Monet; a next-Generation DBMS Kernel For Query-Intensive Applications

Boncz, P.A.

Citation for published version (APA):
Chapter 1

Introduction

In the last decades, almost all sectors of society have come to depend strongly on information technology. The amount of data stored worldwide in information systems has continuously experienced exponential growth. The increased availability of both large data volumes, now often accessible by means of the internet, and commodity computer hardware that is every year cheaper yet more powerful, has created new ways in which information systems are used. The primary goal of early large-scale information systems was the preservation and lookup of data items (recording account balances, tax payments, addresses, etc.). Now that many organizations have created an information system infrastructure and have gathered huge amounts of data over time, information systems are starting to be used more and more for extracting insight from this accumulated data and – in the digital economy – a business advantage for the organization. Therefore, the purpose of many new applications of information systems involves complex analyses over large data volumes, rather than simply preservation and lookup.

1.1 The DBMS and its Applications

Most information systems in operation today are implemented by using a commercially available standard Database Management System (DBMS) product. The DBMS is an – often complex – piece of software that manages the data stored in an information system. It facilitates concurrent access by multiple users to a database, limits access to data to authorized users only, and recovers from system failures without loss of integrity [Ull89a, Ull89b]. The relational DBMS is the most widely used kind of DBMS, that organizes the database in regular tables that consist of rows and columns, and provides access to these tables through the high level query/data manipulation language SQL (Structured Query Language) [Cod70]. An extensive description of relational DBMS concepts is beyond the scope of this thesis, and we assume the reader to be familiar with them [Dat85, UW97].

1.1.1 On-Line Transaction Processing (OLTP)

When relational DBMSs became popular in the early 1980s, their main use was to support on-line transaction processing (OLTP). Typical queries in an OLTP system for e.g., running an insurance company are "show all characteristics recorded about
customer X”, for example to provide the employee receiving a telephone call with basic customer data, or simple updates like: “insert a new record for a sale to customer X of a certain product Y”. The mix between read-only and update queries varies among OLTP applications, but updates tend to form a significant fraction and can even dominate query processing.

OLTP applications involve little or no analysis and serve the use of an information system for data preservation and lookup.

<table>
<thead>
<tr>
<th>contract</th>
<th>client</th>
<th>date</th>
<th>name</th>
<th>price</th>
<th>city</th>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td>1238945</td>
<td>Smith</td>
<td>Doe</td>
<td>Seattle</td>
<td>House</td>
<td>New York</td>
<td>House</td>
</tr>
<tr>
<td>9876543</td>
<td>Jones</td>
<td>Bono</td>
<td>Boston</td>
<td>Car</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1.1: OLTP access pattern.

Figure 1.1 depicts three OLTP queries: an update, a lookup and an insert that each access just one row in the table (grey-shaded). These examples illustrate the access pattern of an OLTP query to a relational database table: the query accesses one or only a few rows, but uses most of its columns. An OLTP query typically is short-running and requires little resources from the DBMS. However, OLTP systems often have a heavy load of many (e.g., hundreds) queries per second, which can be challenging to sustain [GR91].

1.1.2 Query-Intensive Applications

In the 1990s, new DBMS applications became popular that differ from OLTP applications, as they involve complex analysis on large data volumes. Update queries are infrequent here and, if any, typically batch-oriented rather than on-line. We will refer to such applications as query-intensive DBMS applications. Examples of query-intensive DBMS applications are on-line analytical processing (OLAP) and data mining. Let us now discuss both application areas in more detail.

OLAP

OLAP tools summarize and group large amounts of data into small yet insightful results, typically using 2- or 3-D graphics to visualize answers. An example OLAP query is “give accumulated totals per city and per insurance product group of the insurance claims made last year after march 21”. The result of this query could be a 3-D bar-chart with x-axis City, y-axis Product, and vertical bars denoting Claims in the z-axis. Answering this query usually requires the DBMS to go through the entire claims table to filter out all records after march 21, and compute for these
selected records a series of sums representing total claims, with a separate result for each occurring combination of city and insurance product.

Such 3-D visualization of summarized data is called a "data cube", and OLAP data is often thought of as multi-dimensional [Ken95]. One typical operation is "roll-up" of e.g., City along the location hierarchy: City-Region-Country-Continent, which results in increasingly summarized totals with less detail. The inverse operation of roll-up is called "drill-down". Other OLAP operations are "slice-and-dice" (i.e., selection combined with projection in the dimensions), and "pivot" (i.e., a re-orientation of the dimensions).

![OLAP Access Pattern](image)

**Figure 1.2: OLAP access pattern.**

Figure 1.2 shows that the access pattern of OLAP is quite different from OLTP, as one single query from an OLAP tool often requires analysis of a substantial portion of the rows of the database table(s). Answering the query therefore requires the DBMS to do much more work, hence OLAP queries are medium to long-running, as data volumes tend to be large. The load on OLAP systems, however, tends to be limited in terms of number of queries per second in comparison to OLTP loads [CD97, Wid95].

