ACADEMY OF ECONOMIC STUDIES

ISSUE
1

Database Systems Journal

ISSN: 2069 – 3230 Volume I (September 2010)

Journal edited by Economic Informatics Department

DBJOURNAL BOARD

Contact
Calea Dorobanților, no. 15-17, room 2017, Bucharest, Romania
Web: http://dbjournal.ro/
E-mail: editor@dbjournal.ro

Contents

Spatial Operations

Anda VELICANU
Economic Informatics Department, Academy of Economic Studies
Bucharest, ROMANIA
anda.velicanu@ie.ase.ro

This paper contains a brief description of the most important operations that can be performed on spatial data such as spatial queries, create, update, insert, delete operations, conversions, operations on the map or analysis on grid cells. Each operation has a graphical example and some of them have code examples in Oracle and PostgreSQL.
Keywords: *Spatial operations, Querying operations, Spatial data.*

1 Introduction

The values of the objects' spatial attributes represent the spatial data [1].

Spatial data can be divided in point data and regional data. The point data is a point which is completely characterized by its location in a multidimensional space. It may come directly from measurements or by transforming in order to be more easily stored and retrieved [2]. The representation of spatial data in Oracle is done according to the ANSI standard.

Spatial operations are functions that form important components of a spatial data model. They allow the storage of input data, their analysis and obtaining new data as output information. There are special spatial operations that depend as implementation of the operators made available by the database management system, in which the data is stored. No matter who the producing company is, there are certain operations that are common to any spatial products and that are based on queries. There are also regular operations that can be found in other models' data, such as create, insert, update, delete.

Intuitively, spatial operations represent different aspects from the same real world operation. Therefore, spatial operations have space-invariant properties based on which they can be describe [3].

The user views a system as an extended relational database management system. In addition to the usual attributes in relations, the user can define spatial attributes in a homogeneous way. The system supports four types of spatial entities: points, line segments, polygons, and regions. In addition to the standard SQL commands, spatial operations are augmented in order to process and query spatial data. Spatial attribute data is stored in suitable spatial data structures [4].

2 Querying operations

Query processes are generally made to select specific data in order to measure the information from data and to perform some calculations. The queries performed on spatial data can be: Local; Area; Neighborhood (the most difficult because it requires the evaluation of proximity – eg. determining the road which passes closest to a specified region).

There are two important categories of special spatial operations, which satisfy the retrieval requests and are very popular: operations that determine the spatial relations and spatial analysis operations.

Consider a and b, two spatial objects of the same type or not (points, lines, polygons, circles, etc.), that are represented the same system (x0y axis system, latitudinally – longitudinally system, a proper representation

system etc.). With these spatial operations we can:

1. Find the equality between two spatial objects a = b, which means that the two spatial objects have only common points in the system they are represented.

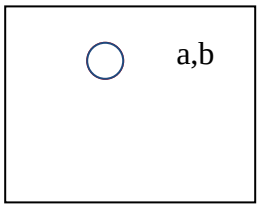


Fig. 1. Equality between two spatial objects

2. Find the disjunction between two spatial objects $a \cap b = \varnothing$, which means that the two spatial objects have no points in common.

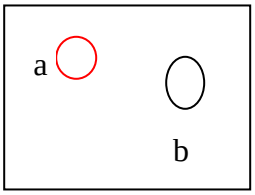


Fig. 2. Disjunction between two spatial objects

3. Calculate the minimum distance between two spatial objects Dist(a,b) = No, where No is calculated as the minimum distance between the polygons which minimum frame a and b objects

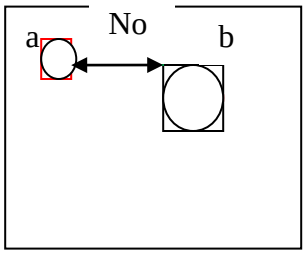


Fig. 3. The minimum distance between two objects

4. Obtain the union of geometries (spatial objects) aUb = c, where c is a spatial object that contains only once all points of a and b objects.

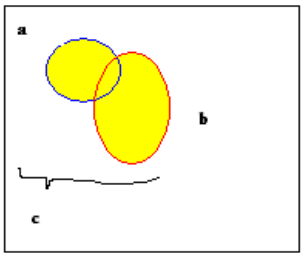


Fig. 4. The union of two spatial objects

5. Obtain the intersection of two or more geometries $a \cap b$ = c, where c is an intersection geometry, which all the common points of a and b geometries.

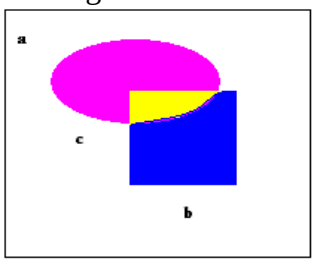


Fig. 5. The intersection of two spatial objects

6. Determine the closest spatial object Close(a) = b, if Dist(a,b) is minimum, no matter where the b object is situated in the same system as a object.

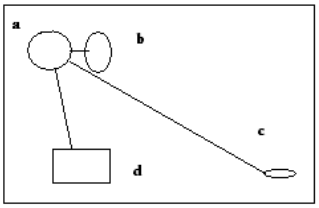


Fig. 6. Determining the closest spatial object

7. Calculate one object's area A(a) = No, where No is the area of a object.
8. Calculate the perimeter of an object P (a) = No, where No is the sum of all sides of the object
9. Calculate the centriod of an object C (a) = p, where p is the central point of object a
10. Calculate the closest geometric area CloseArea (a) = A (b), where Close (a) = b

11. Determine the buffer zone around an object

Buffer (a, No) = {b, c, d...}, where b, c, d ... is the set of points in the vicinity of the a object, at a maximum distance equal to No

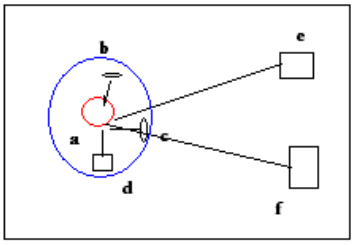


Fig. 7. Determining the buffer area around an object

12. Determine the way two objects relate RELATE(a,b) = true, if objects a and b have common points on the edges or inside them.

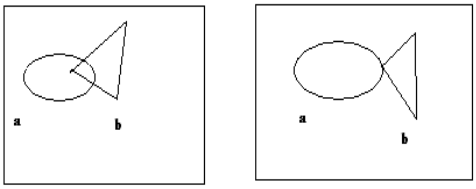


Fig. 8. Example relations between spatial objects

Determine the inclusion of a spatial object in another spatial object $a \subset b$ = true, if all a's points are contained in object b.

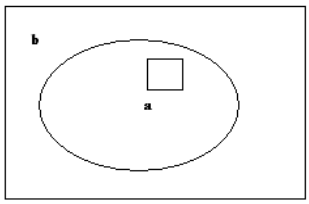


Fig. 9. Inclusion of a spatial object in another spatial object

13. Determine the touch of two objects Touch(a,b) = true, if $a \cap b = \varnothing$ and Dist(a,b) = 0

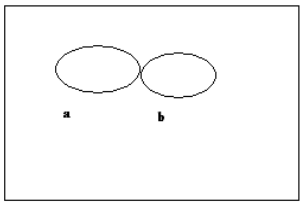


Fig. 10. The touch of two spatial objects

In order to exemplify I can briefly explain the features of spatial operations implemented in Oracle and PostgreSQL. The spatial component of Oracle is called Oracle Spatial and allows the implementation of following special operators:
- SDO_RELATE operator, which evaluates the topologic criterion (determines if two spatial objects interact);
- SDO_WITHIN_DISTANCE operator, which determines whether two spatial objects are from each other at a distance;
- SDO_INTERSECTION operator, which determines the topological intersection of two spatial elements;

An example of implementing this operator is given below, adapted from the book [5].

```
SELECT
SDO_GEOM.SDO_INTERSECTION(eg1.geom,eg2.ge
om, 0.005) as intersection FROM deposits
eg1, deposits eg2 WHERE eg1.ID='1' AND
eg2.ID='2';
```

- SDO_AREA operator, which calculates the area of a geometric figure;

An example of implementing this operator is given below, adapted from the book [5].

```
SELECT eg.name deposit,
SDO_GEOM.SDO_AREA(eg.geom, 0.005) area
FROM deposits eg;
```

- SDO_MAX_MBR_ORDINATE operator, which determines the maximum value for coordinates (x or y);
- SDO_LENGTH operator, that calculates the perimeter of geometric figures;
- SDO_DIFFERENCE operator, which determines the geometry resulting from the difference of two spatial objects;

- SDO_CENTROID operator, which determines the center of a polygon;

The spatial component of PostreSQL is called PostGIS and allows the implementation of following special operators:
- ST_INTERSECTION operator, which returns a geometry that is the common for two spatial elements;
- ST_AREA operator, that returns the calculated area of a geometric figure;
- ST_LENGTH operator, that calculates the perimeter of geometric figures;
- ST_DIFFERENCE operator, which returns a geometry representing a part of a spatial object that does not intersect another object;
- ST_UNION operator, which returns a geometry that represents the common part of two spatial objects;
- ST_GEOMETRYTYPE operator, that returns the geometric type of an object.

3 Create operations

Such operations generate new geometric objects. An example would be generating a buffer, which describes polygons at a specific distance from points, lines or areas.

4 Insert operations

This type of operations allows inserting data in the system.

5 Update operations

These operations change the characteristics of spatial objects.

6 Operations on the map

Such functions are used most often to operations that change a map's scale. Thinning the coordinates of a line means to reduce the number of coordinates that define a line. Similarly it can be applied to a polygon. Edge matching is bonding the maps and arranging them in a single unitary map.

7 Conversions

Conversions refer to rasterizing (transforming vector data into raster data) and vectorizing processes (transforming raster data into vector data).

8 Analysis of grid cells

This type of operation applies only to raster data. Handling grid cell consists of examining one cell or a combination of cells. Basic operations are: operations on a cell, matching operations on two cells and studying the neighbors of a cell. There can be performed different functions: trigonometric, exponential, statistical.

9 Delete operations

Involve the removal of objects from the system.

Aknowledgement

This article is a result of the project „Doctoral Program and PhD Students in the education research and innovation triangle". This project is co funded by European Social Fund through The Sectorial Operational Program for Human Resources Development 2007-2013, coordinated by The Bucharest Academy of Economic Studies.

References

[1] Velicanu A., Olaru S., "Optimizing spatial databases", *Revista de Informatică Economică*, Vol. 14, Nr. 2 / 2010, pag. 61-71, ISSN 1453-1305.

[2] Velicanu M., Lungu I., Muntean M., Ionescu S., "Sisteme de baze de date – Teorie şi practică", Editura Petrion, Bucureşti, 2003, pg. no. 339, ISBN 973-9470-70-X.

[3] Karimipour F., Delava M. Frank A., "An Algebraic Approach to Extend Spatial Operations to Moving Objects", *World Applied Sciences Journal 6 (10):* 1377-1383, 2009, ISSN 1818-4952.

[4] Aref W., Samet H., "Extending a DBMS with Spatial Operations", *Advances in Spatial Databases{2nd Symposium, SSD'91*, vol. 525 of Springer-Verlag Lecture Notes in Computer Science, August 1991, pg. 299-318, Zurich, Switzerland, ISBN 3-540-60153-8.

[5] Velicanu M., Lungu I., Botha I., Bâra A., Velicanu A., Rednic E., "Sisteme de baze de date evoluate", Editura ASE, 2009, nr. pg. 430, ISBN 978-606-505-217-8.

Database Access Through Java Technologies

Ion LUNGU, Nicolae MERCIOIU
Faculty of Cybernetics, Statistics and Economic Informatics,
Academy of Economic Studies, Bucharest, Romania,
ion.lungu@ie.ase.ro , nicu.mercioiu@gmail.com

As a high level development environment, the Java technologies offer support to the development of distributed applications, independent of the platform, providing a robust set of methods to access the databases, used to create software components on the server side, as well as on the client side. Analyzing the evolution of Java tools to access data, we notice that these tools evolved from simple methods that permitted the queries, the insertion, the update and the deletion of the data to advanced implementations such as distributed transactions, cursors and batch files.

The client-server architectures allows through JDBC (the Java Database Connectivity) the execution of SQL (Structured Query Language) instructions and the manipulation of the results in an independent and consistent manner. The JDBC API (Application Programming Interface) creates the level of abstractization needed to allow the call of SQL queries to any DBMS (Database Management System). In JDBC the native driver and the ODBC (Open Database Connectivity)-JDBC bridge and the classes and interfaces of the JDBC API will be described.

The four steps needed to build a JDBC driven application are presented briefly, emphasizing on the way each step has to be accomplished and the expected results. In each step there are evaluations on the characteristics of the database systems and the way the JDBC programming interface adapts to each one. The data types provided by SQL2 and SQL3 standards are analyzed by comparison with the Java data types, emphasizing on the discrepancies between those and the SQL types, but also the methods that allow the conversion between different types of data through the methods of the ResultSet object.

Next, starting from the metadata role and studying the Java programming interfaces that allow the query of result sets, we will describe the advanced features of the data mining with JDBC. As alternative to result sets, the Rowsets add new functionalities that enhance the flexibility of the applications. These are analyzed and the approach is described.

Keywords: *Java, JDBC, Database access, SQL*

1 Introduction

Java plays a dominant role in client-server programming, in the presentation layer of the websites, but also in the business logic on the applications servers. A large contributor to this success is attributed to the ability to interact with data. Starting from these advantages, a description of the JDBC (*Java Database Connectivity*) was needed, also the way these instruments can be used, emphasizing on the new features the latest version have to offer.

The standardization of SQL (*Structured Query Language*) did not block several DBMS creators to develop proprietary extensions to SQL, rezulting in the creation of different interfaces for data manipulation. However, JDBC technology offers a consistent interface for manipulating data, regardless of the format in which the data is stored (fig. 1).

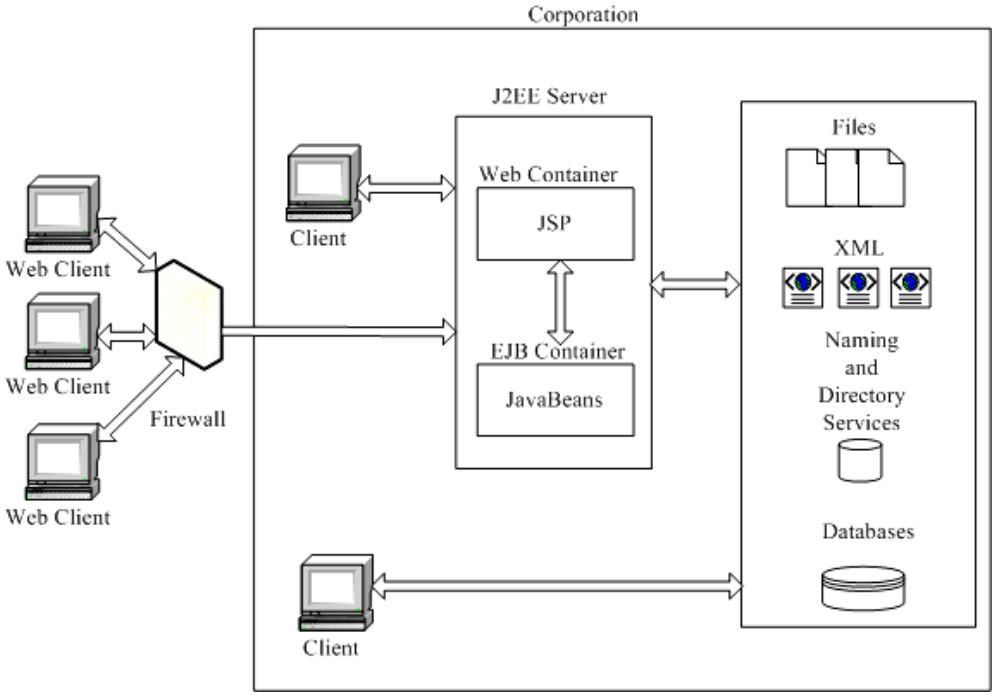


Fig. 1. The role of Java technologies to access data at enterprise level. Derived from [4] , pag. 7

2. The Evolution of Data Access Java Instruments

When Sun Microsystems released the first JDBC API 1.0 (*Application Programming Interface)* in 1997, it had several shortcomings, for instance the interface to access SQL databases. JDBC 2.0 arrived with new features such as cursors and batch files. Also, the *Optional Package*, *javax.sql* as well as other advanced features such as distributed transactions or the *RowSet* interface arrived.

