Data Set Preprocessing and Transformation in a Database System

Carlos Ordonez
University of Houston
Houston, TX 77204, USA [*]

Abstract

In general, there is a significant amount of data mining analysis performed outside a database system, which creates many data management issues. This article presents a summary of our experience and recommendations to compute data set preprocessing and transformation inside a database system (i.e. data cleaning, record selection, summarization, denormalization, variable creation, coding), which is the most time-consuming task in data mining projects. This aspect is largely ignored in the literature. We present practical issues, common solutions and lessons learned when preparing and transforming data sets with the SQL language, based on experience from real-life projects. We then provide specific guidelines to translate programs written in a traditional programming language into SQL statements. Based on successful real-life projects, we present time performance comparisons between SQL code running inside the database system and external data mining programs. We highlight which steps in data mining projects become faster when processed by the database system. More importantly, we identify advantages and disadvantages from a practical standpoint based on data mining users feedback.

Keywords: attribute construction, data transformation, preprocessing, summarization

1 Introduction

In this work we focus on data preprocessing, an important data mining topic that has received scant attention in the literature. Nowadays transaction processing [6] and data warehousing [8] of large databases are performed mostly by relational database systems (DBMSs). Analyzing large databases with data mining techniques is a different story. Despite the existence of many data mining techniques offered by a database system [8, 16], there exist many data mining tasks that are routinely performed with external tools [9, 8]. This is due, among other reasons, to the existence of advanced data mining programs (i.e. open-source, third party commercial software), the availability of mathematical libraries (e.g. LAPACK, BLAS), the lack of expertise of data mining practitioners to write correct and efficient SQL queries and legacy code. Therefore, the database system just acts as a data provider, and users write SQL queries to extract summarized data with joins and aggregations. Once "raw" data sets are exported they are further cleaned and transformed [8] depending on the task at hand, outside the database system. Finally, when the data set has the desired variables (features) and an appropriate number of records (observations, instances), data mining models [9] are iteratively computed, interpreted and tuned. From all the tasks listed above preparing, cleaning and transforming data for analysis is generally the most time-consuming task [8] because it requires significant effort to convert normalized tables in the database into denormalized tabular data sets [8], appropriate for data mining algorithms. Unfortunately, manipulating data sets outside the database system creates many data management issues: data sets must be recreated and re-exported every time required tables change, security is compromised (an essential aspect today due to the Internet). Data sets need to be re-exported when new variables (features) are created, models need to be deployed inside the database system, and different users

[*] © IOS Press, 2011. This is the author's version of the work. The official version of this article was published in Intelligent Data Analysis (IDA Journal). 15(4), 2011. DOI: 10.3233/IDA-2011-

may have inconsistent version of the same data set in their own computers. Therefore, we defend the idea of transforming data sets and computing models inside a modern database system, exploiting the extensive features of the SQL language and enjoying the extensive data management capabilities provided by the database system. Our motivation to migrate data mining preprocessing into the database system, yields the following benefits. The database system provides powerful querying capabilities through SQL and 3rd party tools. Even further, the database system provides the best data management functionality (maintaining referential integrity, transaction processing, fault-tolerance, security). Additionally, the database system server is generally a fast computer and thus results can be obtained sooner and with less programming effort.

We assume the database system provides basic data mining functionality. That is, the database system is capable of performing common data mining tasks like regression [9], clustering [9], classification [19], PCA [3] or association rules [10, 15], among others. On the other hand, we assume the database system supports the SQL language, which is used to store and retrieve information from the database system [6, 8]. Based on experience from data mining projects we have become aware that data mining practitioners have difficulty writing correct and efficient SQL code to extract data from the database system or to transform their data sets for the data mining task at hand. To overcome such issues, practitioners resort to high level programming languages (e.g. Java, C++, SAS) to manipulate and transform their data sets. Thus, practitioners end up creating long programs mixing data transformation and modeling tasks together. Most of the "tedious" programming effort is spent on manipulating and transforming input tables to build the required data set: this is the main aspect studied in this article. We identify practical issues as well as common solutions for data transformation tasks. We present evidence transforming data sets and building models on them is more efficient to do inside the database system than outside with external data mining programs. We motivate much needed research on data preparation. Finally, we identify advantages and disadvantages between external data mining processing outside a database system and internal processing inside the database system. Our lessons can be valuable for data mining practitioners or AI people extracting data from a database system.

