Master's thesis

A Database Language for Data Mining

By
Mohssen ISSA

Supervised By
Mme. Fadila BENTAYEB & Mr. Jérôme DARMONT
Acknowledgments

I would like to express my sincere thanks to people who have helped me and supported me:

My supervisors, Jérôme DARMONT and Fadila BENTAYEB for all their advices, patience, enlightening comments and organizational support.

My thanks to Cécile FAVRE for proof-reading this report and for providing helpful suggestions for improving this manuscript.

Thanks to Professor Djamel ZIGHED for giving me the opportunity to follow this master.

I would like to thank the all members in KDD team.

My thanks to Professor Nicolas NICOLOYANNIS for the chance that he gave to me to work in Eric.

More thanks go to my evaluation team and the master’s thesis committee H. Briand, D. Zighed and Y. Kodratoff.
Contents

1 Introduction 4

2 Some Previous Languages for Data Mining 4
   2.1 MSQL: Query Language for Database Mining 6
   2.2 MINE RULE 6
   2.3 DMQL 7
   2.4 SQL Server 2000: Data Mining Extensions (DMX) 8
      2.4.1 Microsoft OLE DB for Data Mining Specification 9
      2.4.2 DMX Statements 9
   2.5 Oracle Data Miner 10

3 Evaluation and Comparison 12

4 Properties of a Data Mining Language, What we need? 13

5 The proposed model 15
   5.1 Data Mining Structure Model 15
      5.1.1 Pre-processing Model 15
      5.1.2 Data Mining Model 18
      5.1.3 Post-processing Model, Deploying Models 20
   5.2 Prototype, About Implementation 22
      5.2.1 Implementing DM_Table and pre-process functions: 22
      5.2.2 Data Mining Model 23
      5.2.3 Data Mining Parser 25

6 Conclusion 26
Abstract

The amount of collected data (stored in traditional database or data warehouses) increases rapidly, for this reason; the need of analyzing and extracting knowledge is taking an important role. And as we know the integration of knowledge discovery process with DBMS makes the extraction of information easier and faster; one of the key challenges is to enable integration of data mining technology within the framework of traditional database systems and data warehouses. In this domain different solutions have been proposed to integrate Data mining process with DBMS, one of these solutions is data mining query languages proposition, these languages propose to support both data and algorithm manipulations within DBMS.

In our paper we try to propose unifying framework for such systems. This proposal will allow users to write Data mining query to manipulate data training sets, perform data mining algorithms and process the results. The framework we are proposing in this paper is a new data mining query language, this language divide data mining process into three main models: pre-processing model to prepare data and put data in database table, which we will call Data Mining Table (an extension of traditional database table), data mining model to apply a specific data mining algorithm- decision trees, association rules, ... etc- on a data mining table and post-processing model to process and filter the result. To make this language comprehensible for DBMS we propose a parser, this parser translates the statement of this language into SQL and PL/SQL. The implementation of this parser can be done in any programming language able to connect to DBMS. Even we implemented the parser outside the DBMS but all the execution of data query takes place in the DBMS.

Résume

La quantité des données stockées dans des bases de données traditionnelles ou des entrepôts de données augmente rapidement. C’est pourquoi le besoin d’analyser et d’extraire les connaissances de ces données prend un rôle important. L’intégration du processus d’extraction de connaissances dans les systèmes de gestion de bases de données (SGBD) rend l’extraction d’informations plus facile et plus rapide. Un des défis dans ce domaine est l’intégration de technologies de fouille de données dans le cadre de SGBD. Différentes solutions ont été proposées pour cela, dont la proposition de langages de fouille de données. Ces langages supportent la manipulation de données et d’algorithmes dans les SGBD.

Dans notre mémoire, nous proposons un système pour unifier tous les processus de fouille de données. Cette proposition permet aux utilisateurs d’écrire des requêtes de fouille de données pour manipuler des données d’échantillons d’apprentissage, d’exécuter des algorithmes de fouille de données et de traiter les résultats. Le cadre que nous proposons dans ce mémoire est un nouveau langage pour les requêtes de fouille de données. Ce langage divise le processus en trois modèles principaux : une extension du concept de table des bases de données traditionnelles (Data Mining Table) pour le pré-traitement des données; un modèle de fouille de données (Data Mining Model) permettant d’appliquer un algorithme spécifique (arbres de décision, règles d’association, etc.) sur une Data Mining Table; et un modèle de post-traitement pour traiter et filtrer les résultats. Pour rendre ce langage interprétable par un SGBD, nous proposons de le traduire à l’aide d’un Parseur dans les langages SQL et PL/SQL. L’implémentation de ce Parseur peut être faite dans n’importe quel langage de programmation capable de communiquer avec un SGBD.
1 Introduction

The amount of collected and stored data by some organizations increases rapidly. However, this data will be meaningless unless we analyze it. Hence decision-support applications are taking their important place, and in response to the growing importance of these types of applications, relational Data Base Management Systems (DBMS) and SQL were quickly extended to support OLAP queries and data mining. Despite the potential effectiveness of data mining to significantly enhance data analysis, this technology is destined to be a niche technology unless an effort is made to integrate this technology with traditional database systems. This is because data analysis needs to be combined data integrity and management concerns at the data warehouse. Therefore, one of the key challenges is to enable integration of data mining technology seamlessly within the framework of traditional database systems. A lot of works have been done to integrate data mining process into DBMS or Data warehouse. One domain of these solutions tried to implementing data mining algorithm into DBMS, other proposed and implemented special language for data mining functions and data processing.

Some of data mining languages are made for specific DBMS like Oracle Data Miner for Oracle [6] and Data Mining Extensions for Microsoft SQL Server [23], other for specific Data mining algorithm like Mine Rule for association rules [19]. In our proposition we try to unify the entire Data mining algorithms and create a framework that can treat the data on many DBMS. This framework will allow users to prepare the data before they mine into it, and it will give some options to treat and filter the results of a data mining process.

Metadata (the nature of data, conceptual hierarchy information ... etc) helps in data mining process. In some languages like DMQL [10] there are some options to store this information, we will add this option to our proposition, and we will treat this information like any table content (Select, edit, delete ...).

To make our framework reliable we should implement the data mining algorithm inside DBMS. While the language can be implemented outside, and to translate the Data Mining language statement into SQL and PL/SQL we need to create a parser, this parser can be implemented outside DBMS but it shouldn’t make any data treatment outside DBMS. That means the whole data mining process (data preparation, mining process and result deployment) happens in DBMS.

In our work we aim to describe a language that helps data analysts in all data mining process (especially Decision Tree), data preparation (pre-processing), executing and selecting data mining methods and post-processing. We start by related works in the second section, then we analyze some previous languages and we compare them. We mention the properties that a data mining language must have. In the fifth section we start our proposition; it is divided into two parts: Data Mining Structure Model and a Prototype. The data mining structure model contains three stages (Pre-processing Model, Data mining Model and Post-processing model). In our implementation we give an idea about how we can implement this structure, and how to parse the Data mining language statements into SQL and PL/SQL.

