Migration of Data Mining Preprocessing into the DBMS

Carlos Ordonez
University of Houston
Dept. of Computer Science
Houston, TX, USA

Javier García-García
UNAM*
Facultad de Ciencias
Mexico City, Mexico

Michael J. Rote
Teradata
Data Mining Solutions
San Diego, CA, USA

ABSTRACT

Nowadays there is a significant amount of data mining work performed outside the DBMS. This article discusses recommendations to push data mining analysis into the DBMS paying attention to data preprocessing (i.e. data cleaning, summarization and transformation), which tends to be the most time-consuming task in data mining projects. We present a discussion of practical issues and common solutions when transforming and preparing data sets with the SQL language for data mining purposes, based on experience from real-life projects. We then discuss general guidelines to create variables (features) for analysis. We introduce a simple prototype tool that translates statistical language programs into SQL, focusing on data manipulation statements. Based on experience from successful projects, we present actual time performance comparisons running SQL code inside the DBMS and outside running programs on a statistical package, obtained from data mining projects in a store, a bank and a phone company. We highlight which steps in data mining projects are much faster in the DBMS, compared to external servers or workstations. We discuss advantages, disadvantages and concerns from a practical standpoint based on users feedback. This article should be useful for data mining practitioners.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Systems—*Relational Databases*; H.2.8 [**Database Management**]: Database Applications—*Data Mining*

General Terms

Languages, Performance

Keywords

Data preprocessing, denormalization, SQL, translation

*Universidad Nacional Autónoma de México

DMCS'09, June 28, Paris, France.

1. INTRODUCTION

In a modern IT environment transaction processing [7] and data warehousing [10] are managed by database systems. Analytics on the database are a different story. Despite the data mining [8, 10] functionality offered by the DBMS there exist many statistical tasks that are performed outside [11, 10]. This is due to the existence of sophisticated statistical tools and libraries [11], the lack of expertise of statistical analysts to write correct and efficient SQL queries, the limited set of statistical methods available offered by the DBMS and legacy code. In such environments users rely on the database to provide data, and then they write SQL queries to extract data joining several tables. Once data sets are exported they are summarized and transformed depending on the task at hand, but outside the DBMS. Finally, when the data set has the desired variables (features), statistical models are computed, interpreted and tuned. In general, models include both descriptive and predictive techniques, coming from machine learning [14] and statistics [11]. From all the tasks listed above preparing the data set for analysis is the most time consuming task [8] because it requires significant effort to transform typical normalized tables in the database into tabular data sets appropriate for analysis. Unfortunately, in such environments manipulating data sets outside the DBMS creates many data management issues: data sets must be recreated and re-exported every time there is a change, models need to be deployed inside the DBMS, different users may have inconsistent version of the same data set in their own computers, security is compromised. Therefore, we defend the idea of transforming data sets and computing models inside the DBMS, exploiting its extensive functionality. Our motivation to migrate statistical analysis into the DBMS, yields the following benefits. The DBMS server is generally a fast computer and thus results may be obtained sooner. The DBMS provides powerful querying capabilities through SQL and 3rd party tools. The DBMS has extensive data management functionality (maintaining referential integrity, transaction processing, fault-tolerance, security).

We assume the DBMS provides basic statistical and data mining functionality. That is, the DBMS is capable of performing common data mining tasks like regression [11], clustering [11], PCA [14] or association rules, among others. The commercial data mining tool used in our real-life projects is called Teradata Warehouse Miner (TWM) [25]. On the other hand, we assume the DBMS supports the SQL language, which is used to store and retrieve information from the DBMS [7, 10]. Based on experience from data mining

projects we have become aware that statistical analysts have a hard time writing correct and efficient SQL code to extract data from the DBMS or to transform their data sets for the data mining task at hand. To overcome such issues, statistical analysts resort to statistical packages to manipulate and transform their data sets. Since most statistical packages provide a programming language, users end up creating long scripts mixing data transformation and modeling tasks together. In such scripts most of the programming effort is spent on transforming the data set: this is the main aspect studied in this article. We discuss practical issues and common solutions for data transformation tasks. Finally, we present evidence transforming data sets and building models on them is more efficient to do inside the DBMS than outside with an external statistical program.

This article presents a prototype tool capable of translating common statistical language (e.g. SAS [6]) programs into SQL [7]. The tool automates the task of translating statistical language scripts into equivalent SQL scripts, producing the same results. There are certain similarities and differences between both languages that make the problem interesting and challenging. The statistical language is an imperative language that manipulates data sets as tables, but not strictly as relational tables. The language includes syntactic constructs to specify arithmetic expressions, flow control and procedural calls. On the other hand, SQL is a set oriented language that also manipulates data sets as tables, but which allows specifying relationships among tables with primary/foreign keys. A set of tables is manipulated with join operations among their keys. Both languages identify data set columns with names and not with subscripts and both automatically scan all rows in a data set without the need of a loop construct. Query optimization is a fundamental issue in SQL.

The article is organized as follows. Section 2 discusses important practical issues when preparing and transforming data sets for statistical analysis as well as common solutions. Section 3 presents a translator of data transformation statements into SQL. Section 4 presents performance comparisons and users feedback from projects where statistical programs were successfully migrated. Related work is discussed in Section 5. Finally, Section 6 concludes the article.

2. PRACTICAL ISSUES PREPARING DATA SETS IN SQL

We present important practical issues to write and optimize SQL code for data transformation tasks. These issues have been collected from several projects helping users migrate statistical analysis performed in external statistical tools into the DBMS.