JDBC 3.0 brought transactional intermediate saving points and support for SQL99 types of data. The optional packages have been included in the Java 1.4 distribution. JDBC 4.0 provides support for SQL 2003 but also extended support for CLOB (*Character Large OBject*) and BLOB (*Binary Large OBject*).

Currently, the Java API includes a JDBC-ODBC driver (*Open Database Connectivity-Java Database Connectivity*) that allows the JDBC driver access to a native system database, when an ODBC native system driver exists for that database. The Java API does not include drivers for all databases. The existence of a common programming interface brings several benefits. Otherwise, if any database creator would built its own API, that would lead to thousands of ways of programming databases, so any interface would have to be known. Luckily the software industry has chosen the JDBC standard, easying the work of developers, and allowing the creation of robust and scalable applications.

3. The Architecture

In client-server architecture the databases reside on the same or on a different machine on which the client connects to the intranet or Internet. JDBC allows to use SQL instructions and to process results of the queries in an independent and consistent manner. Using a high level of abstractization represented by the JDBC API, the situation of the programmer to handle different SQL calls to a certain DBMS (*Database Management System*) is avoided. (fig. 2).

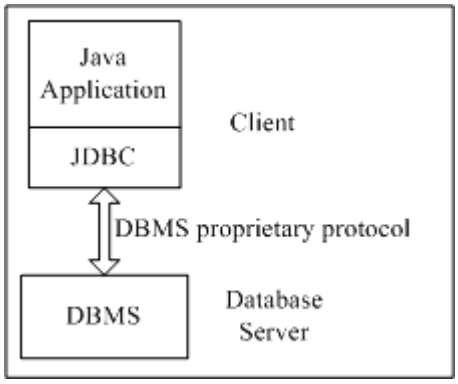


Fig. 2. The role of JDBC in accessing data –Derived from [3] , pag. 442

In order to be able to connect to a certain DBMS we only need to switch the driver, operation that can be done dynamically, even when the application runs, without the recompilation of the application.

The way those drivers are built is standardized through the JDBC specifications which describes the standard interfaces that are to be implemented. During time an evolution of those specifications occured, and the functionalities have been enhanced without compromising the compatibility with previous versions of the specifications. Generally, the JDBC specifications describe a series of interfaces that the people who develop the drivers should implement. Some databases do not allow stored procedures or other functionalities due to non-standard development of databases in general. Therefore, the JDBC specifications that have emerged from time forced the drivers creators to implement a reduced set of interfaces, other things remaining optional, without restricting the real posibilities of existing databases.

JDBC provides object-oriented access to databases through the definition of classes and interfaces that cover several abstract concepts. Also, the JDBC standard defines a series of interfaces that are to be implemented by the drivers creators in order to give the developers informations about the queried database, the DBMS used and so on. Those intels are also known as *metadata*, which means „data about data".

JDBC programming covers many aspects: client-server communication, drivers, APIs, data types and SQL instructions. The JDBC API releaves much of the burden needed to create applications with databases. It comprises many simple and intuitive components that can work for the programmer. In order to create an application one just need to assemble these components. Programming JDBC is also a very methodical way. 90% of the JDBC application uses the same objects and methods.

The first step is to obtain, install and configure the JDBC driver. Afterwards the needed component can be utilized in all the JDBC applications. After that compiling takes place, running the application and solving the eventual issues.

JDBC is an API that encapsulates calls on two levels needed to access database data and interacts through a common interface. JDK *(Java Development Kit)* and JRE (*Java Runtime Environment*) both contain the standard API, the interfaces and classes being contained in two major packages *java.sql* and *javax.sql*. The first package includes standard components, whilst the second includes enterprise level components.

The entire communication with the database occurs through the JDBC drivers. This driver converts the SQL instructions into a database server comprehensible format, using the correct networking protocols. JDBC abstracts the specific communication with the database. (fig. 3).

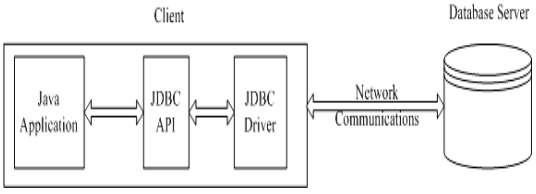


Fig. 3. The relationship between the application-JDBC-database
Derived from [4] , pag. 31

The Java SDK includes the ODBC-JDBC driver, thus allowing the access to ODBC drivers to database. Instead of accessing directly the database, JDBC "talks" to the ODBC drivers, which, in its turn communicates with the database.

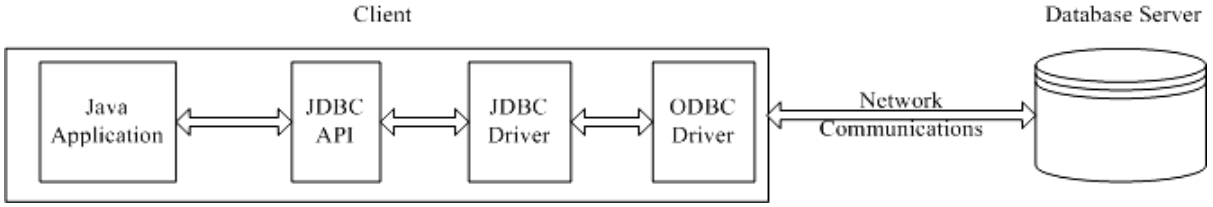


Fig. 4. The relationship application-JDBC-ODBC-database
Derived from [4] , pag. 31

Installing the JDBC driver is similar to installing any other Java API. You only have to add the path of the driver in the CLASSPATH variable when compiling and running the application. When using the ODBC-JDBC bridge driver, this step is not required, however other additional settings have to be done.

First of all, the application should be able to communicate with the database. Afterwards, the application has to be able to establish connections with the database to create a communcation channel in order to send SQL commands and retrieve results. Finallz, the application has to have a mechanism to deal with the errors. In order to accomplish all these things, the JDBC API provides the following interfaces and classes:

- *Driver* – this interface controls the communication with the database server. Rarely one should need to interact with objects of the *Driver* type. Given this, the *DriverManager* objects can be used instead. These have an abstract representation of the details associated to the work with *Driver* objects.

- *Connection* – instantiating objects of this interface represents the physical connection to the database. The result set and tranzactions can be controlled using *Connection* objects.

- *Statement* – objects created with this interface in order to send SQL commands to the database. Some derived interfaces accept supplemental parameters in order to execute stored procedures.

- *ResultSet* – these objects contain the retrieved data from the database after the query has been performed using *Statement* object. These objects allow the browsing of the data like an iterator.

- *SQLException* – a class that traps any error that is encountered in the application

Any Java application that uses databases works directly or indirectly with those four components described earlier.

4. Steps in Writing a Jdbc Application

Practically, the steps that are to be followed when writing a JDBC application are:

1. The registration of the JDBC driver with *Class.forName().newInstance()*.

2. The connection to the database is open with *DriverManager.getConnection()*.

3. A Statement type object is created in order to send SQL commands using the method *Connection.createStatement()* and afterwards *execute()*, *executeUpdate()* or *executeQuery()*

4. The connection is closed using the method *close()*.

The first step that is to be made in order to use a JDBC driver is the exact determination of the class for the driver provided by the creator. Usually, the producers respect the naming conventions of the packages when naming the drivers. The *java.sql.Driver* interface and the *java.sql.DriverManager* class are the tools to work with drivers. Registering a driver means the registration with a

DriverManager object. There are several techniques to register JDBC drivers:

- Class.forName(String *driverName*).newInstance()
- DriverManager.registerDriver(Driver *driverName*)
- jdbc.drivers property

In JDBC, an instance object of the type *Connection* represents a physical connection to the database. The method *Driver.connect()* can be used, being prefered though the *getConnection()* method of the DriverManager class because it allows the choose of the right driver. Also, the method can be overriden in order to allow opening of different menas o open connections. JDBC needs a special name system to be used when connecting to a database. The general format is jdbc:<*subprotocol*>:<*subname*> where <*subprotocol*> represents the specific protocol of the producer and <*subname*> is the source of the data (the logical name of the database we're trying to connect to).

To open connections, the *getConnection()* from the *DriverManager* class returns a valid *Connection* type object. If the method fails, *DriverManger* throws a *SQLException* containing the specific database error.

Closing the connection means the mandatory usage of the *close()* method. The method *Connection.isClosed()* does not check whether the connection is stil open or closed, but returns *true* if the method *close()* has been used. The best way to check a connection is to try a JDBC operation and the trap of the exception to determine whether the connection is still valid.

For the beginning we must be sure that the client's session has been closed on the database server. Some databases cleanse the remains if the sessions terminate unexpectadely. Then, the database sees that the user's session failed and executes a rollback to all the changes between the execution of the programme (for instance sessions that ended in the middle of the tranzaction). Explicitly closing the connections ensures that the client-server medium has been cleansed completely and makes the database administrator happier, conserving the resources used by the DBMS, for instance free licenses used on open sessions. Also RAM and CPU is spared on the server on which the database resides.

We can interact with the database in two ways. This way we can send a SQL query to obtain data about the database schema or to "learn" the values stored in some database fields, all these taking place at runtime. In that case we would need to create parametric JDBC or stored procedures. In this case we need to create. Regardless of what we want to do, the *Statement*, *PreparedStatemen*t and *CallableStatement* object provides the sufficient tools in order to attain our goals.

The corresponding interfaces define models and properties that allow sending commands and receiving data from and to the data database, as well as methods that help creating a bridge between different types of data defined in Java and specific to each type of SQL database types. For instance, the data types that have NULL values in the database in contrast with the *int* type in Java, or the different representation of date and time data between Java and SQL-92. There are methods that allow conversion of data from Java into JDBC. *Statement* objects offer DBMS interaction. They allow all the types of DML (*Data Manipulation Language*), DDL (*Data Definition Language*) commands to be executed, as well as other specific commands, batches and tranzaction management commands.

The three methods of the *Statement* objects are *execute(), executeUpdate()* and *executeQuery()* that allow sending commands to the database and retrieving results. *Execute()* processes DML instructuions, or DML or other specific database commands. It can return one or more *ResultSet* type objects. The method has flexibility, but the processing of the

results is a little bit difficult. *executeUpdate()* is used for INSERT, UPDATE, DELETE or DDL instructions and returns the number of records affected by the sent command. *executeQuery()* queries the database and returns a result set (a *ResultSet* object).

A *ResultSet* type object contains data returned by the SQL queries, run with one of the methods: *executeQuery()* sau *execute()*. Given the fact that many databases use a query language that have supplemental commands other than the DML or DDL, Java has support for a special type of commands format– *JDBC SQL escape* that allows access to specific functions of the database. When used this feature, the driver translates the commands in the specific format of the database. The *execute()* method is the most flexible way to interact with the database because it cas process result sets or number of records. The disadvantage here is that when used you cannot anticipate the type of the results returned – sets of results, number of records, or both.

The next figure shows the way the results returned by the *execute()* method are processed.

The interface set *Statement* can be used to process batches, i.e. sending multiple DML instructions (executed as one command) in a single call, that allows using a tranzactional control over the database. This control allows, for instance, the return to the initial state of all changes if one of the change failed, and by that insuring integrity and database consistency.

The tranzactions allow control of whether and when the changes are applied to the database. They allow the representation of a single instruction or a group of SQL instructions to be treated as a single logical unit, and if just one instruction within fails, the whole tranzaction fails. Tranzactions present both advantages and disadvantages. One advantage is that they allow consistency and integrity of the data. The disadvantage is that the blocking system for each database is different from database to database and the effect of initial blocking of the data initiated by the tranzaction can be sometimes surprising.

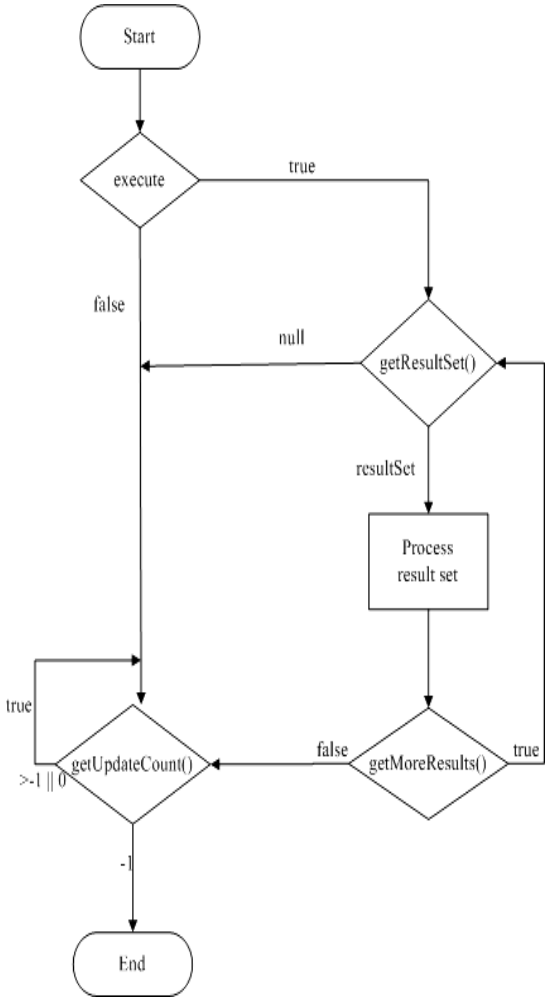


Fig. 5. The process of returning results from the execute() method - [4] , pag. 68

With JDBC tranzactions can be administered through the *Connection* type objects; for instance using the *auto-commit* module and the usage of *rollback()* method. A saving point is actually a logical tranzaction rollback point within the tranzaction. If an error occurs between the last saving point, the *rollback* method can be used to restablish the state of the data at that saving point.

The *PreparedStatement* interface offers some advantages over the classic *Statement* especially because of the feature of adding

parameters dynamically. Still, not all the databases support this feature. Also, all the commands of this type remain in memory in this open session or until the *PreparedStatement* object is closed. This *PreparedStatement* allows input and output stream, allowing us to store files in the database as values.