2 Some Previous Languages for Data Mining

A significant amount of research has discussed the problem of overcoming the limitations of database systems in data mining tasks. Most research propose to support mining tasks by extending the database’s query languages [19]. The wide range of tasks to be supported and the lack of efficient implementation techniques proved serious challenges for these approaches, leading many to believe that general extensions of SQL for data mining might be
difficult to achieve. Therefore, in [22] the more basic question was investigated of whether it was possible for a team of experts to produce an efficient implementation of data mining process and algorithms on state-of-the-art DBMSs!

Extending DBMS, by creating a new query language or extending existing SQL, the database's query languages, in order to support data mining procedures was proposed from the early days of data mining. In [19] they proposed an SQL-Like language for mining using association rules technique. MSQL [11] and Mine Rule [22] are integrated in DBMS to extract association rules from database. Some other researches have proposed an extension for most of data mining techniques (including association rules) such as DMQL [10] (we will speak in detail about this language in the next section), and ODMQL [7] Object Data Mining Query, a Language for object-oriented databases as an extension to the Object Query Language (OQL), this work is proposed by the Object Data Management Group (ODMG) as a standard query language for object-oriented databases, ODMQL gives data miner the possibility to write data mining queries and specify the kinds of knowledge to be discovered using OQL-like syntax.

Several data mining APIs have been developed. The SQL/MM Part 6: Data Mining specifies an SQL interface to data mining applications and services. The Java Specification Request-73 (JSR-73) defines a pure Java API that supports building and using data mining models and access to and process data and metadata [14] in Oracle. The Microsoft's OLE DB for Data Mining is a data mining API for Microsoft-based applications, called Data Mining Extensions (DMX). DMX is a query language for Data Mining Models, much like SQL is a query language for relational databases.

The logic-based approach to the problem of data mining languages is presented in [24, 25]. This approach is realized in the context of deductive databases and rule-based languages. These languages (like Logical Data Language LDL++ [24] or Datalog [25]) provide an intelligent logic-based interface via the definition of meta-rules that specify general shape of searched patterns as second-order predicates. In [8], the concept of inductive databases fits naturally in the framework of rule-based languages and deductive databases; a deductive database can easily accommodate both extensional and intensional data.

Mining in spatial database has been treated in [18]; this article presents a language called SDMOQL (Spatial Data Mining Object Query Language) a spatial mining query language for INGENS (INGENS: Inductive Geographic Information System, is a prototype GIS which integrates data mining tools to assist users in their task of topographic map interpretation) sophisticated users. It is an OQL-based Data Mining Query Language for Map Interpretation Tasks, which permits the specification of the task-relevant data, the kind of knowledge to be mined, the background knowledge and the hierarchies, and the interestingness measures. Koperski designed GMQL (Geo Mining Query Language) [13], it is one of a few attempts in this domain, a language for the formulation of the input to data mining processes. The language uses a unified syntax, which allows a variety of data mining queries. GMQL is based on DMQL (Data Mining Query Language). SQL remains the milestone on which both data mining languages are built.

Some works have been done for data mining aggregation in "Aggregate & Table Language and System-ATLaS " [17], a powerful database language and system that enables users to develop complete data-intensive applications in SQL by writing new aggregates and table functions in SQL, rather than in procedural languages as in current Object-Relational systems. As a result, ATLAS SQL is Turing complete, and is very suitable for advanced data-intensive applications, such as data mining and stream queries [26].

In the following we are going to resume five of data mining languages. We selected this language because: MSQL, Mine rule and DMQL because they are already implemented. DMQL presents some technique for post treatment. DMQL presents a good philosophy
for a data mining language. And we chose DMX and Oracle data miner because these two operators are of the most important in the database market.

### 2.1 MSQL: Query Language for Database Mining

The goal of MSQL\(^1\) is to extract rules that are based on descriptors, each descriptor being an expression of the type \((A_i = a_{ij})\), where \(A_i\) is an attribute and \(a_{ij}\) is a value or a range of values in the domain of the attribute \(A_i\). In this language, the user defines a conjunctset as the conjunction of an arbitrary number of descriptors such that there is no couple of descriptors built on the same attribute. The form of association rules extracted by MSQL is like \(A \rightarrow B\), where \(A\) is a conjunctset (one descriptor or more) and \(B\) is a descriptor, so there is only one attribute in the consequent of each rule. Notice that MSQL defines the support of an association rule as the number of tuples containing \(A\) in the original table \((\text{support}(A) = \text{Count}(A))\) and its confidence as the ratio between the number of tuples containing \(A\), \(B\) and the support of the rule \((\text{Conf}(A, B) = \text{Count}(A, B) / \text{Support}(A))\) [11].

MSQL provides a few primitives for post-processing. Indeed, it is possible to use satisfy clauses to select rules which are supported (or not) in a given table. But there is no pre-processing function (we can use SQL statements to pre-process the data).

MSQL is considered as an extension of SQL. In order to perform association rule mining, some functions and procedures have been added to SQL (given their semantics of association rules). MSQL contains two types of queries: specific queries are used to mine rules (inductive queries starting with GetRules), and the other queries are post-processing queries over a materialized collection of rules (queries starting with SelectRules). The example shows the global syntax of the language for rule extraction is the following one:

\[
\text{GetRules}(C) \text{ INTO } [\langle \text{rulebasename} \rangle]
\]

\[
\text{WHERE } [\langle \text{ruleconstraints} \rangle]
\]

\[
\text{SQL-group-by clause ]}
\]

\[
\text{USING encoding-clause ]}
\]

\(C\) presents the data source (table or view) and rule constraints are conditions on the desired rules, e.g., the kind of descriptors which must appear in rule components, the minimal support or confidence of the rules or some mutual exclusion constraints on attributes which can appear in a rule. The USING part enables to discretize numerical values. rulebase name is the name of the object in which rules will be stored. Indeed, using MSQL, the analyst can explicitly materialize a collection of rules and then query it with the following generic statement where \(\langle \text{conditions} \rangle\) can specify constraints on the body, the head, the support or the confidence of the rule [12, 3] :

\[
\text{SelectRules (rulebase name )}
\]

\[
[\langle \text{whereconditions} \rangle]
\]

As we can see in this example, a user can do some post-processing on the extracted rules using SQL-Like clauses.