In general, the main objection from users to do so is to translate existing code. Commonly such code has existed for a long time (legacy programs), it is extensive (there exist many programs) and it has been debugged and tuned. Therefore, users are reluctant to rewrite it in a different language, given associated risks. A second complaint is that, in general, the DBMS provides elementary statistical functionality, compared to sophisticated statistical packages. Nevertheless, such gap has been shrinking over the years. As explained before, in a data mining environment most user time is spent on preparing data sets for analytic purposes. We discuss some of the most important database issues when tables are manipulated and transformed to prepare a data set for data mining or statistical analysis. We pay particular attention to query optimizations aspects [7]. These issues represent experience that has been used to improve and optimize SQL code generated by a data mining tool tightly integrated with the DBMS [25].

Throughout this section we present example of typical SQL queries. Some examples refer to retail (store sales) databases, whereas some others refer to a bank.

2.1 Main Issues

Summarization

Unfortunately, most data mining tasks require dimensions (variables) that are not readily available from the database. Such dimensions typically require computing aggregations at several granularity levels. This is because most columns required by statistical or machine learning techniques require measures (or metrics), which translate as sums or counts computed with SQL. Unfortunately, granularity levels are not hierarchical (like cubes or OLAP [7]) making the use of separate summary tables necessary (e.g. summarization by product or by customer, in a retail database). A straightforward optimization is to compute as many dimensions in the same statement exploiting the same group-by clause, when possible. In general, for a statistical analyst it is best to create as many variables (dimensions) as possible in order to isolate those that can help build a more accurate model. Then summarization tends to create tables with hundreds of columns, which make query processing slower. However, most state-of-the-art statistical and machine learning techniques are designed to perform variable (feature) selection [11, 14] and many of those columns end up being discarded.

A typical query to derive dimensions from a transaction table is as follows:

```
SELECT
  customer_id
 ,count(*) AS cntItems
 ,sum(salesAmt) AS totalSales
 ,sum(case when salesAmt<0 then 1 end)
  AS cntReturns
FROM sales
GROUP BY customer_id;
```

Denormalization

It is required to gather data from many tables and store data elements in one place. It is well-known that on-line transaction processing (OLTP) DBMSs update normalized tables. Normalization makes transaction processing faster and ACID [7] semantics are easier to ensure. Queries that retrieve a few records from normalized tables are relatively fast. On the other hand, analysis on the database requires precisely the opposite: a large set of records is required and such records gather information from many tables. Such processing typically involves complex queries involving joins and aggregations. Therefore, normalization works against efficiently building data sets for analytic purposes. One solution is to keep a few key denormalized tables from which specialized tables can be built. In general, such tables cannot be dynamically maintained because they involve join computation with large tables. Therefore, they are periodically recreated as a batch process.

```
SELECT
  customer_id
 ,customer_name
 ,product.product_id
 ,product_name
 ,department.department_id
 ,department_name
FROM sales
   JOIN product
     ON sales.product_id=product.product_id
   JOIN department
     ON product.department_id
        =department.department_id;
```

Time window

In general, for a data mining task it is necessary to select a set of records from one of the largest tables in the database based on a date range. In general, this selection requires a scan on the entire table, which is slow. When there is a secondary index based on date it is more efficient to select rows, but it is not always available. The basic issue is that such transaction table is much larger compared to other tables in the database. For instance, this time window defines a set of active customers or bank accounts that have recent activity. Common solutions to this problem include creating materialized views (avoiding join recomputation on large tables) and lean tables with primary keys of the object being analyzed (record, product, etc) to act as filter in further data preparation tasks.

```
SELECT
  customer_id
 ,product_id
 ,sales_amt
FROM sales
WHERE cast(salesDate AS char(10))>= '2009-01-01'
     and cast(salesDate AS char(10))< '2009-03-01';
```

Dependent SQL statements

A data transformation script is a long sequence of SELECT statements. Their dependencies are complex, although there exists a partial order defined by the order in which temporary tables and data sets for analysis are created. To debug SQL code it is a bad idea to create a single query with multiple query blocks. In other words, such SQL statements are not amenable to the query optimizer because they are separate, unless it can keep track of historic usage patterns of queries. A common solution is to create intermediate tables that can be shared by several statements. Those intermediate tables commonly have columns that can later be aggregated at the appropriate granularity levels.

```
CREATE TABLE X AS (
 SELECT
   A,B,C
 FROM
) WITH DATA;

SELECT *
FROM X JOIN T1 ...

SELECT *
FROM X JOIN T2 ...
..

SELECT *
FROM X JOIN TN ...
```

Computer resource usage

This aspect includes both disk and CPU usage, with the second one being a more valuable resource. This problem gets compounded by the fact that most data mining tasks work on the entire data set or large subsets from it. In an active database environment running data preparation tasks during peak usage hours can degrade performance since, generally speaking, large tables are read and large tables are created. Therefore, it is necessary to use workload management tools to optimize queries from several users together. In general, the solution is to give data preparation tasks a lower priority than the priority for queries from interactive users. On a longer term strategy, it is best to organize data mining projects around common data sets, but such goal is difficult to reach given the mathematical nature of analysis and the ever changing nature of variables (dimensions) in the data sets.

Views vs temporary tables

Views provide limited control on storage and indexing. It may be better to create temporary tables, especially when there are many primary keys used in summarization. Nevertheless, disk space usage grows fast and such tables/views need to be refreshed when new records are inserted or new variables (dimensions) are created.