In this article, we provide practical guidelines to translate common data preparation tasks programmed in a traditional high-level programming language into SQL, which have quite different computing models behind. However, there are certain similarities and differences between both kinds of languages that make the problem interesting and challenging. A traditional language is an imperative language that manipulates data sets as record files, but not strictly as relational tables. A high-level programming language includes syntactic constructs for arithmetic expressions, flow control and function calls. In such languages, arrays can be used to implement matrix operations. In addition, some languages (e.g. C++) allow pointer manipulation. On the other hand, SQL is a set oriented language that manipulates data sets as tables, but which allows specifying relationships among tables with primary and foreign keys relationships. A set of tables is manipulated with join operations among their keys. In general, SQL does not support arrays and thus matrix manipulation is more difficult. Even further, pointer manipulation is forbidden. Scanning the data set needs to be carefully programmed in a traditional language, whereas it is automatic in SQL. Query optimization is a fundamental issue in SQL, whereas in a traditional language the goal is to optimize mathematical operations and memory usage.

The article is organized as follows. Section 2 discusses important practical issues and common solutions when preprocessing and transforming data sets for data mining analysis. Section 3 presents guidelines to code common data transformations in SQL. Section 4 presents performance comparisons and users feedback from projects where external data preparation programs were successfully migrated into the database system. Related work is discussed in Section 5. Finally, Section 6 concludes the article.

2 Issues Preprocessing and Transforming Data Sets

We present important practical issues to preprocess and transform tables in a database in order to build a data set suitable for analysis. These issues have been collected from several projects helping users migrate analysis performed in external data mining tools into a database system. We present issues in three groups: (1) creating and transforming variables for analysis; (2) selecting (filtering) records for analysis; (3) developing and optimizing SQL queries.

In general, the main objection from users against using SQL is to translate existing legacy code. Commonly such code has existed for a long time (legacy programs), it is extensive (there exist many programs) and it has been thoroughly 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 database system provides good, but still limited, data mining functionality, compared to sophisticated data mining and 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. In the following paragraphs, we discuss some of the most important database issues when tables are manipulated and transformed to prepare a data set for data mining. We pay particular attention to building the data set with the right set of columns, getting correct results, and optimizing queries [6]. The issues we will discuss represent experience that has been gained by testing, adapting and optimizing SQL code that has been manually written or automatically generated by data mining tools. Certain data preparation tasks can be partially automated, whereas other tasks require significant manual work by the user. We will carefully distinguish both cases. Throughout this section, we present examples of typical SQL queries in a store.

2.1 Issues Creating and Transforming Data Set Variables

We explain issue to create variables (features) for analysis. We also consider the case where the variable is already available as a column, but needs to be transformed. Creating and transforming variables is the most important aspect to understand variables.

Summarization and Aggregation

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 data mining techniques require measures (or metrics), which translate as sums or counts computed with SQL. Unfortunately, granularity levels are not hierarchical (like cubes or OLAP [6]) making the use of separate summary tables necessary (e.g. summarization by product or by customer, in a retail database). Thus, the user needs to decide: which tables and columns need to be summarized, the primary key(s) of output tables (i.e. grouping columns) and which kind of aggregations (e.g. sum(), average(), min()) are required.

For a data mining practitioner it is best to create as many variables (features, dimensions) as possible to identify those that can help computing a more accurate model. Then summarization tends to create tables with many columns (sometimes hundreds), which make data mining analysis harder and query processing slower. This task is related to attribute relevance, which determines which attributes are more important for a given model. Attribute relevance is commonly determined by dimensionality reduction and feature selection techniques. Most data mining techniques, especially for classification and regression, are designed to perform variable (feature) selection [9]. A straightforward optimization is to compute as many dimensions in the same statement exploiting the same GROUP-BY clause, when possible.

