	large
views	pre-defined	user-defined
typical write operation	update, insert, delete	bulk insert
"age" of data	current (60 - 90 days)	historical, current, predicted, 5 - 10 years
tables	flat tables	multidimensional tables
number of users	high	low-med
data availability	high	low-med
database size	med $(10^9\,B - 10^{12}\,B)$	high $(10^{12}\,B - 10^{15}\,B)$
query optimizing	much experience	new

The target database is usually implemented with a relational DBMS (RDBMS) because this technology is well understood and able to handle large sets of data. However, the index structures investigated in this thesis can be applied to other kinds of DBMSs, too. OLAP applications that are based on RDBMS are also called Relational ROLAP (ROLAP).

2.4. OLAP vs. OLTP

Once the data is stored in the data warehouse, it is used to create new information for decision making processes. Typical OLAP operations are drill down, roll up, and slice & dice [Inmon, 1996], [Inmon et al., 1997].

Table 2.2 summarizes the differences between classical OLTP applications and OLAP applications. In OLTP operations the user changes the database via transactions on detailed data. A typical transaction in a banking environment transfers money from one account to another account. The four ACID properties are essential for this kind of application, because otherwise money may get lost or get doubled. In OLAP applications the typical user is an analyst who tries to select data needed for decision making. He is primarily not interested in detailed data, but usually in aggregated data over large sets of data. A typical OLAP query is to calculate the average amount of money that customers between age of 20 and 30 withdraw from ATMs in a certain region. For that kind of query the DBMS does not change any data. Hence no locking is necessary. Since the result is calculated by summing up

values from many different tuples, fast access to the data has to be supported. The user directs any query in an ad-hoc manner to the system. There exist approaches to model the behavior of the user [Sapia, 1999]. Based on such models the next queries of a user is predicted and pre-computed. This pre-computation decrease the waiting time for the user significantly and the interaction of a user with a data warehouse will become more efficiently.

OLAP and data warehouses are similar to statistical databases that have been discussed for many years [Lenz, 1993], [Lamersdorf et al., 1996], [Shoshani, 1997].

2.5. Accelerating query speed

The previous paragraph demonstrates the importance for having fast access to the data stored in the data warehouse. In the typical data warehouse environment, some requirements are less important than in the OLTP applications, *e. g.* locking and normalization. The overall goal of a data warehouse is to give the user a tool for interactive decision support. This implies that the user needs fast access to large sets of data integrated from different sources. In the remaining part of this chapter, five techniques are discussed that decrease the response time of the system. The speed-up of access to data is achieved if typical requirements of transaction-oriented systems are relaxed. Also redundancy of data increases the storage and update cost, but can reduce the query response time.

2.5.1. Denormalized schemas

The choice of a data model is essential for any database design. Most OLTP systems are designed on the conceptual level with entity relationship models and then transformed into relational models. The relation schemas are normalized to avoid insert-, update-, or delete-anomalies and redundancy. In data warehouse applications it is more important to have fast access to data than to avoid anomalies. Therefore, the relation schemas are usually not normalized. This implies redundancy in the data and makes manipulative operations more expensive than in databases with normalized schemas.

In a data warehouse multidimensional data is stored. A dimension is defined over a dimensional schema which is a set of functionally interrelated dimensional attributes [Lehner et al., 1998]. Dimensional attributes describe categorical attributes and properties like “brand”.

The most popular denormalized schema of a data warehouse is the star schema [Chaudhuri and Dayal, 1997]. A star schema consists of one base or fact table in the center surrounded by dimension tables . Figure 2.3 shows an example of a star schema. The denormalization becomes evident in the following examples: the same state belongs always to the same country (*e. g.* LA and SF belong both to CA and therefore both to the USA). The dependency between CA and USA is stored several times in the relation `region`. There are other dependencies in the relation `time`