Level of detail

Transaction tables generally have two or even more levels of detail, sharing some columns in their primary key. The typical example is store transaction table with individual items and the total count of items and total amount paid. This means that many times it is not possible to perform a statistical analysis only from one table. There may be unique pieces of information at each level. Therefore, such large tables need to be joined with each other and then aggregated at the appropriate granularity level, depending on the data mining task at hand. In general, such queries are optimized by indexing both tables on their common columns so that hash-joins can be used.

For instance, in a store there is typically one transaction table containing total amounts (sales, tax, discounts) and item counts, and another transaction detail (or line) table containing each individual item scanned at the register. For certain data mining analysis (market basket analysis the detailed purchase information may be required). On the other hand, in a bank there is one table with account summaries (current and by month) and another table with individual banking transactions (deposits, withdrawals, payments, balance inquiry).

Left outer joins for completeness

For analytic purposes it is always best to use as much data as possible. There are strong reasons for this. Statistical models are more reliable, it is easier to deal with missing information, skewed distributions, discover outliers and so on, when there is a large data set at hand. In a large database with tables coming from a normalized database being joined

with tables used in the past for analytic purposes may involve joins with records whose foreign keys may not be found in some table. That is, natural joins may discard potentially useful records. The net effect of this issue is that the resulting data set does not include all potential objects (e.g. records, products). The solution is define a universe data set containing all objects gathered with union from all tables and then use such table as the fundamental table to perform outer joins. For simplicity and elegance, left outer joins are preferred. Then left outer joins are propagated everywhere in data preparation and completeness of records is guaranteed. In general such left outer joins have a "star" form on the joining conditions, where the primary key of the master table is left joined with the primary keys of the other tables, instead of joining them with chained conditions (FK of table T1 is joined with PK of table T2, FK of table T2 is joined with PK of T3, and so on).

```
SELECT
 ,record_id
 ,T1.A1
 ,T2.A2
 ..
 ,Tk.Ak
FROM T_UNIVERSE
   LEFT JOIN T1 ON T_UNIVERSE.record_id= T1.record_id
   LEFT JOIN T2 ON T_UNIVERSE.record_id= T2.record_id
   ..
   LEFT JOIN Tk ON T_UNIVERSE.record_id= Tk.record_id;
```

Filtering Records from Data Set

Selection of rows can be done at several stages, in different tables. Such filtering is done to discard outliers [18], to discard records with a significant missing information content (including referential integrity [22], to discard records whose potential contribution to the model provides no insight or sets of records whose characteristics deserve separate analysis. It is well known that pushing selection is the basic strategy to accelerate SPJ (select-project-join) queries [3], but it is not straightforward to apply into multiple queries. A common solution we have used is to perform as much filtering as possible on one data set. This makes code maintenance easier and the query optimizer is able to exploit filtering predicates as much as possible.

Statistics columns

Many times users build data sets with averages, which unfortunately are not distributive. Common examples are computing the mean and standard deviation of some numeric column. A simple solution is to keep sums and counts for every data set, from which it is easy to derive descriptive statistics. In particular, sufficient statistics [26, 2, 17, 19] prove useful for both simple statistics as well as sophisticated multidimensional models. This solution is represented by the sufficient statistics L, Q [17, 19], explained in more detail in Section 3.6. Another particularly useful optimization is to perform aggregations before joins, when data semantics allow it. This optimization has been studied in the database literature [3].

```
SELECT
 ,count(*) AS n
 ,sum(X1) AS L1
 ,sum(X2) AS L2
 ..
 ,sum(Xd) AS Ld
 ,sum(X1*X1) AS Q1
 ,sum(X2*X2) AS Q2
 ..
 ,sum(Xd*Xd) AS Qd
FROM DataSetX;
```

Multiple primary keys

Different sets of tables have different primary keys. This basically, means such tables are not compatible with each other to perform further summarization. The key issue is that at some point large tables with different primary keys must be joined and summarized. Join operations will be slow because indexing involves foreign keys with large cardinalities. Two solutions are common: creating a secondary index on the alternative primary key of the largest table, or creating a denormalized table having both primary keys in order to enable fast join processing.

For instance, consider a data mining project in a bank that requires analysis by *customer id*, but also *account id*. One customer may have multiple accounts. An account may have multiple account holders. Joining and manipulating such tables is challenging given their sizes.

Model deployment

Even though many models are built outside the DBMS with statistical packages and data mining tools, in the end the model must be applied in the database [19]. When volumes of data are not large it is feasible to perform model deployment outside: exporting data sets, applying the model and building reports can be done in no more than a few minutes. However, as data volume increases exporting data from the DBMS becomes a bottleneck [19]. This problem gets compounded with results interpretation when it is necessary to relate statistical numbers back to the original tables in the database. Therefore, it is common to build models outside, frequently based on samples, and then once an acceptable model is obtained, then it is imported back into the DBMS. Nowadays, model deployment basically happens in two ways: using SQL queries if the mathematical computations are relatively simple or with UDFs [19], if the computations are more sophisticated. In most cases, such scoring process can work in a single table scan, providing good performance.

2.2 Lessons Learned: Most Common Queries

Data transformation is a time consuming project, but the statistical language syntactic constructs and its comprehensive library of functions make such task easier. In general, users think writing data transformations in SQL is not easy. Despite the abundance of data mining tools users need to understand the basics of query processing. The most common queries needed to create data sets for data mining are (we omit transformations that are typically applied on an already built data set like coding, logarithms, normalization, and so on):

- Left outer joins, which are useful to build a "universal" data set containing all records (observations) with columns from all potential tables. In general, natural joins filter out records with invalid keys which may contain valuable information.

- Aggregations, generally with sums and counts, to build a data set with new columns.
- Denormalization, to gather columns scattered in several tables together.