### 2.2 MINE RULE

We can define this language (Meo et al., 1998) as an extension of SQL coupled with a relational DBMS. This allows users to select and prepare data by using the full power of SQL. Mined association rules are materialized into relational tables as well. MINE RULE extracts association rules between values of attributes in a relational table [20]. However,

\(^1\)There is another language called mSQL (Mini SQL) is a lightweight client/server database from Hughes Technologies. Originally developed in 1994
it is a flexible language allows user to specify the form of the rules to be extracted. The user can specify the contents of body and head of the desired rules and the attributes on which rule components can be built [12]. MINE RULE gives the possibility to work on more than one level of grouping during the extraction process (in a similar way as the GROUP BY clause of SQL). If there is one level of grouping, rule support will be computed with respect to the number of groups in the table (using the group by clause as we will see). The definition of clusters allows a second level of grouping (sub-groups). Rules components can be taken in two different clusters, eventually ordered, inside a same group. It is thus possible to extract some elementary sequential patterns (by clustering on a time-related attribute). For instance, grouping purchases by customers and then clustering them by date, we can obtain rules like $(\text{Butter} \land \text{Milk}) \rightarrow \text{Oil}$ to say that customers who buy first Butter and Milk tend to buy Oil after [12]. Concerning interestingness measures, MINE RULE enables to specify minimal frequency and confidence thresholds. The general syntax of a MINE RULE query for extracting rules is:

```
MINE RULE (TableName) AS
    SELECT DISTINCT [(Cardinality)](Attributes)
    AS BODY,
    [(Cardinality)](Attributes)
    AS HEAD
    FROM (Table)
    WHERE (WhereClause)
    GROUP BY (Attributes)
    HAVING (HavingClause)
    [ CLUSTER BY (Attributes)
      HAVING (HavingClause)]
EXTRACTING RULES WITH
    SUPPORT: (real), CONFIDENCE: (real).
```

### 2.3 DMQL

DMQL (Han et al., 1996) has been designed in DBMiner project for mining several kinds of knowledge in relational databases. The philosophy of data mining may strongly influence the design of a data mining language. The following philosophical considerations will serve as guidelines in the design of this data mining language:

- The set of data relevant to a data mining task should be specified in a data mining request.
- The kinds of knowledge to be discovered should be specified in a data mining request.
- Background knowledge could be generally available for data mining process.
- Data mining results should be able to be expressed in terms of generalized or multiple-level concepts.
- Various kinds of thresholds should be able to be specified flexibly to filter out less interesting knowledge.

DMQL adopts an SQL-like syntax to facilitate high level data mining and natural integration with relational query language, SQL. The DMQL language is defined in an extended BNF grammar \[^2\] , where "[ ]" represents 0 or one occurrence, "{ }" represents 0 or more

[^2]: Backus-Naur form: is a metasyn tax used to express context-free grammars: that is, a formal way to describe formal languages.
occurrences, and words in sans serif font represent keywords, as shown below.

\[
\langle \text{DMQL} \rangle ::= \\
\quad \text{use database} \langle \text{databasename} \rangle \\
\quad \{ \text{use hierarchy} \langle \text{hierarchyname} \rangle \text{ for} \langle \text{attribute} \rangle \} \\
\quad \langle \text{rulespec} \rangle \\
\quad \text{related to} \langle \text{attroragglist} \rangle \\
\quad \text{from} \langle \text{relation}(s) \rangle \\
\quad \{ \text{where} \langle \text{condition} \rangle \} \\
\quad \text{order by} \langle \text{orderlist} \rangle \\
\quad \{ \text{with} \langle \text{kindsof} \rangle \text{ threshold} = \langle \text{thresholdvalue} \rangle \} \\
\quad \{ \text{for} \langle \text{attribute}(s) \rangle \} \\
\]

In \(\langle \text{DMQL} \rangle\), "use database \langle \text{databasename} \rangle" directs the mining task to a specific database "\langle \text{databasename} \rangle", and the optional statement, "use hierarchy \langle \text{hierarchy} \rangle \text{ for} \langle \text{attribute} \rangle", assigns \langle \text{hierarchy} \rangle \text{ to} a particular attribute \langle \text{attribute} \rangle \text{ (otherwise, a default hierarchy is used). The statement, \langle \text{rulespec} \rangle\text{, is the specification of the kind of rules to be discovered (Association rule, Classification rule, Discriminant rule, Characteristic rule, Generalized relation). For discovering different kinds of rules, the rule specification should be in different formats. The related-to statement, \langle \text{related to} \langle \text{attroragglist} \rangle \rangle\text{, selects a list of relevant attributes and/or aggregations for generalization. The \"from\" and \"where\" clauses, \"from} \langle \text{relation}(s) \rangle \text{ [where} \langle \text{condition} \rangle \rangle\text{, form an SQL query to collect the set of relevant data. The \"order by\" clause simply specifies the order of rows to be printed. The \"with-threshold\" statement specifies various kinds of thresholds.}"

\[\text{2.4 SQL Server 2000: Data Mining Extensions (DMX)}\]

\[\text{OLE DB}\text{ DM has been designed at Microsoft Corporation. It is an extension of the OLE DB Application Programming Interface (API) that allows any application to easily access a relational data source under the Windows family OS. The main motivation of the design of OLE DB for DM is to ease the development of data mining projects with applications that are not stand-alone but are tightly-coupled with the DBMS} [1]\text{. Indeed, research work in data mining focused on scaling analysis and algorithms running outside the DBMS on data exported from the databases in files. This situation generates problems in the deployment of the data mining models produced because the data management and maintenance of the model occurs outside of the DBMS and must be solved by ad-hoc solutions. OLE DB DM aims at ease the burden of making data sources communicate with data mining algorithms (also called mining model provider) [10].}\\
\]

\[\text{Microsoft Data miner supports both predictive and descriptive mining functions. Predictive functions, and there is an implementation in MS SQL server for the common Descriptive and Predictive functions (Decision tree, Association rules, Naive Bayes... etc).}\\
\]

\[\text{2.4.1 Microsoft OLE DB for Data Mining Specification}\]

\[\text{The data mining features in Analysis Services are built to comply with the Microsoft OLE DB for Data Mining specification, which was first released to coincide with the release of Microsoft SQL Server 2000.}\\
\]

\[\text{The Microsoft OLE DB for Data Mining specification defines the following:}\\
\]

---

3Object Linking and Embedding Database, sometimes written as OLEDB or OLE-DB: is an API designed by Microsoft for accessing different types of data stores in a uniform manner. DM stands for Data Mining.
• A structure to hold the information that defines a data mining model.
• A language for creating and working with data mining models.

The specification defines the basis of data mining as the data mining model virtual object. The data mining model object encapsulates all that is known about a particular mining model. The data mining model object is structured like an SQL table, with columns, data types, and meta information that describe the model. This structure lets user use the DMX language, which is an extension of SQL, to create and work with models.

2.4.2 DMX Statements

The user can use DMX statements to create, process, delete, copy, browse, and predict against data mining models. There are two types of statements in DMX: data definition statements and data manipulation statements. the user can use each type of statement to perform different kinds of tasks.

DMX statements consist of:
• Data Definition Statements
• Data Manipulation Statements
• Query Fundamentals

The use data definition statements in DMX helps create and define new mining structures and models, to import and export mining models and mining structures, and to drop existing models from a database. Data definition statements in DMX are part of the data definition language (DDL).

the user can perform the following tasks with the data definition statements in DMX:

• Create a mining structure by using the CREATE MINING STRUCTURE statement, to add a mining model to the mining structure we use the ALTER MINING STRUCTURE statement.

• Create a mining model and associated mining structure simultaneously by using the CREATE MINING MODEL statement to build an empty data mining model object.

