DEFINITION OF CROSS-DOMAIN INDEXES AND ORDERING FUNCTIONS IN RELATIONAL ALGEBRA AND ITS USAGE IN RELATIONAL DATABASE MANAGEMENT SYSTEMS

Ph. D. Thesis

Paulo Jorge Gonçalves Pinto

This thesis is submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Software Technology Research Laboratory
De Montfort University
2010

Declaration

I declare that the work described in this thesis is original work undertaken by me between February 2004 and July 2010 for the degree of Doctor of Philosophy, at the Software Technology Research Laboratory (STRL), De Montfort University, United Kingdom. It is submitted for the degree of Doctor of Philosophy. Apart from this degree, no other academic degree or award was applied for based on this work.

Publications

Two articles extracted from this thesis were published at University Lusíada of Lisbon (Lusíada: Economia & Empresa, number 10 Series 2, ISSN: 1645-6750, August 2010).

P. Pinto, H. Yang, “Definition of sort function in relations and its usage in Relational Database Management Systems”

P. Pinto, H. Yang, “Foreign Keys and Multi-domain indexing”

Acknowledgments

The researcher wishes to thank a group of people that could make this thesis possible.

Among these people are, in first place for their support, the researcher's whole family because they had patiently stood some bad humours and some lack of personal availability whilst doing this research.

Following the family are all the researcher's working colleagues for their patience and some tips in benefit of the present research.

Next and as important as all the previous is all the DMU personnel involved in the researcher's travels from Portugal to the United Kingdom and back, namely, DMU Housing, all the S.T.R.L. personnel.

At last, not for being less important but because the researcher wishes to make a special acknowledgment, the first supervisor, Professor Hongji Yang mainly for accepting the researcher's ideas and believing in the outcome of this research since the first moment.

Definition of cross-domain indexes and ordering functions in relational algebra and its usage in Relational Database Management Systems

Abstract

In this thesis, a mathematical model that describes a “Unique Constraint Domain” is defined. Following, the “Ordered Unique Constraint Domain” is also mathematically defined. With those definitions, a cross-domain ordering is also defined.

Then it is shown that relationships between tables in a Relational Database Management System can be defined in other forms than the usual ways, using cross-domain indexes, based in cross-domain ordering. It is shown that all foreign keys in a database can be transformed in indexes with the benefit of speeding data access. It is also shown that this technique is consistent with actual modelling techniques.

It is shown how the index structure, with indexes defined as functions, can provide support for relationship roles. In addition, it is also shown how this can provide support for more than two tables in one relationship and for supporting special sorting order. The addition of a mathematical function to a relation that could sort that relation, demonstrating that the closure property of relations are still kept, shows that this mathematical model can be used as extension of the base relational model.

Next, it is shown that with this new technique, commercial database engines should not degrade performance because all supporting structures are already present and, in some cases, a better performance might be achieved.

Code for a prototype based in a Commercial Database Engine has been added, as an annexe, to show how this new technique can be used.

Finally, future work can be done in this area considering that objects other than text and number need to be sorted (e.g. images, videos, audio data) developing new ways to find semantics to define order.

The major contribute is the mathematical definition of the “Unique Constraint Domain” and the “Ordered Unique Constraint Domain” since they are mathematical models for candidate and primary keys.

Table of Contents

List of Figures

Listings

Chapter 1

Introduction

When we look to raw data, no matter where it had come from a computer or it was written in a piece of paper, we always have a first row and a last row [37] (unless, of course, the list is empty, but then again, in this case, we have no data to look to). The order that data is presented to us might or might not be relevant, but in the case when it is relevant, we only have low-level implementation of database engines to deal with ordering. There is little support, theoretically speaking, for ordering data.

One of the most important things in Database Engines is to find data. However, unless the data is sorted how can we find anything?

One way (which is very time consuming) is doing a full scan until we find what we want, but, if we are dealing with large databanks, scanning all data might not be feasible, at least, in useful time.

Improving how order should be treated, and providing mechanisms to achieve independence between the Database Engines and how to order data should be done, could be a good step to provide better relational engines with higher compatibility levels.

In a scenario where all data ordering could be done by a defined set of rules can, in the same way that SQL is a standard for language, software evolution could be made easier with a larger set of standards to stand on.

Some authors already have addressed this issue, but from different points of view. One of the approaches [37] is a pure SQL approach disregarding the theoretical aspects of ordering. Another one [33] is a pure theory approach disregarding some particular aspects involving data ordering. Although both

works are very important, they both seem incomplete when the whole scenario of ordering is taken in account.

With these grounds, it might seem easy to pick up both approaches and “tie” them together to make a sound and complete theory that could define, once for all, how order should be made.

Unfortunately, it is not so simple, since there are some loose ends that must be defined in first place. As a simple example, ordering domains is not the same as ordering data (although every piece of data belongs to a domain) and on the other side, ordering data does not always means that the domain is previously ordered (such as random order).

The main purpose of this research is to make clear that order in relations is not a trivial matter, since order can make any relation meaningful. Besides that, since order can be viewed as a property of a given relation, we can indeed prove that the mathematical structure composed by a relation and a given sort order is different from the same relation with another sort order.

The sort order, later in this document, shall be defined through a function. This function acts as a black box, taking as sole argument the relation and producing an ordered relation as output.

This approach will allow that “ordering functions” could be implemented by database engine and used to sort any relation that meets the requirements for that particular function.

Order has been disregarded since the very beginning of the relational model, as defined by Codd [8] mainly because the abstract relation didn’t need any sorting and because every field on the databases (by then) were textual or numeric (as seen by the examples provided in those documents).

But with the success of relational databases, end users wanted to store everything inside those databases. Therefore, they started to store pictures, videos, audio and many other objects. At first the objects were found by means of textual or numeric data associated with them, as for instance, find an employee given his number and then access to his/her photo, but not looking for the photo itself.

The only way to perform a search in that kind of data was doing a full scan retrieving every piece of data and comparing it to a given document. To search for a matching photo of someone in a photography database, it will be necessary to get every row and thus every photo. Then use some kind of software outside the database engine that, according to a set of parameters, would match both photos looking for a set of common point and decide if it was a match or not.

The database engine only acts as a support for storing and retrieval but not processing. Since the database engines cannot process this data, they are unable to find anything of this kind.

The solution is to provide the engine for a function that applied to that data (and the function must know the data) and return something that the engine knows about: the sort order, meaning, and a sequence of numbers for that data. Then converting any piece of data to a sequence number makes it searchable.

Therefore, this work may also provide grounds to search and find within non-textual data, because, since the function is a black box, it can provide order to the new types of data regardless of the relations where it come from. In this context, since order for any type of data can be defined, search and find becomes trivial.

Nevertheless, this it is the future work to be done when this part of this research is completed.

Chapter 2

Research Methodology

The research area for this thesis was software engineering. This area is very sensitive and therefore very rigorous in aspects such as safety, security, time to solution and cost, among several other minor aspects. The main area of the research falls into the field of constructive research since it began with the observation of case studies and then a new theory was constructed.

The approach for the method that allows the theory to be demonstrated was a typical scientific research technique based on several stages:

The first stage was, based on the observation of cases, the definition of the research question.

After, during the first three years a general and specialized literature review was done, in first place to find out if the research was truly innovative or if there were already works on the subject, and also to provide background on the subject to the researcher.

After the mathematical model that supports the theory was built and tested, proving to explain all the cases in the study and beyond (universality of the theory).

Then a modelling stage followed to achieve a practical usage of the theory, proving that not only it solves the cases but also was technically feasible.

An algorithm based on the theory was built so that all the cases could be tested and the responses to those cases could be evaluated.

After the algorithm was defined, it was necessary to implement it through a prototype. The prototype was then tested against the cases and against other hypothetical situations and proved reliable. The reason for the reliability was the

solidness of the theory. During the coding phase of the prototype, some limitations were defined in order not to jeopardize the timing of the construction and they were referred in the proper place.

Finally, conclusions were taken and, based on those conclusions a future work goals were defined.

Chapter 3

The Research Question

Observing day-by-day database use it can be easily verified that relations are made useful when ordered.

Moreover, sometimes the use of order is mandatory. However, when looking to the relational model as defined by Codd in the early 70's order was not relevant because sets are unordered by definition and a relation, being a set, does not have to be ordered. Therefore, to a relation it is irrelevant how its data is ordered.

"Why order is not defined in an abstract level and only used in practical situations?"

Another questions followed that one:

"Is order important to be defined at an abstract level or the sorting that database engines provide us is sufficient?"

"The knowing of order inside a data collection can or cannot be useful to the underlying meaning of the data?"

"Is order more than just a sequence of data or can it have some more meaning?"

"Can order be defined is some abstract way that can be implemented and used by the database engines?"

And the last one was the origin of this research work.

Chapter 4

General Review

Before starting this section is was stated that there is a great lack of literature in this particular area (database sorting) mainly because ordering is already provided by database engines and it's suitable for most uses, and what was found mainly were cases of uses of ordering within some "special" data. These are countless and of reduced use or even of no use in this research, so they were omitted.

On the other hand, some theoretical subjects stood as they were defined. It is included in this category Codd's original paper and countless papers from that epoch regarding mainly relational calculus, so some of the literature is quite old[1] but, nevertheless, actual.

The following are the documents reviewed that were found with any benefit to this research and in the following section, the most important reading besides the latter will be detailed.

[1] In a matter of fact some of the readings are 30 years old or more, but still actual. One example is a mathematical book [23] whose theory is still applicable.

Hall et al. (1975), in his paper called “Relations and Entities”[24], first treated the fundamental concept of surrogate keys in detail. Surrogate keys are keys in a common sense but they have some specific properties such as they are always simple keys (never composite ones). Their values serve solely as surrogate (hence the name) for entities they stand for, meaning that their values are only for represent the fact that the entity exists and nothing else. Since they carry no additional information, and finally when a new entity is inserted in the database it is given a surrogate key value that has never been used before and will never be used again even if the entity that it represents is deleted.

By definition, surrogate keys are used as primary keys. Since primary keys are implemented as an index, data is fully ordered by the surrogate keys. In addition, if no other order is specified in a simple query involving that data, it will be presented by the order imposed by the surrogate key.

The use of surrogate keys is a relation occurs when a relation does not have any clear candidate keys. In the relational model, all the tuples should make a composite key, but that is not always feasible. The use of surrogate keys can ease the burden of carrying several attributes to implement a foreign key. In this case, a surrogate key, generally hidden from the eyes of the end user, could replace a list of attributes composing a very large key.