3. TRANSLATING DATA TRANSFORMATIONS INTO SQL

This section explains the translator, summarizes similarities and differences between the statistical language and SQL and presents common sufficient statistics for several models.

3.1 Definitions

The goal of data manipulation is to build a data set. Let $X = \{x_1, \ldots, x_n\}$ be the data set with n points, where each point has d attributes (variables, features), where each of them can be numeric or categorical. The statistical language is a high-level programming language, based on: data set, observation and variable. We assume SQL is well understood, but we give a summary. In SQL the equivalent terms are table, row and column. A table contains a set of rows having the same columns. The order of rows is immaterial from a semantic point of view. A set of tables is interrelated by means of primary/foreign key relationships.

3.2 General Framework

A statistical language program produces data sets which are basically tables with a set of columns. Columns can be of numeric, string or date data types. On the other hand, SQL manipulates data as tables, with the additional constraint that they must have a primary key. In general, data sets in the statistical language are sequentially manipulated in main memory, loading a number of rows. On the other hand, in SQL relational operations receive tables as input and produce one table as output. There exist two main kinds of statements: (1) Data manipulation (data set transformation). (2) Subroutine calls (functions, procedures, methods).

Translating subroutine calls requires having stack-based calling mechanisms in SQL, which are not generally available, or if they are available parameters are not standardized. That is, the translator produces a "flat" script in SQL.

3.3 Data Manipulation

3.3.1 Importing Data

Importing is a relatively easy procedure which is used mostly to integrate external data sources into the statistical analysis tool. Despite its simplicity it is important to carefully analyze importing statements because they may involve data not stored in the database. Importing data comes in the form of a statement specifying a data set with several columns and an input file.

3.3.2 Arithmetic Equations

Columns in the data set are treated as variables. An assignment creates a new variable or assigns a value to an existing one. If the variable does not exist the type is inferred from the expression. In SQL there is no assignment expression. Therefore, the assignment expression must be converted into SELECT statements with one column to receive the result of each expression.

A sequence of variable assignments creates or updates variables with arithmetic expressions. The first assignment is assumed to use an expression with all instantiated variables. The sequence of assignment statements assumes a variable cannot be used before it has a value. Given the dynamic nature of the sequence of expressions it is necessary for the SQL run-time evaluation algorithm to determine the type of the resulting column. The alternative approach, defining a DDL and then the SQL with expressions would require doing extensive syntactic and semantic analysis when the expression is parsed. Most math functions have one argument and they can be easily translated using a dictionary. String functions are more difficult to translate because besides having different names they may have different arguments and some of them do not have an equivalent in SQL.

Let C be the set of original variables and let V be the set of variables created or updated by assignment. An initial pass on all assigned variables is needed to determine which columns are overwritten by computing $C \cap V$. Each column that belongs to $C \cap V$ is removed from C. Then it is unselected from the original list of variables. Assume then that the input columns become a subset of the original columns: $\mathcal{F} = F_1, F_2, \ldots, F_p$, where $\mathcal{F} \subseteq C$. Then the sequence of expressions can be translated as $\mathcal{F}$, followed by the arithmetic expressions assigned to each variable.

It is important to observe the data types are dynamically inferred by SQL at run-time and that the table is defined as multiset. Performing a static analysis would require a more sophisticated mechanism to infer data types storing variables in a symbol table like a traditional compiler.

3.3.3 IF-THEN and WHERE

A high-level programming language provides great flexibility in controlling assignments. This is more restricted in SQL because only one column can be manipulated in a term. We consider three cases: (1) Chained IF-THEN statements with one assignment per condition; (2) Generalized IF-THEN-DO with IF-THEN-DO nesting and two or more assignments per condition. (3) A WHERE clause. A chained IF-THEN statement gets translated into an SQL CASE statement where each IF condition gets transformed into a WHEN clause. It is important to watch out for new columns when new variables are created. The IF-THEN-DO construct is more complex than the previous case for several reasons: More than one assignment can be done in the IF body; IF statements can be nested several levels. There may be loops with array variables. This case will be analyzed separately. The system uses a stack [1] to keep track of conditions given an arbitrary number of levels of nesting. For every nested IF statement boolean expressions are pushed into the stack. For each assignment each additional boolean expression are popped from the stack and are concatenated using a logical "AND" operator to form an SQL WHEN expression. In other words, nested IF-THEN statements are flattened into "WHEN" substatements in a CASE statement. The WHERE clause translates without changes into SQL. Comparison for numbers and strings use same operators. However, date comparisons are different and therefore special routines. Comparison operators have similar syntax in both languages, whose translation requires a simple equivalence table. Negation (NOT), parenthesis and strings, require similar translation (compilation) tech-

niques.

3.3.4 Looping Constructs

In the statistical language there may be arrays used to manipulate variables with subscripts. SQL does not provide arrays, but they can be simulated by generating columns whose name has the subscript appended. A FOR loop is straightforward to translate when the subscript range can be determined at translation time; the most common case is a loop where the bounds are static. When an array is indexed with a subscript that has a complex expression (e.g. $a(i*10-j)$) then the translation is more difficult because the target column name cannot be known at translation time.

3.3.5 Combining Different Data Sets

We focus on two basic operations: (1) Union of data sets; (2) Merging data sets.

