1. Introduction

During the last decades many companies have invested much effort in the area of information technology. Most of the effort went into the optimization of transaction-oriented operational systems. The main goal of these systems is to transport data as quickly and as cost efficiently as possible. Operational systems model and represent the dynamic behavior of processes. In this area much research has been conducted and sophisticated solutions have been developed. Typical operational systems store data about customers and transactions in databases. Database management systems (DBMSs) have been developed to support the processing of transactions on this data efficiently.

But apart from the use in operational systems, valuable strategic information is also hidden in the operational data. The extraction of this information by analysts may help companies to operate more efficiently. In particular, complex analytical queries on historic data over extended time periods are useful. Since the operational systems cannot deal with additional work load caused by complex analytical queries, the data has to be transfered from the operational systems to some other system dedicated only for analysis. These systems are called *Data Warehouses*.

Usually the data for these systems is stored with DBMSs. Relational DBMS (RDBMS) are the best understood technique to deal with large data sets. However they were not primarily designed for these new kinds of data and applications. There are three major differences between transaction-oriented operational systems and data warehouse systems:

- Size of the data: Fast access to GB ($\approx 10^9$ bytes) or TB ($\approx 10^{12}$ bytes) of data is crucial in providing interactive decision support. For these large sets, table scans, even on parallel systems, should be avoided whenever possible.

- Dynamics of data: In a typical data warehouse, data is inserted, but exceptionally updated or deleted. Furthermore, insertion only takes place at certain time windows when the system is not accessible for the analysts. Outside these time windows, analysts use the system only for reading data. This strategy is typical for *read-mostly* environments.

- Type of queries: In data warehouse and in operational systems different queries are processed. Typical queries in an operational system access data on a very detailed level, such as the *balance of a specific bank account*. Typical queries in data warehouse environments calculate aggregated data over large

sets of data, such as *sum of sales on product groups for some time period*. Therefore, the access to aggregated data over large sets of data has to be supported efficiently.

Since data warehouses are used for *interactive* decision support and not in batch mode, the response time of queries should be as short as possible. The analysts submit any query in ad-hoc fashion against the data warehouse. Different techniques are applied to reduce the query execution time. One technique to increase the performance is the use of *index structures*. Index structures avoid full table scans if only a small fraction of the stored tuples is used for result computation. Different methods, such as B-trees, have been investigated in large detail for operational databases during the last decades, but not for data warehouse systems. Therefore, the research question of this thesis is: What index structures are best suited for data warehouse systems?

1.1. Goals

This thesis investigates which index structures are well suited for data warehouse systems. We provide in detail answers to the following four questions:

1. What should an *optimal* index structure for data warehouses look like; *e. g.,* how can *optimality* be defined and what is the time complexity to calculate such an *optimal* solution?

   We describe one approach which guarantees finding an *optimal* index structure by mapping the problem of finding an *optimal* index structure to a Mixed Integer Problem (MIP) and solving the MIP with scientific standard software.

2. How can existing index structures be *improved* to support typical data warehouse queries more efficiently?

   We discuss in detail an extension of index structures where aggregated data is materialized in the inner nodes of an index structures. This extension can be applied to most tree-based index structures.

3. How can the performance of such extended structures be *predicted*?

   We extend performance models known from literature to model the behavior of the structure with aggregated data in the inner nodes. Furthermore, we develop a new model which considers the actual distribution of data and the distribution of queries.

4. How can different index structures be *compared*?

   We describe two approaches for comparing index structures. We apply these techniques and present results of comparisons of different index structures. Especially the time/space tradeoff and the evolving technology for secondary memories are considered in our comparisons.

1.2. Outline

Synopses of Chapter 2 through Chapter 8 follow below:

Chapter 2. Data warehouse systems, as mentioned before, differ greatly from traditional transaction-oriented database systems. This chapter describes the major differences between the two types of systems. Operational systems do not efficiently support the extraction of information through complex analytical queries. Data warehouse systems need to be designed and implemented which support the decision making process. We describe the specific properties of these data warehouse systems. The chapter also discusses several approaches on how to accelerate query processing in a typical data warehouse environment.