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;
```

Transforming Categorical Variables

Many statistical and machine learning models require categorical variables to be transformed into numeric ones. One common approach to solve such problem is to code each categorical value as one binary attribute. Given k categorical values they are coded as $k-1$ binary attributes in order to create an uncorrelated set of variables (i.e. the kth value can be deduced from the other $k-1$). Such transformation is solved as a list of $k-1$ CASE statements in SQL. From a database perspective, this transformation is simple and fast to compute since the number of records remains the same.

This transformation does not require significant manual work by the user since the coding phase can be automated. Figuring out which categorical values are more relevant for a certain model requires computing correlations or contingency tables.

Denormalization

It is well known that on-line transaction processing (OLTP) database systems update normalized tables. Therefore, it becomes necessary to gather data from several tables and store attributes in one table. The explanation is simple. Normalization makes transaction processing faster and ACID [6] 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 essential 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.

In general, denormalizing does not require significant user involvement since joins are computed by the database system and the logical model dictates which columns should be used to compute joins. Denormalization may introduce unnecessary attributes because those attributes may be repeated in multiple tables. Therefore, it is necessary to detect and remove redundant attributes: extra attributes having unitary correlation should be removed.

```
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;
```

Cross-tabulation and data set flattening

This data transformation is a combination of summarization and denormalization. Data mining models generally require having all dimensions for analysis in a single data set containing a set of records having several attributes. The distinguishing characteristic of the table storing such data set is that in general it has only one column for its primary key (e.g. product id, customer id) and several numeric and discrete attributes. In certain cases the table may have a composite primary key formed by an identifier and some time-dependent column. Since tables tend to be available normalized, certain columns need to be aggregated whereas other columns are used to transpose (pivot) the input table. In other words, aggregation and pivoting are combined to build the data set. This transformation effectively transforms the input table into a "flattened" data set, where several values are cross-tabulated. Such kind of SQL statement has been called a horizontal aggregation [12]. A typical example is to aggregate numeric columns converting date and time values into multiple columns for one year of sales:

As noted above summarization requires a lot of user manual work, with the additional required denormalization. Therefore, getting all the pieces together in one data set requires careful code development and testing.

```
SELECT
  customer_id
 ,sum(CASE WHEN extract(month from salesDate)=1  THEN salesAmt END) AS salesAmtJan
 ,sum(CASE WHEN extract(month from salesDate)=2  THEN salesAmt END) AS salesAmtFeb
  ..
 ,sum(CASE WHEN extract(month from salesDate)=12 THEN salesAmt END) AS SalesAmtDec
FROM customerSales
WHERE extract(year from salesDate)=2009
GROUP BY customer_id;
```

Statistics columns

It is fair to say that univariate statistics are the main tool to understand the data set before computing a model, and even as the model is being validated and tuned. Such statistics include mean, variance, standard deviation and extreme values (min, max). Also, statistics include the mode and histograms for discrete variables. Unfortunately, some statistics like the mean are not distributive [17]. In other words, they cannot be computed from existing means or standard deviations. 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 [2, 16] prove useful for both simple statistics as well as sophisticated multidimensional models. This solution is represented by the sufficient statistics n, L, Q [16], explained in detail in Section 3.5. This task can be automated for the most part (i.e. little or no user manual work) since these sufficient statistics can be automatically computed as explained in the literature.

2.2 Issues Selecting Data Set Records for Analysis

For the issues discussed below we assume we have an acceptable set of variables. So we now concentrate on explaining how to select the best records from the database for analysis. Selecting and filtering records is the second most important aspect to understand the data set. Many times selecting the right set of records produces much better models than using all existing records. In general, this task requires excellent domain knowledge on the problem. Thus it cannot automated. We should emphasize again preparing data is an iterative process. Therefore, it may be necessary to add variables and then select records again multiple times.

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 a date column 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.

This task is generally guided by users with domain knowledge. Query writing is fairly simple since it only requires specifying date/time ranges.

```
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';
```

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 data mining on one data set only. 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).

More detailed data generally means more precise models, but also more manual work summarizing and aggregating. Therefore, the level of detail existing in the database is correlated to the amount of manual effort required for preparing data sets.

2.3 Issues Developing and Optimizing SQL Code

We now consider tasks related to developing SQL queries to transform and build data sets. Such aspects are relevant only in the context of a database system. As such these aspects do not help understanding the data set, but being able to efficiently manipulate it. Manual work by the data mining practitioner can be significant if there is no tool to automate SQL query writing. Manual work can become more complicated if query optimization is involved, especially with large tables.

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. Data mining 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).

User involvement is small. The main issue is getting the main table that includes all records. This issue can be easily solved if there is one table that is guaranteed to have every record (e.g. every customer).