Union: This is the case when the user wants to compute the union of data sets where most or all the variables are equal in each data set, but observations are different. Assume D_i already has observations and variables. The main issue here is that such statement does not guarantee all data sets have the same variables. Therefore, the translation must make sure the result data set includes all variables from all data sets setting to null those variables that are not present for a particular data set. First, we compute the union of all variables. Let p be the cardinality of $R.B = \{B_1, \ldots, B_p\}$. For each data set we need to compute the set of variables from R not present: $U.B - D_i.A$. A total order must be imposed on columns so that each position correspond to one column from U. Such order can be given by the order of appearance of variables in D_i. At the beginning variables are those from $R.A = D_1.A$. If there are new variables, not included in $R.A$ then they are added to $R.A$. This process gets repeated until D_m is processed. Then we just need to insert nulls in the corresponding variables when the result table is populated. The ith "SELECT" statement has p terms out of which n_i are taken from D_i and the rest are null.

Merging: This is the case where two data sets have a common key, some of the remaining columns are the same and the rest are different. If there are common columns among both data sets columns from the second one take precedence and overwrite the columns from the first data set. In contrast, SQL requires the user to rename columns with a new alias. In general, one of the data sets must be sorted by the matching variables. Let the result columns of M be $B_1, \ldots, B_p$. The dat sets D_1 and D_2 both contain the matching columns $A_1, A_2, \ldots, A_k$. A filtering process must be performed to detect common non-key columns. If there are common columns the column from D_2 takes precedence. The process is similar to the filtering process followed for arithmetic expressions or the union of data sets. This translates into SQL as a full outer join to include unmatched rows from D_1 and D_2.

3.4 Translating Subroutine Procedural Calls

We discuss translation of common procedural calls to equivalent calls using the data mining tool and then we discuss translation of embedded SQL. Translating procedural calls can be done with several alternative mechanisms: A first alternative is to call a data mining tool application programming interface (API) to automatically generate SQL code for univariate statistics or data mining algorithms. This issue is that several intermediate mathematical computations are left outside the final SQL script. Therefore, this alternative does not produce self-contained scripts. A second alternative is to replicate automatically generated SQL code generation in the translator. The benefit of this approach is that we can generate SQL scripts that can manage an entire data mining process, going from basic data transformations and descriptive statistics to actually tuning and testing models. The last alternative is to leave the translation open for manual coding into SQL, including the statistical language code in comments. We include this alternative for completeness because there may be specific statements and mathematical manipulations that cannot be directly translated. User intervention is required in this case.

Interestingly enough, the translation process may be faced with the task of handling embedded SQL statements. Such SQL is in general used to extract data from different DBMSs. We now explain the translation process: The most straightforward translation is a PROC SQL if the SQL corresponds to the same DBMS (in our case Teradata). Care must be taken in creating appropriate temporary tables whose names do not conflict with those used in the code. If data elements are being extracted from a different DBMS then the translation process becomes more complicated: it is necessary to check each referenced table exists in the target DBMS and SQL syntax may have differences. Therefore, it is best to manually verify the translation. If the SQL corresponds to some other DBMS there may be the possibility of finding non-ANSI features. This aspect may be difficult if queries are complex and have several nesting levels combining joins and aggregations. Many script versions need to be maintained.

3.5 Similarities and Differences

In Section 3 we explained how to translate code in the statistical language into SQL statements. Here we provide a summarized description of common features of both languages and where they differ.

Language Similarities

We present similarities going from straightforward to most important. In the statistical language there is no explicit instruction to scan and process the data set by observation: that happens automatically. In SQL the behavior is similar because there is no need to create a loop to process each row. Any SQL statement automatically processes the entire table. However, in the DBMS sometimes it is necessary to perform computations without using SQL to exploit arrays. Then regular looping constructs are required. Processing happens in a sequential fashion for each observation. In the statistical language each variable is created or updated as new assignment expressions are given for each row. In SQL a column is created when a new term in a "SELECT" statement is found. A column cannot be referenced if it has not been previously created with the "AS" keyword or it is projected from some table. Broadly speaking each new procedural call (PROC) reads or creates a new data set. Therefore, this aspect can be taken as a guideline to create temporary tables to store intermediate results.

Language Differences

We discuss differences going from straightforward to those we consider most challenging when making syntactic and semantic analysis.

Macros are different in both languages, being represented by stored procedures in SQL. In the statistical language a data set name or variable name can start with underscore, which may cause conflicts in translation. This can be solved by enclosing the table name or column name in SQL in quotes (e.g. "_name"). In the statistical language a missing value is indicated with a dot '.', whereas in SQL it is indicated with the keyword "NULL". A missing value can compared with the equality symbol '=' like any number, whereas SQL requires specialized syntax using the "IS NULL" phrase. Since a number can start with '.' a read-ahead scanner needs to determine if there is a digit after the dot or not. However, equations involving with missing values, in general, return a missing value as well. Both languages have similar semantics for missing information. In the statistical language variable name conflicts are solved in favor of the last reference. In SQL that conflict must be solved by qualifying ambiguous columns or by explicitly removing references to columns with the same name. To store results a table cannot contain columns with the same name. Therefore, for practical purposes duplicate column names must be removed during the SQL code generation. In the statistical language sorting procedures are needed to merge data sets. Sorting is not needed in SQL to join data sets. In fact, there is no pre-defined order among rows. Merging works in a different manner to joins. New variables are added to a given data set for further processing. A data set is always manipulated in memory, but new variables may not necessarily be saved to disk. In SQL a new table must be created for each processing stage. Tables are stored on disk. Some tables are created in permanent storage, whereas the rest of tables have to be created in temporary storage. The statistical language allows creating a data set with up to hundreds of thousands of variables. whereas SQL allows table with up to thousands of columns. This limitation can be solved by counting variables and creating a new table every one thousand variables; this will vertically partition the result table. Joins are required to reference columns from different partitions. The DBMS performs more careful retrieval of rows into main memory processing them in blocks, whereas the statistical language performs most of the processing in main memory based on a single table at one time.