• Export a mining model and associated mining structure to a file by using the EXPORT statement. Import a mining model and associated mining structure from a file that is created by the EXPORT statement by using the IMPORT statement.

• Copy the structure of an existing mining model into a new model, and train it with the same data, by using the SELECT INTO statement.

• Completely remove a mining model from a database by using the DROP MINING MODEL statement. Completely remove a mining structure and all its associated mining models from the database by using the DROP MINING STRUCTURE statement.

Use data manipulation statements in DMX to work with existing mining models, to browse the models and to create predictions against them. Data manipulation statements in DMX are part of the data manipulation language (DML). the user can perform the following tasks with the data manipulation statements in DMX:

• Train a mining model by using the INSERT INTO statement. This does not insert the actual source data into a data mining model object, but instead creates an abstraction that describes the mining model that the algorithm creates.
• Extend the SELECT statement to browse the information that is calculated during model training and stored in the data mining model, such as statistics of the source data. Following are the clauses that the user can include to extend the power of the SELECT statement:

\[
\begin{align*}
&\text{SELECT DISTINCT FROM} \ <\text{model}\ >\ (\text{DMX}) \ \text{SELECT FROM} \ <\text{model}.\ \text{CONTENT} \ (\text{DMX}) \\
&\text{SELECT FROM} \ <\text{model}.\ \text{CASES} \ (\text{DMX}) \\
&\text{SELECT FROM} \ <\text{model}.\ \text{SAMPLE CASES} \ (\text{DMX}) \\
&\text{SELECT FROM} \ <\text{model}.\ \text{DIMENSION CONTENT} \ (\text{DMX})
\end{align*}
\]

• Create predictions that are based on an existing mining model by using the PREDICTION JOIN clause of the SELECT statement.

• Remove all the trained data from a model or a structure by using the DELETE (DMX) statement.

The SELECT statement is the basis for most DMX queries. Depending on the clauses that the user use with such statements, can browse, copy, or predict against mining models. The prediction query uses a form of SELECT to create predictions based on existing mining models. Functions extend your ability to browse and query the mining models beyond the intrinsic capabilities of the data mining model.

2.5 Oracle Data Miner

Oracle Data Mining (ODM) embeds data mining within the Oracle database. ODM algorithms operate natively on relational tables or views, thus eliminating the need to extract and transfer data into standalone tools or specialized analytic servers. ODM's integrated architecture results in a simpler, more reliable, and more efficient data management and analysis environment. Data mining tasks can run asynchronously and independently of any specific user interface as part of standard database processing pipelines and applications. Data analysts can mine the data in the database, build models and methodologies, and then turn those results and methodologies into full-fledged application components ready to be deployed in production environments. The benefits of the integration with the database cannot be emphasized enough when it comes to deploying models and scoring data in a production environment. ODM allows a user to take advantage of all aspects of Oracle’s technology stack as part of an application.

ODM supports both predictive and descriptive mining functions. Predictive functions, known as supervised learning, use training data to predict a target value. Descriptive functions, known as unsupervised learning, identify relationships intrinsic to the data. Each mining function identifies a class of problems to be solved, and each can be implemented with one or more algorithms.

In any case, the process of developing a data mining application is iterative (not only in ODM). It involves testing the model, evaluating test metrics, making adjustments in the model, and re-evaluating.

Although it is common to try different approaches to solve a data mining problem, each application must accomplish several basic tasks.

1. Prepare the data. One data set is needed for building the model; additional data sets may be necessary for testing and scoring the model, depending on the algorithm. In most cases, the data must be prepared with transformations that enhance or facilitate the effectiveness of the model. Each data set must be prepared in the same way.

2. Create a model using the build data.
3. Evaluate the model.

- For classification and regression models, this is the application of the model to a set of test data, and the computation of various test metrics.
- For clustering models, this is the examination of the clusters identified during model creation.
- For feature extraction models, this is the examination of the features identified during model creation.
- For attribute importance and association models, evaluation is the final step in the mining process. These models cannot be scored against new data.

4. Apply the model. This is the process of deploying the model to the data of interest.

- For classification and regression models, scoring is the application of the "trained" model to the actual population. The result is the best prediction for a target value in each record.
- For clustering models, scoring is the application of clusters identified by the model to the actual population. The result is the probability of cluster membership for each record. For feature extraction models, scoring is the mapping of features defined by the model to the actual population. The result is a reduced set of predictors in each record.

ODM integrates data mining with the Oracle database and exposes data mining through the following interfaces:

- Java interface: Java Data Mining (JSR-73) compliant interface that allows users to embed data mining in Java applications.
- PL/SQL interface: The packages DBMS_DATA_MINING and DBMS_DATA_MINING_TRANSFORM allow users to embed data mining in PL/SQL applications.
- Automated data mining: The DBMS_PREDICTIVE_ANALYTICS PL/SQL package, automates the entire data mining process from data preprocessing through model building to scoring data.
- Data mining SQL functions: The SQL Data Mining functions (CLUSTER_ID, CLUSTER_PROBABILITY, CLUSTER_SET, FEATURE_ID, FEATURE_SET, FEATURE_VALUE, PREDICTION, PREDICTION_COST, PREDICTION_DETAILS, PREDICTION_PROBABILITY, and PREDICTION_SET) support deployment of models within the context of existing applications, improve scoring performance, and enable pipelining of results involving data mining predictions.
- Graphical interfaces: Oracle Data Miner and Oracle Spreadsheet Add-In for Predictive Analytics are graphical interfaces that solve data mining problems.

The end result of data mining is a model. Often this model is deployed so that its results can be embedded in an application. ODM Data Table Format: ODM data must reside in a single table or view in an Oracle database. The table or view must be a standard relational table, where each case is represented by one row in the table, with each attribute represented by a column in the table. The columns must be of one of the types supported by ODM.

**Column Data Types Supported by ODM**: ODM does not support all the data types that Oracle supports. Each attribute (column) in a data set used by ODM must have one of the following data types:
• INTEGER
• NUMBER
• FLOAT
• VARCHAR2
• CHAR
• DM_NESTED_NUMERICALS (nested column)
• DM_NESTED_CATEGORICALS (nested column)

The supported attribute data types have a default attribute type (categorical or numerical).