The knowledge discovery process for which OLAP tools are used, is an interactive process. OLAP queries therefore should exhibit interactive response, rather than medium to long running times observed in today's DBMS technology.

**Data Mining**

Data mining is an even more demanding application area for a DBMS, as the query load generated by a data mining tool can be considered a kind of "repeated OLAP". In data mining, the objective is to locate sub-groups that have statistically significant differences from the mean and are interesting in some respect. An example query on a car insurance customer database could be: "what are profiles of dangerous drivers?". It is left to the data mining tool to determine what are the characteristics of those dangerous customer groups. This is typically done using a combination of statistical measures and automated search techniques from the field of artificial intelligence.
(AI). The answers found by data mining can be unexpected, i.e., a user never would have found them using an OLAP tool manually, as OLAP tools just give answers to specifically asked questions. A data mining tool finds these unexpected and interesting groups by building a hypothesis (e.g., identifying “male drivers” as a sub-group in the database whose members have a higher probability of causing traffic accidents) and step-by-step improving it (e.g., “male drivers in company-leased cars”). This search for the best hypothesis can be compared to the search process a computer chess program performs while looking for the best chess move. In data mining, however, judging each “move” (i.e., hypothesis describing a sub-group) on its merits, corresponds to executing an OLAP-like DBMS query that checks the statistical validity of the hypothesis by looking in the historical data. In our example case, such a query would determine the percentage of accidents among male drivers with respect to the form of car-ownership. Answering one data mining query implies finding a good model and this typically requires testing many hypotheses, hence can amount to hundreds or thousands of OLAP-like queries, all of which together are supposed to finish in interactive time. Therefore, data mining is a truly query-intensive application area.

We should note here, that we consider that data mining tools should use a DBMS to execute queries on-the-fly. The extreme intensity of query loads has until now scared away many data mining tools vendors for implementing such solutions [Cha98]. DBMS use of such tools is limited to importing a (small) DBMS table into the tool, which performs the data mining analysis, while all data is memory resident in a data structure specific to the data mining algorithm chosen. We reject this approach as it goes against the main advantages of using a DBMS, which is providing efficient management of shared data and data independence. In particular, the before mentioned ad-hoc approach of processing data outside the DBMS in algorithm-specific in-memory structures is both limited in scalability (all data should fit in memory) and hard to extend with new algorithms, as each new algorithm requires specific data structures for handling mass data. In our view, one should use DBMS technology to support the information need [Cha98] of the data mining tool to provide both scalability for mining huge data volumes, and re-use of the same DBMS mass-data manipulation infrastructure for all data mining algorithms. Therefore, when we discuss data mining as an DBMS application area, we refer to solutions that perform statistical validation of hypothesis inside using DBMS queries.
1.2 Thesis Outline

We use the term DBMS architecture with the meaning of the design of a DBMS from the software engineering standpoint. This thesis is about DBMS architecture, where we investigate the question:

- how to design DBMS software that is capable of supporting query-intensive applications with high performance?

This thesis is structured as follows. We start in Chapter 2 with a short historic overview of DBMS architecture. Here, we motivate our choice to focus on the issue of performance, by showing why achieving good performance on query-intensive applications is a major problem for current DBMS products. In Chapter 3 we formulate specific research goals, and describe a number of ideas that we decided to try out in our Monet research DBMS. The first papers published on Monet were:


We then describe the Monet system in detail: Chapter 4 introduces the algebraic MIL language, which is the query language for the Monet system, designed to support query-intensive loads in extended database application areas. Chapter 5 describes how this language was implemented in Monet to provide high performance. Both chapters are based on:


In Chapter 6, we focus on the question how the relational join operator can best be implemented in Monet, and formulate hardware-conscious join algorithms, where we provide detailed insight\(^1\) into the reasons why Monet is successful in exploiting the power of modern hardware, through cost modeling and experimentation. This research has been published in:


\(^1\) Joint research with Stefan Manegold; certain parts will overlap in his Ph.D. thesis.
\(^2\) Re-published extended version of the VLDB 1999 Conference paper, as Best-of-VLDB 1999 paper.
In Chapter 7, we show how design and implementation of MIL in Monet provide the required back-end functionality to construct a full-fledged and well-performing SQL-speaking RDBMS front-end.\(^3\) While not directly based on them, this chapter is related to other papers published on using Monet to support an object-oriented query language and a data mining front-end.


The thesis is concluded in Chapter 8 (Conclusion), which contains a retrospective on Monet and recommendations for future work.

\(^3\)This SQL front-end will be part of the upcoming open-source release of Monet.