A surrogate keys can be the result of an ordering function since the returning values are unique, hence not needed to be defined elsewhere.

Although Codd has defined what a database should be in his first paper (referred later in chapter 5), the concept of Data Model was defined in “Data Models in Database Management” (June 1980)[9]. The main question addressed in this paper is what purposes data models in general and the relational model in particular intended to serve. The whole idea of defining what a database should do *before* it was implemented is the foundation for a good model and Codd, in this paper claims that the relational model was, in fact, the first data model to be defined.

In Date's paper caller "Referential Integrity" (September 1981)[19] introduces the concepts of referential integrity methods, namely CASCADE and RESTRICT addressing the problem with the UPDATE and DELETE operations when changes to a given primary key could bring the database to an inconsistent state for keeping now invalid foreign keys.

This paper, for itself, alerts us for the problem of problematic foreign keys. Since foreign keys are values from valid primary keys residing elsewhere, one must always be aware if a certain foreign key always match its counterpart primary key. Although there are no problem changing foreign keys, we cannot tell the same about the primary keys that are referenced. Primary keys should be handled with care since an update or a delete can cause a fatal error on a reference on a table holding a foreign key for that primary key.

Nowadays most commercial database engines gives full support for referential integrity, such as cascade delete or cascade update, but some of them don't, meaning that any approach that tends to improve how relationships amongst tables should be defined could possibly reduce the number of errors by misplaced foreign key values.

Another approach is, as it will be seen later, to build a structure that could hold both keys identifying both tuples related. Since we are talking about two sets of attributes that makes a primary key on their base tables, we are talking a about a new index structure that could be used to implement relations based on a relation dictionary.

Codd's document called "Domain, Keys and Referential Integrity on Relational Databases" (Spring 1988)[10] discusses the concepts of domain, primary key and foreign key. In Codd's point of view, and despite he is the author of these concepts, he thinks that a great deal of unresolved and unexplained issues are still present. One of the major arguments in favour of the discipline of choosing one candidate key to be the primary key is the need to define an addressing

schema. This discussion is not final because surrogate keys are still matter of further discussion.

As an example of that, Ambler [i11] shows advantages and disadvantages of surrogate keys, such as natural keys having business meaning whereas surrogate keys, by definition do not. On the other hand every attribute with business meaning, such as natural keys, are subject to change in structure over time. Such change in a primary key carries similar changes in all its foreign keys cascading the problem and, in extreme, causing damaging downtime in a database. Surrogate keys, since they have no business meaning are not affected by change on business rules.

Still the advantages of surrogate keys are also approached by Walker [i12], in an on-line article dated 2006, January, where he states that surrogate keys are more stable than natural keys since they are immutable, hence the indexes created are compact. These structures allow simpler join architecture (only one attribute to join) and with faster performance due to the fact that indexes are very compact.

Larson [i13], in 2011, January, also discusses the use of natural keys versus surrogate keys pointing out the advantages and the disadvantages of each approach. There are several comments about this on-line article clearly showing that this discussion is still an issue not settled.

Negri et al, in their book called “Formal Semantics of SQL Queries” (September 1991)[32] said "(…) the semantics of SQL queries are formally defined by stating a set of rules that determine a syntax direct-driven translation of an SQL query to a formal model called Extended Three-Valued Predicate Calculus (E3VPC), which is largely based on well-known mathematical concepts. Rules for transforming a general E3VPC expression to a canonical form are also given:

[in addition,] problems like equivalence analysis of SQL queries are completely solved"[2]

However, SQL used in this document was the dialect defined on the first version of the SQL (SQL/86).

The presence of null values has several problems, starting with it is not contained in any domain of values generally defined for an attribute in a database table, but it might still be used on attributes without the “NOT NULL” constraint.

The null value should not be interpreted as a default value, as, for instance, “a null value on the field PayDate means that the payment is not done” but (in this example) as “the null value in the PayDate means that we don’t know when it was paid if it was paid”. Thus, it would be preferable to define a “not known” value for that domain or, even better, a “non paid date” to achieve the meaning of the first sentence.

When an index is built, the null values constitute a problem since they are not comparable to any other value in the domain because of the simple fact that they do not belong to the domain. This generally means that entries with null values should be discarded from the index, which, in turn, poses another problem, which is the existence of rows without entries in the index.

This poses an awkward situation where null values, on one hand are used as transactional values and on the other hand, they are treated as missing values. Database engines implements the null value according to Codd’s definition [8], but even Codd, later, proposed that the null value defined in SQL-93 should be replaced by two null markers [12]. This approach was not generally accepted because of the complexity added to the treatment of null values.

[2] Quoted from the abstract.

It is generally accepted that a record that contains a null value in a field means that is an object similar to other objects simply with an unknown value in a field. Therefore, although database engines allow null processing, the sole operation that should be allowed over nulls should be the test for nullity.

Date's paper "A Normalization Problem" (1995)[17] shows a simple problem of normalization and uses it to make considerations about database design and explicit constraint declaration. Among other things, it calls one's attention to a "right" database design can seldom be decided based on the normalization principles alone.

The normalization is very important to remove from the database redundancy responsible for a great deal of anomalies in insert, update or delete operations. However, normalization has a great cost in the implementation of a database because the higher the normalization goes, the more tables are constructed each one with a smaller level of redundancy. Nevertheless, to rebuild a document, a greater number of tables should be joined together thru a greater number of foreign keys to obtain the necessary data.

Proposing a new model for treating foreign keys, using indexes, then all we have to do is invoke all the relations needed without having to specify all the connection keys, as nowadays is needed.

Windom's "Active Database Systems: Triggers and Rules for Advanced Database Processing" (1996)[45] is actually a compendium of research and tutorial papers on what the authors call "active database systems", meaning database systems that automatically carry out predetermined actions in response to specific events, triggered procedures as they are called nowadays.

Several descriptions are included for several prototype systems. The book summarizes the relevant aspects of SQL/92, the then called SQL/3 (now ISO/IEC 9075 – SQL[t11]) and certain commercial products like Oracle, Informix and Ingres among others.

In the book “Foundation for Object/Relational Databases: The Third Manifesto” (1998)[13], Date and Darwen detail a rigorous proposal for the future direction of databases and DBMS's. The book shows what should be (in the vision of the authors) a DBMS design and the language interface to it. Some concepts are there defined such as strong types in database fields and, accordingly what operations could be possible with each data types. It is, then, a formal document that proposes a more precise model (although relational) to manage databases (a DBMS).

On this document, the authors defended that SQL should be abolished and replaced by a language (called “D” in the document) that would gradually replace the actual SQL. The main purpose is to eliminate from SQL all the “bad” things it has such as allowing duplicate rows in a table, the use of null and several other things not mentioned here.

They propose some prescriptions, some proscriptions and some “very strong recommendations” prefixed “RM” when they derived from the Relational Model or “OO” (meaning “Other Orthogonal”) when not derived from the Relational Model.

Besides Chen's E-R model [6], the Unified Modelling Language (UML) is yet another graphical notation to support the task of application design and development, as defined by Reed’s book “The Unified Modelling Language Takes Shape” (July 1998)[39]. It can also be used to develop SQL schemas. It is more and more significant nowadays mainly because it has been adopted as a standard by the Object Management Group (OMG) and it is already supported by several commercial products.

UML supports the modelling of both data and processes but it does not say much about integrity constraints. It is more concerned in representing data structures as a snapshot (not in runtime) and a rigorous definition of the processes that manipulate such data. The whole idea of the UML schema is to bind behaviour (processes) to entities and describes them not just as a set of

attributes but as an object. However, in the matter of integrity constraints it still presents a gap.

In a communication for SIGMOD (1976) called “An Architecture for High-Level Language Database Extensions”[21], Date tries to launch an informal approach to a new high level language that could be used with all the major database models at the time (relational, hierarchical and network). His idea was to have a common construct so that any model could transparently use as its own. It was partially focused on the programmer’s view of a database and its constructs and functions could be easily be mapped in the concrete syntax of the most common programming languages at that time.

Although this document it is not applicable nowadays, the concept of fully ordered data for the datasets is still present, and used in this research.

Martin, on “Principles of Data-base Management” [30], a four-part book speaks about database management and the interactions between rational information systems and managers. The first part of the book takes us to a comprehensive set of chapters justifying why corporate databases. The author explains to the reader concepts such as data basics and categories of data usage. He explains, also, some other concepts like flexibility and data independence as a foundation for success corporate databases. Next, he talks about the new view of data and its impact on the corporate information systems and finally he compares operations system against information systems. On the second part, Martin takes us through some logical/physical aspects of data organization. These aspects includes schemas (and subschemas), tree and plex structures, file addressing and searching. He also approaches some information about relational databases and distributed databases. On the third part, the author moves to database software (database management systems). He deals with subjects as types of databases languages such as CODASYS data description language and IBM’s Data Language/I. At last, he enters the chapter of Query Languages, where he discusses some of the query languages existing then with their advantages and disadvantages. On the last part Martin enters several

considerations about management, namely about the roles of the Data Administrator and Database administrators, information quality, security and privacy. Afterwards the author speaks about management information systems and how the management can benefit from it. At last, he writes about the impact of new technologies in old corporate environments. His main concern in this part of the book is the irrational resistance to rational systems, exposing the main motifs for resistance from established circuits inside the corporations that may lead to unsuccessful implementation of new systems. He then ends with a small guide “to success” for new information system, written as a checklist with the “dos” and “don’ts” in an implementation of a rational system in a large corporation.

The concern for sorting relations with duplicate rows (called “multirelations” by Lehman and Kung)[27], approached by Abdelguerfi and Sood in a paper called “Computational Complexity of Sorting and Joining Relations with Duplicates” [1], is important because it can lead to optimizations in database engines. When a projection is made over a relation, in the result, since relations have the closure property [8], is still a relation (and not a multirelation) duplicate rows should be eliminated. Although “the attribute elimination phase of the projection operation is computational inexpensive” the duplicate removal phase is more intensive (according to the authors). In a matter of fact, the removal of duplicates implies that that result must be sorted in order to get all duplicates together and only one of each value could be copied to the output. This problem arrives mainly with foreign keys where the table where they come have a small number of rows and they relate with a great number of records in another table. We can consider, for instance a table classifying customers with, for example, ten rows, related to a customer table with, for example, ten thousand rows. Obviously when we make a projection over the customer table getting the foreign key we will have, at first, a multirelation with thousands of duplicate rows that needed sorting and then elimination.