The *CallableStatement* object allows us to execute stored procedures in the database from the application. These objects utilize parameters as *OUT* or *INOUT*. The result sets are nothing more than rows and columns obtained from the *ResultSet* objects, creating a logical view of the data from the database. JDBC provides a class that implements the *ResultSet* interface that offers method to allow data interactivity.

Although a result set contains multiple records, at a time it is possible to get access to only one record, the "active" record. Accessing this record means moving the cursor with specific methods. Populating a set of results, the cursor initially is positioned before the first record. Obviously, in order to access data, the cursor has to be moved onto this first record. The set of records can be browsed forwards and backwards, also beyond the last records. Depending on the type of the result set, it is possible or not to go back to a previous record. If not possible, in order to access data again it is necessary to recreate the result set by executing a SQL query.

There are several types of results:

The predefined type *Standard* that allows only sequential and forward browse of the set. The data cannot be updated. It is useable to populate a simple list or other simple operations.

The second type would be *Scrollable*, which allows the browsing of the results forth and back and jumping to a specific records. This one reflects the changes in the database, so it can be used in real-time applications.

The third type would be *Updateable* that allows the update of the result set without additional SQL instructions.

The *Scrollable* and *Updateable* must be used only when really needed as they can affect the application's performance.

5. Main Types of Data

Generally, the databases support a limited types of data. If SQL2 (SQL92) offered support for limited standard types, SQL3 allows customized built types, the dimension of data that can be accomodated in a column is now bigger than 1GB of binary or character data. Also, SQL offers complex object support in business modelling and multimedia applications as well as object identifiers, abstract data and inheritance. However, not all the databases support SQL3 standard.

There are discrepancies between Java types and known database types, that requires the conversion of those Java types into SQL types and viceversa. These conversions are made through the *getXXX(), setXXX()* and *updateXXX()* methods that belong to the *ResultSet* object. It is important to know that every JDBC data type has a corresponding recommended Java type. Still, these methods are not very strict, they allow conversions from more precise into more loose types of data and even into other types (for instance: *getString()*).

Given the fact that primite Java types do not have to be defined, they store directly information, remaining constant from one application to another and from one virtual machine to another. Because primitive objects cannot be instantiated, Java offers the *wrapper* class that allows treating the primitive values as objects.

In SQL, NULL represents a data with unknown or undefined value. In Java, this NULL can present a problem, especially for numeric data types. For instance, the integer type from Java cannot have NULL values. Using the *ResultSet.getInt()* method JDBC will translate this NULL value into

0, which untreated can lead to an erroneous interpretation of the data. Objects, on the other hand, can have NULL values in Java. The *ResultSet.wasNull()* methods determines whether the last column read from the database returned a NULL value.

Data returned from SQL queries must be formatted as JDBC types. The conversion to Java types must be made before assignation to variables.

SQL UDTs (*User-Defined Types*) allows the developers to create their own definition of data types in the database. These are exlusively defined with SQL instructions, but JDBC offers support for UDT in Java applications. The custom types are materialized on the client, so the access is not directly to the value, but through an intermediate *LOCATOR* that references a value in the database. The UDTs allow the usage of large data and the way those can be used will be presented later on.

The *DISTINCT* data type allows the assignation of the new custom data type with another type of data, in a similar manner classes are extended in Java.

STRUCT is a data type built that has several members, named attributes, each of them carrying different types of data. A Java class without methods is an analogical representation of a *STRUCT*. In SQL3 *STRUCT* types of data can be constructed, each being able to hold any type of data, including other *STRUCT*.

Example: Declaring a *STRUCT:*

```
CREATE TYPE Sal_DATA(
CNP Number(9),
Nume VARCHAR(20),
Prenume VARCHAR(20),
Salariu NUMBER(9,2) )
```

JDBC allows the creation of Java classes to mirror UDTs on the database servers. The process of creation and usage of a Java class is called mapping of types. The advantages are: control of access through classes, data protection, the possibility to add new methods and attributes.

6. Data Mining With Jdbc

Understanding the concept of data mining implies the knowledge of the role of the metadata.

These are data about data. In databases, metadata represents information about data and structure and applications that deal with data. An example would be tables and attributes of the columns.

The JDBC API allows the descovery of the metadate about a database through the query of the result set using the *DatabaseMetaData* and *ResultSetMetaData* interfaces. The first one allows gathering information about the database attributes and allows taking decision at runtime upon these information. The second interface allows gathering the attributes such as number of columns, name and type of data of the result set. This information can be used, for instance, to populate a report with the name of the column and to determine which kind of *getXXX()* method should be used.

A *ResultSetMetaData* object can be used to create a generic method for processing result sets. This way, the types of the data from the column of the result set and the correct version of *getXXX()* methods to obtain data. *DatabaseMetaData* allows the creation of tools which database administrators can use to inspect databases, the structure of the tables and the users schemas.

JDBC 3.0 defines a new interface for metadata - *ParameterMetaData*. This one describes the number, type and properties of the parameters used in prepared statements.

The *ResultSetMetaData* provides information about the columns in the result set, such as number and type. The interface does not provide information about the number of records in the set of results.

The *DatabaseMetaData* interface is useful when inspecting the structure of the database. Creating a *DatabaseMetaData* object can be done by using the *getMetaData()* method of the *Connection* object. A *DatabaseMetaData* object has

many methods and properties, all those can be grouped in two categories: refering to the characteristics of the database, or refering to the structure of the database.

The first category of methods and properties answers to questions such as:

- Does the database support batches?
- What is the user which I am connected to the database?
- What kind of SQL data types does the database support?
- What are the SQL keywords supported by the database?

The methods from this category refer to information about the database return *String* results; the ones that offer information about database limitations return *int*.

The methods from the second category return a *ResultSet* object which depends on the method used to query the database. The majority of the methods are simple and allow the usage of replacement wildcards: "_" is used to replace a single character, while "*%*" can be used to replace zero, one or more characters.

Information pertaining to the tables and columns can be obtained only if sufficient access rights are given. It is recommended to use pattern of strings to avoid obtaining a result set that is too big and unusable. The *getUDT()* method offers information about the UDTs in the database. *getPrimaryKeys()* and *getImportedKeys()* provide information about the primary key and the foreign keys. The *getProcedures()* and *getProcedureColumns()* methods provide information about the stored procedures in the database.

7. *Rowsets*

Rowsets represent an alternative to result sets. The *RowSet* interface extends the *ResultSet* offering the same functionalities for viewing and manipulating data, but adds among features, functionalities that enhance the flexibility and the power of the application. Rowsets implement the *JavaBean* architecture, can operate without a coninuous connection to the data source and can offer tabulary data about any data source being in constrast with result sets that can only work with databases.

Extending the *ResultSet* interface, the *RowSet* allows access to the same methods and properties. A *RowSet* object can obtain data from a source in many other ways. The main differences betwessn these two interfaces are:

- The *RowSet* interface supports the *JavaBean* component model, allowing the developers to use the visual tools for *Beans*. *RowSet* can inform the "listeners" about events that appear.

- The row sets can operate connected or disconnected. The first way is similar to the result sets, but the disconnected stores the rows and the columns in memory, allowing the manipulaton of data in this manner.

Because the *RowSet* is in the *javax.sql* package, Sun Microsystems does not provide a standard implementation. Still, in JDBC 2.0 there are some implementations like: *JdbcRowSet*, *CachedRowSet* and *WebRowSet*.

The development of *RowSet* object based applications implies a different technique than the one with standard components. Mainly we need a single object to implement the *RowSet* interface. The steps to be pursuit when using a row set are:

1. Registering the JDBC driver.
2. Setting the connection parameters.
3. Populating the row set.

Because the *RowSet* object supports the *JavaBean* model it is impossible to access the properties of the object directly, which requires the usage of the methods *get* and *set* to configure the properties of the class that implements the *RowSet* interface.

RowSet objects can generate *JavaBean* events and allow the notification of other components the events that appear in the *RowSet* object. A row set, acts differently than a result set, because it automatically

connects to the data source when it has to retrieve or update data.

Both DDL and DML commands can be used with *RowSet* objects but the execution is different than in standard JDBC. First of all, it is not necessary to instantiate *Statement, PreparedStatement* or *CallableStatement* objects to execute SQL instructions. The object determines if there is a parametrized query or a stored procedure to be executed.

The retrieve of the data from the row sets is done with the *getXXX()* methods, where *XXX* refers to the Java type of data in which we store the value.

Since the *RowSet* interface extends the *ResultSet* interface, for browsing the rows the same methods exposed by the *ResultSet* can be used. Also, the properties can be controlled with *scrollable* and *updateable* by the *setType()* method.

After usage, the *RowSet* object must be closed in order to free the database resources used. The *RowSet.close()* method frees all the resources of the database. Closing the object is critical when this is of *JdbcRowSet* type, because this object maintains an open connection to the server once the *execute()* method is called. Objects of type *CachedRowSet* and *WebRowSet* connect to the data source when needed. However to eliminate the possibility of unwanted closing by the garbage collector, they must be explicitly closed.

Using a *JdbcRowSet* object is simple because it is a *JavaBean* component. Once the row set is populated, the methods inherited from *ResultSet* can be used to work on data. This object does not require a JDBC driver, or an open connection to the database.

The *CachedRowSet* object provides a disconnected and serializable implementation of the *RowSet* interface. Once the object is populated, it can be made serialized so we can share information with other users. It is not recommended for large amounts of data since it can exhaust the system memory.

The *WebRowSet* can work independent and is able to serialize data. This object is able to generate an XML (*eXtensible Markup Language*) file which can be used as it is or use an XML file to repopulate itself. Having a row set represented as an XML file, the data can be presented on various devices and browsers. The *WebRowSet* class works great with HTTP (*HyperText Transfer Protocol*). The client and the server exchange XML documents that represent *WebRowSet* objects. The construction of such an object is not trivial, but the browsing and he manipulation of the data set is done in a similat manner to the other types of objects presented earlier.

Conclusions

The designers of IT systems choose the combination of Java and JDBC because it allows the disemination of the information contained within databases in a simple and economic way. The operations within the organization can go on by utilizing existing databases even if these are used on different operating systems. The time used to develop new applications is shorter and the installation and versioning control is simplified. All these advantages determined us to try to describe in a non-exhaustive manner the concepts, the methods and the techniques related to JDBC technology to access databases, offered by the Java platform from Sun Microsystems.

References

[1] Leția T., Programare *avansată în Java*, Editura Albastră, 2002;
[2] Patel P., Java *Database Programming with JDBC***,** The Coriolis Group, 1996;
[3] Tanasă Ș., Olaru C., Andrei Ș., *Java de la 0 la expert*, Polirom, 2003;
[4] Thomas T., Java *Data Access JDBC, JNDI, and JAXP*, M&T Books, 2002;
[5] Văduva C., *Programare în Java*, Editura Albastră, 2002;
[6] Sun Microsystems, *JDBC Data Access API*, http://java.sun.com/products/jdbc;

Optimization of Data Requests Timing by Working with Matrixes under MSAccess Environment

Alexandru ATOMEI
Accountancy and Management Information Systems Faculty
Academy of Economic Studies, Bucharest, Romania

Abstract: This paper is going to emphasize an optimised code in order to manage matrix calculus under MSAccess. The economic impact of using such a method is the optimal cost-benefit solution, and optimised timing for data management. As well, matrix calculus is the base of Variance-Covariance method used by financial corporations as an advanced method for estimation of market risk movements with direct impact over the capital required by prudential bodies.

***Keywords**: Visual Basic, DAO (Data Access Objects) Recordset, System DSN (Data Source Name) driver, Variance-Covariance Matrix, Value at Risk.*

1 Optimization of data requests timing by working with matrixes under MSAccess Environment

A. Current stage of matrix calculus facilities

In order to use the matrixes for economic purposes, there are a series of software solutions more or less integrated in executive management of the financial corporations.

The proposed method for optimised management of data requests by working with matrixes is a similar method to the UNIX concept, which deals with its platform as a core base, and whose interation with other applications is performed through shells. Similarly, in this paper it will be presented this facility developed under Windows environment.

The econometric softwares (Matlab, SAS, Eviews, SPSS) offer some flexible solutions, but the complete integrated solutions require, in initial phase, licences, implementation projects and testing procedure, which exceed the forecasted budgets of most of up-to-medium financial corporations, and on a daily basis, a proffessional team permanently available.

The development of integrated solutions require a business study, whose implementation could not be done fastly, in order to generate advanced calculus for executive management.

A database permits the transfer of data through a XML (Extensible Markup Language) file, but there is needed a transformation of XML file into an Standardised XML file. Nowadays, last verisons of MSAccess and Oracle offers a good interaction with XML files, so that, XML files could be an alternative to the present solution presented in this paper.

The matrixes could be easily developed, also, in MSExcel programs. The MSAccess offers good database facilities, but, in terms of calculus is not so powerful compared with MSExcel. In this paper, we shall present how the MsAccess and MSExcel could be linked dinamically for serving a Variance-Covariance matrix determination.

B. The role of matrixes in accountancy

The fair value concept is often invoked in International Accountancy Standards. The fair value (market value in most of the cases) generates volatility for a certain type of asset. This volatility represents the base for a specific modeling requiring a matrix calculus.

A simple covariance between two series of data could be calculated as follows (formula 1):

Formula 1- Covariance of two series of data

$$\operatorname{cov}(X,Y) = \sum_{i=1}^{N} \frac{\left(x_i - x_a\right)\left(y_i - y_a\right)}{N}$$

Where:

Cov(X,Y) = covariance between two series of data ('xi' and 'yi');

xa = average of 'x' seria;

ya= average of 'y' seria;

N = number of observations.

A Variance-Covariance matrix is formalised as follows (formula 2):

Formula 2 – Variance-Covariance matrix

$$\mathbf{COV} = \begin{bmatrix} Ó x_1^2 / N & Ó x_1 x_2 / N & . & Ó x_1 x_c / N \\ Ó x_2 x_1 / N & Ó x_2^2 / N & . & Ó x_2 x_c / N \\ .. & .. & . & ... \\ . & . & . & \\ Ó x_c x_1 / N & Ó x_c x_2 / N & . & Ó x_c^2 / N \end{bmatrix}$$

Where:

COV = Variance-Covariance matrix

x_i = deviation from the *ith* data set

Ó x_i^2 / N = variance of elements from the *i*th data set

Ó x_i x_j / N = covariance of elements from the *i*th and *j*th data sets

N = number of observations for each of the *c* data sets

The using of matrixes into an integrated economic application of a financial corporation is required for at least two main purposes: a) Determination of market value (by multiplying the vector line of cash-flows with column vector of discount factors); b) Determination of Value at Risk ratios through Variance- Covariance method.

The Variance-Covariance method is used for Value at Risk ratio determination, being the reference element of integrated financial reporting under the International Financial Reporting Standards (IFRS) 7 of IASB (International Acccounting Standard Board).

The capital adequacy process of financial corporations is directly influenced by the type of models used for potential losses determination. The usage of matrixes will introduce a better estimation of potential losses rather than roughly applying a pre-determined coefficient which is oftenly too high, in order to be conservative.

C. Case study

Is is supposed a MSAccess application storing data about financial assets, eg. equities / indexes listed on Stock Exchange. The database have daily recordings for one year period for 8 types of stock-exchange indexes.

The Variance-Covariance matrix requires two series of data. In our case, the series of data are represented by one seria of daily yields against the average of daily yields and second seria consisting from transposed data of previous seria.