```
SELECT  T0.record_id
       ,T1.A1
       ,T2.A2
        ..
       ,Tk.Ak
FROM T0 LEFT JOIN T1 ON T0.record_id= T1.record_id
        LEFT JOIN T2 ON T0.record_id= T2.record_id
        ..
        LEFT JOIN Tk ON T0.record_id= Tk.record_id;
```

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 T
 ..
) WITH DATA;

SELECT * FROM X JOIN T1 ...
SELECT * FROM X JOIN T2 ...
..
SELECT * FROM X JOIN Tn ...
```

Choosing between views and 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.

Filtering records at multiple data preparation stages

Selection of rows can be done at several stages, in different tables. Such filtering is done to discard outliers [13], to discard records with a significant missing information content (including referential integrity), 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 [6], but such optimization is not easy to apply to 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.

Multiple primary keys

Different subsets of large tables have different primary keys. This issue basically means such tables are not compatible with each other to join and aggregate directly. At some point in the SQL code such large tables with different primary keys must be joined and later summarized by the primary key of one of them. Join operations will be slow because indexing involves foreign keys with large cardinalities, in contrast to common star joins in star schemas. 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. Data mining models may be built by customer or by account, having some statistical variables in common.

Model deployment

Even though many models are built outside the database system with data mining tools, in the end the model must be applied or deployed in the database [16]. 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 database system becomes a bottleneck [14, 16]. This problem gets compounded with results interpretation when it is necessary to relate models 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 database system. Nowadays, model deployment basically happens in two ways: using SQL queries if the mathematical computations are relatively simple or with UDFs [14, 16], if the computations are more sophisticated. In most cases, such scoring process can work in a single table scan, providing good performance.

Database system server resources 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.

Lessons Learned: Most Useful Queries

Data transformation is a time consuming task in any data mining project since the data set needs to be built tailored to the problem at hand. In general, data mining practitioners think writing data transformations in SQL is not easy. The reason is simple: despite the abundance of data mining tools users still need to understand the basics of query processing. The most common queries needed to create data sets for data mining are listed below.

- 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.
- Cross-tabulation, which combines transposition (pivoting) and aggregation to create a data with multiple dimensions.
- Variable transformation in which the number of records remains the same, but the number of columns may increase or decrease. Such transformations include categorical coding, logarithms, rescaling, normalization, merging categorical values and null replacement.
- Rewriting complex queries to perform aggregations before joins, when query result correctness is preserved. Many times this optimization is handled by the query optimizer.

3 Translating Data Transformations in a High-level Programming Language into SQL Queries

An important goal is to automate data transformation whenever there is code already written. This section gives guidelines to translate common transformations in a high-level programming language into SQL. We point out similarities and differences between both languages. Finally, we explain how to efficiently compute common sufficient statistics in SQL that benefit several models, which effectively summarize large data sets.

3.1 Definitions

In general, the goal of data preparation (and transformation) is to build a so-called "analytic" data set. Let $X = \{x_1, \ldots, x_n\}$ be the data set with n records, where each record has d attributes (variables, features). Each attribute can be numeric or categorical.

In a high-level programming language the data set is manipulated as a file of records, as defined above. In general, data sets are stored as flat files (i.e. plain text files), where each line contains one record. In each record values may be separated by a special character (a comma for instance) or they may have a fixed length (i.e. values are located by position). We do not consider the case where data sets are stored as binary files for efficient random access since such format is not commonly used by data mining practitioners. In SQL the terms used to manipulate a data set 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 linked by foreign key [6] relationships.

3.2 Data Set Storage

In a high level programming language (e.g. C++, Java, SAS) a program produces data sets which are basically files with a set of records, where each record has fixed columns. Columns can be of numeric, string or date (time) data types. On the other hand, SQL manipulates data as tables, with the additional constraint that they must have a primary key (sometimes artificially generated). In general, data sets in a high-level programming language are sequentially scanned, loaded and processed in main memory row by row. On the other hand, in SQL relational operations receive tables as input and produce one table as output, making internal processing in blocks of rows.

3.3 Statement Translation

We distinguish two main classes of statements making a program that require different translation mechanisms: (1) Data manipulation (data set transformation). (2) Function calls (e.g. procedures, methods). Most statements we discuss are for data manipulation.