3.6 Sufficient Statistics

We now explain fundamental statistics computed on the data set obtained from the data transformation process introduced in Section 3. These statistics benefit a broad class of statistical and machine learning techniques. Their computation can be considered an intermediate step between preparing data sets and computing statistical models. In general, most statistical data mining tools provide functionality to derive these statistics. In the literature the following matrices are called sufficient statistics [2, 11, 19] because they are enough to substitute the data set being analyzed in mathematical equations. Therefore, it is advantageous they are available for the data set to be analyzed.

Consider the multidimensional (multivariate) data set defined in Section 3.1: $X = \{x_1, \ldots, x_n\}$ with n points, where each point has d dimensions. Some of the matrix manipulations we are about to introduce are well-known in statistics, but we exploit them in a database context. We introduce the following two matrices that are fundamental and common for all the techniques described above. Let L be the *linear* sum of points, in the sense that each point is taken at power 1. L is a $d \times 1$ matrix, shown below with sum and column-vector notation. $L = \sum_{i=1}^{n} x_i$. Let Q be the *quadratic* sum of points, in the sense that each point is squared with a cross-product. Q is $d \times d$. $Q = XX^T = \sum_{i=1}^{n} x_i x_i^T$.

The most important property about L and Q is that they are much smaller than X, when n is large (i.e. $d << n$). However, L and Q summarize a lot of properties about X that can be exploited by statistical techniques. In other words, L and Q can be exploited to rewrite equations so that they do not refer to X, which is a large matrix. Techniques that directly benefit from these summary matrices include correlation analysis linear regression [11], principal component analysis [11, 10], factor analysis [11] and clustering [14]. These statistics also partially benefit decision trees [14] and logistic regression [11].

Since SQL does not have general support for arrays these matrices are stored as tables using dimension subscripts as keys. Summary matrices can be efficiently computed in two ways: using SQL queries or using UDFs [19]. SQL queries allow more flexibility, are portable, but incur on higher overhead. On the other hand, UDFs are faster, but they depend on the DBMS architecture and therefore may have specific limitations such as memory size and parameter passing. Having an automated way to compute summary matrices inside the DBMS simplifies the translation process.

4. EXPERIENCE FROM REAL PROJECTS

This section presents a summary of performance comparisons and main outcomes from migrating actual data mining projects into the DBMS. This discussion is a summary of representative successful projects. We first discuss a typical data warehousing environment; this section can be skipped by a reader familiar with the subject. Second, we present a summary of the data mining projects presenting their practical application and the statistical and data mining techniques used. Third, we present time measurements taken from actual projects at each organization, running data mining software on powerful database servers. We conclude with a summary of the main advantages and accomplishments for each project, as well as the main objections or concerns against migrating statistical code into the DBMS.

4.1 Data Warehousing Environment

The environment was a data warehouse, where several databases were already integrated into a large enterprise-wide database. The database server was surrounded by specialized servers performing OLAP and statistical analysis. One of those servers was a statistical server with a fast network connection to the database server.

First of all, an entire set of statistical language programs were translated into SQL using Teradata data mining program, the translator tool and customized SQL code. Second, in every case the data sets were verified to have the same contents in the statistical language and SQL. In most cases, the numeric output from statistical and machine learning models was the same, but sometimes there were slight numeric differences, given variations in algorithmic improvements and advanced parameters (e.g. epsilon for conver-

gence, step-wise regression procedures, pruning method in decision tree and so on).

4.2 Organizations and Data Mining Projects

We now give a brief discussion about the organizations where the statistical code migration took place. We also discuss the specific type of data mining techniques used in each case. Due to privacy concerns we omit discussion of specific information about each organization, their databases and the hardware configuration of their DBMS servers. We can mention all companies had large data warehouses managed by an SMP (Symmetric Multi-Processing) Teradata server having a 32-bit CPU with 4GB of memory on each node and several terabytes of storage. Our projects were conducted on their production systems, concurrently with other users (analysts, managers, DBAs, and so on).

The first organization was an insurance company. The data mining goal involved segmenting customers into tiers according to their profitability based on demographic data, billing information and claims. The statistical techniques used to determine segments involved histograms and clustering. The final data set had about $n = 300$k records and $d = 25$ variables. There were four segments, categorizing customers from best to worst.

The second organization was a cellular telephone service provider. The data mining task involved predicting which customers were likely to upgrade their call service package or purchase a new handset. The default technique was logistic regression [11] with stepwise procedure for variable selection. The data set used for scoring had about $n = 10$M records and $d = 120$ variables. The predicted variable was binary.

The third organization was an Internet Service Provider (ISP). The predictive task was to detect which customers were likely to disconnect service within a time window of a few months, based on their demographic data, billing information and service usage. The statistical techniques used in this case were decision trees and logistic regression and the predicted variable was binary. The final data set had $n = 3.5$M records and $d = 50$ variables.

4.3 Performance Comparison

We focus on comparing performance doing statistical analysis inside the DBMS using SQL (with Teradata data mining program) and outside using the statistical server (with existing programs developed by each organization). The comparison is not fair because the DBMS server was in general a powerful parallel computer and the statistical server was a smaller computer. However, the comparison represents a typical enterprise environment where the most powerful computer is precisely the DBMS server.