The reason of transposed seria introduction is given by the neccessity for introduction of an random factor in daily yields seria, having the same average, standard deviation and normal distribution. The random factor is founded on the supposition that a certain evolution could be contrary to the trend.

The first step in our calculation consist of using the DAO recordset facilities in order to transform the daily prices in daily yields. The second step deals with average yields calculation. The third step is performing the deduction of average yield from each daily yield.

There is presented hereinafter, the Visual Basic Code from MSAccess in order to generate the calculation of daily yields against average:

```
   Function eq_delta() 'Calculation of
daily yields against average'
   Dim r As DAO.Recordset, t As
DAO.Recordset, q As DAO.Recordset
   Dim strSQL As String
   'Local variables
```

```
      Dim data As Date
      Dim deltaBETI, deltaBETC,
deltaBETFI, deltaVAB, deltaROTXUSD,
deltaROTXEUR, deltaRSQALL As Double
      Dim BETI, BETC, BETFI, VAB,
ROTXUSD, ROTXEUR, RSQALL As Double
   Set r =
CurrentDb.OpenRecordset("SELECT * FROM
information_indexes, local_time WHERE
(((information_indexes.data)<[end] And
(information_indexes.data)<>#12/31/2008#
))")
   Set t =
CurrentDb.OpenRecordset("local_eq_delta"
)
   Set q =
CurrentDb.OpenRecordset("information_ind
exes")
   DoCmd.SetWarnings False
   DoCmd.RunSQL "Delete * From
local_eq_delta"
   DoCmd.SetWarnings True

    'First step - Daily yields
determination
   Do While Not r.EOF
       t.AddNew
       t!data = r!data
       t!deltaBETI = r!BETI / q!BETI - 1
       t!deltaBETC = r!BETC / q!BETC - 1
       t!deltaBETFI = r!BETFI / q!BETFI
- 1
       t!deltaVAB = r!VAB / q!VAB - 1
       t!deltaROTXUSD = r!ROTXUSD /
q!ROTXUSD - 1
       t!deltaROTXEUR = r!ROTXEUR /
q!ROTXEUR - 1
       t!deltaRSQALL = r!RSQALL /
q!RSQALL - 1
       t.Update
       q.MoveNext
       r.MoveNext
   Loop
     t.Close
     q.Close
     r.Close
   Set t = Nothing
    Set q = Nothing
    Set r = Nothing
   Dim m As DAO.Recordset
   Set t =
CurrentDb.OpenRecordset("local_eq_delta"
)
   Set m =
CurrentDb.OpenRecordset("local_eq_varian
ce")

   ' Second step – average daily yields
   DoCmd.SetWarnings False
   DoCmd.RunSQL "Delete * From
local_eq_variance"
   DoCmd.SetWarnings True
   DoCmd.SetWarnings False
   DoCmd.RunSQL "SELECT
Avg(local_eq_delta.deltaBETI) AS
AvgOfdeltaBETI,
Avg(local_eq_delta.deltaBETC) AS
AvgOfdeltaBETC,
Avg(local_eq_delta.deltaBETFI) AS
AvgOfdeltaBETFI,
Avg(local_eq_delta.deltaVAB) AS
AvgOfdeltaVAB,
Avg(local_eq_delta.deltaROTXUSD) AS
AvgOfdeltaROTXUSD,
Avg(local_eq_delta.deltaROTXEUR) AS
AvgOfdeltaROTXEUR,
Avg(local_eq_delta.deltaRSQALL) AS
AvgOfdeltaRSQALL INTO local_eq_delta_avg
FROM local_eq_delta"
   DoCmd.SetWarnings True

    'Third step - daily yields against
average'
   Do While Not t.EOF
       m.AddNew
       m!data = t!data
       m!deltaBETI = t!deltaBETI -
AvgOfdeltaBETI
       m!deltaBETC = t!deltaBETC -
AvgOfdeltaBETC
       m!deltaBETFI = t!deltaBETFI -
AvgOfdeltaBETFI
       m!deltaVAB = t!deltaVAB -
AvgOfdeltaVAB
       m!deltaROTXUSD = t!deltaROTXUSD -
AvgOfdeltaROTXUSD
       m!deltaROTXEUR = t!deltaROTXEUR -
AvgOfdeltaROTXEUR
       m!deltaRSQALL = t!deltaRSQALL -
AvgOfdeltaRSQALL
       m.Update
       t.MoveNext
   Loop

     m.Close
     t.Close
   Set m = Nothing
    Set t = Nothing
   End Function
   The user of MSAccess application is
performing a visualition of a specific
report in order to see the final figures
of Value of Risk. When clicking to open
a specific file, the previous code is
run as a macro (Table 2 – first command)
and is stored in local tables.
   Secondly, after the macro is run,
there is generated a new procedure as
follows:
   Private                          Sub
Command58_Click()'Transfer   of   daily
yields ag. Average in Excel, refresh
driver, report visualisation'
   On Error GoTo Err_Command58_Click
   Dim stDocName As String

    'MSAccess macro launch
       stDocName = "create_eq_variance"
       DoCmd.RunMacro stDocName

    'MSExcel launch
   Dim xlApp As Excel.Application
   Dim xlWb As Excel.workBook
   Dim xlWk As Excel.workSheet
   DoCmd.SetWarnings False
   Set xlApp = New Excel.Application
```

```
   xlApp.Application.DisplayAlerts     =
False
   xlApp.Visible = False
   Set               xlWb              =
xlApp.Workbooks.Open("d:\a\equity_var.xl
s")
   Set xlWk = xlWb.Worksheets("eq_var")
       xlWk.Range("a1").Select

xlWk.Range("a1").QueryTable.Refresh

xlWk.Range("a1").QueryTable.BackgroundQu
ery = False
       xlWk.Calculate
       xlWb.Save
       xlWb.Close
       xlApp.Quit
    Set xlWb = Nothing
    Set xlApp = Nothing

   'Report visualitation
       stDocName = "local_eq_var"
       DoCmd.OpenReport       stDocName,
acPreview
   Exit_Command58_Click:
       Exit Sub
   Err_Command58_Click:
       MsgBox Err.Description
       Resume Exit_Command58_Click
   End Sub
```

At the refresh stage of previous code there is downloaded into an Excel file (which is opened in invisible mode), through a system DSN driver, the daily yields against average that have been calculated through the procedure explained in Table 1.

The calculus for Variance-Covariance Matrix are performed due to Excel function 'CoVar'. As well, there is calculated in MSExcel the Correlation Matrix and Value at Risk figures. Each of these data are defined with names and linked to MSAccess application. Finally, the report is seeing data from a local MSAccess table which is linked to a MSExcel file, which is refreshed through a specific press of a button. The process is performed in real-time and is optimised from the cost-benefit purpose and timing of access to information compared with generation of data exclusively through SQL, or by other engines.

D. Conclusions

The results of matrixes calculus are generated each time the request is perfomed by each user. The intermediary data is not stored, so that database storage facilities are not affected.

This operation of dinamically transfer of data between MSAccess and MSExcel could create an integrated environment of exploiting the database facilities by MSExcel or other MSOffice application and, on other hand, of calculus facilities by the the MSAccess.

The data requests when working with matrixes is crucial. This solution of complex calculus generation in real-time is a real benefit for the end user, basically a financial modeller, who needs to solve and adjust frequently market data and formulas.

I. References

[1] Gheorghe Sabău, Vasile Avram, Ramona Bologa, Mihaela Muntean, Marian Dardal, Răzvan Bologa - *Baze de date*, Editura Matrix Rom, Bucureşti, 2008

[2] Manole Velicanu, Ion Lungu, Iuliana Botha, Adela Bâra, Anda Velicanu, Emanuil Rednic - *Sisteme de baze de date evoluate*, editura ASE, Bucureşti, 2009

[3] Marin Fotache, Cătălin Strâmbei, Liviu Crețu - *Oracle – Ghidul dezvoltării aplicațiilor profesionale* , Editura Polirom, Iaşi, 2003

[4] Philippe Jorion - *Financial Risk Management Handbook*, John Wiley and Sons, 2005

[***] IASB (2009) *Standardele Internaționale de Raportare Financiară,* CECCAR.

SEO Techniques for Business Websites

Alexandru ENACEANU
Romanian-American-University

In the world of website marketing, search engines are an essential key to success. They are the most important way to bring traffic to websites. Understanding how search engines work and what they require is an important first step to harnessing their marketing power. There are proven methods to search engine marketing involving website design, content adaptation, and keyword strategy. The primary goal of these methods is to bring traffic to your site. The secondary goal is for that traffic to be targeted to your product. In the internet marketing game, exposure is essential. But marketing efficiency requires effective exposure to the right prospects.
Keywords: *SEO, search engine optimization, pagerank, business website, Internet*

1 Introduction

When first building a website, you are excited to get it up on the web. You are anticipating how to handle all the new business that will be generated by thousands and thousands of visitors. Cut to a few weeks later, and you realize that no one is finding you!

Surely, by the time you have made the plans for the new website, you have heard that you must use some technique named search engine optimization to be found on Google or Yahoo.

Concepts :

SEO (Search Engine Optimization) – a subset of search engine marketing that seeks to improve the number and quality of visitors to a web site from "natural" ("organic" or "algorithmic") search results.

PR (PageRank) - a link analysis algorithm which assigns a numerical weighting to each element of a hyperlinked set of documents, such as the World Wide Web, with the purpose of "measuring" its relative importance within the set. The algorithm may be applied to any collection of entities with reciprocal quotations and references.

Web directory - a web directory is a directory on the World Wide Web. It specializes in linking to other web sites and categorizing those links. Web directories often allow site owners to directly submit their site for inclusion, and have editors review submissions for fitness.

The most common methods of SEO include on-page optimization or utilizing keywords and meta tags and link strategies. Here are some steps to take in order from not being listed far away from the top or not being listed at all:

2. Search Engine Submission

The first thing to do is submitting the website's homepage. Many search
engines will promise to find and crawl the rest of your website automatically. But if they don't disagree from doing so, submitting several of the important pages will help. For example, a site map is definitely something to submit, since it should have direct links to the rest of the website.
Also, it is recommended to ask for another webmaster to link the new site to his already submitted website. That way the search engines will recognize that this resource has changed.

WHERE TO SUBMIT

It is recommended to submit your home page to the major search engines individually, at least initially :

- Submit to Google http://www.google.com/addurl/?continue/addurl
- Submit to Yahoo

http://submit.search.yahoo.com/free/request

- Submit to MSN http://beta.search.msn.com/docs/submit.aspx?FORMWSUT

A Yahoo account is needed to submit to the Yahoo search engine. Immediate results should not be seen immediately. Your site should normally exist in MSN within about 6 weeks, in Yahoo in 8-12 weeks, and in Google within about 3 months. In the long run, Google will normally give you about 60 - 70% of the search engine traffic if you follow the hereafter steps.

There are several services that do groups of them for you - and is a big time saver for the rest of your site. The following is one of the free and well-known website submitter: http://www.freewebsubmission.com/.

3. Directory Submission

A web directory is not a search engine, and does not display lists of web pages based on keywords, instead it lists web sites by category and subcategory. The categorization is usually based on the whole web site, rather than one page or a set of keywords, and sites are often limited to inclusion in only one or two categories.

The first directory to submit into is **DMOZ** (http://www.dmoz.com/add.html) which is closed for submission for the moment, but all people hope it will come up again . The Open Directory Project is the largest, most comprehensive human-edited directory of the Web.

This is a massive directory that is republished in several other websites. It is managed by volunteer humans, and is therefore considered to be of special relevance by other search engines (especially Google).

Read all their rules before submitting - and follow them closely. Make sure that you try to get listed in only one category - the most relevant one for your business. It can take a month or two to get listed, but it really helps with your backlinks and overall accuracy as a website.

After DMOZ, here are the most important list of directories to be listed in :

- Yahoo Directory website submission ($299 annual fee) https://ecom.yahoo.com/dir/reference/submit/
- Business.com website submission ($199 annual fee) http://www.business.com/
- Best of Web website submission ($40 annual fee) http://www.botw.org/
- wowdirectory.com website submission ($25 lifetime fee) http://www.wowdirectory.com/howtoadd.php
- LOCAL directories from your own country

There are specialized directories that focus on a particular category of links. These can be valuable - you will just have to do a bit of searching to find them. These may be considered as part of your overall strategy.

Being listed in a search engine doesn't guarantee that you will have a good ranking - this is just the first step - letting them know that you exist.

4. Technical recomandations

a.Site Design

Use the "Keep It Simple" principle. Employ an external CSS file, clean

up any Java Scripts by referring to them off the page in an external file, don't use frames, use flash the way you would an image, and no matter what, do not create a flash site.

Page Size - Your web page's speed is important to your visitors and the search engines, because the robots will be able to spider your web page faster and easier. Try your best to keep your web page over 5k and under 15k in size.

b. Validate your site

Run a website validator on the pages intended for submitting - to keep the search engine spiders from choking on your website. (http://validator.w3.org/)

c. HEAD part of the page

Title Tag - The title tag is the most powerful on-site SEO technique you have, so use it creatively! What you place in the title tag should only be one thing, the exact keyword you used for the web page that you are trying to optimize. Every single web page should have it's own title tag.

Keyword Density - This is also vital and should be used with research. You should use the keyword(s) once in the title tag, once in the heading tag, once in bold text, and get the density between 5% to 20% (Don't over do it!). Also use your keyword(s) both low and high on the web page, keyword(s) should be in the first sentence and in the last one.

d. Internal links

Internal links are the easiest to attain links. That would be, the ones right there on your own site and those which you have total and complete control of. Properly used internal links can be a useful weapon in your SEO arsenal.

The internal linking structure can:

- Insure that your website gets properly spidered and that all pages are found by the search engines
- Build the relevancy of a page to a keyword phrase
- Increase the PageRank of an internal page

Your web pages should be no more than three clicks away from the home page. Link to topic related quality content across your site. This will also help build you a better theme through out your web site. On every page you should link back to your home page and your main service(s).

e. External links

External links are links coming from other websites to your pages.

External links could be reciprocal or one-way links. Reciprocal links should be avoided, because of the Google's new policy to ignore reciprocal linking – named JAGGER.

There are a number of tactics for building one-way links: articles, press releases, paid links.

Digging for External links

Instead of looking around for nice sites, and then asking if they're interested in a link exchange with you, just scout around and look at where other people are getting links from. Visit a site that's similar in topic to your site (competitors).

Go to Google.com and type "link:www.nameofyourcompetitor.com" (this will list all the pages that have links pointing to the current page).

Click on every link, opening each one in a new window. Close all the pages with Page Rank less than 5.

Visit the remaining sites and see if they accept subscription or paid advertising.

f. Site Map

Build a site map with a link to each of your pages. Keep it up to date. This will allow the spiders to get to every page. Do not include session IDs in the links advertised. Session IDs confuse search engines.