Arithmetic Equations

Columns in the data set are treated as variables. In a compiled programming language every variable must be declared, which can be used as columns in a temporary table. An assignment statement assigns a value to an existing variable. 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 updates variables with arithmetic expressions. Most math functions in a high level programming language 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.

If-then-else

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-ELSE with nesting and two or more assignments per condition. A chained IF-THEN statement is translated into an SQL CASE statement where each IF condition is transformed into a WHEN clause. It is important to watch out for new columns when new variables are declared. IF statements can be nested several levels. There may be loops with array variables. This case will be analyzed separately. Nested IF-THEN statements are flattened into "WHEN" substatements in a CASE statement. Comparison for numbers and strings use similar operators. However, date comparisons are different and therefore require special functions. Comparison operators have similar syntax in both languages, whose translation requires a simple equivalence table. Negation (NOT), parenthesis and strings, require similar translation (compilation) techniques.

Looping Constructs

We pay particular attention to FOR loops, which are particularly useful to manipulate matrices and vectors. In general, the number of iterations is fixed. On the other hand, WHILE loops, with an undetermined number of iterations, are generally required to scan the input data set, which is done automatically by SQL. In a high-level programming language there are 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.

Combining Different Data Sets

We focus on two basic operations: (1) Union of two data sets; (2) Merging two data sets. Further combinations used these two operations as primitives.

Union: This is the case when the user wants to compute the union of data sets where most or all the columns are equal in each data set, but records are different. The main issue here is that such statement does not guarantee all data sets have the same columns. 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. A total order must be imposed on columns so that each position correspond to one column from U. Then we just need to insert nulls in the corresponding variables when the result table is populated.

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 other common columns (other than the key) between both data sets they must be renamed in order to have unique column names. Similarly, SQL requires the user to rename columns with a new alias. In general, in a high-level programming language both data sets must be sorted by the matching column(s). Such sorting process is unnecessary in SQL since the join operation serves that purpose. A filtering process must be performed to detect common non-key columns. This translates into SQL as a full outer join to include unmatched rows from either data set.

Translating Function and Subroutine Calls

Translating subroutine calls requires defining a stack-based calling mechanism, which is difficult in SQL. A solution is to create separate SQL stored procedures for each major function and then integrate their calls with additional stored procedures. An alternative is to create a flat script in SQL that contains all "unfolded" function calls (i.e. substituting the function call with corresponding SQL code).

3.4 Similarities and Differences

In the previous sections, we explained how to translate code in a high-level programming language into equivalent SQL statements. We now provide a summary of common features of both languages and where they differ.

Language Similarities

We present similarities going from straightforward to most important. Values can be stored on a declared variable only. In both languages it is necessary to declare variables and columns, respectively. However, a high-level programming language may include dynamic storage allocation with pointers. Such cases require paying close attention to which variables get stored on disk. In a high level programming language each variable is updated as new assignment expressions appear in the code. In SQL it is necessary to create a column for each declared variable. In SQL a column cannot be referenced if it has not been previously created with the "AS" keyword or it comes from some joined table. In most cases, each new function reading or writing an entire file, 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

In a high-level programming language there is explicit code to scan and process the data set record by record. In contrast, in SQL there is no need to create a loop to process each row: any SQL statement automatically processes the entire table. However, in the database system sometimes it is necessary to perform computations without using SQL to exploit arrays. In other words, regular looping constructs are still required (stored procedures partially solve the issue).

High-level programming languages do not have support for managing missing information. SQL provides specialized syntax and operators for null values. Files do not necessarily contain column names, although it is common to include them as a header or in a separate metadata file. On the other hand, SQL requires defining column names before populating them with values. 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 writing. In a high level programming 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. 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.

3.5 Sufficient Statistics

We now explain fundamental statistics computed on the data set obtained from the data transformation process. These statistics benefit a broad class of data mining models. Their computation can be considered an intermediate step between preparing data sets and computing statistical models. In the literature the following matrices are called sufficient statistics [2, 9, 14, 16] because they are enough to substitute the data set being analyzed in mathematical equations, preserving its most important statistical properties. Therefore, it is preferable 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.

$$L = \sum_{i=1}^{n} x_i. \tag{1}$$

Each entry is the linear sum of each dimension:

$$L = \begin{bmatrix} \sum X_1 \\ \sum X_2 \\ \vdots \\ \sum X_d \end{bmatrix}$$

Let Q be the *quadratic* sum of points, in the sense that each point is squared with a cross-product. Q is a $d \times d$ matrix.