Nested table columns can be used for capturing in a single table or view data that is distributed over many tables (for example, a star schema). Nested columns allow to capture one-to-many relationships

3 Evaluation and Comparison

The following table 1 shows a comparison between the previous languages in the some domains:

<table>
<thead>
<tr>
<th></th>
<th>MSQL</th>
<th>MR</th>
<th>DMQL</th>
<th>DMX</th>
<th>ODM</th>
</tr>
</thead>
<tbody>
<tr>
<td>The selection of data to be mined</td>
<td>SQL</td>
<td>SQL</td>
<td>SQL</td>
<td>Rich</td>
<td>Rich</td>
</tr>
<tr>
<td>The specification of the type of patterns to be mined</td>
<td>AR</td>
<td>AR</td>
<td>Many DM patterns</td>
<td>Many DM patterns</td>
<td>Many DM patterns</td>
</tr>
<tr>
<td>Meta data</td>
<td>No</td>
<td>No</td>
<td>Yes/hierarchy</td>
<td>Yes/contenntype</td>
<td>No</td>
</tr>
<tr>
<td>The definition of constraints of the extracted pattern</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>The post-processing of results</td>
<td>Rich</td>
<td>Poor</td>
<td>Poor</td>
<td>Rich</td>
<td>Rich</td>
</tr>
<tr>
<td>Using API</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 1: A comparison between some data mining languages

The advantages of the proposed languages are that they are all designed as extensions of SQL. It facilitates the work for database experts and it is useful for data manipulation (or the needed standard queries) [4]. They all satisfy the closure property. Indeed, even if all the languages do not systematically provide operators for manipulating extracted rules, it is always possible to access materialized collections of rules using SQL queries. Notice, however, that most of the needed pre-processing or post-processing techniques will need not only SQL queries but also PL/SQL statements. Some languages provide primitives to
simplify some typical preprocessing, e.g., the discretization of numerical values. Even it is quite preliminary, it is an important support for the practical use of the association rule mining technique. Finally, the concept of OLE DB for DM is quite relevant as it enables external providers to plug-in new solvers to the existing systems [12].

Another advantage is that we can build user friendly interface (GUI) over most of the previous languages. As in DMX, and Oracle, both of them released user friendly software. Although Oracle has provided users with a package called "od_miner.jar" written in java, this package helps user to perform most common data mining process, on oracle database tables and on external database files.

The most critical limitation of the proposed languages is: pre- and post-processing operations are not treated very well. Indeed, they are essentially designed around the extraction step and mainly provide primitives for rule extractions, these primitives being generally fixed, e.g., the possibilities to specify minimal thresholds for a few selected objective measures of interestingness or to define syntactical constraints on the rules. Only MSQL, OLE DB for DM and ODM propose restricted mechanisms for discretization. Typical preprocessing techniques for, e.g., sampling or boosting, are not supported. The lack of primitives for post-processing is also obvious. Only MSQL provides a SelectRules operator which enables to query rule databases and primitives for crossing-over operations between rules and data. Meanwhile ODM propose a method to write mining results as XML or PMML format. The others rely on SQL and its programming extensions for accessing and manipulating the rules. Not only the SQL post-processing queries are hard to write but also difficult to optimize given the current state of the art for SQL optimization [12, 3].

4 Properties of a Data Mining Language, What we need?

The philosophy of data mining may strongly influence the design of a data mining language. We have the following philosophical considerations which will serve as guidelines in our design of a data mining language [10, 7]:

1. The set of data relevant to a data mining task should be specified in a data mining request. Since a user may be interested in any portion of data in a database, a data miner should be able to work on any specific subset of data. This implies that a data miner may take a database query as a subtask to first retrieve the relevant set of data before data mining. If a user cannot identify precisely the relevant set of data, a superset of the data can be collected, and certain mechanisms can be developed to help identify or rank the relevant set of data and/or attributes. Therefore, a data mining language should take a query language as its subtask in the specification.

2. Background knowledge (metadata) could be generally available for data mining process.

3. The kinds of knowledge to be discovered specified in a data mining request. Clearly, real-life KDD processes need for different kinds of patterns like various types of descriptive rules, clusters or predictive models.

4. The definition of constraints that the extracted patterns must satisfy. This implies that the language allows the user to define constraints that specify the interesting patterns (e.g., using measures like frequency, generality, coverage, similarity, etc).

5. The post-processing of results. The language must allow to browse the patterns, apply selection templates, cross over patterns and data, e.g., by selecting the data in which some patterns hold, or aggregating results.
5 The proposed model

A data mining framework is divided into pre-processing of data, data mining procedure and post-processing of the data. Thus it can support an integrated and efficient view to the process for the user [9]. This framework has to provide a unifying description of models and patterns.

5.1 Data Mining Structure Model

In our proposal, we are going to respect the flow in figure 1, and we will try to cover the properties of data mining language defined in section 4.

5.1.1 Pre-processing Model

For pre-processing, we divide it into two parts: data mining table in which we will store the dataset (learning set), and the second is pre-processing functions.

Data mining table likes classic table, in its body we defined fields' source (source table(s)), also we defined a flag column called "sample_id", in which the pull id is going to store, to allow users to use sampling methods in learning process.

To describe this language we will use an extended BNF grammar - the same notation is used in DMQL specification-, where "[]" represents 0 or one occurrence, "{}" represents 0 or more occurrences, and words in bold represent keywords.

The following syntax is used to define a data mining table:

```
Create DM_Table ( Data mining table name )
{
    ([Columns list]);
    ([Hierarchy definitions])
}
```

Column list defined by:

```
Columns list ::= [select {column name}(data type)(content type)]
from (table names)
where (condition)];[[([column name](data type)(content type)])]
```

Column name: is a unique name in a dm_table.
**Data type:** user can define the data types for the mining table columns on which the model is based. These types (Integer, Text, Varchar, ...etc) must be supported by data mining algorithms.

**Data contents:** this to allow users to define the data types for columns in a mining table, to affect how algorithms process the data in those columns during the data mining algorithm process. However, defining the data contents of columns gives algorithms information about the type of data in the columns [23]. Each data mining data type supports one or more content types, which the user can use to describe the behavior of the content that the columns contain. In the following we list must common content types:

- **DISCRETE:** The column contains discrete values. The values in a discrete attribute column do not imply ordered data, even if the values are numeric; the values are clearly separated, and there is no possibility of fractional values. Telephone area codes are a good example of discrete data that is numeric.

- **ENUMERATION:** The column contains enumeration values, the values can be number, chars or strings. But they are like DISCRETE type, the values are clearly separated, the only reason we propose the type is add meaning to discrete values and to define them before any use takes place. The definition of ENUMERATION will be in column list, after the word enumeration we add the values and their description like the following:

  ```
  (column name)(data type) enumeration ([(value, description)])
  ```

- **CONTINUOUS:** The column contains values that represent a continuous set of numeric data. Unlike a discrete column, which represents finite, counted data, a continuous column represents measurement data, and it is possible for the data to contain an infinite number of fractional values. An income column is an example of a continuous attribute column.

- **KEY:** The column uniquely identifies a row (primary key/unique).

**Select statement:** is the same select in SQL, so user can drive profit from the powerful of SQL in all ways. Except that user defines a data type (optionally, if not data type will be the same in the original table) and content type. Defining a data type different from the original in source table forces type casting to run (change type).