We now describe the computers in more detail. The DBMS server was, in general, a parallel multiprocessor computer with a large number of CPUs, ample memory per CPU and several terabytes of parallel disk storage in high performance RAID configurations. On the other hand, the statistical server was generally a smaller computer with less than 500 GB of disk space with ample memory space. Statistical and data mining analysis inside the DBMS was performed only with SQL. In general, a workstation connected to each server with appropriate client utilities. The connection to the DBMS was done with ODBC. All time measurements discussed herein were taken on 32-bit CPUs over the course of several years. Therefore, they cannot be compared with each other and they should only be used to understand performance gains within the same organization.

Table 1: Comparing time performance between statistical package and DBMS (time in minutes).

Task	Statistical package (outside DBMS)	DBMS (inside)
Build model:		
Segmentation	2	1
Predict propensity	38	8
Predict churn	120	20
Score data set:		
Segmentation	5	1
Predict propensity	150	2
Predict churn	10	1

Table 2: Time to compute linear models inside the DBMS and time to export X with ODBC (secs).

$n \times 1000$	d	SQL/UDF	ODBC
100	8	4	168
100	16	5	311
100	32	6	615
100	64	8	1204
1000	8	40	1690
1000	16	51	3112
1000	32	62	6160
1000	64	78	12010

We discuss tables from the database in more detail. There were several input tables coming from a large normalized database that were transformed and denormalized to build data sets used by statistical or machine learning techniques. In short, the input were tables and the output were tables as well. No data sets were exported in this case: all processing happened inside the DBMS. On the other hand, analysis on the statistical server relied on SQL queries to extract data from the DBMS, transform the data to produce data sets in the statistical server and then building models or scoring data sets based on a model. In general, data extraction from the DBMS was performed using the fast utilities which exported data records in blocks. Clearly, there exists a bottleneck when exporting data from the DBMS to the statistical server.

Table 1 compares performance between both alternatives: inside and outside the DBMS. We distinguish two phases in each project: building the model and scoring (deploying) the model on large data sets. The times shown in Table 1 include the time to transform the data set with joins and aggregations and the time to compute or apply the actual model. As we can see the DBMS is significantly faster. We must mention that to build the predictive models both approaches exploited samples from a large data set. Then the models were tuned with further samples. To score data sets the gap is wider, highlighting the efficiency of SQL to compute joins and aggregations to build the data set and then compute statistical equations on the data set. In general, the main reason the statistical server was slower was the time to export data from the DBMS and then a secondary reason was its more limited computing power.

Table 3: Project Outcomes.

Outcome	Insurance	Phone	ISP
Advantages:			
Decrease data movement	X		
Prepare data sets more easily	X	X	X
Build models faster	X		
Score data sets faster	X	X	X
Increase security	X	X	
Improve data management		X	
Objections:			
Traditional progr. lang.		X	X
Sampling on large data sets	X	X	
Lack statistical techniques		X	

Table 2 compares time performance to compute a linear model inside the DBMS and the time to export the data set with the ODBC interface. The DBMS runs on a Windows Server with 3.2 GHz, 4 GB on memory and 1 TB on disk, and represents a small database server. The linear models include PCA and linear regression, which can be derived from the correlation matrix of X in a single table scan using SQL and UDFs [19]. Clearly, ODBC is a bottleneck to analyze X outside the DBMS regardless of how fast the statistical package is. Exporting small sample of X may be feasible, but analyzing a large data set without sampling is much faster to do inside the DBMS.

4.4 Data Mining Users Feedback

We summarize main advantages of projects migrated into the DBMS as well as objections from users to do so. Table 3 contains a summary of outcomes. As we can see performance to score data sets and transforming data sets are positive outcomes in every case. Building the models faster turned out not be as important because users relied on sampling to build models and several samples were collected to tune and test models. Since all databases and servers were within the same firewall security was not a major concern. In general, improving data management was not seen as major concern because there existed a data warehouse, but users acknowledge a "distributed" analytic environment could be a potential management issue. We now summarize the main objections, despite the advantages discussed above. We exclude cost as a decisive factor to preserve anonymity of users opinion and give an unbiased discussion. First, many users preferred a traditional programming language like Java or C++ instead of a set-oriented language like SQL. Second, some specialized techniques are not available in the DBMS due to their mathematical complexity; relevant examples include Support Vector Machines, Non-linear regression and time series models. Finally, sampling is a standard mechanism to analyze large data sets.

5. RELATED WORK

There exist many proposals that extend SQL with data mining functionality. Teradata SQL, like other DBMSs, provides advanced aggregate functions to compute linear regression and correlation, but it only does it for two dimensions. Most proposals add syntax to SQL and optimize queries using the proposed extensions. UDFs implementing common vector operations are proposed in [21], which shows UDFs are as efficient as automatically generated SQL queries with arithmetic expressions, proves queries calling scalar UDFs are significantly more efficient than equivalent queries using SQL aggregations and shows scalar UDFs are I/O bound. SQL extensions to define, query and deploy data mining models are proposed in [15]; such extensions provide a friendly language interface to manage data mining models. This proposal focuses on managing models rather than computing them and therefore such extensions are complementary to our UDFs. Query optimization techniques and a simple SQL syntax extension to compute multidimensional histograms are proposed in [12], where a multiple grouping clause is optimized. Computation of sufficient statistics for classification in a relational DBMS is proposed in [9]. Developing data mining algorithms, rather than statistical techniques, using SQL has received moderate attention. Some important approaches include [13, 23] to mine association rules, [20, 18] to cluster data sets using SQL queries, [20, 17] to cluster data sets using SQL queries and [24] to define primitives for decision trees. Sufficient statistics have been generalized and implemented as a primitive function using UDFs benefiting several statistical techniques [19]; this work explains the computation and application of summary matrices in detail for correlation, linear regression, PCA and clustering.