On this paper, the authors have defined a model where the finding of duplicates is optimized, and could be used to improve query optimizers within database engines.

This problem is also addressed, with another perspective by Paulley and Larson [35]. These researchers defined the misuse of the “DISTINCT” SQL clause, implementing a method to find out if the “DISTINCT” keyword in a particular query affects the results, case when not it will be discarded by the database engine since it is not necessary. The same applies to nested queries with unnecessary “DISTINCT” clauses. These researchers define a set of theorems that allow database engines to analyse an input query and decide if the use of the “DISTINCT” keyword is valid or not.

They also defined that in most cases sub-queries, as the ones specified by the “EXIST” SQL clause can be transformed into joins allowing query optimizers to perform an optimized join operation instead of a different sub-query. The concepts in this document apply, according to the authors, to non-relational databases as well.

In the scope of this research, although we must consider the existence of duplicate values in some projections (needed, for instance, to build a composed primary key), the research main issue is not the engine optimization.

Early books, such as the Delobel’s “Bases de données et systèmes relationnels” [22], were written with the intention of clarify what was a relational database and how conventional programs should interact with them. In the early days, the relational databases were seen for most programmers as just another repository of data. In this book is clearly stated how the DBMS was organized, showing in a very clear way how the three database levels (external, conceptual and internal) interact with each other and how the application programs should interact with them. In another way, it was referred that even with low-level access programs should not override rules enforced by the DBMS.

Christment in his book “Prática de Bases de Dados” (Portuguese edition) [7] compares several database management systems showing how each one works. The author shows how the inverted model, based only in index lists works, showing its strong and weak aspects. This model, entirely based on indexes was poorly performing mainly because for each value was built a list of indexes that, in turn, pointed to data. The author then showed how the hierarchical model worked and how well it behaved when a parent-child access was made and how poorly it behaved when a list of child (with no need for parents) was needed. The author goes on and next it shows how the network model tries to solve what the hierarchical model could not and how it works. Finally, the author approaches the relational model, showing not also the database model but also the mathematical model associated with it.

The technical book “Directions in Database Management Systems: Selection and implementation” [t03] is a comprehensive document specifying how a database management system should behave and lead us to a set of conditions that one should verify in order to use a DBMS adjusted to one’s needs.

On Alagic’s book “Relational Database Technology” [2], the author does not restrict his writings to the relational model but explores also, in comparison, the hierarchical and the network model also. He deeply explains not only the relational model but also the relational algebra showing in a very detailed way all the set operations, the usual and the special ones. Further, he explains in a very mathematical way what are functional dependencies and their types (trivial, transitive, multivalued). The text is thoroughly filled with examples and graphical images showing in a very intuitive way all that was demonstrated in mathematical language on the text. This book is very clear (although it still uses some terms later abandoned such as “time-varying relation”, nowadays replaced by the expression “relvar” meaning “relation variable”, as defined by Date [18]). Another interesting chapter is the one about distributed technology, since nowadays, with the growth of communication technologies, distributed transactions are beginning to take part of everyday’s work. In this chapter, Alagic explains the architecture a distributed system, modeling it as “a set of

nodes and a set of connections among them". All the transaction modules and database modules communicate in the set of connections through a communication module that has the control over the individual transactions on the involving databases. He then explains what Serial Distributed Execution (where every transaction preserves the integrity constraints) and Nonserial Executions are. He explains what Distributed Query Processing is explaining what strategies should be adopted to achieve a distributed query with a minimal cost (as defined for every single transaction). Distributed updating of the database is also approached, in this case with special care because of the possibility of rolling back a transaction that has already finished in one of the nodes.

In Stanczjk book "Theory and Practice of Relational Databases" [42], the author approaches some good practices regarding database design, namely normalization which he explains as a general method to solve the problem of finding better relations that are free of anomalies, such as insertion, update and delete anomalies. He explains in detail, first in mathematical terms then with database samples all the normalization steps, eliminating relation attributes, and reaching first normal form. Moving forward, detecting functional dependencies on parts of composite keys, projecting them into new relations, and so reaching the second normal form. The author skips directly to the Boyce-Codd Normal form (skipping the third normal form) since the Boyce-Codd normal form supersedes the 3rd. He explains that a relation where every determinant is a key is in the Boyce-Codd normal form. Then the author moves towards the higher normal forms considering the multivalue dependencies and, if they are not functional dependencies, they should be decomposed in order to verify that, and reaching the fourth normal form. After, he approaches the fifth normal form, verifying join dependences and decomposing the relation in a set of relations that can verify that there are no join dependencies. In this last normal form, unlike the others, there are no set of axioms that are sound and complete was found, so finding join dependencies in a relation can be a very time consuming task.

The next book (Database: structured techniques for design, performance and management) [3], Atre describes a set of techniques to efficiently design a database. However, he does not rest on the design and he approaches a group of techniques to improve performance on the database design. He then takes us through the “mechanical” problems with database physical implementation such as backup policies, reorganization of the database files, restructuring data in order to improve performance, monitoring performance and tuning the database, such as adding indexes to speed up frequent operations. He also considers approaches to performance regarding transactions with the use and control of checkpoints, in such a way that if a crash occurs in a database a transaction can be resumed from the last checkpoint instead of being replayed from the beginning. He also approaches techniques for accessing data in the physical store, considering different needs (sequential, direct with pointers). His main concern is the optimization of data access through several means.

Date, on his book “A Guide to SQL Standard” [20], presented us with a complete guide to SQL, starting with the Data Definition Language, covering the design of the schema, the DDL keywords and its syntax and approaching themes like constructing views, procedures, modules, and so on, going through the Data Manipulation Language with its keywords and syntax. On the very first chapter the author explains thoroughly “why is SQL important” (in his own words) showing the fact that a standard language for data creation and manipulation can improve communicability among different database systems and reduce training costs because the programmers can adapt their source code to new database engines without significant changes on database access.

In “Data Base Management Systems”[44], Tsichritzis and Lochovsky presents this book as a manual for advanced students and divides it in two parts. The first part (divided in nine chapters) is a more theoretical approach with the study of the relationships between the Information Systems and Database Management Systems. It follows a chapter concerning Data Models (the network data model and the relational data model) then a chapter about Data Languages (Network Selection and Relational Algebra). He then defines what a

Database Management System should present, as facilities, namely the ability of defining schemas (and associated sub-schemas). The next three chapters are a comprehensive study of the most popular (at the time) database systems: the hierarchical, the network and the relational system. The authors concludes this part of the book with several considerations about the database management systems implementation approaching, again, the most popular implementations, i.e., the hierarchical, the network and the relational database management system. They conclude this part with the operational requirements of any Database Management System, namely about security, integrity, concurrency and performance.

The next set of chapters is intended to have a more practical content than the previous, since they are mainly composed with database management systems samples. They use several real database engines to explore and test the samples that were used to explain the theoretical concepts of the first part. The authors use the Information Management System (IMS), a hierarchical system from IBM, the MRI System 2000, also a hierarchical system from MRI Systems Corporation, the Integrated Database Management System (IDMS), a network system from Cullinane Corporation, Total, also a network system from Cincom Systems, Inc. and, at last, Adaptable Data Base System (ADABAS), from Software AG, which uses sets of “flat-files” that can be coupled to each other through a common attribute. The way in which files are linked to each other can define databases according to the hierarchical or network model.

Inmon, in his “Effective Data Base Design” [25], tries to approach the main problems that rise from building database applications, with “recurring pattern of very serious problems”. This book is mainly written for IMS users, hence hierarchical database users, but some types of problems referred in this book, along with some new ones are applicable to the relational model of data. This book was written for a broad audience since the author tries to reach the professional audience, such as managers, data administration personnel, data analysts, application personnel for example, along with academic audience because some topics of interest for that community are addressed such as data

base design, data elasticity and the achievement of flexibility in data base design.

Martin, in his course book “Computer Data-Base Organization” [31], in Part I he takes the reader through the concepts of logical organization of database elements such as schemas and sub-schemas, entities and attributes, data models and DBMS (Database Management Systems). He uses these concepts to integrate them with the purpose of database organization. Afterwards he goes through a set of chapters regarding data structures definition, namely Data Description Languages, CODASYL, IBM’s Data Language/I and relational databases.

Within the field of data structures, the author explains normalization (third normal form) and canonical data structures. These canonical data structures are, in concept, the seed for the conceptual level in a database management system since, according to the author, the data structures should be application independent. He defines a set of rules to help to determinate which attributes depends on what, hence helping the normalization task. The author ends Part I with the varieties of data independence where he points out the advantages and disadvantages of data independence.

In Part II he deals with the physical organization (and differences to logical organization) of the database elements, dealing with matters such as addressing, sequential indexes organization, pointers, chains and ring data structures and tree structures, among other structures. He approaches, through a set of chapters, techniques for searching (index searching), multiple-key retrieval, inverted file systems, data compaction and memory management (virtual memory, virtual memory hierarchies, virtual storage, volatile files and associative memory)

Another source of information was the documentation accompanying the database engines. Several books concerning how commercial (and some non-commercial) engines behaves were very important.

"Programming Microsoft SQL Server 2005" [t02] from Brust, "Inside Microsoft SQL Server 2005: Query Tuning and Optimization" [t04] and "Inside Microsoft SQL Server(TM) 2005: The Storage Engine" [t05] from Delaney, Professional SQL Server 2005 Integration Services (Programmer to Programmer)" [t13] from Knight, and the more recent "Inside Microsoft SQL Server 2008: T-SQL Querying" [t01] from Ben-Gan are technical books that allows us to have a glance at a commercial database engine (Microsoft's SQL Server, versions 2005 and 2008) and mainly are reference books for some particular aspects of that database engine.

In the very some way, "MySQL Cookbook" [t07] and "MySQL (4th Edition)" [t06] from Dubois, "MySQL Stored Procedure Programming" [t08] from Harrison and Feuerstein, "Understanding MySQL Internals" [t17] from Pachev and "MySQL Administrator's Guide and Language Reference (2nd Edition)" [t16] from MySQL AB were mainly reference books for the Open Source Database Engine MySQL, that allowed a very comprehensive look at the low level programming of that engine.

Other engines internals (namely Informix and DB2 from IBM) were also considered as a source of information for this research, namely the "Programming Informix SQL/4GL: A Step-By-Step Approach" [t12] by Kipp, "Informix SQL Reference Library" [t09] by Informix Software, "DB2 Developer's Guide" [t15] by Mullins and "DB2: The Complete Reference" [t14] by Melnyk and Zikopoulos.