Submit it to search engines or put a text link to the site map on the main pages.

g. Short URLS

Keep the URLs short with page names having the keywords that best reflect page's content. Use a delimiter like underscore or dash to separate the keywords or products model from each other.

h. Fresh content

Add a new product/service or a new review every 2-3 days: 200-500 words. Create original content, don't copy others. The more original and useful it is, the more people will

read it, link to it, and most importantly of all keep coming back for more.

i. NO Spam

Stay away from black hat optimizing techniques. Black hat optimization

consists of using any method to get higher rankings that the search engines would disapprove of, such as keyword stuffing, doorway pages, invisible text, cloaking and more. Stick to white hat methods for long-term success. People who use black hat optimization are usually there for the short-term (just look at your email spam for more black hat markets). These black hat industry sites are usually around just long enough to make quick incomes.

j. Statistics

Make sure your server has a good statistics program. If you

don't have access to a good program, then pay for one. Without the

knowledge of who is coming to your site, where from, and how often, you

will be missing out on some essential tools to improve your site.

k. RSS Feeds

RSS (Real Simple Syndication or Rich Site Summary) is becoming a powerful tool for Internet marketers. You can quickly and easily add fresh content to your website. Article feeds are updated frequently, so you can give your visitors (and the search engines) what they want - fresh content! You can use RSS to promote any new content, such as new products, special offers, articles or reviews.

l. Text browser

The final step is to use a text browser like **Lynx** (http://lynx.isc.org).

This helps you to see how your website "looks like" for search engines. Try to find good keywords within the pages.

Conclusion

80 to 90% of Internet users turn to search engines such as Google, Yahoo, msn to find information they need. Therefore, the importance of search engines should be treated accordingly in the marketing campaign. Even if the exact formulae that the top search engines use to calculate rankings are usually a closely guarded secret, you can apply SEO techniques because it is free, it is easy, it is targeted marketing, and do not need to be constantly monitored or funded as they are self-sustaining once you set them into motion.

Bibliography

[1]SeoPedia Forum - http://forum.seopedia.ro/
[2] Aaron Wall's SEO Book.com – http://www.seobook.com/
[3] WikiPedia – http://www.wikipedia.com
[4]EzineArticles – http://www.ezinearticles.com
[5] SEO Company Canada - http://www.seocompany.ca/
[6] Fallows, Deborah; Rainie, Lee, The Popularity and Importance of Search Engines, Pew Internet&American Life Project, 2004
[7] Foster, Peter, The perils of the keyword search, http://www.telegraph.co.uk , June 16, 2006
[8] Thurow, Shari, Search Engine Visibility, Macmillan Computer Pub 2003
[9] Brad, Konia, Search Engine Optimization W/webposition, WWPD Library

Solutions for improving data extraction from virtual data warehouses

Adela BÂRA
Economic Informatics Department, Academy of Economic Studies
Bucharest, ROMANIA
bara.adela@ie.ase.ro

Abstract: The data warehousing project's team is always confronted with low performance in data extraction. In a Business Intelligence environment this problem can be critical because the data displayed are no longer available for taking decisions, so the project can be compromised. In this case there are several techniques that can be applied to reduce queries' execution time and to improve the performance of the BI analyses and reports. Some of the techniques that can be applied to reduce the cost of execution for improving query performance in BI systems will be presented in this paper.
***Keywords**: Virtual data warehouse, Data extraction, SQL tuning, Query performance*

1 Introduction

The Business Intelligence (BI) systems manipulate data from various organizational sources like files, databases, applications or from the Enterprise Resource Planning (ERP) systems. Usually, data from these sources is extracted, transformed and loaded into a primary target area, called staging area which is managed by a relational database management system. Then, in order to build analytical reports needed in a BI environment, a second ETL (extract, transform and load) process is applied to load data into a data warehouse (DW). There are two ways to implement a DW: to store data in a separate repository which is the traditional method and to extract data directly from the relational database which manages the staging repository. The last solution is usually applied if the ERP system is not yet fully implemented and the amount of data is not as huge as the queries can be run in a reasonable time (less than 30 minutes). The implementation of a virtual data warehouse is fastest and requires a low budget then a traditional data warehouse. Also this method can be applied in a prototyping phase but after the validation of the main functionalities, data can be extracted and loaded into a traditional data warehouse. It's a very good practice to use the staging area already build for the second ETL process to load data into the data warehouse.

This paper presents some aspects of the implementation of a virtual data warehouse in a national company where an ERP was recently setup and a set of BI reports must be developed quickly. Based on a set of views that collects data from the ERP system, a virtual data warehouse based on an ETL process was designed. The database management system (DBMS) is Oracle Database 10g Release 2 and the reports were developed in Oracle Business Intelligence Suite (OBI). After the development of the analytical BI reports, the project team run several tests in a real organizational environment and measured the performance of the system. The main problem was the high cost of execution. These reports were over 80% resource consuming of the total resources allocated for the ERP and BI systems. Also, the critical moment when the system was breaking down was at the end of the month when all transactions from functional modules were posted to the General Ledger module. After testing all parameters and factors, the team concluded that the major problem was in the data extraction from the relational database. So, in order to improve the query performance, some of the main optimization techniques are considered.

2 An overview of the SQL execution process

The query performance depends on one side on the technology and the DBMS that are used and on the other side on the way queries are executed and data are processed. So, first let's take a look on the way Oracle Database manages the queries. There are two memory structures which are responsible with SQL processing: the System Global Area (SGA) - a shared memory area that contains data and control information for the instance and the Program Global Area (PGA) - a private memory region containing data and control information for each server process.

The main component in the SGA that affect query optimization process is the Shared pool area which caches various SQL constructs that can be shared among users and contains shared SQL areas, the data dictionary cache, and the fully parsed or compiled representations of PL/SQL blocks. A single shared SQL area is used by multiple users that issue the same SQL statement. The size of the shared pool affects the number of disk reads. When a SQL statement is executed, the server process checks the dictionary cache for information on object ownership, location, and privileges and if it is not present, this information is loaded into the dictionary cache through a disk read. The disk reads and parsing are expensive operations; so it is preferable that repeated executions of the same statement find required information in memory, but this process require a large amount of memory. So in conclusion, the size of the shared pool leads to better SQL management by reducing disk reads, shared SQL queries, reducing hard paring and saving CPU resources and improving scalability.

The PGA is a non-shared memory area that is allocated for each server process that can read and write to it. The Oracle Database allocates a PGA when a user connects to an Oracle database. So, a PGA area contains the information about: the user session that initiated it, the cursor that is executed in the PGA and the SQL work areas. The main components which affect the query execution are the SQL work areas. A SQL query is executed in a SQL work area based on an execution plan and algorithm: hash, sort, merge. Thus, the SQL work area allocates a hash area or a sort area or a merge area in which the query is executed. These algorithms are applied depending on the SQL operators, for example a sort operator uses a work area called the sort area to perform the in-memory sort of a set of rows. A hash-join operator uses a work area called the hash area to build a hash table from its left input. If the amount of data to be processed by these two operators does not fit into a work area, then the input data is divided into smaller pieces. This allows some data pieces to be processed in memory while the rest are spilled to temporary disk storage to be processed later. But the response time increases and it affects the query performance. The size of a work area can be controlled and tuned, but in general bigger database areas can significantly improve the performance of a particular operator at the cost of higher memory consumption. The best solution that can be applied is to use Automated SQL Execution Memory (PGA) Management which provides an automatic mode for allocating memory for SQL working. Thus the working areas that are used by memory-intensive operators (sorts and hash-joins) can be automatically and dynamically adjusted. This feature of Oracle Database offers several performance and scalability benefits for analytical reports workloads used in a BI environment or mixed workloads with complex queries. The overall system performance is maximized, and the available memory is allocated more efficiently among queries to optimize both throughput and response time [1].

Another important component of the Oracle Database is the Query optimizer that creates the execution plan for a SQL statement. The execution plan can greatly affect the execution time of a SQL query

and it consists in a series of operations that are performed in sequence to execute the specified statement. The Query optimizer considers many factors related to the objects referenced and the conditions specified in the statement such as: statistics gathered for the system related to the I/O operations, CPU resources and schema objects; information in the data dictionary; conditions in WHERE clause; execution hints supplied by the developers. Based on the evaluation of these factors the Query optimizer decides which is the most efficient path to access data and how to join tables (full-scan, hash, sort, and merge algorithms). In conclusion the execution plan contains all information of a SQL statement execution and in order to improve query performance we have to analyze this plan and to try to eliminate some of the factors that affect the performance.

3 Optimization solutions

3.1. Materialized views

To reduce the multiple joins between relational tables in a virtual data warehouse, the first solution was to rewrite the views and build materialized views and semi-aggregate tables on the staging area. Data sources are loaded in these tables by the ETL (extract, transform and load) process periodically, for example at the end of the week or at the end of the month after posting to the General Ledger. A benefit of this solution is that it eliminates the joins from the views and the ETL process can be used to load data in a future data warehouse that will be implemented after the prototype validation.

After re-write the queries in terms of materialized views, the project team re-test the system under real conditions. The time for data extraction was again too long and the costs of executions consumed over 50% of total resources. So, on these materialized views and tables some of optimization techniques must be applied. These techniques are: table partitioning, indexing, using hints and using analytical functions instead of data aggregation in some reports.

3.2 Partitioning

The main objective of the partitioning technique is to decrease the amount of disk activity and limiting the amount of data to be examined or operated on and enabling parallel execution required to perform queries against virtual data warehouses. Tables are partitioning using a partitioning key that is a set of columns which will determine by their conditions in which partition a given row will be store. Oracle Database 10g on which our ERP is implemented provides three techniques for partitioning tables [1]:

- Range Partitioning - specify by a range of values of the partitioning key;
- List Partitioning - specify by a list of values of the partitioning key;
- Hash Partitioning - a hash algorithm is applied to the partitioning key to determine the partition for a given row;

Sub partitioning techniques can be applied and first tables are partitioned by range/list/hash and then each partition is divided in sub partitions:

- Composite Range-Hash Partitioning – a combination of Range and Hash partitioning techniques, in which a table is first range-partitioned, and then each individual range-partition is further sub-partitioned using the hash partitioning technique;
- Composite Range-List Partitioning - a combination of Range and List partitioning techniques, in which a table is first range-partitioned, and then each individual range-partition is further sub-partitioned using the list partitioning technique.
- Index-organized tables can be partitioned by range, list, or hash.

In our case we consider evaluating each type of partitioning technique and choose the best method that can improve the queries' performance. Some of our research can be found also in [2] and [3].

For the loading process we created two tables based on the main table and compare the execution cost obtained by applying the same query on them. First table TEST_A

contained un-partitioned data and is the target table for an ETL process. It counts 100000 rows and the structure is shown below in the scripts. The second table TEST_B is a range partitioned table by column T_DATE which refers to the date of the transaction. This table has four partitions as you can observe from the script below:

```
create table test_b
( T_DATE     date not null,
  PERIOD varchar2(15) not null,
  DEBIT  number,
  CREDIT  number,
  ACCOUNT     varchar2(25),
  DIVISION  varchar2(50),
  SECTOR  varchar2(100),
  UNIT varchar2(100))
partition by range (T_DATE)
(partition QT1  values less than
(to_date('01-APR-2009', 'dd-mon-
yyyy')),
partition QT2 values less than
(to_date('01-JUL-2009', 'dd-mon-
yyyy')),
partition QT3  values less than
(to_date('01-OCT-2009', 'dd-mon-
yyyy')),
partition QT4  values less than
(to_date('01-JAN-2010', 'dd-mon-
yyyy')));
```

Then, we create the third table which is partitioned and that contained also for each range partition four list partitions on the column “Division” which is very much used in data aggregation in our analytical reports. The script is showed below:

```
  create table TEST_C
(  T_DATE     date not null,
  PERIOD varchar2(15) not null,
  DEBIT  number,
  CREDIT  number,
  ACCOUNT     varchar2(25),
  DIVISION  varchar2(50),
  SECTOR   varchar2(100),
  UNIT varchar2(100))
partition by range (T_DATE)
subpartition by list (DIVISION)
(partition QT1  values less than
(to_date('01-APR-2009', 'dd-mon-
yyyy'))
(subpartition QT1_OP values
('a.MTN','b.CTM','c.TRS','d.WOD','e.DM
A'),
 subpartition QT1_GA values ('f.GA
op','g.GA corp'),
 subpartition QT1_AFO values ('h.AFO
div','i.AFO corp'),
 subpartition QT1_EXT values
('j.EXT','k.Imp') ),
partition QT2 values less than
(to_date('01-JUL-2009', 'dd-mon-
yyyy'))
(subpartition QT2_OP values
('a.MTN','b.CTM','c.TRS','d.WOD','e.DM
A'),
 subpartition QT2_GA values ('f.GA
op','g.GA corp'),
 subpartition QT2_AFO values ('h.AFO
div','i.AFO corp'),
 subpartition QT2_EXT values
('j.EXT','k.Imp')),
partition QT3  values less than
(to_date('01-OCT-2009', 'dd-mon-
yyyy'))
(subpartition QT3_OP values
('a.MTN','b.CTM','c.TRS','d.WOD','e.DM
A'),
 subpartition QT3_GA values ('f.GA
op','g.GA corp'),
 subpartition QT3_AFO values ('h.AFO
div','i.AFO corp'),
 subpartition QT3_EXT values
('j.EXT','k.Imp')),
partition QT4  values less than
(to_date('01-JAN-2010', 'dd-mon-
yyyy'))
(subpartition QT4_OP values
('a.MTN','b.CTM','c.TRS','d.WOD','e.DM
A'),
 subpartition QT4_GA values ('f.GA
op','g.GA corp'),
 subpartition QT4_AFO values ('h.AFO
div','i.AFO corp'),
 Subpartition QT4_EXT values
('j.EXT','k.Imp')));
```

After loading data in these two partitioned tables we gather statistics with the package DBMS_STATS. Analyzing the decision support reports we choose a sub-set of queries that are always performed and which are relevant for testing the optimization techniques. We run these queries on each test table A, B and C and compare the results in table 1.

In conclusion, the best technique in our case is to use table C instead table A or table B, that means that partitioning by range of T_DATE and then partitioning by list of DIVISION with type VARCHAR2 is the most efficient method. Also, we obtained better results with table B partitioned by range of T_DATE than table A non-partitioned.

Table 1 Comparative analysis results for simple queries

TABLE:	TES	TEST_B	TEST_C

	T_A					
QUERRY:	Not partitioned	Partition range by date on column "T_DATE"		Partition range by date with four list partions on column "DIVISION"		
		Without partition clause	Partition (QT1)	Without partition clause	Partition (QT1)	Sub-partition (QT1_AFO)
Select * from TEST_	**170**	184	-	346	-	-
where extract (month from T_date) =1;	183	197	91	357	172	172
… and division='h.AFO divizii'	173	199	91	**25**	**12**	172
select sum(debit) TD, sum(credit) TC from test_ where extract (month from t_date) =1 and division='h.AFO divizii'	173	199	91	**25**	12	172
… and unit ='MTN'	173	199	91	350	172	172

Note: The grey marked ones have the best execution cost of the current query

3.3 Using hints and indexes

When a SQL statement is executed the query optimizer determines the most efficient execution plan after considering many factors related to the objects referenced and the conditions specified in the query. The optimizer estimates the cost of each potential execution plan based on statistics in the data dictionary for the data distribution and storage characteristics of the tables, indexes, and partitions accessed by the statement and it evaluates the execution cost. This is an estimated value depending on resources used to execute the statement which includes I/O, CPU, and memory [1]. This evaluation is an important factor in the processing of any SQL statement and can greatly affect execution time.