For background knowledge. Hierarchy (or lattice) concept is a very useful concept which provides background knowledge for expressing data mining results in brief [10]. Concept hierarchies can be specified based on database attribute relationships, on customized grouping operation, etc. However, here we will treat only the hierarchy that is defined from attribute relations. As the following:

```hierarchy
definitions ::= (Hierarchy (Hierarchy name) on (attribute name) as (attribute1, attribute2...));
```

Example:
Create DM_Table dmTable_demo
(
  Id autoincrement integer KEY,
  Sample_id integer Discrete,
  (  
    Select date continuous, field1 continuous, field2 discrete, ...
    From Table1
    Where some condition
  )
) (Hierarchy hierar_Date on date as (day, month, year); )

Of course, there must be some specification to delete and modify data mining table and its components, for these options we use the same SQL statements:

- To delete records from data mining table: `Delete from <DM_Table> where <Condition>;`
- To delete hierarchy concept from data mining table: `Delete hierarchy from <DM_Table> where <Condition>;`
- To insert some tuples in a data mining table we can use the same SQL insert statement. And we extend it include a hierarchy insertion statement: `Insert [hierarchy] into <DM_Table> values (...);`
- To modify a unit, update statement can be used: `Update <DM_Table> set column name=new value, where <condition>;` and to update a hierarchy concept: `Update <DM_Table> set hierarchy <hierarchyname> value (<hierarchyvalue>) where <condition>;`
- To delete a data mining table permanently: `DropDM_table;`

A new SQL statement should be added about changing content type, this will be useful especially when a user needs to change content type from continuous to discrete.

- To change content type: `Update content type <> set <columnname> = <newcontenttype>`

The second proposition for data mining table is some associated useful functions, to facilitate data set treatment. Users can use those functions to perform frequent and necessary operations.

- `Rest <DM_Table>`: if a data mining table contains SQL statements to select data from some tables, the rest function empties the data mining table and executes SQL statement then inserts the data into the data mining table. Else rest function just empties the table. If user uses "sample_id" column then on rest all values set to 0. Briefly, this function returns a data mining table to its default status.
- `Show missing <DM_Table>`: show a statistic resume about missing values for each column.

Data Mining Table allows user to use the complete power of SQL data selection statements, although it gives the options of using SQL clauses to treat the training set (delete, insert, update,...), another advantage is now we have all information of training data and metadata in one object, so we can edit these data easily. The column of "sample_id" is
used by sampling algorithm to mark the number of sample; the sampling algorithm can be called before or during the mining process, in any case it will be good issue if we create an index on sample_id in order to facilitate sample extraction from a big database.

Finally data mining table covers the first and second points in "Properties of a Data Mining Language". The select statement covers first point, and hierarchy and data contents cover the second point.

5.1.2 Data Mining Model

We try to support both predictive and descriptive mining functions. Predictive functions, known as supervised learning, use training data to predict a target value. Descriptive functions, known as unsupervised learning, identify relationships intrinsic to the data. Each mining function identifies a class of problems to be solved, and each can be implemented with one or more algorithms, we will suppose that all algorithms are implemented in DBMS or we can call them from DBMS (see [2] for decision tree implementation).

First we deal with predictive mining functions, using decision tree as template. This type of data mining function needs a training set (table, view, data file or a data mining table ...) as input. In our article it will always be the proposed data mining table, also a predictive attribute must be determined.

```plaintext
Create Predictive DM_Model < DM_Modelname > as
Apply Decision_Tree on < DM_Table(columnslist) >
{
    Predict < column >,
    (< Trainingsetparameters >),
    (< algorithmParameters >),
    (< OutputOptions >)
}
```

Create DM_Model builds an object(set of tables and procedures) in DBMS to save the model metadata in it, and the predictive model (the result model). Columns list contains some or all columns name from a DM_Table including the column against which we want to predict, if we mentioned the DM_Table name in the DM_Model only, that means the algorithm must to use all columns in DM_Table:

```plaintext
Column list ::==(column1, column2,...)
```

After columns list, we have to mention the attribute against which we want to predict, the content type attribute must be compatible with the data mining algorithm (most implementation doesn’t predict against continuous attribute).

```plaintext
<Training set parameters> ::= (  
    [size=<integer number>,],
    [start=<number>],
    [end=<number>],
    [selection=random|first|last],
    [mark_sample=true|false],
    [number_of_iteration=number],
    [sample_id={number}],
    [criteria=conditions])
```

In Training set parameters, size presents the number of examples used by data mining function by default size equal to DM_Table size. Start and End to determine the first and
the last record number (we supposed that the data are arranged by date of insertion). The
default value is zero for start and record count for end. Of course if we have a column in
the training set contains information about row id (a primary key or a unique attribute) we
can omit this tow option and use the row id in the criteria, this is better especially when
there is indexing on the row id.

Selection specifies the way to select examples; here we use sample_id that we mentioned
in the previous section to mark the selected examples. Hence we can perform "Select with
and without replacement" sampling. We can use sampling in two ways, first we can run a
sampling method on the training set, this method gives each selected row an id then we can
use the id in the select criteria or in sample_id clause or we can select the examples using
the selection clause this way allows to mark the selected examples (if we set mark_sample
to true-Select with and without replacement- or set it to false to perform select with replace-
ment). To avoid any incompatible condition it is recommended not to use size and selection,
if we have already performed a sampling method on the training set. In another word we
cannot use size and selection clause when we use sample_id clause, because the examples
have been divided into training sets.

If we want to use posting and bagging, we have to set the number of iteration (by
default it is 1), this option gives the ability to create multi learning model and store them in
the result tables, it is better to have a separate table for each iteration, so we can generate
more than one predict model and we can apply a voting method to decide which is more
convenient [16].

Algorithm parameters present the definitions of constrains, thresholds and other data
mining function parameters, for example in the implementation of ID3 in [2], we need to
define the minimum gain for node building as threshold, and an output table name, class
attribute, ...etc, of course a good implementation is the one that define a default value for
each parameter (if possible).

<Algorithm parameters>::= ([<parameters name>=<value>]);
Ex: (gain=0,...)

Output Options present how to deal with the result, indeed, it depends on the data
mining functions implementation, therefore we suppose that all implementations write the
result in a specific table format. It is enough to give the function a table name then it will
build a table using the given name and a known structure for each type of function, for
example, in [2] and [13] the functions write the result directly into a table which facilitate
the post-processing (filtering, selection, exporting the rules into another format). When the
results are written in a table, we have the ability to use the power of SQL in output editing
and management. So the question now is to find a convenient structure for each type of data
mining function, in this proposition, we will use the result table in [2] to store the result of
decision tree table (see table 2):

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>The depth of the current node (0 for root)</td>
</tr>
<tr>
<td>Node ID</td>
<td>The node ID number</td>
</tr>
<tr>
<td>Parent</td>
<td>The ID number of the node’s parent</td>
</tr>
<tr>
<td>Rule</td>
<td>The rule that lead to the creation of this node</td>
</tr>
<tr>
<td>Frequency</td>
<td>The frequency for the considered value of the attribute in this node</td>
</tr>
</tbody>
</table>

Table 2: Decision Tree result’s table
The second case is descriptive mining functions. We will use association rules as template. The same model of predictive functions can work on descriptive function all we need is to remove the predictive clause.