Besides the printed resources, on line resources were also considered. Some web sites contained relevant material for this research namely, Microsoft's web site [i01], IEEE web site [i02] (where some of the referenced documents were found), Oracle's web site [i03], Postgres web site [i04] and MySQL web site [i05].

Regarding OODB the ODBMS[i07] website was the root of information and search. Every single link on this site was followed and some documents were retrieved that helped this research. This particular site is dedicated to Object

Oriented Databases Engines and some relevant documentation was found. Since this research is mainly about relational databases and RDBMS, researching deeply in object-oriented databases could be part as future work, since the model defended here can be applied to ODBMS. For the moment, the documentation found in this site was enough.

In the field of online resources, the search engines proved to be a very important resource for finding information. It is not possible to mention every piece of material that was found in the Internet with some interest but not used directly.

This includes tips for developing code when the researcher was developing his model, small snippets of code (quoted whenever possible) and so on.

The database engine manufacturers were particular relevant in order to choose what database engine would be suitable for testing the model. This represented a significant part of the research although the content is not included or referenced elsewhere in this document mainly because they addressed technical issues related with the functioning of the database engine but not directly with the issue addressed with this research.

Chapter 5

Detailed Review

In Codd's base document – "A Relational Model for Large Shared Data Banks" (June 1970)[8] – the relational model is defined, the principles of data independence are enumerated and a high-level language is defined so it can use that data independence. Furthermore, it is demonstrated that the relational model is superior in concern to abstraction than other existing models at that time. In this document is clearly stated that "ordering dependence" should be avoided, meaning that programs should not depend on stored ordering, but in the other hand "order of presentation" is not a dependency and in most systems "(…) fail to make a clear distinction between order of presentation on one hand and stored ordering on the other".

The concept of "normal form" is also defined in such a way that nonsimple domains could be eliminated from the final relations making the whole relational model more flexible since redundancy could be eliminated whilst preserving all data needed to represent some reality.

In this document, operations with relations are defined. Some of these operations – permutation, projection, join, composition, restriction – are the foundation of SQL itself since SQL implements some of these operations as defined here.

In "The Relational Model for Database Management Version 2" (1990)[12], Codd enlarges what was defined in the previous document after about ten years of development of the relational model. In the first document only structure, integrity and manipulation was defined as the result of an approach to deal with a specific problem – definition and manipulation of large databanks. In this book all aspects of the database management system is approached. The book contains eighteen parts (instead of three of the first book) and addresses each

on of them so that a full relational database management can be clearly defined.

The eighteen parts are: Authorization, Basic Operations, Catalogue, Principles of DBMS design, Commands for the DBA, Functions, Integrity, Indicators, Principles of Language Design, Manipulation, Naming, Protection, Qualifiers, Structure, Data Types, Views, Distributed Database, Advanced Operators.

Codd's main concern in both documents was to define the relational model in such way that could be implemented by the industry with strict rules of operation. After the first document, along with some true relational database engines a lot of so called "relational" software was released without been truly relational even after Codd has published his "12 rules" [11] as a guide to truly relational database engines.

This document brought a set of rules that must be abided to a database engine can call itself relational. With this document most of the so called "relational" just disappeared from the market. The ones who did not disappeared have dropped the "relational" on its name.

In this document, and in order to be precise, some practical aspects were omitted such as sorting or indexing data. By 1990 (when this latter document was written) databases mostly contained "textual" data, meaning text and numbers but not multimedia or binary objects. With domains that were subset of ordered domains such as text or numbers, sorting data was never a problem. Indeed data has been sorted on programs even before the definition of the Relation Model. However, with the evolution the lack of theory to sustain order began to appear more clearly, because another type of data is being held in the databases. In addition, that data needed to be found on its own and not by the means of "descriptive fields" that are no more than findable texts connected to "unfindable" data.

A relation (according to Codd's definition) can have any order on the tuples. It is true. But, back then, it was possible without hassle to sort and find data independently from the definition. Today we have more data types and a growing need to order it and to find within it. Therefore, the relational model must be extended to incorporate order but in such a way that the base model holds and the extension does not pervert those principles.

In another fundamental paper – Chen's "The Entity-Relationship Model – Toward a Unified View of Data" (1976)[6] – a data model, called the entity-relationship model, is proposed. This model incorporates some of the important semantic information about the real world. A special diagrammatic technique is introduced as a tool for database design. An example of database design and description using the model and the diagrammatic technique is given. Some implications for data integrity, information retrieval, and data manipulation are discussed. The entity-relationship model can be used as a basis for unification of different views of data: the network model, the relational model, and the entity set model. Semantic ambiguities in these models are analyzed. Possible ways to derive their views of data from the entity-relationship model are presented.

This document is a proposed path for turning the abstract relations of the relational model into practical databases. Although is not in the scope of this research, a possible future work, or even a side paper could propose an adaptation of these diagrammatic techniques to incorporate ordered entities.

Date's "An Introduction to Database Systems – 7th Edition" (2000)[18] is divided in six major parts: Basic Concepts, The Relational Model, Database Design, Transaction Management, Further Topics, and at last, Object and Object/Relational Databases.

Each part, in turn, is divided in several chapters. Part I (Basic Concepts) is an introduction to the database systems concepts in general. Also the relational model as a particular model is approached along with SQL[t11], the standard database language. Part II (The Relational Model) gives a very detailed

description of what is the relational model showing that this model in not only the theoretical foundation for the relational model but for database field in general. Part III (Database Design) discusses the database design, mainly describing the normalization process and, in the last part, the Chen's entity-relationship model[6]. Part IV (Transaction Management) is mainly concerned with the management of transactions, namely recovery and concurrency. Part V (Further Topics) talks about several other aspects of databases technology not found in other chapters, namely, security, distributed databases, temporal data, decision support among others. Part VI (Object and Object/Relational Databases) shows the impact of object technology on database systems (namely on relational database systems). In the last chapter, a rapprochement between object and relational technologies is considered and a object/relational system is discussed. This topic was later approached in another book[13].

Professor Braumann's book "Teoria da Medida e da Probabilidade" (1987)[4] starts with some general considerations and then he moves on to the study of the six main operations concerning sets contained in a common space and the study of some associations made by some of these operations.

Next, the book deals with the main operations concerning sets from different spaces and the deductions of formal properties. The next section of the book starts with the restriction to a sub-space because it will demonstrate some properties that otherwise would be very difficult to demonstrate. The study of the Cartesian product of sets follows and some particular aspects deserves particular attention. The projection operation (among sets) is studied next, having in account the geometric representation of such operation.

The book the follows with the definition of set classes contained in a Ω space. The most important are reviewed namely the algebras and the σ-algebras.

The additive decompositions of a measurable space (Ω,A) are studied next and this chapter ends with the study of the Borel line.

The book continues with several aspects related with the multi-dimension Borel spaces. Chapter III starts with the generic study of measurable functions and continues with the detailed study of such functions and borelian functions.

The book ends with the use of these definitions in the classical study of real and complex numbers.

The definition of algebraic structures is important because relations are sets and since they are sets with the operations defined (such as union and Cartesian product) they prove to be true σ-algebras. In this manner, the book can give a more precise math view over the relational model. The properties found for the algebras, namely the closure property are, in this book, clearly demonstrated.

In “An Extension of the Relational Database Model to Incorporate Ordered Domains” (September 2001)[33], Ng extends the relational data model to incorporate linear orderings into data domains, which he called the ordered relational model. The conventional Functional Dependencies (FDs) are examined in the context of ordered relational databases by using the notion of System Ordering Independence (SOI), which refers to the desirable scenario that the ordering of tuples in a relation is independent of the implementation of the underlying DBMS. The author also extends Armstrong's axiom system for FDs to object relations, which are a subclass of ordered relations that allow him to view tuples as objects. He formally defines Ordered Functional Dependencies (OFDs) for the extended model by means of two possible extensions of domains, pointwise-orderings and lexicographical orderings. He first presents a sound and complete axiom system for OFDs in the case of pointwise-orderings and then establishes a sound and complete set of chase rules for OFDs in the case of lexicographical orderings. His main result shows that the implication problems for both cases of OFDs are decidable, and that it is linear time for the case of pointwise-orderings.

The author does not solve in a simple manner the problem of changing the ordered domain for a given set of attributes (e.g. random ordering) neither the problem of adding new values to a domain along with new data, rebuilding the entire ordered domain.

This approach is an approach to domains and defined how a domain can be ordered. But although data belongs to certain domain, data extracted from a domain can have duplicate values so we need something else to order that data. On the other hand, although abstract, the domain has some kind of order and there must be a definition for that order. To have the same data randomly obtained each time a query is issued it means that we have to change the domain every time. This is never addressed in this document.

Nevertheless, the axiom system developed by the author is a main piece for this research development, since the ordered domains are used to sort data when certain conditions are met.

In another document dated 1998 ("SRQL: Sorted Relation Query Language) [37] the authors start to address the problem that SQL does not properly answers questions such as moving averages because the language can not support "rich class of queries". Therefore, they approach the concept of "sequence" in order to implement ordered relations (at database level) and use these relations seamless with the "unordered" ones. This model great advantage is, according to the authors, "[…] is that queries involving relations and sequences are easier to express". This advantage is, namely, over previous approaches such as the new ADT or EADT.

Although the authors have defined a new set of operators to extend the relational model with a new group of relational operators (Sequence ψ, Shift δ, Shiftall Δ and WindowAggregate ω), this extension only addresses how order can be handled but not how it can be defined. The proposed SQL keyword "SEQUENCE" acts as a variant of the "ORDER BY" SQL clause without really defining in a higher form how order should be created (it has a similar syntax as

the “ORDER BY” keyword). Nevertheless, the proposed handling addresses the manipulation of order, but still keeps open the gap where order should be defined. Now we know how to handle a random order, but we still do not know how to create it.

In another communication (“Sequence query processing” in Proceedings of the ACM SIGMOD Conference on Management of Data) [38] the authors addressed how database engines could be optimized to deal with ordered data thus defining a complex model for query analysis and optimizations but leaving aside how order can be built in the first place.

These authors do not address non-textual data as a source to ordering. Nor the semantic order is addressed. These documents, although they tried to implement new SQL words more functionality, do not address higher issues, such as domain orderings (as opposed to Ng [33]) with its consequent lack of generality. The underlying model might be considered superseded by Ng’s work.