We can override the execution plan of the query optimizer with hints inserted in SQL statement. A SQL statement can be executed in many different ways, such as *full table scans, index scans, nested loops, hash joins and sort merge joins*. We can set the parameters for query optimizer mode depending on our goal. For BI systems, time is one of the most important factor and we should optimize a statement with the goal of best response time. To set up the goal of the query optimizer we can use one of the hints that can override the OPTIMIZER_MODE initialization parameter for a particular SQL statement [1]. The optimizer first determines whether joining two or more tables having UNIQUE and PRIMARY KEY constraints and places these tables first in the join order. The optimizer then optimizes the join of the remaining set of tables and determinates the cost of a join depending on the following methods:

- Hash joins are used for joining large data sets and the tables are related with an equality condition join. The optimizer uses the smaller of two tables or data sources to build a hash table on the join key in memory and then it scans the larger table to find the joined rows. This method is best used when the smaller table fits in available memory. The cost is then limited to a single read pass over the data for the two tables.

- Nested loop joins are useful when small subsets of data are being joined and if the join condition is an efficient way of accessing the second table.
- Sort merge joins can be used to join rows from two independent sources. Sort merge joins can perform better than hash joins if the row sources are sorted already and a sort operation does not have to be done.

We compare these techniques using hints in SELECT clause and based on the results in table 2 we conclude that the Sort merge join is the most efficient method when table are indexed on the join column for each type of table: non-partitioned, partitioned by range and partitioned by range and sub partitioned by list.

Table 2. Comparative analysis results using hints

TABLE:	**TEST_A**	**TEST_B**		**TEST_C**		
QUERRY**:**	Not partitioned	Partition range by date on column "T_DATE"		Partition range by date with four list partions on column "DIVISION"		
		Without partition clause	Partition (QT1)	Without partition clause	Partition (QT1)	Sub-partition (QT1_AFO)
select /*+ *USE_HASH (a u)*/* a.*, u.location,u.country, u.region from TEST_t a , d_units u where a. unit=u. unit and extract (month from T_date) =1	176	182	95	294	152	152
… and a.division = 'h.AFO divizii'	175	181	94	28	19	150
…/*+ *USE_NL (a u)*/*	281	287	151	170	18	171
…/*+ *USE_NL (a u)*/* --*WITH INDEXES*	265	235	110	120	12	143
…/*+ *USE_MERGE (a u)*/* --*WITH INDEXES*	174	180	94	27	18	150
…and u. unit ='MTN'	174	180	94	151	19	150
…/*+ *USE_NL (a u)*/*	174	180	94	21	18	150
…/*+ *USE_NL (a u)*/* --*WITH INDEXES*	172	178	86	21	12	144
…/*+ *USE_MERGE (a u)*/* --*WITH INDEXES*	172	178	86	21	12	144

The significant improvement is in sub partitioned table in which the cost of execution was drastically reduce at only 12 points compared to 176 points of non-partitioned table. Without indexes the most efficient method is hash join with best results in partitioned table and sub partitioned table.

3.3 Using analytical functions

In the latest versions in addition to aggregate functions Oracle implemented analytical functions to help developers building decision support reports [1]. Aggregate functions applied on a set of records return a single result row based on groups of rows. Aggregate functions such as SUM, AVG and COUNT can appear in SELECT statement and they are commonly used with the GROUP BY clauses. In this case Oracle divides the set of records into groups, specified in the GROUP BY clause. Aggregate functions are used in analytic reports to divide data in groups and analyze these groups separately and for building subtotals or totals based on groups. Analytic functions process data based on a group of records but they differ from aggregate functions in that they return multiple rows for each group. The group of rows is called a window and is defined by the analytic clause. For each row, a sliding window of rows is defined and it determines the range of rows used to process the current row. Window sizes can be based on either a physical number of rows or a logical interval, based on conditions over values [4]

Analytic functions are performed after completing operations such joins, WHERE, GROUP BY and HAVING clauses, but before ORDER BY clause. Therefore, analytic functions can appear only in the select list or ORDER BY clause [1].

Analytic functions are commonly used to compute cumulative, moving and reporting aggregates. The need for these analytical functions is to provide the power of comparative analyses in the BI reports and to avoid using too much aggregate data from the virtual data warehouse. Thus, we can apply these functions to write simple queries without grouping data like the following example in which we can compare the amount of current account with the average for three consecutive months in the same division, sector and management unit, back and forward:

```
   select period, division, sector,
unit, debit,
   avg(debit) over (partition by
division, sector, unit
   order by extract (month from
t_date)
   range between 3 preceding and 3
following) avg_neighbours
   from test_a
```

3.4. Object oriented implementation

A modern RDBMS environment, such as Oracle Database 10g, supports the object type concepts that can be used to specify the multidimensional models (MD) constrains. An object type differs from native SQL data types in that it is user-defined, and it specifies both the underlying persistent data (attributes) and the related behaviours (methods).

The object type is an object layer that can map the MD model over the database level, but data is still stored in columns and tables. Internally, statements about objects are still basically statements about relational tables and columns, and you can continue to work with relational data types and store data in relational tables. But we have the option to take advantage of object-oriented features too. Data persistency is assured by the object tables, where each row of the table corresponds to an instance of a class and the table columns are the class's attributes. Every row object in an object table has an associated logical object identifier. There can be use two types of object identifiers: a unique system-generated identifier of length 16 bytes for each row object assigned by default by Oracle in a hidden column, and primary-key based identifiers specified by the user and in which we have the advantage of enabling a more

efficient and easier loading of the object table [1].

The object oriented implementation can be used to reduce the execution cost by avoid the multiple joins between the fact and the dimension tables. For exemplification we'll present here only the classes of management unit dimension (table unit in our previous examples) and the fact table – balance_R.

We'll use a super type class to define the management unit dimension. We'll call it as UnitSpace_OT. For hierarchical levels of the dimension, as you can observe, there are two major hierarchies:

- geographical locations (H1): zone->region->country->location-> unit
- organizational and management (H2): division->sector-> unit.

So, the final object in both hierarchies is unit which will have two REFs, one for H1 and one for H2 hierarchies. The script is shown below:

For the first hierarchy (H1):

```
create or replace type unitspace_ot as object (unitspace_id number, unitspace_desc varchar2 (50), unitspace_type varchar2 (50)) not instantiable not final;
create or replace type zone_o under unitspace_ot (/*also add other attributes and methods*/) not final;
create or replace type region_o under unitspace_ot (zone ref zone_o /*also add other attributes and methods*/) not final;
create type country_o under unitspace_ot (region ref region_o /*also add other attributes and methods*/) not final;
create type location_o under unitspace_ot (country ref country_o /*also add other attributes and methods*/) not final;
The second hierarchy (H2):
create or replace type division_o under unitspace_ot (/*also add other attributes and methods*/) not final;
create type sector_o under unitspace_ot (division ref division_o /*also add other attributes and methods*/) not final;
create or replace type unit_o under unitspace_ot (sector ref sector_o, location ref location_o /*also add other attributes and methods*/) final;
```

The orders' fact is implemented also as an object type INSTANTIABLE and NOT FINAL:

```
create or replace type balance_r as object
    ( T_DATE date not null,
      PERIOD varchar2(15) not null,
      DEBIT number,
      CREDIT number,
      ACCOUNT varchar2(25),
      DIVISION varchar2(50),
      SECTOR varchar2(100),
      UNIT_ID varchar2(100)));
```

The methods of each MD object type are implemented as object type bodies in PL/SQL language that are similar with package bodies. For example the unit_o object type has the following body:

```
create or replace type body unit_o as
static function f_unit_stat(p_tab varchar2,p_gf varchar2, p_col_gf varchar2, p_col varchar2, p_val number) return number as
  /* the function return the aggregate statistics from fact tables for a specific unit */
  v_tot number;
  text varchar2(255);
  begin
   text:= 'select '|| p_gf || ' ('||p_col_gf||') from '||p_tab||' cd where '||p_col||'= '||p_val;
   execute immediate text into v_tot;
    return v_tot;
  end f_unit_stat;
/*others functions or procedures*/
end;
/
```

We can use this function to get different aggregate values for a specific unit, such as the total amount of quantity per unit or the average value per unit. Data persistency is assured with object tables that will store the instances of that object type, for example:

```
CREATE TABLE unit_t OF unit_o;
```

For example the static function *f_unit_stat* from the *unit_o* class can be use to retrieve the total debit value for each unit:

```
(1) select unit_id, description,
unit_o.f_unit_stat('balance_rt', 'SUM',
'debit', 'unit_id', unit_id) total
from unit_t;
```

instead of using the join between the unit_t and the balance_r tables:

```
(2) select t.unit_id, t.description,
SUM(debit) total_debit
from unit t, balance_r b
where t.unit_id=b.unit_id
```

We'll use for testing two types of tables: object tables (unit_t and balance_rt) and relational tables (unit and balance_r). The cardinality of these tables is about 100000 records in balance and about 100 in unit tables.

We analyze the impact of calling the function in the SQL query in different situations, as we present in the following table:

Table 3. The execution costs of the queries

No	Query	Cost
	select unit_id, description, unit_o.f_unit_stat('balance_rt', 'SUM', 'debit', 'unit_id', unit_id) total from unit_t	35
	select t.unit_id, t.description, SUM(debit) total from unit t, balance_r b where t.unit_id=b.unit_id	171
	Example (1) with an index on unit_id on both object tables	30
	Example (2) with an index on unit_id on both relational tables	171
	Example (1) with an index on unit_id on both relational tables with *use_nl* hint	79

We analyze the execution cost of the function's query, it has 35 units, but the SQL Tuning Advisor makes a recommendation: "consider collecting statistics for this table and indices". We used DBMS_STATS package to collect statistics from both object and relational tables. Then we re-run the queries and observe the execution plans; there is no change and the Tuning Advisor doesn't make any recommendation.

By introducing these types of functions we have the following *advantages*:

- The function can be used in many reports and queries with different types of arguments, so the code is re-used and there is no need to build another query for each report;
- The amount of joins is reduced; the functions avoid the joins by searching the values in the fact table;
- Soft parsing is used for the function's query execution instead of hard parsing in the case of another SQL query.

The main *disadvantage* of the model is that the function needs to open a cursor to execute the query which can lead to an increase of PGA resources if the fact table is too large. But the execution cost is insignificant and does not require a full table scan if an index is used on the corresponding foreign key attribute.

Through an ETL (extract, transform and load) process data is loaded into the object tables from the transactional tables of the ERP organizational system. This process can be implemented also through object types' methods or separately, as PL/SQL packages. Our recommendation is that the ETL process should be implemented separately from the object oriented implementation in order to assure the independency of the MD model.

4 Conclusions

The virtual data warehouse is based on a set of objects like views, packages and program units that extracts, joins and aggregates rows from the ERP system's database. In order to develop a BI system we have to build analytical reports based on this virtual data warehouse. But the performance of the whole system can be affected by the data extraction process which is the major time and cost consuming job. A possible solution is to apply the optimization techniques that can improve the performance. Some of these techniques are presented in this paper. The results that we've obtained are relevant for decreasing the execution cost. Also, for developing BI reports an important option is to choose analytic functions for predictions, subtotals over current period, classifications and ratings. Another issue discussed was the OO implementation which is very flexible and offer a very good representation of the business aspects that are essential for BI projects. Classes like dimensions or fact tables can be implemented together with the attributes and methods, which can be a very good and efficient practice concerning both the performance and business modelling.

References

[1] Oracle Corporation - *Database Performance Tuning Guide 10g Release 2 (10.2),* Part Number B14211-01, 2005

[2] Ion Lungu, Manole Velicanu, Adela Bâra, Vlad Diaconita, Iuliana Botha – Practices for designing and improving data extraction in a virtual data warehouses project, Proceedings of ICCCC 2008, International conference on computers, communications and control, Baile Felix, Oradea, Romania, 15-17 May 2008, pag 369-375, published in International Journal of Computers, Communications and Control, volume 3, 2008, ISSN 1841-9836

[3] Adela Bâra, Ion Lungu, Manole Velicanu, Vlad Diaconița, Iuliana Botha – Improving query performance in virtual data warehouses, WSEAS TRANSACTIONS ON INFORMATION SCIENCE AND APPLICATIONS, May 2008, ISSN: 1790-0832

[4] Ion Lungu, Adela Bara, Anca Fodor - Business Intelligence tools for building the Executive Information Systems, 5thRoEduNet International Conference, Lucian Blaga University, Sibiu, June 2006

[5] Donald K. Burleson, Oracle Tuning, ISBN 0-9744486-2-1, 2006

The Optimization of Algorithms in the Process of Temporal Data Mining Using the Compute Unified Device Architecture

Alexandru PIRJAN
The Bucharest Academy of Economic Studies, Romania
alex@pirjan.com

Considering the importance and usefulness of real time data mining, in recent years the concern of researchers to discover new hardware architectures that can manage and process large volumes of data has increased significantly. In this paper the performance of algorithms for temporal data mining that are implemented in the new Compute Unified Device Architecture (CUDA) from the latest generation of graphics processing units (GPU) will be analyzed and reviewed. The performance will be evaluated taking into account the type of algorithm, data access, the problems` size, the GPU's processor generation, the number of threads processed.
Keywords: *Temporal data mining, MapReduce, CUDA, GPU, Fermi, thread, kernel.*

1 Introduction

Real time data mining will enable scientists to develop researches on a scale that seemed unimaginable until recently. Both hardware architectures and data mining algorithms must properly manage and process huge volumes of data, otherwise data analysis risks becoming irrelevant in certain fields such as that of neuroscience. A possible solution to overcome these difficulties is the development and implementation of new parallel processing algorithms and novel hardware architectures.

New techniques and data mining methods have been developed in recent years, used to discover new patterns, clusters and to classify different types of data. In order to optimize a data mining algorithm one should aim to improve the quality of the data extraction process and to streamline it by reducing the response time.

Parallel hardware architectures have proved to be viable solutions in this respect. Graphics processing units (GPUs) have a real potential in optimizing the data mining process, as they are multithreaded and multicore processing units. Unlike central processing units (CPU's), the cores of a GPU are virtualized at a hardware level and its threads are hardware managed, so the programs' scalability and portability improves substantially. A GPU has a computational capacity and memory bandwidth far beyond than those of a CPU, which help accelerate most of the databases operations and streamline the entire data mining process. These graphics processing units combine hundreds of simplified parallel processing cores, which can be very useful in the data mining process, reducing the necessary time for extracting knowledge from data analysis. This high computational power is currently being used successfully in various scientific fields: image processing, geometric processing and database, overcoming the most powerful CPUs. Another essential aspect is the performance per watt consumed, obtained from the GPU when compared to the CPU processors.

Based on their high performance, low cost and on the increasing number of features offered, GPU processors are powerful tools capable of solving an increasingly wide range of applications. In this paper, the research is focused on the study of the temporal data mining process. This technique is becoming increasingly important and widely used in applications from different fields such as: financial data prediction, telecommunication control, neuroscience, medical data analysis, even if temporal data mining is a relatively new field.

For example, researchers in neuroscience may determine how neurons are connected and related to each other in the human brain.

For this purpose, besides traditional methods, whose main disadvantages are the restricted area of the brain on which you can get information, modern fast methods can be used, which offer real-time images and information. These lead to a series of huge opportunities, patients can be screened, diagnosed and operated by extremely rapid procedures, based on huge GPU processing performance.