Create Descriptive DM_Model <DM_Model name> as
Apply Association_Rules on <DM_Table (columns list)>

Training Set Parameters, Algorithm Parameters and Output Options have the same purpose in the predictive model. Except the algorithm parameters are different, for example for association rules we need to determine the Minimum Probability, Minimum Support...etc. although the result table structure is different for each type of data mining algorithm. We can use table structure in like the one in [13]:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Body</td>
<td>The variables in the Body on current rule</td>
</tr>
<tr>
<td>Head</td>
<td>The variables in the Head on current rule</td>
</tr>
<tr>
<td>Support</td>
<td>the Support for current rule</td>
</tr>
<tr>
<td>Confidence</td>
<td>the Confidence for current rule</td>
</tr>
</tbody>
</table>

Table 3: Association Result's Table

As we are using the same options here, we have the same technique (sampling, posting ...).

5.1.3 Post-processing Model, Deploying Models

Now the results of data mining functions are stored as raw data in some tables. The last step in the data mining process is to write the result in comprehensible format, to make the result useful, and finally to deploy a data mining model production. Clearly, the deployment depends on the mining type (predictive or descriptive), so for each type we have to create a specific procedure to deploy (In our case we will focus on decision trees as example). Apply (score) the model. This is the process of deploying the model to the data of interest.

- For classification and regression models, create a prediction model (application). The result is the best prediction for a target value in each record.
- For clustering models, scoring is the application of clusters identified by the model to the actual population. The result is the probability of cluster membership for each record.
- For feature extraction models, scoring is the mapping of features defined by the model to the actual population. The result is a reduced set of predictors in each record.

In [5] they publish a poll (Data Mining Deployment / Sep 2006/: "How do you usually deploy data mining results?") (See figure 2)
The count of "Deploy in production" probably should be higher, since some of the answers included use of data mining tool for scoring and other scoring methods, but did not include the "Deploy in production choice". However, we did not modify the reported results. Of those who deployed in batch mode, the most popular deployment methods were:

- Use data mining tool for scoring (60%)
- Convert model to SQL (29%)
- Convert model to C or Java (19%)
- Convert model to another language (19%)

Among those who deployed in real-time mode, about 60% used data mining tool for scoring. However, 86% also deployed in batch mode, so it is hard to separate real-time deployment from batch deployment with this poll.

For **Descriptive Functions**, the deployment will be only results publishing. As the results in a table we can perform SQL statements to filter and export them. For example in association rules we want to see/export the rules that have support more than x and confidence more than y, we can filter the results using the classic SQL statements. The descriptive data mining result should be interpreted and reported by data mining analyzer.

For **Predictive Functions**, now we have the result in a table and we need to create a predictive function using this data, in decision tree we can convert the result into if-else format or any other predictive format and we implement it into a stored procedure associated with the data mining model. To keep the SQL-like format we propose the following model:

```
Select dm_Model.Predict ([input data], [options])
From dm_Model
```

The input data can be a raw data or a selected row from a table, then the result will display the predicted value and all the other output of this function. In ODM [6] they propose some other functions to predict:

- **PREDICTION** Returns the best prediction for the target.
• **PREDICTION_COST** Returns a measure of the cost of false negatives and false positives on the predicted target.

• **PREDICTIONDETAILS** Returns an XML string containing details that help explain the scored row.

• **PREDICTIONPROBABILITY** Returns the probability of a given prediction.

• **PREDICTIONSET** Returns a list of objects containing all classes in a binary or multi-class classification model along with the associated probability (and cost, if applicable).

These functions provide users with most function and information they may need but the use is somehow complex, so in our proposition we will try to simplify these functions and include the essential information. We can implement the predict function in more than one format to give the "Cost" and all the details about predicted data.

If we use iteration as option in data mining model, we would get a result table for each iteration, so we need to create one specific function to predict. As we have more than one model to predict we suggest a voting strategy, we call this function PREDICTION_VOTE.

The work on deployment could be a huge task if we want to include every thing about prediction functions like probability, the measurement of errors, cost ...etc. We can create a specific stored procedure (embedded procedure) for each task.

5.2 Prototype, About Implementation

Each part of the proposition can be implemented alone. DM_Table, preprocessing functions and result tables must be implemented in DBMS, although we prefer to save all information about any object in this language in tables (in DBMS), and create a platform with a parser to retrieve these information [21], while the other objects could be implemented outside or inside DBMS, and call DM_Tables and all necessary objects via API interface.

5.2.1 Implementing DM_Table and pre-process functions:

Data mining tables can support only data types that are supported by data mining algorithm and DBMS. As we use DM_Table to store data we can use the DBMS table for the same purpose. In there are some specification for Oracle Database system tables, they explain how Oracle store information in system tables. We will use the same concept to store our DM_Table and its information. But we have to extend the classical system table (or create new tables) in order to store all meta data (table information, columns type) to give the option to store data content type (*DISCRETE, CONTINUOUS, KEY*). For hierarchy we need to save the hierarchy information and edit saved data. This task is not a heavy one because in the most database system (like oracle [15]), they store table's information in a relation tables. So what we need is to add a new column to store content type only, or we can create a similar table with the new column. And make this table a like system table.

figure 3 shows how we can keep DM_Table metadata in the DBMS using tables.

After we created DM_Table we can write the functions we proposed earlier like: rest and show missing in PL/SQL. If the DBMS doesn't support PL/SQL, we write a sequence of SQL statements and execute them instead of calling a PL/SQL procedure.

In this model we can store all metadata of a data mining table, of course the SQL statements are applicable on these tables and on the data table. This facilitates the treatment. When we write "Create DM_Table" statement, the information in this statement
must translated into SQL statement, and the metadata information saved into data mining tables' information, this work done through data mining parser (we will speak about the parser later in this section). Clearly for each object, we prefer to store the code than rebuild it from the information we have; it is time issue.

5.2.2 Data Mining Model

This part is the functional part of data mining model, meanwhile the DM_Table part is physical. Here we need to know how to use DM_Table and how to achieve the results. Figure 4 shows how to store the information and options of data mining model. Most entities are easy to understand, so we will speak only about the data mining result entity. It contains references to tables that contain the raw results, we may have more than table if we used iteration for posting and bagging in our model. Of course there will be a result table for each sample. Also, according to data mining model's entity, we can see that we can apply more than one data mining model on a single data mining table.

After we created a model and executed it, we have to make a predict function for each result table, as we said earlier "for each iteration there will be a result table, and for each result table we need to create a data mining function".

Creating Data Mining Functions: In this part we discuss how to translate a data mining model results into predict function, decision tree as example, basically the predict function is statements of "If... Then ... Else ". If we consider the proposed output format in 5.1.2, to convert this results to predict function is the same process of printing a decision tree, all we need is to add the condition statement in each level. The next example shows a prediction according to weather to tell if we can play (football) or not:
As we said to convert this tree to a stored procedure in PL/SQL, it is enough to add the condition statement, so after adding we will have:

```sql
if outlook = sunny then
    if humidity = normal then return yes
    else if humidity = high then return no
else
    if outlook = overcast then return yes
else
    if outlook = rainy then
        if windy = TRUE then return no
        else if windy = FALSE then return yes
```
in order to have the necessary information.