ISO/IEC 9075[t10] defines the SQL language. The scope of the SQL language is the definition of data structure and the operations on data stored in that structure. Parts 1, 2 and 11 encompass the minimum requirements of the language. Other parts define extensions. The asterisk means a group of related documents described next:

ISO/IEC 9075-1:2003 describes the conceptual framework used in other parts of ISO/IEC 9075 to specify the grammar of SQL and the result of processing statements in that language by an SQL-implementation.

ISO/IEC 9075-2:2003 defines the data structures and basic operations on SQL-data. It provides functional capabilities for creating, accessing, maintaining, controlling, and protecting SQL-data. Both static and dynamic variants of the language are proved. In addition to direct invocation, bindings are provided for the programming languages Ada, C, COBOL, FORTRAN, M, Pascal and PL/I.

ISO/IEC 9075-3:2003 defines the structures and functions that may be used to execute statements of the database language SQL from within an application written in a standard programming language in such a way that the functions used are independent of the SQL statements to be executed.

ISO/IEC 9075-4:2003 specifies the syntax and semantics of statements to add a procedural capability to the SQL language in functions and procedures. It includes statements to direct the flow of control, define variables, make assignments and handle exception conditions.

ISO/IEC 9075-9:2003 defines extensions to SQL to support management of external data using foreign-data wrappers and datalink types.

ISO/IEC 9075-10:2003 defines extensions to the SQL language to support embedding of SQL statements into programs written in the Java programming language (Java is a registered trademark of Sun Microsystems, Inc.). In addition, it specifies mechanisms to ensure binary portability of resulting applications.

ISO/IEC 9075-11:2003 specifies an Information Schema and a Definition Schema that describes the structure and integrity constraints of SQL-data, the security and authorization specifications relating to SQL-data and the features supported by an SQL-implementation together with other sizing information.

ISO/IEC 9075-13:2003 specifies the ability to invoke static methods written in the Java programming language as SQL-invoked routines and to use classes defined in the Java programming language as SQL structured user-defined types. (Java is a registered trademark of Sun Microsystems, Inc.)

All the previous documents were revised in 2005 and eight other documents were published with several amendments to its predecessors made. Those documents were named "/Cor 1:2005" after their original name. The final names for these four documents were: ISO/IEC 9075-1:2003/Cor 1:2005, ISO/IEC

9075-2:2003/Cor 1:2005, ISO/IEC 9075-3:2003/Cor 1:2005, ISO/IEC 9075-4:2003/Cor 1:2005, ISO/IEC 9075-9:2003/Cor 1:2005, ISO/IEC 9075-10:2003/Cor 1:2005, ISO/IEC 9075-11:2003/Cor 1:2005 and ISO/IEC 9075-13:2003/Cor 1:2005.

ISO/IEC 9075-14:2006 defines ways in which SQL can be used in conjunction with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database and publishing both XML and conventional SQL-data in XML form. In addition, it provides facilities that permit applications to integrate into their SQL code the use of XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.

All document definitions, except for the correction ones, are the documents abstracts as published by ISO and found on the site http://www.iso.org

Chapter 6

Ordered Unique Constraint Domain (OUCD) Definition

Previous works dealt with data domains that were “partially ordered” [33] but to build a true strict total order we need a fully ordered domain. Although the domain can be or not fully ordered, the data that belongs to that domain must also be fully ordered. This means that it should be possible to extract from a lattice order a subset with total order. We know that in a data domain every value only appears once because the domain holds the possible values for the underlying data [8] and for some domains, it is possible to extract a subset fully ordered. These are the domains chosen for keys.

Therefore, it is possible to have a fully ordered domain but the uniqueness of the data from that domain presented in a relation does not hold. An example of that is a foreign key. Although its domain is fully ordered, (after all it is the primary key somewhere), it may show duplicates on a concrete relation. It is sufficient that the foreign key implements a “one-to-many” relationship to be possible to show up more than once.

To guarantee that data can be fully ordered it is important that each element of the domain only appears once or none. Only with this condition, we can have a fully ordered dataset, or, in other words, we can have a unique constraint over that data.

A “Unique Constraint Domain” (UCD, from this point forward) is a domain that, in a particular case of a given relation, returns only a given value for an attribute just once. The domain itself can be or not a UCD depending on the relation where it is used.

An example is a primary key and its foreign key. On the relation where the attribute is the primary key it is a UCD since each value appears only once (the values are a subset of the domain) whereas the same domain in another table

as a foreign key might appear with duplicates. In the latter, the domain is not a UCD because the values could appear more than once.

Let us consider a Domain D. Let us consider a_d as an element of domain D.

Let us consider a relation R. Let us consider an attribute A_R belonging to the relation R. Consider a value V_a as an occurrence of the element a_d in A_R

The following statement is true for all relations:

$$\forall V_a \in A_R, \exists! \, a_d \in D : V_a = a_d$$

However, the following statement is true only for a UCD in that relation.

$$\forall a_d \in D, (\exists! \, V_a \in A_R : V_a = a_d) \vee (\nexists \, V_a \in A_R : V_a = a_d)$$

That means that all values are unique in that particular relation and can identify any occurrence. It can be stated that all candidate keys belongs to an UCD.

In addition, we can always define order in that UCD, by defining an ordering function (even a random one) case when every a_d has a position (rank) inside the domain (verifying all the properties for strict total orders). In this case, we have an Ordered UCD (OUCD). The properties of such function are addressed in chapter 9.

The rank of each tuple of the UCD should be provided by a function that we will call forward the “ordering function” or simply “function” when the context will allow without any ambiguities. The properties of such function will be addressed in chapter 9.

To be an OUCD, besides being a UCD each a_d must obey to the following rules:

- $a_{d1} < a_{d2}$ and $a_{d2} < a_{d3}$ then $a_{d1} < a_{d3}$ (transitive property)
- $a_{d1} < a_{d2}$ or $a_{d2} < a_{d1}$ (linear property)

Opposite to order sequences, the anti-symmetric property never occurs because a_{d1} is always different from a_{d2}, so it can never be verified.[3]

- If a_{d0} is the smallest element ($\forall a_d \in D, \exists! a_{d0} \in D : a_{d0} < a_d$) then $a_{dn} < a_{d(n+1)}$ for every n. If a_{d0} is the largest element ($\forall a_d \in D, \exists! a_{d0} \in D : a_{d0} > a_d$) then $a_{dn} > a_{d(n+1)}$ for every n

Since D is a domain, it is assumed that there are no equal values because all elements of the domain are different hence they have different values. All elements compared to each other are either lesser or greater. One element can only be equal to itself.

These properties in conjunction with the impossibility of having duplicate values (by the definition of a domain) make the OUCD a fully ordered domain.

Primary keys on tables on databases are attributes that belong to an OUCD, so they can be ordered by their values on its domain.

UCD's and OUCD's can be classified in reducible and irreducible ones. If the domain is composed with more that one composing domains, there might be a smaller set of domains that can still be a UCD (or an OUCD). In this case, we call the UCD reducible. In the decomposition, at least a set of the resulting domains will be a UCD. If it is not possible to decompose without loosing the UCD definition then the UCD is irreducible.

Since the primary key (or any candidate key) of any relation is the "smallest set of attributes the can identify an occurrence" [18], then it is an irreducible UCD. When we combine any domain from any other attribute with that UCD, we create a newer UCD, only this new one is reducible.

Let us demonstrate

[3] The anti-symmetric property states that $a_{d1} < a_{d2}$ and $a_{d2} < a_{d1}$ then $a_{d1} = a_{d2}$

An UCD, as stated, is defined by: $\forall a_d \in D, (\exists! V_a \in A_R : V_a = a_d) \vee (\nexists V_a \in A_R : V_a = a_d)$. Let us assume that the domain D is irreducible.

Let us consider another domain D' with the values $a_{d'}$. The composition of a new domain which we call DD' and it all its values are result of a Cartesian product from each domain and represented as $a_{dd'}$ meaning we have an a_d element from the first domain and an $a_{d'}$ from the second. Since the domain only holds unique values, the domain resulting from a Cartesian product are still unique. Let's assume that the value $a_{dd'}$ is equal the value $a_{ed'}$, meaning that duplicates in the resulting domain could appear. Since $a_{d'}$ is equal in both expressions, the values a_d e a_e must be equal so the statement that $a_{dd'}$ is equal to $a_{ed'}$ holds. Since in each composing domain there are no duplicate values, the only way of a_d be equal to a_e is if they are the same. And if they are the same, there is no possibility of $a_{dd'}$ can be equal do $a_{ed'}$ if a_d is different from a_e.

Is also easy to prove that the same principle occurs on the D' domain, so we consider proved that the result of a Cartesian product over two domains is yet another domain. And if the elements of DD' are unique, any subset holds equally unique values, so any subset of DD' is a data domain. This is what occurs when a composite primary key is made of primary keys, such as a key for an association many to many between relations.

But, do we have always a UCD defined in a relation? The answer is yes. Let us suppose that we could not define a UCD in a relation. The UCD cannot be defined only if there is no set of attributes that could provide a unique combination so the values in the domain would appear only once. If that could not be guaranteed then we cannot guarantee that every row is unique (since there are no set of attributes that provides uniqueness) so it was not a relation, according to the Relational Model definition of a relation [8].

Another important property is that any subset of data from an OUCD is still an OUCD.

Let D be an OUCD. For the domain X to be an OUCD is must be over a relation R and over some of that Relation attributes.

By definition $\forall a_d \in D, (\exists! V_a \in A_R : V_a = a_d) \vee (\nexists V_a \in A_R : V_a = a_d)$, because D is an UCD and $a_{d1} < a_{d2}$ and $a_{d2} < a_{d3}$ then $a_{d1} < a_{d3}$ and also $a_{d1} < a_{d2}$ or $a_{d2} < a_{d1}$ because it is ordered (OUCD).

A subset of an OUCD is also a Domain, because any subset of a set with unique values has also unique values, hence being a domain. We will call that subset D_0.

It is always possible to create a subset in the relation that the values of the attribute A which belongs to the OUCD are all part of the domain D_0, excluding all the other tuples where their values of attribute a does not belong to D_0. This represents a new relation because it is the result of a selection operation over the primitive relation. We will call that relation R_0. Let us consider an attribute A_R belonging to the relation R_0. Consider a value V_a as an occurrence of the element a_d in A_R

The expression $\forall a_d \in D_0, (\exists! V_a \in A_R : V_a = a_d) \vee (\nexists V_a \in A_R : V_a = a_d)$ still holds because every value of the domain is either present once in the values of the attribute A in the relation R_0 or is not present at all.