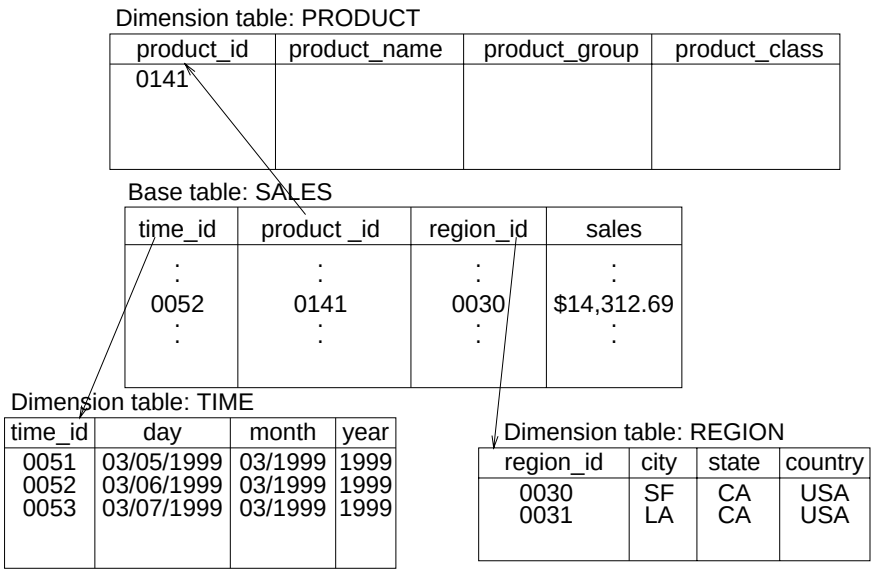

Figure 2.3.: Instance of a star schema

between `month` and the `year`.

Other schemas used for data warehouses include galaxy schema and snowflake schema [Anahory and Murray, 1997]. A set of base tables with some mutual dimension tables is called a galaxy schema. A star schema with normalized dimension tables is called a snowflake schema.

2.5.2. Materialized views

One widely used strategy of accelerating the access to aggregated data is to pre-compute *materialized views* [Gupta et al., 1997]. Base tables of a data warehouse may contain several millions of tuples. Therefore, scanning these base relations can take a significant amount of time. If there is some knowledge on what kind of queries the analysts will ask, these queries are pre-computed and the results are stored in materialized views. The access to pre-computed data is much faster than computing data on demand. However, the main technical problems are that the pre-computation takes time, the pre-computed data needs space, and it is difficult to predict what kind of data the user is interested in. There are additional semantical problems, *e. g.* the integrity constraints have to be satisfied. The data for pre-computation has to be selected along three criteria. First, an aggregation function (*e. g.* sum, max, min, avg, median, mostFrequent) is selected. Second, the dimensions are chosen; the data is aggregated with the group-by-statement. Third, in case of hierarchical attributes, the aggregation level is fixed. Data that is stored on a daily base can be summarized to data for weeks, months, or years. The size of the different views can be estimated without calculating the views [Shukla et al., 1996].

Assume three-dimensional sales data exist with time dimension, product dimen-

sion, and region dimension RSALES(time, product, region, sales). The three attributes time, product, and region are the categorical attributes. They define the three dimensions of the data. The attribute sales is the summary attribute and holds the information about the sales in a certain time about a certain product in a certain region. Figure 2.4 shows the base relation with all three dimensions at the bottom. The base relation is denoted by (t, p, r). From this base relation various marginal relations (subcubes) are computed by summarizing the data. For example RSALESpr(product, region ,sales) is computed by the following SQL statement:

```
CREATE VIEW RSALESpr AS
SELECT product,region,SUM(sales)
FROM RSALES
GROUP BY product,region
```