For most of predictive methods it is almost the same, on the other hand, for descriptive there is no deployment, no need to create a predictive function, what we have to do here is filter the results and publish them, or create a report that contains the results.

5.2.3 Data Mining Parser

The goal of data mining parser is to convert Data mining language statements into SQL and PL/SQL statements and execute them in DBMS, and insert/update the information of DM_Table and DM_Model in their appropriate system tables. The good point in this parser is that we can implement it outside DBMS, therefore it can connect to different DBMS (Oracle, SQL server,...). Clearly for each DBMS there will be a specific parser, especially when we work with PL/SQL.

![Data Mining Parser schema](image)

Figure 5: Data Mining Parser schema

As we can see from figure 6, the parser has two main functions: First fill the system tables with appropriate information, and then generate the SQL statements using a specific translation for each statement in Data mining language SQL. This translation depends on the DBMS system. After the parser has created the appropriate statements, it can ask the DBMS to execute them. For more information about parsing pre-processing model see Appendix 1.

Because of the similarity between Data mining language and SQL, parsing of Create, Update and Delete statement will be easy. In Figure 6 we have a Create DM_Table statement, the parser will rewrite it in Create Table statement using the convenient SQL, also it will generate the DM system statement to store the table information and the meta information (if there is any). If we use in a `select` clause while we are creating a DM_Table, the parser must searching for the original type of each column in the source table.

The implementation of Posting and Baging must be done in PL/SQL procedure, in which we call the data mining function n times, where n is the number of iteration. And store the results for each call in the result's tables, and fill the result information in results system tables.

One of the problems we might have is where and how to implement Data mining algorithms. If we have the algorithm outside DBMS, then the parser should transfer the data to the algorithm, which may take a lot of time and resource, therefore it is better to implement them in DBMS. But if the DBMS doesn’t support PL/SQL then the implementation of predict functions and Data mining algorithms must be out side DBMS, which is not good.

Another problem is the duplication of information, in DBMS system tables and in tables for data mining object, this problem has been created because we add some system tables for our language without removing the system tables (impossible to remove system table). The solution for this problem is quite easy, we have to prevent users from editing these
tables inside DBMS, the changes of any type of information must be done via data mining language.

The specification we have here is not sufficient to describe the parser’s function, as we said for each must be a specific translation. But here we gave an idea about how the parser will work.

6 Conclusion

From the needs of data mining procedure (unifying data mining procedures, integrate data mining algorithm into DBMS and facilitate the data treatment) we proposed a language that contains the most essential functions. This language is divided into three main parts: Pre-processing Model, Data Mining Model and Post-processing Model (Deploying Models), this language allows data mining users to apply more than one data mining algorithm on the same training set (or the same algorithm with different options), the pre-processing contains some tables in which we can store training set examples and metadata like content type and hierarchy, this model gives the option to use the power of SQL inside it. Data mining model stores the algorithm options and perform the mining process. Post-processing Model is to deploy, export and publish the results. The techniques of sampling and voting make the data mining process easy and rich; these algorithms will be implemented in RDBM.

To make this language acceptable by DBMS we proposed a language parser, this parser translates all the statements of our language into SQL and PL/SQL. The idea of parser makes this language work with many DBMS, we can implement it outside DBMS but all the operations go in the database system.
The implementation of this language can be done in any programming language, but because this language can work with many DBMS, it is recommended to use platform independence programming language that works on multiple operating systems. This language can be test on any database server that supports SQL and PL/SQL.

This language can be extended to support every data mining techniques (Boosting, sampling algorithms...) and the new data mining functions. Because it is based on using data mining algorithms already implemented in DBMS. So And to facilitate the use of this language we can build a graphical interface based on this language to interact with the DBMS.
References


Appendix 1

In this appendix we try to show how to parse DML statements into SQL, PL/SQL statement:

- **Pre-processing Model**
  - Create Data Mining Table – In DML:

**DML:**
Create DM_Table <name>
{
  [{<Columns list>}];
  [{<Hierarchy definitions}>]
}

**SQL (Oracle):**
**DM Sys:**
- Insert into Data_Mining_Tables (name,..)
  Values (<name>,..);
- Insert Into Data_Mining_Colums (name,type,content,..)
  Values
    (<Column1 info>)
    (<Column2 info>)
    (...)
- Insert Into Data_Mining_Hierarchy (name,..)
  Values (<hname>,...)
- Insert Into Data_Mining_Hierarchy_Columns (name,..)
  Values (<hname>,...)

**DBMS:**
- Create Table <data_mining_table_name> ({{<Columns list>}})

If we use in Column list select statement we have to retrieve the columns information from DBMS table:

**DML:**
{{select {<column name> <data type> <content type>} from <table names> where <condition>}} || {{<column name> <data type> <content type>}}

**SQL (Oracle):**
- Create temporary table results_temp as
Select COLUMN_NAME, DATA_TYPE
From ALL_TAB_COLUMNS
WHERE TABLE_NAME = <table names> and column_name in
(<column list>);

Now we columns information a temporary table, then we read them and put them in
Data_Mining_Columns table (Data mining System table). If we mentioned the data type
in DML we have to override the data type in the original table. The after creating the
table the next step in this case is to insert the data into Data_Mining_Table.

**SQL (Oracle):**
- Create Table <data_mining_table_name> ([(<Columns list>)]
- Insert Into <data_mining_table_name> ([(<Columns list>)]) values
  (Select <Columns list>
   From <table names>
   Where <some conditions>
  )

- **Data Mining Model:**
  In this model, we have to save the data mining model information: Train set
  parameters, algorithm parameters and output options, and then we can execute the
  algorithm on the training set using the parameters.

**DML:**
Create Predictive DM_Model <DM_Model name> as
Apply Decision_Tree on <DM_Table (columns list)>
{
  Predict <column>,
  (<Training set parameters>),
  (<algorithm Parameters>),
  (<Output Options>)
}

First we need to fill the DM system tables:

**Oracle SQL:**
- insert into Data_Mining_Model
  (Name, Predictive_Desc, Algorithm, DM_Table,...) Values
  ("name",true," Decision_Tree ","DM_Table",...)
- insert into Data_Mining_Algorithm_Parameters
  (Name, Option_value ,...) Values
  (<option 1>)
  (<option 2>)
  ...
- insert into Data_Mining_output_Parameters
  (Name, Option_value ,...) Values
  (<option 1>)

Before calling the data mining function we need to know the header of Decision_tree algorithm and all other predictive algorithm. In our proposition all the algorithms must take the same parameters as input. We propose this header, in Oracle's PL/SQL:

```sql
Oracle PL/SQL:
Create or replace procedure Decision_Tree (DM_Model IN VARCHAR)
AS
<local_var_declarations>
Begin
<Algorithm Body>
End
```

Because the decision_tree procedure takes DM_Model as input, it can extract all necessary information from DM system tables (Data Mining Algorithm Parameters, Data Mining output Parameters, Data Mining training set Parameters). Of course we can reuse the written Data mining algorithms implementation, and adapt it with our approach.