Since every value of the domain D_0 is also a value of the domain D the properties for each element holds so $a_{d1} < a_{d2}$ and $a_{d2} < a_{d3}$ then $a_{d1} < a_{d3}$ and $a_{d1} < a_{d2}$ or $a_{d2} < a_{d1}$ where a_{d1}, a_{d2} and a_{d3} belongs to the domain D_0.

This proves that the subset D_0 of an OUCD is also an OUCD over a subset of the relation R. Since there were no restrictions how the subset was created, this is valid for any subset of an OUCD.

Also the subset of D defined as $\forall a_d \in D_0, (\exists! V_a \in A_R : V_a = a_d)$ meaning that all values of D_0 exists on data and excluding the elements of D that are not present in data is an UCD. If D is an OCUD then D_0 is also an OUCD.

Therefore, since any subset of the relation created by the selection operator is also a relation, the values the attribute A are also a subset of the values of the attribute A in the primitive relation, hence holding unique values, we can always construct a subset of the original domain D where $\forall a_d \in D_0, (\exists! V_a \in A_R : V_a = a_d)$ holds, this subset is an UCD. If D on the primitive relation was an OUCD then this subset is also, by definition, an OUCD.

Chapter 7

Cross-Domain Ordering Definition

A cross-domain ordering occurs when a composite key from two different domains are merged into a single irreducible OUCD.

Each candidate key (single or composite) has its attributes belonging to, at least, an OUCD, since they are unique. The primary key, by definition, only belongs to a single irreducible OUCD.

If the primary key attributes belonged to more than one OUCD, which would mean that different combinations of the composite attributes would belong to different OUCD (a single attribute must only belong to a single OUCD).

In this case, at least one subset of the key would belong to an OUCD hence being a smaller attribute combination than the primary key that truly is an identifier. This is incompatible with the definition of primary key, which is the “smallest set of attributes that can identify an occurrence” [18], or, as it can be redefined, the “smallest set of attributes that composes an irreducible OUCD”.

When two foreign keys are merged together to compose an irreducible OUCD as, for instance, the key for the association between Authors and Books (an “Author” writes many “Books” and a “Book” might be written by several “Authors”) although in the association relation none of the keys belongs to an OUCD, they constitute a cross-domain ordering.

Now we can define a cross-domain ordering as an OUCD composed by two or more domains. This new OUCD can be irreducible if the domains that compose it are not. If at least one of the domains that compose that OUCD also belongs to another OUCD that would mean it would be a decomposition of that domain making it a reducible OUCD.

This property is useful when we need to extend our order to more than one domain, but we do not want to loose the identity of the relation. We can leave the primary key untouched and build a unique index based on a reducible OUCD that can provide us what order we need.

We can, therefore, use OUCD, building them as indexes, from data and use all the properties of the OUCD to find, locate and use data inside a relation, no matter what kind of data we are using. This is true regardless of the type of data is involved in the definition of the OUCD.

Now we can use Ng definition of OFD's [33] to combine data at the data level, making a truly ordered dataset as a result.

All data is presented in some kind of order [37] so when we look at a dataset, this dataset has an OUCD that orders it.

Formally, $\forall a_{dd'} \in DD', (\exists! V_{ab} \in AB_R : V_{ab} = a_{dd'}) \vee (\nexists V_{ab} \in AB_R : V_{ab} = a_{dd'})$ where $a_{dd'}$ is the composed value of the DD' domain and V_{ab} is a composed value from the composition of attributes A and B belonging the relation R (AB_R)

For DD' to be an OUCD it must hold true the following statements:

- $a_{dd'1} < a_{dd'2}$ and $a_{dd'2} < a_{dd'3}$ then $a_{dd'1} < a_{dd'3}$ (transitive property)
- $a_{dd'1} < a_{dd'2}$ or $a_{dd'2} < a_{dd'1}$ (linear property)
- If $a_{dd'0}$ is the smallest element ($\forall a_{dd'} \in DD', \exists! a_{dd'0} \in DD' : a_{dd'0} \leq a_{dd'}$) then $a_{dd'n} < a_{dd'(n+1)}$ for every n. If $a_{dd'0}$ is the largest element ($\forall a_{dd'} \in DD', \exists! a_{dd'0} \in D : a_{dd'0} \geq a_{dd'}$) then $a_{dd'n} > a_{dd'(n+1)}$ for every n

The way in which we define "$a_{dd'n} < a_{dd'(n+1)}$ for every n" or "$a_{dd'n} > a_{dd'(n+1)}$ for every n" may or may not depend on the base domains of each attribute. This is not relevant since ordering the composed attribute might not be the same as the ordering of the base attributes, if there was any order on the base attributes. It

must be defined on its own. We can assess this just by composing two distinct OUCD simply by switching the base attributes.

For instance, in the Book-Author model building an OUCD with BookID/AuthorID would result in an OUCD different than the one built with AuthorID/BookID. Although both hold the same data, although both are OUCD, the first one represents the list of author for each book and the second implements the list of books per author.

Chapter 8

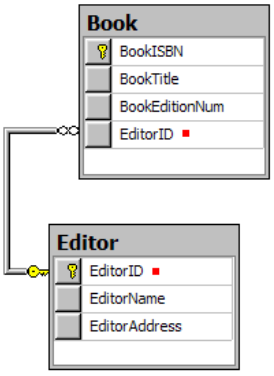
Foreign Keys as Cross-Domain Structures

The Entity-Relation model, as presented by Chen in the early 70's [6], stood as an independent model to represent conceptual entities and the relations amongst them. It is independent because, regardless of the technology employed, it would always be applicable.

In that model, when a relationship was defined, a role for that relation (associated with its counterpart foreign key) could always be specified. The point is that role was never properly defined outside this modelling technique.

To implement the E-R model for an information system, or more precisely, to implement relationships between entities we use foreign keys.

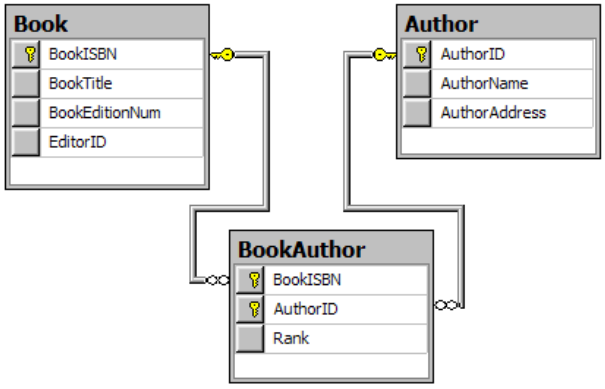
The use of foreign keys, in database design, is widely spread as good practice for implementing relations amongst tables [18][39][30]. However, a foreign key is what is says: the primary key (or even a candidate one) of a table placed as an attribute on another table to enforce a relationship between those two.



Picture 1: A Relationship enforced with a foreign key

Nevertheless, we should only place a foreign key in a table if we have a relation in which each tuple of the target table matches only one of the referred tables.

As we know when we have multiple associations between tuples (for instance in a many to may relationship such as authors and books, in which an author can write many books and a book can be written by several authors) we have to adopt another strategy by building a new table with both keys. In this new table, we have our relation “dictionary” [30], because we have the references to the tuples in the original tables that should match.



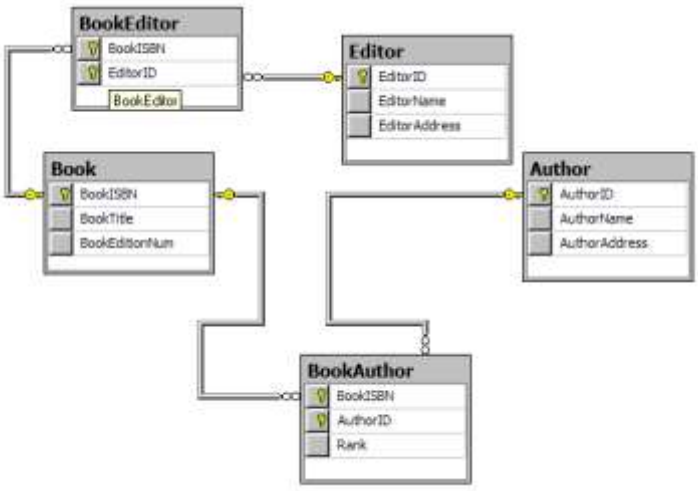
Picture 2: A Relationship enforced with the aid of an additional table

In addition, we can build these “dictionaries” for any kind of relationship. This includes those that we use to create foreign keys directly (such as the costumer id on an invoice).

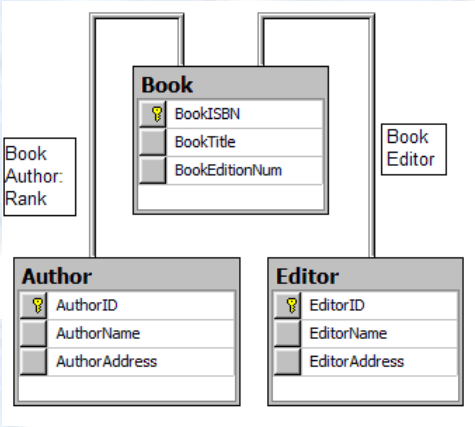
We can think we can lose some performance doing more tables than necessary since we are now using a table for the relation, but what can we surely gain?

The relation table is built with rules, and we can have in a more clear way what rules connect to instances of data together. This would show in a more meaningful way in which data relate amongst them.

We also would eradicate foreign attributes from target tables (no longer an Editor ID in a Book record), so tables could mirror their conceptual counterpart.



Picture 3: Full diagram with no foreign keys in the main tables and with the connections tables (Book Editor and BookAuthor)



Picture 4: Diagram with no connection tables (BookEditor and BookAuthor). The relations now are between the Book and Author as well as the Book and Editor directly. The meaning of the relations is noted beside them. In the Book-Author relation, there is also a relation attribute: Rank. This attribute does not belong neither to the Author table neither to the Book table but to the relation itself.

Sometimes relationships have attributes (for instance the attribute “rank” which is the relative position of an author within the group of authors that wrote the book: 1st, 2nd, and so forth).

The attribute should be declared when the relationship is built and should have the very same rules as an attribute in a base table. It will be provided when a

particular instance of a relationship is made, through the SQL keyword “SET” in the context of that relation[4].

As noted, the relation must have a name and a definition, and, together with it, a set of relationship attributes.

Perhaps the reader is thinking that this can reduce overall performance. We will see that that is not quite true, as it might seem.