Some related work on exploiting SQL for data manipulation tasks includes the following. Data mining primitive operators are proposed in [4], including an operator to pivot a table and another one for sampling, useful to build data sets. The pivot/unpivot operators are extremely useful to transpose and transform data sets for data mining and OLAP tasks [5], but they have not been standardized. Horizontal aggregations were proposed to create tabular data sets [16], as required by statistical and machine learning techniques, combining pivoting and aggregation in one function. For the most part research work on preparing data sets for analytic purposes in a relational DBMS remains scarce. To the best of our knowledge there has not been research work dealing with the migration of data mining data preparation into a relational DBMS. Also, even though the ideas behind the translator are simple, they illustrate the importance of automating SQL code generation to prepare data sets for analysis.

6. CONCLUSIONS

We presented our experience on the migration of statistical analysis into a DBMS, focusing on data preprocessing (cleaning, transformation, summarization), which is in general the most time consuming, not well planned and error-prone task in a data mining project. Summarization generally has to be done at different granularity levels and such levels are generally not hierarchical. Rows are selected based on a time window, which requires indexes on date columns. Row selection (filtering) with complex predicates happens on many tables, making code maintenance and query optimization difficult. To improve performance it is necessary to create temporary denormalized tables with summarized data. In general, it is necessary to create a "universe" data set to define left outer joins. Model deployment requires importing models as SQL queries or UDFs to deploy a model on large data sets. We also explained how to compute sufficient statistics on a data set, that benefit a broad class

of data mining and machine learning techniques, including correlation analysis, clustering, principal component analysis and linear regression. We presented a performance comparison and a summary of main advantages when migrating statistical programs into the DBMS by translating them into optimized SQL code. Transforming and scoring data sets is much faster inside the DBMS, whereas building a model is also faster, but less significant because sampling can help analyzing large data sets. In general, data transformation and analysis are faster inside the DBMS and users can enjoy the extensive capabilities of the DBMS (querying, recovery, security and concurrency control).

We presented a prototype tool to translate statistical scripts into SQL, to automate and accelerate the migration of data preparation, which is the most time consuming phase in a data mining project. The tool main goal is to generate SQL code that produces data sets with the same content as those generated by the statistical language: such data sets become the input for statistical or machine learning techniques (a so-called analytical data set).

7. REFERENCES

[1] A. Aho, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1986.

[2] P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *ACM KDD Conference*, pages 9–15, 1998.

[3] S. Chaudhuri. An overview of query optimization in relational systems. In *ACM PODS Conference*, pages 84–93, 1998.

[4] J. Clear, D. Dunn, B. Harvey, M.L. Heytens, and P. Lohman. Non-stop SQL/MX primitives for knowledge discovery. In *ACM KDD Conference*, pages 425–429, 1999.

[5] C. Cunningham, G. Graefe, and C.A. Galindo-Legaria. PIVOT and UNPIVOT: Optimization and execution strategies in an rdbms. In *VLDB Conference*, pages 998–1009, 2004.

[6] L.D. Delwiche and S.J. Slaughter. *The SAS little book: a primer*. SAS, 4th edition, 2003.

[7] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Addison/Wesley, Redwood City, California, 3rd edition, 2000.

[8] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27–34, November 1996.

[9] G. Graefe, U. Fayyad, and S. Chaudhuri. On the efficient gathering of sufficient statistics for classification from large SQL databases. In *ACM KDD Conference*, pages 204–208, 1998.

[10] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, 1st edition, 2001.

[11] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning*. Springer, New York, 1st edition, 2001.

[12] A. Hinneburg, D. Habich, and W. Lehner. Combi-operator-database support for data mining applications. In *VLDB Conference*, pages 429–439, 2003.

[13] R. Meo, G. Psaila, and S. Ceri. An extension to SQL for mining association rules. *Data Min. Knowl. Discov*, 2(2):195–224, 1998.

[14] T.M. Mitchell. *Machine Learning*. Mac-Graw Hill, New York, 1997.

[15] A. Netz, S. Chaudhuri, U. Fayyad, and J. Berhardt. Integrating data mining with SQL databases: OLE DB for data mining. In *IEEE ICDE Conference*, 2001.

[16] C. Ordonez. Horizontal aggregations for building tabular data sets. In *ACM DMKD Workshop*, pages 35–42, 2004.

[17] C. Ordonez. Programming the K-means clustering algorithm in SQL. In *ACM KDD Conference*, pages 823–828, 2004.

[18] C. Ordonez. Integrating K-means clustering with a relational DBMS using SQL. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 18(2):188–201, 2006.

[19] C. Ordonez. Building statistical models and scoring with UDFs. In *ACM SIGMOD Conference*, pages 1005–1016, 2007.

[20] C. Ordonez and P. Cereghini. SQLEM: Fast clustering in SQL using the EM algorithm. In *ACM SIGMOD Conference*, pages 559–570, 2000.

[21] C. Ordonez and J. García-García. Vector and matrix operations programmed with UDFs in a relational DBMS. In *ACM CIKM Conference*, pages 503–512, 2006.

[22] C. Ordonez and J. García-García. Referential integrity quality metrics. *Decision Support Systems Journal*, 44(2):495–508, 2008.

[23] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: alternatives and implications. In *ACM SIGMOD*, pages 343–354, 1998.

[24] K. Sattler and O. Dunemann. SQL database primitives for decision tree classifiers. In *ACM CIKM Conference*, pages 379–386, 2001.

[25] Teradata. *Teradata Warehouse Miner Release Definition Release 5.1*. Teradata (NCR), 2006.

[26] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *ACM SIGMOD Conference*, pages 103–114, 1996.