In this paper the performance of algorithms for temporal data mining will be analyzed and reviewed considering that the algorithms are implemented on the new Compute Unified Device Architecture (CUDA) from the latest generation of graphics processing units (GPU). The performance will be evaluated taking into account the type of algorithm, data access, the problems` size, the GPU's processor generation and the number of threads processed.

Research shows that GPU processors can provide the desired performance, but it is required to address specific technical issues for each temporal data mining problem. Taking into account the size of the problem and the type of the algorithm implemented on the GPU, one can determine the optimal algorithm, the data access model and the number of threads that are necessary to achieve the desired performance. These results confirm but also contrast with previous research about the temporal data mining, implemented on graphics processors, such as research on MapReduce algorithms [1]. While most papers provide conclusions based on optimum choice of configurations, this article presents some general characterizations that help explain how data mining applications can benefit from the parallel architecture of the latest graphics processors.

2. Compute Unified Device Architecture

For a long time, GPU processors have been used to accelerate graphics rendering on computers. Following the increasing need for improved three-dimensional rendering at a high resolution and a large number of frames per second, the GPU has evolved through specialized architecture, from one-purpose components to multiple purposes complex architectures, able to do much more than just provide video rendering. This development allowed the acceleration of a broad class of applications. The architecture and characteristics of NVIDIA GPUs are summarized in Figure 1.

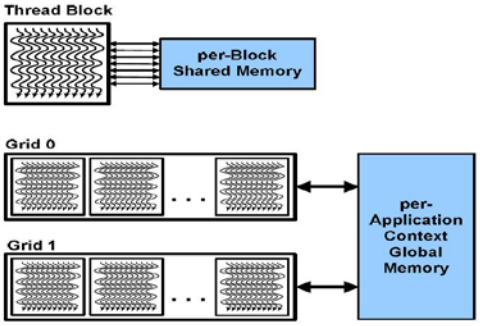


Fig. 1. NVIDIA Compute Unified Device Architecture (CUDA) [2]

CUDA is a software and hardware architecture that allows the NVIDIA graphics processor to execute programs written in C, C++, FORTRAN, OpenCL, Direct Compute and other languages. A CUDA program calls parallel program kernels. A kernel executes in parallel a set of parallel threads. The programmer or compiler organizes these threads into thread blocks and grids of thread blocks. The GPU processor instantiates a kernel program on a grid containing parallel thread blocks. Each thread from the block executes an instance of the kernel and has an unique ID associated to registers, to thread's private memory from the thread block [2].

The Compute Unified Device Architecture hierarchy of threads is mapped to the hierarchy of the graphics processing units hardware processor; a GPU executes one or more kernel grids; a streaming multiprocessor (SM) executes one or more thread blocks; the CUDA cores contained in the SM run the threads within blocks. SM can perform up to 32 groups of threads called warp. Regarding memory hierarchy, each multiprocessor contains a set of 32-bit registry with a zone of shared memory,

which is easily accessible for each core of the multiprocessor but hidden from other multi-processors. Depending on the generation of a GPU, the number of registry and the size of shared memory varies. Besides shared memory, a multiprocessor contains two read-only memory cache, one for texture and another one for constants.

3. The Optimization of Algorithms In the Process Of Temporal Data Mining Using the Compute Unified Device Architecture

When algorithms are developed in the CUDA programming model, the basic concern of developers is to divide the work required in fragments that can be processed by a x number of thread blocks, each containing n threads. For optimum performance, it is recommended that the number of thread blocks matches the number of processors, although the threads within a block will be executed by more cores within a multiprocessor SM. The repartition of tasks to be performed between the x thread blocks is the most important factor in achieving performance.

A single thread block can be considered as equivalent to a PRAM model (parallel random-access-machine) which allows processors to behave arbitrarily asynchronous CRCW (concurrent-read, concurrent-write) [3].

Thus, PRAM algorithms are most efficient at block level [4] and they have to be decomposed into separate kernels because of the need for global synchronization of data flows, synchronization that can be achieved only by successive calls of the kernel.

The technique of data mining through association is an usual method used to discover how certain subsets of elements are associated with other subsets. Temporal data mining is a restricted version of that technique, in which temporal relationships between elements are taken into account.

A specific problem of temporal data mining is the mining of frequent episodes in which we find sequences of frequent items (episodes) appearances in a timed ordered database.

An episode is defined as a partially ordered set of events for consecutive time intervals, embedded in a sequence [4]. The frequent episode mining is defined below [5]:

- $\boldsymbol{D} = \{\boldsymbol{d}_1, \boldsymbol{d}_2, ..., \boldsymbol{d}_n\}$ is a database of ordered items;
- $\boldsymbol{d}_i$ is an element of the alphabet $\boldsymbol{I} = \{\boldsymbol{i}_1, \boldsymbol{i}_2, ..., \boldsymbol{i}_n\}$;
- an episode $\boldsymbol{A}_j$ is a sequence of k elements $< \boldsymbol{i}_{j_1}, \boldsymbol{i}_{j_2}, ..., \boldsymbol{i}_{j_k} >$;
- it is defined an appearance in the database $\boldsymbol{D}$ of the episode $\boldsymbol{A}_j$ if there is a sequence of indices $< \boldsymbol{r}_1, \boldsymbol{r}_2, ..., \boldsymbol{r}_k >$ in ascending order so that $\boldsymbol{i}_{j_1} = \boldsymbol{d}_{r_1}, \boldsymbol{i}_{j_2} = \boldsymbol{d}_{r_2}, ..., \boldsymbol{i}_{j_k} = \boldsymbol{d}_{r_k}$;
- the total number of appearances of $\boldsymbol{A}_j$ in $\boldsymbol{D}$ is called the count of an episode, $\boldsymbol{Number}(\boldsymbol{A}_j)$;
- the purpose of frequent episodes mining is to find all episodes $\boldsymbol{A}_j$ for which $\boldsymbol{Number}(\boldsymbol{A}_j)/\mathrm{n}$ is greater than a threshold α.

In the following, the standard algorithm for frequent episodes mining is presented.

- Input: the threshold α and the sequential database $\boldsymbol{D} = \{\boldsymbol{d}_1, \boldsymbol{d}_2, ..., \boldsymbol{d}_n\}$;
- Output: the set of frequent episodes $\boldsymbol{A} = \boldsymbol{A}_1, \boldsymbol{A}_2, ..., \boldsymbol{A}_m$;
- Stages:

1. *on generate candidate episode for each level*

$$k \leftarrow 1, S \leftarrow \phi$$

$$level\ k \leftarrow 1, A'_k \leftarrow \{\{i_1\},\{i_2\},...,\{i_m\}\})$$

2. *the count of candidate episodes*

while $A' \neq \phi$ ***do***

is calculated $Number(A'_{k_j})$ *for all episodes* A'_{k_j} *of* A'_k

3. *non-frequent episodes are eliminated*

$[Number(A'_j)]/n \leq \alpha$ *from the set* A'_k

4. *frequent episodes are stored in the set* S_A*:*

$$S_A \leftarrow S_A \cup A'_k$$

5. *on generate candidate episode for next level*

$$A \leftarrow A + A', \quad k \leftarrow k + 1$$

end while

6. *it is returned the set* S_A *which contains frequent episodes:*

return S_A

While the elimination phase and generating steps include only the relevant subsets, the counting step may increase exponentially in respect with the size of a subset A_j and the alphabet I. So, the potential number of episodes of length k is $\frac{n!}{(n-k)!}$ for every $k \in \{1,2,...,n\}$.

Run time can be reduced by the use of advanced algorithms and hardware, implemented on parallel processing architectures in order to increase computational power. Although a number of data mining algorithms have already been implemented on graphics processing units, very few are for temporal data mining. An example is presented in [6].

In the following four algorithms based on CUDA programming model [6] will be presented. They are based on the MapReduce programming model (which will be presented below) and for each of them some kind of parallelism is implemented. In Algorithm 1, each thread is looking for a single episode using data stored in graphics memory. In Algorithm 2, each thread is looking for a single episode, but uses shared memory to create first a data buffer. In Algorithm 3, threads in a block are looking for the same episode, but different blocks are looking for different episodes using data from the graphic card memory. In Algorithm 4, threads in a block are looking for the same episode and different blocks are looking for different episodes using shared memory to create first a data buffer.

MapReduce is a software framework developed and implemented by Google [1], which provides programmers the necessary means to process large sets of data using large parallel systems. It is not necessary for the programmers, which use the MapReduce framework to have advanced knowledge in the field of parallel systems. In achieving real-time data mining, the ability to process data sets in parallel is extremely useful.

The MapReduce algorithm uses two functions: "map" and "reduce." The first one, the function "map" applies to a set of input, which consists of a key/value pair in order to create a set of pairs of intermediate key/value. The function "reduce" applies to all pairs of intermediate key/value containing the same key intermediate to produce a set of outputs. Each of the two functions "map" and "reduce" can be parallel executed in order to use the available resources in large data centers (Figure 2).

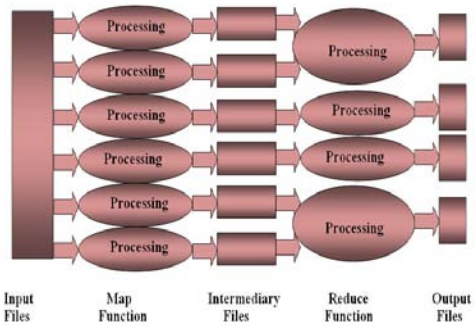


Fig 2. The parallelism in MapReduce algorithm.

MapReduce was originally developed and optimized by Google to run on its private computer data centers. Currently there are different versions of the MapReduce framework for multicore processors, for the Cell processor and for graphics processing units as well. Achieving high performance in these frameworks is quite difficult.

The four algorithms analyzed in this article follow the MapReduce programming model to efficiently benefit from the parallel processing advantage. The "map" function returns the number of appearances of an A_j within a database D. The "reduce" is applied differently, considering if parallelism of threads or parallelism of thread blocks is used.

The first two algorithms implement thread level parallelism to assign a thread for searching an episode A_j in the database D. Using a thread for searching an episode makes the “reduce” function to become the identical application, which returns the value given by the “map” function itself as an output.

Algorithm 1 (without buffering). As each thread will scan the entire database, the first algorithm places the database in the texture memory, so each thread can use the high bandwidth of the GPU. Consequently, threads are allocated in thread blocks one by one until the maximum number of threads per block is reached. For example, if the maximum number of threads per block is 256, then threads from 1 to 256 are allocated to the first block of threads, those from 257 to 512 correspond to the second block of threads and so on until all threads have been used.

Algorithm 2 (with buffering). The second algorithm also uses thread level parallelism, but instead using the texture memory, this algorithm loads a block of data from the database into a buffer of shared memory, processes data from the buffer, then loads another block of data in the buffer and the process is repeated until the entire database has been processed. Thread allocation within the thread blocks is achieved in the same way as in Algorithm 1.

The Compute Unified Device Architecture programming model implements also a block level parallelism. The two algorithms assign a block of threads to find an episode. Within a block, threads collaborate in searching so every thread is looking in a portion of the database.

Algorithm 3 (without buffering). Similar to Algorithm 1, threads within each block access data through the texture memory. Unlike the first algorithm, threads within a block are starting at different positions within the database, while threads with the same ID from different blocks are starting from the same position.

Algorithm 4 (with buffering). The fourth algorithm analyzed in this article uses block-level parallelism with shared memory database buffering. The starting point for each thread of Algorithm 4 depends on buffer size and not on the size of the database (as in Algorithm 3). A thread will always access the same shared memory area during all searches, but data from the shared memory will change when buffer updates.

4. Experimental results

In order to analyze the performance of implementing the characteristics of MapReduce algorithms within the graphics processing units, the various existing NVIDIA CUDA properties should be taken into account.

Table 1. Characteristics of graphics cards used.

Graphics Card	8800 GTS 512	9800 GX2	GTX 280	GTX 480
GPU	G92	2xG92	GT280	GF100
Memory (MB)	512	2x512	1024	1536
Memory Bandwidth (GBps)	57.6	2x64	141.7	177.4
Multiprocessors	16	16	30	48
Cores	128	128	240	480
Processor Clock (MHz)	1625	1500	1296	1401

Compute Capability	1.1	1.1	1.4	2
Registers per Multiprocessor	8196	8196	16384	32768
Registers per thread	10	10	16	21
Threads per Block (Max)	512	512	512	512
Active Threads per Multiprocessor (Max)	768	768	1024	1536
Active Warps per Multiprocessor (Max)	24	24	32	48

For this purpose four different NVIDIA GeForce graphics cards have been subjected to a series of tests. These cards were chosen to represent the latest developed technologies nowadays. A brief description of the chosen graphics cards is presented in Table 1.

The most relevant experimental results on the performance of algorithms (presented in the previous section), implemented on CUDA architecture, running on the four graphics cards in Table 1, are presented below [6], [7]. This article aims to study, compare and analyze these results, highlighting the specific characteristics of each selection.

In tests the following configuration has been used: E4500 Intel Core2 Duo operating at 2.2 GHz with 4 GB (2x2GB) of 200 MHz DDR2 SDRAM (DDR2-800). Programming and access to the GPUs used the CUDA toolkit and SDK 2.0 with NVIDIA driver version 197.75. In addition, all processes related to graphical user interface have been disabled to reduce the external traffic to the GPU.

The experimental results and interpretations on the performance of algorithms mentioned above, using different graphics cards for episodes at different levels with different numbers of threads per block are presented below.

At the L level of an episode, an algorithm searches an episode of length L. In the considered cases, L can be 1, 2 or 3.

The alphabet in which the searching is performed consists of capital English alphabet letters and the database contains 393,019 letters. Different scenarios have been chosen: the first level contained 26 episodes, level 2 contained 650 episodes and level three contained 15,600 episodes [6].

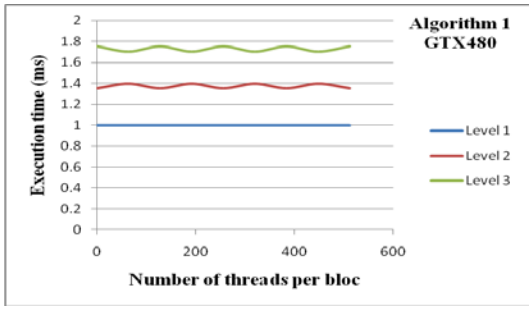
A test consists of selecting an episode's level, an algorithm, a graphics card and the block size. The execution period (measured in milliseconds) is considered the period of time between the moment when the kernel is invoked and the moment when it returns the answer.

Although the GPU's access to graphics has been limited by disabling all non-essential services, each test was performed ten times and the average time obtained during the tests was calculated.

In the following some characteristics that result from these tests on the three criteria (the chosen level, the algorithm and graphics card used) and their impact on the execution time are presented.

The effect of level selection on execution time

To assess the impact of the problem's size on execution time, a series of tests have been done, in which the hardware and the algorithm remained stable and the level L was varied. Because the number of episodes that must be searched increases exponentially when L increases (as noted earlier), the scalability of an algorithm regarding the problem's size is important (Figure 3).