These pairs of keys are only pointers to data, so this new structure is no more that a multi-domain index. It points to two pieces of data and can be effectively built as an index. We already have indexes for foreign keys in order to “speed up” the verification of referential integrity, so no extra overhead is required. In addition, if it is built by rules we could rebuild them by applying the very same set of rules we had. In this manner, we could effectively implement the idea of a role in this relationship.

Obviously that some kind of data recording should be done, but this would only be done at the database engine level, not in the conceptual level.

We would have to state our rules in a more precise way, we could have these rules building the relations between data and we can free ourselves out of the foreign keys.

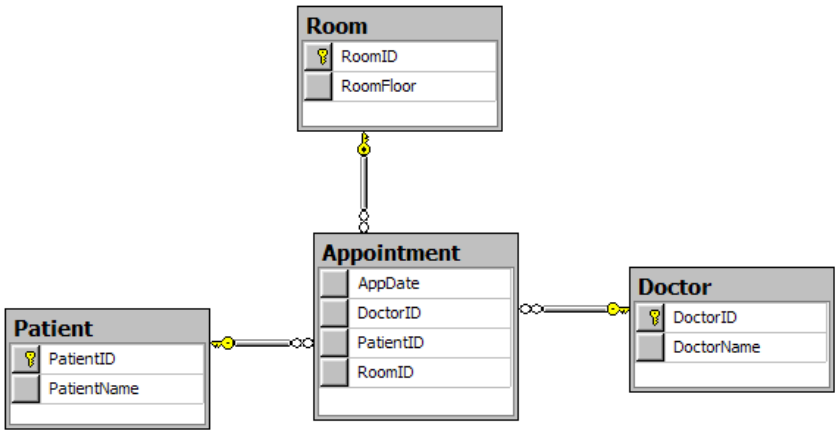
In the present database engines implementations, if we have a one to many relationship and want to change it to a many to many relationship, we have to build the new table, copy data to it, and change every view or stored procedure that accesses that data to accommodate the new table and provide the very some results as before.

[4] The SQL syntax in the context of a relation will be shown later in this document.

If we had a model where all relationships between tables were built with multi-domain indexes, we would only have to change the rules how data can be paired and nothing else.

Besides, there is nothing in this model that prevents the accommodation of three, four or more keys in a relation providing true associations between more than two tables (as opposite to modern relational database engines that allows us only to define a single relationship with just two tables).

With this solution, the classical clinic problem to associate patients to appointments and attending doctors could be eased. This is because we could associate all three keys (DoctorID, PatientID and AppointmentHour) and establish as a rule that we could not have duplicates in DoctorID & AppointmentHour and in PatientID & AppointmentHour. With no further restrictions, the model can validate all the main issues in this situation: Not to appoint more than one patient for hour for the same doctor, not to appoint more than one doctor for hour for the same patient. Notice that although we are dealing with the same relation, the pair DoctorID and PatientID can have duplicates.

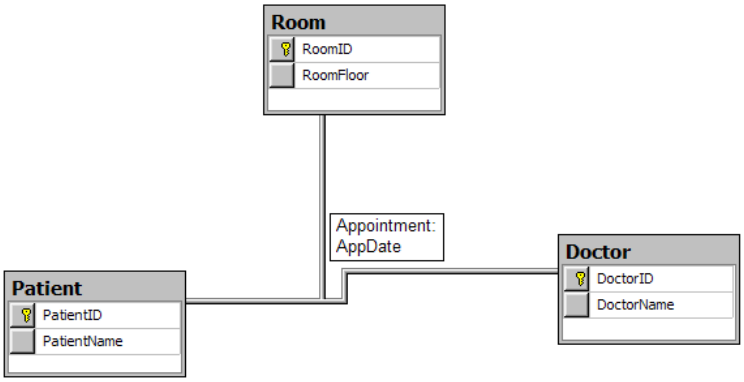


Picture 5: The Patient/Doctor/Room/Appointment problem

To create such relations we should provide an SQL statement like CREATE RELATION ON Doctor,Patient,Room WITH (AppDate Datetime NOT NULL) CONSTRAINT UNIQUE DoctorID,AppDate, UNIQUE PatientID,AppDate

This statement should create an internal table with the attributes DoctorID, PatientID, RoomID and an extra AppDate. It would also create unique indexes for the pairs DoctorID,AppDate and PatientID,AppDate. This structure would implement the conceptual relation among these entities.

With a multi domain index, all the rules should be on that relationship and it would look like this:



Picture 6: Doctor/Patient/Room with an Appointment Relation between the three data tables

To relate/un-relate data, we should use the plain INSERT / UPDATE / DELETE SQL statement applied to the relation. Examples of SQL to manage relations follow:

INSERT Appointment (PatientID, DoctorID, RoomID, AppDate) VALUES (100,23,4,#5-May-2010#)

UPDATE Appointment SET AppDate = #11-May-2009# WHERE DoctorID = 21 AND AppDate = #5-May-2010#

DELETE Appointment WHERE DoctorID = 21 AND AppDate = #10-May-2010# AND PatientID = 95

As you see, there is no need to further keywords in SQL. They are all applied in the context of one relationship (Appointment). This is because internally the

Appointment relation should be build as a table (as all the indexes are), so we can manipulate it from outside, as it was a regular table.

As seen the set of foreign keys are now building another structure that implements a cross-domain index, as defined earlier. This cross-domain index also proves to be an OUCD, since a multi-domain order is also defined.

The multi domain indexes can also be used to implement some hierarchy among data, because an index (whether it be a single or multi-domain index) will eventually order data in some way, adding a little more meaning to the relation it applies. If we have, for instance, an employee-manager relationship, besides its semantic we can add a job hierarchy to this just by ordering it properly. In addition, this can be achieved because this structure is an index structure and we can have this additional meaning added just as simply defining how that index should order its data.

It seems clear that the information about these relationships no longer resides on the tables, but instead, on the index structure of the database. This also means that in the backup strategy of the table these indexes must also be kept in order to reconstruct all the data.

On the other hand, building these structures as indexes can lead to have them permanently in memory (as databases engines already do that for indexes) [30] and reduce in a significant way accesses to related data because their physical pointers are already in memory, hence improving global database performance. Access to raw data can then be performed by one of the traditional techniques like (hash tables, clustered keys, etc.) as they are right now. For very large datasets, however, the amount of RAM can be insufficient. Generally, in these cases, the database engine spans the indexes through disk, with costs in performance that may prove to be significant.

This approach is consistent with Chen's definition of primary keys as functions that would return the corresponding set of data (row) for each key given [6]. This

is not only true for primary keys, but for all indexes in general. Even when we have duplicate indexes, we can add to the index the primary key (or one of the candidates) and make it unique, even if it is only in an internal database engine procedure.

Chapter 9

Definition of an Ordering Function

To achieve independence between the relations and their sort order, ordering functions should be defined so several relations can use the same sort method, if applicable.

In order to do that, and from a strict theoretical point of view, the “Ordering Function” must be mathematical defined as a functional implementation of an OUCD.

These abstract functions can then be associated (if compatible) with a relation taking values from a UCD defined for that relation. We define that function is compatible with a relation if the arguments for the function are a subset of the attributes of the UCD. Associating such function to a UCD defined over a relation creates an OUCD, as defined in chapter 6.

In most cases, ordering functions uses domains that are not OUCD sorting two (or more) “identical” records in an arbitrary way. If such order is requested the domain should be extended to include one OCUD so a unique ordering could be achieved. This extension should be done by the database engine if not explicitly requested by the user.

Consequently an order function φ on a set of attribute $(A,B,\ldots,Z)_R$ over the relation R must have the following properties:

- Its domain is an OUCD.
- It returns the ordinal (rank) within that domain for each value ($V_{ab..z}$) belonging to its domain.
- It is invertible since for each point in its domain (OUCD) has only an image in its counter domain (relation data), by definition of OUCD. The

inverted function gives, for a given rank, the corresponding values of the OUCD.

With these definitions an ordered relation is a structure composed by a “regular” relation, the definition of an UCD (and it was proved that at least one UCD can be defined over a relation) and associating an ordering function to that UCD to transform it into an OUCD.

We also define an extended relation as structure that is composed by a set of data (a relation - R) and one sorting function (φ). Since the function only applies to one relation because of the OUCD, the OUCD implements the connection between the relation’s data and its domain. Every relation will have one or more functions that can sort it. However, the some relation (i.e. the same data set) composed with two different sorting functions are also two different extended relations even if they have equal OUCD defined on them. The ordering function based on that OUCD can be used to sort both relations, but, still, they are independent.

If that relation is operated with another relation through any relational operator, the result will be a new relation (closure property of the relations) that will present its data with any kind of order even if there is no previous specification of that order.

However, since the result set will have some kind of order, so it can be claimed that such sorting function exists (although it might be unknown) because the results are presented in some kind of order. Since the result is a relation and we can always have at least one UCD defined over that relation, an unknown function is defined over that UCD that provides order to the result.

Then this extended model also has a closure property as long as relations (regardless of their sorting functions) are operated with a relational operator and a sorting function is assigned to the resultant relation. From a mathematical point of view, it is formally an algebra, as defined in Braumann’s book [4].

Let us demonstrate:

An algebra must have the followings properties:

- The set must be closed, in respect to intersection and, in addition, chain condition must respected (semiring).
- The set must also be closed regarding to subtraction and binary intersection or union operation (ring).
- Next, the set must be closed to complementation (algebra).

Let us consider Ω as the universe of all possible ordered relations composed with a relation R and an ordering function φ (R,φ).

The intersection of sets is also defined for relations since they are sets. For ordered sets the intersection operation is also applicable as shown before.

The chain condition states that for a non-empty class K, two sets $A \in K$ and $B \in K$ and $A \leq B$ there is a finite number of sets that observes $A_m \in K$ (m=0,1,2,3...,M) such as $A_0 = A$ and $A_m = B$ and $A_m \leq A_{m+1}$ for $m < M$ and A_{m+1}-$A_m \in K$, also for $m < M$.

For an extended relation it is also possible to define a relation B with a sorting function φ, as element (B,φ) and a class K where all sets are derived from B with the same sorting function φ. Any subset of B with the same sorting function φ will be the A set and, therefore, element (A,φ). Let us consider that A has M less rows than B (therefore a selection of B) so we consider A_0 as A, and A_1 as A_0 plus a row belonging to B but not to A, and so forth until the M rows added to A will give the B relation. It is trivial that the extended relation is closed for intersection operations since the intersection of a subset of B with another subset of B is still a subset of B. Since the ordering function is the same for every element of the class, the closure property is verified. The chain condition is also verified, so the class K of every subset of B is a semiring.