Chapter 3. Among the most important properties of DBMSs is the ability to handle large amounts of data. Different technologies are used to store and retrieve this data efficiently. Large sets of data do not fit into main memory and, therefore, are stored on secondary memory. Fast accesses to the data have to be provided to retrieve the data. The mechanics of the disks influence the performance of the index structures. In order to process queries efficiently, different access structures have been developed in traditional academic research. This chapter presents different kinds of structures that organize multidimensional data efficiently. We focus on the R-tree family, on the multi-component equality encoded indexes, and on the range encoded bitmap indexes because of their flexibilities.

Chapter 4. Since there is a strict separation of insert operations and read operations in data warehouses, one method of increasing the query processing speed is to invest more effort in organizing the data such that read operations perform quickly. We describe an approach for finding optimal index structures. We first define optimality, and then solve the problem of computing an optimal structure by transforming the problem into a Mixed Integer Problem (MIP). This MIP is solved with scientific standard software. Experiments show a serve problem: The time complexity of the MIP increases exponentially with the number of tuples and the number of clusters. Because of this exponential growth, this approach is in general infeasible. Therefore, in the following chapters of this thesis, we apply heuristic techniques for improving index structure for data warehousing. One application of computing optimal index structures with a MIP is to evaluate for small data sets how closely the heuristic approaches attain their optimum.

Chapter 5. The results of typical queries in data warehouse systems are aggregated data from large sets of data and not values of specific tuples. To support such queries we examine a technique where we materialize aggregated data in the inner nodes of a tree-based index structure. This data could also be calculated from data stored in leaf nodes, but the access to the aggregated data in the inner nodes is certainly faster than accesses to successor nodes. This concept of aggregated data in the inner

nodes is generic and is applicable to most tree-based index structures. We show how to modify algorithms to maintain and to use the aggregated data. In particular we modify the R^*-tree in such a way which leads to the so called R_a^*-tree. We present results of experiments comparing the R_a^*-tree and the R^*-tree. Results show that the performance of queries on aggregated data is significantly increased by using the extended structure.

Chapter 6. DBMSs can create different index structures on the same table. In order to choose the fastest index structure, it is desirable to have analytical models which predict the performance. For some index structures these models have been proposed in literature. Chapter 6 deals with performance models for tree-based index structures. We analyze models for index structures with the extension of aggregated data in the inner nodes and index structures without the extension. Then we apply three performance models for index structures without aggregated data which are known from literature. We extend the models so that they are applicable to model index structures with aggregated data. Then we introduce a new and better model: PISA (*P*erformance of *I*ndex *S*tructures with and without *A*ggregated data). The PISA model considers the distribution of data and the distribution of queries. Experiments determine the accuracy of models for different data sets. The PISA model is in most scenarios more accurate than the other models. We can decide under which conditions the use of the index structure with aggregated data increases the performance of the index structure to the highest degree.

Chapter 7. For range queries on aggregated data, the R_a^*-tree promises to be an efficient structure. However, tree structures have one major drawback: Tree structures degenerate when the number of dimensions increases as usually this is the case in data warehouse systems. Another class of index structures, the bitmap indexes, try to overcome the problem of degeneration by storing the data of each dimension separately and allowing fast access to those dimensions necessary to answer queries. The question arises which structure is best suited for certain applications. We select and investigate the parameters which influence the performance of the index structures. We apply two techniques to compare different index structures. A first method uses classification trees to visualize rules which are generated from a number of cases. Classification trees show which parameters influence the performance of the index structure more than other parameters. Then a second technique project multidimensional spaces into two dimensions which can be visualized in scatter diagrams. Our results show that by exploiting the evolving disk technology bitmap indexes become more time efficient in comparison to the tree-based index structures.

Chapter 8. The last chapter summarizes the contributions of this thesis. We briefly point out possible further research directions.