$$Q = XX^T = \sum_{i=1}^{n} x_i x_i^T. \tag{2}$$

Matrix Q has sums of squares in the diagonal and sums of cross-products off the diagonal:

$$Q = \begin{bmatrix} \sum X_1^2 & \sum X_1 X_2 & \ldots & \sum X_1 X_d \\ \sum X_2 X_1 & \sum X_2^2 & \ldots & \sum X_2 X_d \\ \vdots & & & \vdots \\ \sum X_d X_1 & \sum X_d X_2 & \ldots & \sum X_d^2 \end{bmatrix}$$

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 several data mining 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 [9], principal component analysis [9, 8], factor analysis [9] and clustering. These statistics also partially benefit decision trees [8] and logistic regression [9].

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 [14, 16]. SQL queries allow more flexibility, are portable, but incur on higher overhead. On the other hand, UDFs are faster, but they depend on the database system architecture and therefore may have specific limitations such as memory size and parameter passing. Having an automated way to compute summary matrices inside the database system simplifies the translation process. The SQL code to derive sufficient statistics for a full matrix Q on X is:

```
SELECT
 ,count(*) AS n
 ,sum(X1) AS L1
 ,sum(X2) AS L2
 ..
 ,sum(Xd) AS Ld
 ,sum(X1*X1) AS Q1
 ,sum(X1*X2) AS Q12
 ,..
 ,sum(X2*X1) AS Q21
 ,sum(X2*X2) AS Q2
 ..
 ,sum(X(d-1)*Xd) AS Q(d-1)d
 ,sum(Xd*Xd) AS Qd
FROM X;
```

This statement is fast, but may present limitations when d is high (given a maximum number of columns allowed by the database system), in which case a vertical layout for X is required. For further details consult [16].

4 Experience and Feedback from Real-Life Projects

This section presents performance comparisons, advantages and disadvantages when migrating actual data mining projects into the database system. 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 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 external code in a high-level programming language into the database system.

4.1 Data Warehousing Environment

The computing environment was a data warehouse, where several databases were already integrated into a large enterprise-wide database. The database server was surrounded by specialized computers, performing OLAP and data mining analyses, connected by a high-speed network. Some of those computers were used for data mining.

First of all, an entire set of high-level programming language programs was translated into SQL. Second, in every case the data sets were verified to have the same contents in the external files and in SQL. In most cases, the numeric output from data mining models was the same, but sometimes there were slight numeric differences, given variations in algorithmic improvements and advanced parameters (e.g. epsilon for convergence, step-wise regression procedures, pruning method in decision tree, outlier handling and so on).

4.2 Background on Companies and Their Data Mining Projects

We now give a brief discussion about the companies where the 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

Table 1: Project Outcomes.

	Insurance	Phone	ISP
Advantages:			
Decrease data movement	X	X	
Security and information assurance	X	X	X
Prepare data sets more easily and faster	X	X	X
Build models faster (when DBMS has technique)	X	X	X
Score data sets faster (with SQL queries)	X	X	X
Reduce data management outside DBMS		X	
Disadvantages:			
Prefer a traditional programming language		X	X
Dependence on SQL expert	X	X	X
Sampling large data sets is acceptable	X	X	
DBMS lacks advanced data mining techniques		X	

of specific information about each company (i.e. their name), their databases (size, specific table names) and the hardware configuration of their database system servers (in the range of terabytes). We can mention all companies had large database systems managed by an SMP (Symmetric Multi-Processing) Teradata server having several GB of memory on each node and several terabytes of disk storage. Our projects were conducted on the data warehouse production systems, concurrently with other users and their applications (e.g. 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 data mining 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 [9] 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 data mining 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 Users Feedback

We summarize main advantages of projects migrated into the database system as well as objections from users to do so. Table 1 contains a summary of pros and cons. 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. A second important advantage was security: there was no need to export sensitive information outside the database system. A third advantage was simplified data management as databases and their analysis was centralized. We now summarize the main objections, despite the advantages discussed above. We exclude cost as a decisive factor since all companies had enough budget to implement a large data warehouse and we did not have specific cost information on hardware and software. First, many users

Table 2: Comparing time performance between external data mining program and database system (time in minutes).