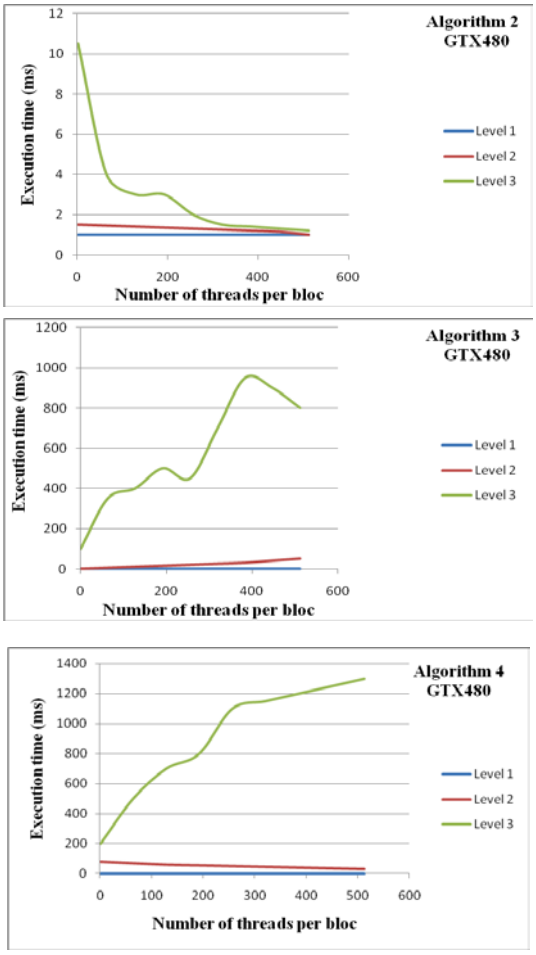


Fig. 3. The effect of level selection on execution time.

a) Parallel thread algorithms provide constant time per episode. This is the case of Algorithms 1 and 2. Regardless of the number of searches, the time required to complete each individual search is essentially the same. A search for each episode is completely independent of other searches and every search is assigned to a thread. The complexity of searching a single episode in a set of data remains constant regardless of the chosen level. For this reason, the execution time is spent entirely for the execution of the "map" function across the entire database. Since these algorithms require a constant time per search, if we consider the parallel processing capability, one can observe that time remains constant even if a large number of searches are executed via the graphics processing unit. Even if there are 30 or 700 or thousands of searches, the process requires the same period.

b) The increasing time in parallel thread processing caused by buffering may be amortized. Algorithm 2 uses a buffer zone to combine the memory bandwidth of all threads in a memory block to reduce the texture load. This implies a long execution time because only one block can be resident on a multiprocessor at a time during the loading phase and other processing cannot be done. As more threads are added to a block, the execution time for Algorithm 2 decreases exponentially. This feature shows that Algorithm 2 is able to use the processing power of a large number of threads. As the number of threads increases, more results will be quickly calculated since all threads can access the shared memory block without additional resource consumption (until the moment when planning a large number of processes on the multiprocessor exceeds the total calculation time).

c) The execution time increases along with the number of threads, when using Algorithm 4 and the 3rd level. Unlike Algorithm 2, Algorithms 3 and 4 lose performance per episode as they increase the number of threads per block and the episode's length. Studying the experimental results, we can observe an increase in the execution time along with the number of threads when using Algorithm 4 and the level 3. Therefore, the execution time increases when switching from the first to the second level and from the second to the third. These two trends are due to the complexity of finding the episodes and due to the increased resources consumption

when loading more blocks that can be active simultaneously on the graphics card.

4.2. The effect of algorithm selection on execution time

It is very important that the chosen algorithm matches the size of the considered problem. However, a programmer often wants to solve a problem of a certain size but has access only to a certain type of hardware. The programmer can modify only the algorithm and the number of threads that he uses within this algorithm. In this situation, he will want to use the fastest algorithm for the problem. Tests were done on GTX480 graphics card, because from all the tested cards, it has the highest computational capability (Figure 4).

d) One thread per episode is not enough for small problems (L = 1). When small lower levels problems are assessed, there are not enough episodes to generate sufficient threads to use the graphic card's resources. As the number of episodes is fixed and there is just one thread per episode, in the case of Algorithms 1 and 2 one can observe the tendency to increase the execution time along with the number of threads. Therefore, for these algorithms, the search time slowly decreases. Algorithm 4 on GTX480 obtains a search time of a milliseconds order. Therefore, it is noted that when using the GTX480, real-time data mining can be achieved and this becomes a certainty for servers incorporating more of these parallel cards and for future GPU architectures, when dealing with significant size databases.

e) Block level depends on its size for medium size problems (L = 2). Algorithms 1 and 2 describe the number of blocks while the number of threads per block increases, because there are a fixed number of episodes and thus a fixed number of threads, so the number of blocks changes in the same time with the number of threads per block. At the second Level, the number of blocks will vary depending on the number of threads per block.

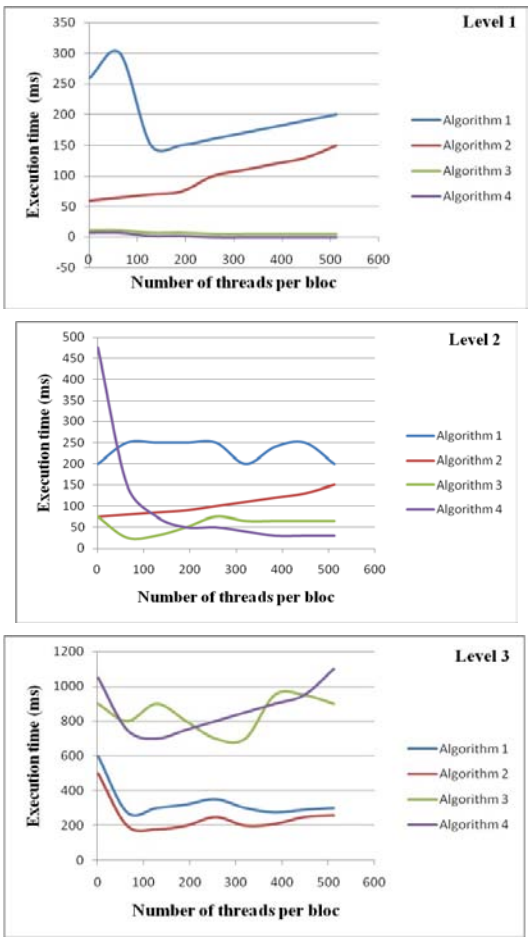


Figure 4. The effect of algorithm selection on execution time

f) Thread level parallelism is enough for large problems (L = 3). The graphic card used, GTX480 has 48 microprocessors, a maximum number of 1,536 active threads per multiprocessor and a total of 73,728 active threads available. For L = 3, there are 25,230 episodes to search. Parallel thread processing algorithms (Algorithms 1 and 2) are much faster than block-level algorithms (Algorithms 3 and 4) because the Algorithms 1 and 2 have to search simultaneously for more episodes than Algorithms 3 and 4 for a certain number of threads per block. Algorithms 3 and 4 are limited to 384 episodes that can be searched due to the limitation of eight blocks per each of the 48 available multi-processors in GTX480 and each block is searching for a single episode. Algorithms 1

and 2 may search more episodes because each thread within a block will look for one episode. Practical resources used by each thread and the available resources per each multiprocessor determine the number of active episodes for Algorithms 1 and 2.

4.3. The effect of graphics card selection on execution time

The hardware configuration is one of the major influencing factors of the algorithms' performance (Figure 5).

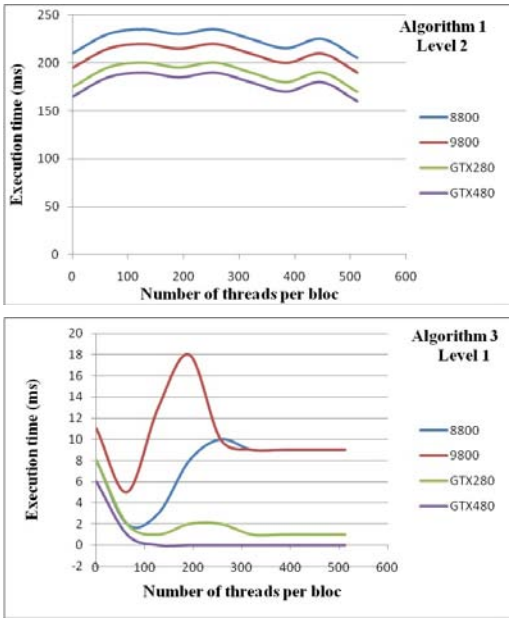


Fig. 5. The effect of graphics card selection on execution time.

g) The frequency of CUDA processors implemented in GPU influence performance for small and medium problems. Algorithms 1 and 2 depend heavily on the CUDA processing cores frequency for small or medium size problems. The frequencies of the four graphic cards in question are 1401 MHz (for GTX480), 1296 MHz (for GTX280), 1500 MHz (for 9800GX2) and 1625 MHz (for 8800 GTS512). For the first two levels, there are no important differences, but starting from the third level, the 480 cores of the GTX480 significantly exceed those 256 of GX2 and those 128 of 8800GTX.

h) Block-level algorithms are affected by memory bandwidth. Algorithm 3 is considerably affected by the memory requirements when large sets of data are processed. The total number of threads that require memory is given by the total number of episodes related to the total number of threads in a block. The algorithm needs to store a large number of threads per multiprocessor over a long period of time and this produces a huge memory consumption. GTX480 gets the highest performance due to the 177.4 Gbps bandwidth followed by GTX280 with a bandwidth of 141 Gbps.

5. Conclusions

In this article, we analyzed and compared temporal data mining algorithms implemented on the latest NVIDIA CUDA architectures. As expected, the best execution time for the analyzed algorithms is the one obtained on the latest architecture, Fermi, that was released by NVIDIA on March 26, 2010. As experimental results outlined, an implementation based on the MapReduce framework must dynamically adapt the type and parallelism level in order to obtain an increased performance.

In order to design efficient temporal data mining algorithms implemented on CUDA parallel processing architectures, one must take into account the eight criteria mentioned above:

- parallel thread algorithms provide constant time per episode;
- the increasing time in parallel thread processing caused by buffering may be amortized;
- the execution time increases along with the number of threads, when using Algorithm 4 and the 3rd level;
- one thread per episode is not enough for small problems (L = 1);
- block level depends on its size for medium size problems (L = 2);

- thread level parallelism is enough for large problems (L = 3);
- the frequency of CUDA processors implemented in GPU influence performance for small and medium problems;
- block-level algorithms are affected by memory bandwidth.

There are many difficulties regarding the practical implementation of data mining algorithms on a GPU architecture. A CUDA programmer must have thorough knowledge of how threads work and how thread blocks are mapped, must know in detail six different areas of memory and especially inter-threading communication. Software development for the CUDA architecture began to be facilitated by new development environments such as NEXUS, but programmers are still forced to write source code for low-level resources and kernels control for each processing operation that is implemented on the GPU, which requires a large amount of time.

There are also other limitations on the performance of data mining algorithms described above. Most important of them are the limitations in memory size and the transfer time between the GPU and the memory. Current NVIDIA cards support memory sizes up to 6 GB, the size being extended from 4GB with the launch of the new Fermi architecture, but even this is far below from the required size when it comes to huge dimensions data warehouses. Transfer of memory blocks between the CPU and GPU still consume a considerable amount of execution time which influences the performance when applying temporal data mining algorithms.

Although the results offer a much improved performance compared to conventional architectures based on CPUs and a tremendous potential for improving the performance of temporal data mining process, there are hardware issues that have obviously limited the implemented algorithms' performance, limitations that can be overcome since the new Fermi hardware architecture has been launched. An important limitation that has a direct impact on algorithms runtime performance happens when dealing with dynamically accessed arrays (which cannot be accessed directly by an index at compile time). Dynamically accessed arrays are automatically stored in local memory and cannot be stored in the registry memory within the CUDA programming model. Since local memory is an abstraction that refers to memory in the scope of a single thread, it has the same latency time as global memory of GPU and is up to about 140 times slower than registry memory [8]. In the above presented algorithms, this type of arrays addressing is frequently needed and the fact that the registry memory cannot be used is a significant restriction.

Also, certain functions in CUDA, such as "atomicAdd ()" are implemented only for integer values. The support for other types of data would facilitate communication between thread blocks.

Considering the possibilities offered by CUDA (depicted in the official documentation "Official CUDA Programming Guide" [2]) the limitation of memory can be managed in two ways. The first option is the memory paging technique used in the algorithms above, successively moving portions of memory and then processing them. Another way to manage memory limitation is to use the CUDA direct access memory option, "zero-copy ".

Besides the fact that the bandwidth available for this technique is very low, the memory should be declared "pinned", thus allowing the memory pages to be maintained in real memory all the time. In practice, both the GPU and the operating system have limits concerning the pinned memory that is under 4 GB and thus makes this method less effective than the paging one.

Fermi, the new generation of NVIDIA architecture, can overcome all the limitations mentioned above. Even when writing this article significant efforts are

made to develop the CUDA programming environment to provide the necessary facilities for the typical programmer. An unified memory hierarchy address space makes it possible to run genuine C++ code on Fermi GPUs. Dynamic arrays can also be accessed in registry memory. Enhancements made in this new architecture allow improved execution times for the algorithms.

The small size of the memory supported by the GPU is a significant limitation of the hardware, which is indeed increased but the 6GB is still insufficient considering that in practice many databases` sizes are of the order of terabytes or even petabytes. TESLA products based on the new Fermi architecture use 40 bits of space address and thus allow addressing of up to a terabyte of memory, but until the first benchmarks will be available, it remains only a theoretical statement. Real time temporal data mining becomes slowly but surely a reality.

References

[1] J. Dean, S. Ghemawat - *MapReduce: Simplified Data Processing on Large Clusters*, OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, 2004.

[2] NVIDIA CUDA Compute Unified Device Architecture - Programming Guide, Version 3.1, 2010.

[3] C. Martel, R. Subramonian, A. Park http://portal.acm.org/author_page.cfm?id=81363604006&coll=GUIDE&dl=GUIDE&trk=0&CFID=89531225&CFTOKEN=84483493 - *Asynchronous PRAMs are (almost) as good as synchronous PRAMs*, Proceedings of the 31st Annual Symposium on Foundations of Computer Science, Pages: 590-599 vol.2, 1990.

[4] N. Satish, M. Harris, M. Garland - *Designing Efficient Sorting Algorithms for Manycore GPUs*, Proc. 23rd IEEE International Parallel and Distributed Processing Symposium, 2009.

[5] R. Agrawal, T. Imielinski, A. N. Swami - *Mining Association Rules between Sets of Items in Large Databases*, Proc. ACM SIGMOD International Conference on Management of Data, 1993.

[6] J. Archuleta, Y. Cao, W. Feng, T. Scogland - *Multi-Dimensional Characterization of Temporal Data Mining on Graphics Processors*, Technical Report TR-09-01, Computer Science, Virginia Tech, 2009.

[7] W. Fang, K. Lau, M. Lu, X. Xiao, C.Lam, P. Yang Yang, B. He1, Q. Luo, P.Sander, K. Yang - *Parallel Data Mining on Graphics Processors*, Technical Report HKUSTCS0807, 2008.

[8] P. Bakkum, K. Skadron - *Accelerating SQL Database Operations on a GPU with CUDA*, Vol. 425, Proc. of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, pg. 94-103, 2010.