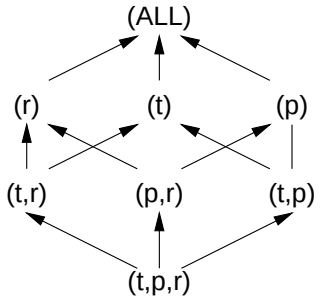

Figure 2.4.: Lattice of base table and subcubes/margins

In general, the base data is stored in a fact table $R(a_1, \cdots, a_n, s)$. The variables a_1 to a_n denote the categorical attributes and s is the summary attribute. Figure 2.4 presents the whole lattice of relations. An arrow from (p, t, r) to (p, r) denotes the fact that (p, r) is computable from (p, t, r). (p, t, r) can be seen as a three-dimensional data cube and (p, r) as one of its two-dimensional subcubes. (ALL) denotes the aggregate over all values. In general a n-dimensional data cube (without any hierarchical attributes) has $\binom{n}{k}$ k-dimensional subcubes. Therefore, a n-dimensional data cube has $\sum_{k=0}^{n} \binom{n}{k} = 2^n$ subcubes. If hierarchical attributes are considered, the number of possible subcubes is even bigger. The (automatic) selection of views that should be materialized is an actual research topic and known as the *view selection problem* [Gupta et al., 1997]. There are several different ways to choose the views for materialization such as Exhaustive search, Greedy algorithms [Gupta et al., 1997], $\mathbf{A}^*$-Algorithm [Labio et al., 1997], Integer programming [Yang et al., 1997], and Genetic algorithms [Lee and Hammer, 1999].

None of the five approaches has proven to be dominant. Once the views are selected and are materialized, another problem arises. Each time a base table is

changed, the materialized views and indexes built on it have to be updated (or at least have to be checked whether some changes have to be propagated or not). The views and the indexes can be updated incrementally or from the scratch. The problem of updating the views is known as the *view maintenance problem* [Huyn, 1997].

2.5.3. No locking

Since analysts only read data of a data warehouse and do not change tuples, a locking mechanism is not necessary. Data is changed only when it is inserted from external data sources. This is scheduled for example at night when no analyst is allowed to access the data. Overhead is decreased and query processing accelerated if no locking mechanism is applied.

2.5.4. On-line aggregation

The execution of typical OLAP queries can take much time and the user waits for the final answer until the query is processed completely. There are applications where the user is more interested in having an approximate result after a short time than getting a very precise answer after a long time period. One approach is the use of on-line aggregation where the user is informed about the actual status of his query during query processing [Hellerstein et al., 1997a], [Haas, 1999]. He can stop the execution of the query at any time and gets the result that is computed so far. This possibility of interactive query control significantly reduces the work load of the database server. However, some techniques have to be applied to guarantee that the data is processed in the system in random order.

2.5.5. Index structures

During the last three decades a great deal of research has been performed in the area of index structures for DBMSs. Starting with the B-tree [Bayer and McCreight, 1972] many new structures have been developed [Gaede and Günther, 1998]. The B-tree is an optimal structure for dynamic indexing of one-dimensional data for many applications. Almost all DBMSs implement the B-tree. No tree structure has been published that has the same properties for the multidimensional case as the B-tree in one dimension.

Due to the large sets of multidimensional data that are stored in data warehouse systems, table scans should be avoided whenever possible by the use of index structures. Because the kind of data and the typical queries that are processed in data warehouse systems differ much from traditional transaction-oriented applications, evaluation is important which index structures are suited best for these applications.

This thesis focuses on finding index structures that index typical data warehouse data efficiently. Chapter 5 describes an extension to improve multidimen-

sional tree structures which perform significantly faster for range queries on aggregated data. Performance models are developed to show how this extension influences the performance of tree structures. Chapter 6 investigates these performance models in detail. DBMSs that are designed for data warehousing have implemented bitmap indexing techniques besides the classic index structures. (*e. g.* [Sybase, 1997]). The bitmap indexing techniques are promising techniques for indexing typical data warehouse data and processing data warehouse queries efficiently. Chapter 7 describes and applies techniques for comparing different index structures. We present results of the comparative study.

2.6. Summary

This chapter described the major research topics in data warehousing. We defined the term *data warehouse* and investigated the differences between OLAP and OLTP applications. The main task in data warehousing is to provide the analyst with a tool for having fast access to aggregates over large sets of data that are integrated from different sources. Usually, this data is stored in a relational DBMS. The facts are stored in a fact table like $R(\underline{time}, \underline{product}, \underline{region}, sales)$. In general, the data is stored in a fact table $R(a_1, \cdots, a_n, s)$. One of the main goals of a data warehouse system is to decrease the query response time as much as possible. Five different mechanisms were described to speed up queries. The main part of the thesis focuses on what kind of index structures are used to speed up queries in data warehouse systems and how the behavior of such structures can be estimated and compared.