Task	External program (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

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 database system due to their mathematical complexity; relevant examples include Support Vector Machines, non-linear regression, neural nets and time series models. Finally, sampling is a standard mechanism to get approximate models from large data sets, although model error needs to be controlled.

4.4 Processing Time Comparison

We focus on comparing processing time running data mining processing inside the database system with SQL code (generated by a data mining program) and outside with existing programs developed by each organization. Such programs were developed mainly in Java and SAS and ran on the data mining practitioner workstation. The comparison is not fair because the database system server was in general a powerful parallel computer and the external computer was slower and smaller. However, the comparison represents a typical enterprise environment where the most powerful computer is precisely the database system server.

We now describe the computers in more detail. The database system 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 external computer was generally a smaller computer with less than 500 GB of disk space with ample memory space. Data mining processing inside the Teradata DBMS was performed only with SQL and UDFs. In general, a workstation connected to each server with appropriate client utilities. Client computers connected with ODBC/JDBC. Data sets were exported with fast bulk utilities. 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.

We discuss tables from the database in more detail. There were several input tables coming from a large normalized enterprise data warehouse that were joined, transformed and denormalized to build data sets used by data mining techniques. In short, the input were many tables and the output were a few tables (in many cases only one). No data sets were exported in this case: all processing happened inside the database system exploiting SQL and UDFs [16]. On the other hand, analysis on the data mining computer relied on bulk utilities, ODBC and SQL to extract data from the database system. Most transformations happened outside the database system. Data mining models were computed in an external data mining program (e.g. statistical package, data mining program, custom code). Once an acceptable was obtained, data sets were scored inside the database system with SQL code (manually written or generated by a tool). In general, data extraction from the database system was performed using fast bulk utilities which exported data records in blocks.

Table 3: Time to compute linear models inside the DBMS (with SQL and UDFs) and time to export X with BULK/ODBC (secs).

$n \times 1000$	d	DBMS	BULK	ODBC
100	8	4	35	168
100	16	5	64	311
100	32	6	119	615
100	64	8	235	1204
1000	8	40	353	1690
1000	16	51	617	3112
1000	32	62	1216	6160
1000	64	78	2558	12010

Table 2 compares performance between both alternatives: inside and outside the database system. As explained above, we distinguish two phases in each project: building the model and scoring (deploying) the model on large data sets. The times shown in Table 2 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 database system 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 model equations on the data set. In general, the main reason the data mining computer was slower was the time to export data from the database system and to a second extent its more limited computing power.

We decided to analyze if exporting data sets is indeed a bottleneck on a separate computer. The database system runs on a Windows Server with CPU running at 3.2 GHz, 4 GB on memory and 1 TB on disk. Table 3 compares time performance to compute a linear model inside the database system and the time to export the data set with the ODBC interface and bulk utilities. Clearly, there exists a bottleneck when exporting data from the database system to the data mining server, even with bulk utilities. 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 [16]. Clearly, exporting a data set is a bottleneck to analyze X outside the database system, regardless of how fast an external program is. Exporting a small sample from X can accelerate processing, but analyzing a large data set without sampling is much faster to do inside the database system.

5 Related Work

Creating new attributes for analysis has received attention. One important technique is constructive induction [1, 11], where relevant attributes to a learning task are created. Such attributes expand the representation space and enable a descriptive summarization of input records. In our case, building attributes with aggregations is the closest transformation to constructive induction since it provides a summarized representation of records. However, such construction is guided by the practitioner and domain knowledge. Deciding which new attributes are more relevant and setting the best aggregation granularity are aspects which could be improved with constructive induction. On the other hand, new numerical attributes are constructed from binary ones in the IGLUE system [11] through constructive induction; this approach is restricted to binary attributes (representing concepts). In our case, this technique could be applied to summarize combinations of categorical attributes, coded as binary attributes.

There exist many proposals that extend SQL with data mining functionality. Relational database systems provide advanced aggregate functions to compute linear regression and correlation, but it only does it for two dimensions. Most research proposals add syntax to SQL and optimize queries using the proposed extensions, but have limited applicability as they require extensive changes to the database system source code.

UDFs implementing common vector operations are proposed in [16], 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. Several SQL extensions to define, query and deploy data mining models are discussed in [8]; 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. Computation of sufficient statistics for classification in a relational database system is proposed in [7]. Sufficient statistics have been generalized and implemented as a primitive function using UDFs benefiting several data mining techniques [16]; 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 and unpivot operators are 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 [12], as required by data mining techniques, combining pivoting and aggregation in one function. For the most part research work on preparing data sets for analytic purposes in a relational database system remains scarce. To the best of our knowledge there is scarce research work on the migration of data mining data preparation into a database system. For further reference on preprocessing data sets in practice, consult [8]. A preliminary version of this article appeared in [18].

6 Conclusions

We presented our experience migrating data set preprocessing and transformation performed by external programs into a database system exploiting the SQL language. In general, data preprocessing is the most time consuming, not well planned and most error-prone task in a data mining project. Summarization generally has to be done at different aggregation granularity levels and such levels are generally not hierarchical with respect to each other (they are best represented by a lattice). Input data records are selected based on a time window, which requires indexes on date columns. Row selection (filtering) with complex predicates happens on many tables (including temporary 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 compute left outer joins in order to preserve every available record. Model deployment requires importing data mining models as SQL queries or UDFs to apply the model on new records. We briefly explained how to compute a powerful set of sufficient statistics on a data set, that benefit a broad class of data mining techniques, including correlation analysis, clustering, principal component analysis and linear regression. We presented a summary of main advantages when migrating programs written in traditional programming languages into the database system by translating them into optimized SQL code. Also, we presented a processing time comparison. We showed transforming and scoring data sets is much faster inside the database system, whereas building a model is also faster, but less significant because sampling can help analyzing large data sets. In summary, users can enjoy the extensive data management capabilities provided by the database system (querying, recovery, security and concurrency control) and data set preprocessing, transformation and analysis are faster inside the database system.

There are several opportunities for future work. We would like to create automated SQL code generation for common data preparation tasks like summarizing, cross-tabulation and denormalization to create data sets. Constructive induction can help creating relevant attributes. We would like to automate the translation of existing data transformation programs in high-level programming languages into SQL, identifying useful primitive operations. We would like to create a tool that can combine external processing in a high level programming language with SQL running on a database system, benefiting the user with both external mathematical libraries and the speed and query capabilities provided by SQL.

Acknowledgments

This work was partially supported by National Science Foundation grants CCF 0937562 and IIS 0914861. This article summarizes experience and lessons learned while the author was working for Teradata Corporation, providing data mining consulting services to companies having large data warehouses. The author thanks Michael J. Rote from Teradata Corporation for his valuable guidance and support to migrate many data mining projects into the DBMS.

References

[1] E. Bloedorn and R.S. Michalski. Data-driven constructive induction. *IEEE Intelligent Systems*, 13(2):30–37, 1998.

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

[3] S. Chitroub, A. Houacine, and B. Sansal. A new PCA-based method for data compression and enhancement of multi-frequency polarimetric SAR imagery. *Intelligent Data Analysis*, 6(2):187–207, 2002.

[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 *Proc. VLDB Conference*, pages 998–1009, 2004.

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

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

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

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

[10] F. Liu, Z. Lu, and S. Lu. Mining association rules using clustering. *Intelligent Data Analysis*, 5(4):309–326, 2001.

[11] E.M. Nguifo and Njiwoua. P. IGLUE: A lattice-based constructive induction system. *Intelligent Data Analysis*, 5(1):73–91, 2001.

[12] C. Ordonez. Horizontal aggregations for building tabular data sets. In *Proc. ACM SIGMOD Data Mining and Knowledge Discovery Workshop*, pages 35–42, 2004.

[13] 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.

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

[15] C. Ordonez. Models for association rules based on clustering and correlation. *Intelligent Data Analysis*, 13(2):337–358, 2009.

[16] C. Ordonez. Statistical model computation with UDFs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(12):1752–1765, 2010.

[17] C. Ordonez and Z. Chen. Evaluating statistical tests on OLAP cubes to compare degree of disease. *IEEE Transactions on Information Technology in Biomedicine (TITB)*, 13(5):756–765, 2009.

[18] C. Ordonez, J. García-García, and M.J. Rote. Migration of data mining preprocessing into the DBMS. In *Proc. ACM KDD Data Mining Case Studies Workshop (DMCS)*, 2009.

[19] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.