To verify if the class K defined as all the subsets of any ordered relation is a ring it is necessary to verify if this class is not only closed regarding the set subtraction but also closed regarding to the binary union and binary intersection.

The subtraction of any two subsets of B, as defined for sets, is still a subset of B by definition. Also the union of any subsets of B is still a subset of B. Since the intersection of any two subsets of B is also a subset of B and all subsets share the same ordering function φ, the class of any subset of B with an ordering function φ is a ring.

Finally to be an algebra, as stated, it is necessary that the class is closed to complementation.

For any subset of B, (A) exists another subset of B called A- that verifies the following equations:

$$A \cap A^- = \{\} \text{ and } A \cup A^- = B$$

Since the meaning of A- is all the tuples belonging to B not existing in A, A- if beyond doubt a subset of B, thus belonging to the class.

So any ordered relation is, as stated, an algebra.

The ordering function must have a set of attributes to be considered as one.

A function to be considered an ordering function must:

- Accept a subset of the UCD where it is defined
- Return the rank of each tuple inside that UCD
- It must be invertible and returns the element of OUCD given the rank

For instance, consider the pair E1=(R1, φ1) and E2=(R2, φ2) with (R1, φ1) being an extended relation in which R1 is a relation associated with an sorting function φ1 and (R2, φ2) is another extended relation in which R2 is a relation associated to an sorting function φ2. The relational operation θ between E1 and E2, (E1 θ E2) can be defined giving the result as an extended relation E3 = (R3, φ3) in which R3 = R1 θ R2 and φ3 its sorting function.

Then (E1 θ E2) = (R1, φ1) θ (R2, φ2) = ((R1 θ R2), φ3) = (R3, φ3) = E3

Please note that the sorting function φ3 does not have to be defined using the other two ordering functions (φ1 and φ2). As an example, taking a relation, for instance, customer order by “Name” (R1, φ1), joining it with another relation

such as invoice, order by “Invoice#” (R2, φ2), the join result might be order by InvoiceDate (R3, φ3). The latter is not related on its composition from the previous ordering functions. But nothing prevents the function from being derived from the previous as, for instance, if the same join result was ordered by “Name” concatenated with “Invoice#” (R3, φ’3) this sorting function is indeed defined from the previous ones.

We can also rewrite all of the principles of Relational Calculus (either Domain or Tuple) considering the extended relations.

In domain Relational Calculus we have $\{\langle X_1, X_2, \dots, X_n\rangle | p(\langle X_1, X_2, \dots, X_n\rangle)$ as the form for a general query where X_i is either a Domain Variable or a constant and $p(\langle X_1, X_2, \dots, X_n\rangle)$ denotes a formula.

Since the ordering function is only applied to the output, the general formula of the Relational Calculus can be rewritten considering a general ordering function φ like this: $\{\varphi(\langle X_1, X_2, \dots, X_n\rangle) | p(\langle X_1, X_2, \dots, X_n\rangle)$. The output is the rank for an ordered relation in the form of (R, φ) where R is $\langle X_1, X_2, \dots, X_n\rangle$. To obtain the relation the inverse must be used. So an OUCD is returned with $\{\varphi^{-1}(\varphi(\langle X_1, X_2, \dots, X_n\rangle)) | p(\langle X_1, X_2, \dots, X_n\rangle)$

There is no distinction being made between Domain Relation Calculus and Tuple Relation Calculus because the principles are the same. The ordering function only changes the left part of the query and only in presentation order.

On the other hand, and by definition of relation, at most all domains makes a UCD. With the ordering function, the outcome is an OUCD by definition.

We can derive that any formulae that are safe in Relational Calculus are still safe with the extended relation. This is because the ordering function does not change how the expression is constructed but only affects how the outcome is presented.

Then how could this help to solve the problems that were referred as cases?

Let us see. In the first case, it would be sufficient to associate to the resulting relation a sorting function with a random output (no arguments is a subset of

any set of attributes), so that each time the function was associated with that particular set of data would produce a different extended relation.

In the second case, since the sorting function is invertible it is not only possible to know what particular order has a tuple, but also it should be possible to know the value of any field from any row. It would be possible then have access to the running sum stored in the previous row and, without remaking all the calculations, to add only the value in a line to the previous running sum to obtain the new running sum.

It's now perceivable what the advantages of this model are since the industry, regardless of the theoretical model, has implemented its own data sorting (needed since ever) through SQL, being SQL itself a standard (ORDER BY clause).

What then this new model brings that SQL's ORDER BY does not have?

First of all, the ORDER BY clause can only be applied to the columns (fields) of the result set. The sorting function, although it receives as an argument the all tuple, doesn't need to use any of the raw data to assign a order number (e.g. The "natural" order, meaning, whatever tuple is presented the sorted results would always be 1,2,3,…,N).

On the other hand, the ORDER BY clause uses only the binary content of the field to sort the results out (ascending or descending), whilst the sorting function doesn't have that limitation. It can assign an order number to a row depending only the way it was defined.

Finally, the sorting function can be used to define non-linear business rules, complex mathematic expressions, random expressions or any other kind of expressions like results from software for optical image recognition, since the model has no restriction in its internal logic neither how the ranks are evaluated.

Therefore, in the first case it is only executed a simple database query with the following *possible* syntax:

```
SELECT HotelName, RoomFee
FROM Hotel
WHERE RoomType IN (@TypeList) AND RoomFee BETWEEN @LowPrice
AND @HighPrice
ORDER BY SqlRandom()
```
[5]

Since that statement its not possible to issue, with the present database engines, a procedure was built that assigns to a temporary table to store the results a new field filled with a random number between 1 and ROWCOUNT() (i.e. the number of rows the query is returning). Finally, the procedure queries that temporary table sorting out the results by that random field.

This is a so complex solution that it required a row level processing which must be avoided at all cost, as stated in the introduction.

Since row level processing must be avoided, row identification is nevertheless useful and can be used to avoid some of that “nasty” row level processing.

With row identification the concept of “first”, “last”, “previous”, “next”, “n-th” when referring to any tuple is clearly defined.

Although it seems like a going back, in concern with the relational model – in a matter of fact, after have been defined the operations between relations with the due independence regarding its tuples – getting row references (pointers) it seems like a going back. Moreover, it would be if row level processing of the relation were used. However, what is wanted is to define is some new aggregate functions that can use the relative positioning of the rows (their order) in a way that they could be integrated in the SQL so data can be manipulated.

One practical example would be the function ORDERPREV (domain) that would return the value in the domain “domain” in the previous row[6]. If ORDERPREV (domain) is to be defined, according with other SQL aggregate functions, would

[5] Bold formatting will be used whenever proposed syntax is included in SQL statements

[6] Note that it is not the previous value for that domain, but the value for that domain from the previous tuple.

return a NULL value for the first row, the following query (in MS-SQL) could be issued to solve the second case:

```
SELECT DtBudg, MonthValue, ISNULL(ORDERPREV(Accumul),0) + MonthValue as Accumul
FROM Budget
WHERE DtBudg BETWEEN '1-Jan-2009' AND '31-Dec-2009'
ORDER BY DtBudg()
```

DtBudg() is a sorting function based on a table field (it would be equivalent to the actual ORDER BY DtBudg).

This query would then give for a given time slice, the monthly values of the budget, sorted by budget date with the running sums of the monthly values along with these.

Given the following table

DtBudg	MonthValue
31-10-2008	1.000,00 €
1-1-2009	500,00 €
1-6-2009	2000,00 €
1-1-2010	750,00 €

The result of such a query would be:

DtBudg	MonthValue	Accumul
1-1-2009	500,00 €	500,00 €
1-6-2009	2000,00 €	2.500,00 €

The SQL to execute the very same function, using Microsoft's T-SQL, which also solves the proposed case, is the following:

```
SELECT B.DtBudg, B.MonthValue, Accumul = (SELECT SUM(MonthValue)
        FROM Budget BB
        WHERE BB.DtBudg BETWEEN '1-Jan-2009' AND B.DtBudg)
FROM Budget B
WHERE B.DtBudg BETWEEN '1-Jan-2009' AND '31-Dec-2009'
ORDER BY B.DtBudg
```

Although, technically it can be executed, in a matter of fact, beyond the statement is much more difficult to understand, the sum is calculated for each row instead of using the last value computed from the last row, as is proposed in this model, and affecting the overall performance.

Using such an extension could provide some theoretical support in a way to suppress the gap between the relational model most rigorous mathematical definitions and the standard SQL in what sorting is concerned.

Chapter 10

Definition of an Ordered Relation as a Structure with an Unordered Relation and a Ordering Function

Since any relation regardless of the order of columns or rows contain the same data, we can refer to such relation as R

When we look into attributes and their domains, we can have attributes belonging to nominal domains, ordinal domains or scalar domains. In order to find anything without a full scan the scale must be at least ordinal. All search algorithms are based in order and data must have, at least, binary order.

We can, in the other hand, always define an UCD because there are no duplicate rows in a relation [8][15][10]. If we define how the elements of this UCD are ordered then we will have an irreducible OUCD. Extending this OUCD to several attributes, we can build as many irreducible OUCD as needed.

For each OUCD built this way we can associate a specific function that allows us to know the rank of each value but also the value for a certain rank. Since the OUCD is connected to the relation data, the functions defined earlier can always be used with that relation. We can even say that the functions were defined for that relation.

Now, the same relation with different functions shows the very same data in different ways, meaning, how they are sorted. It is imperative that we can show that although we have the same data we are sorting it out differently. The “natural” approach is a mathematical structure that represents, as a whole, the relation and its function for a particular sort order.

So instead of referring to the relation R with a particular ordering function φ we shall refer them as (R, φ) meaning that R is the “unordered” relation and φ is the ordering function with an OUCD defined over R as its domain.

This base structure approaches somewhat has been done in the commercial database engines, since indexes are defined over data structures such as tables or views.

In a matter of fact, it was common sense that an index should be built over a relation, so dropping a table would drop all the indexes defined for that table.

On the other hand there were no ways of defining cross-domain indexes other than indexes over foreign keys or composition of foreign keys whether they reside as foreign keys in a table or in an association structure to implement a many-to-many relationship.

The extended relation as defined here can be the support not only for sorting inside a relation, but as shown in chapter 7, as a basis for cross-domain implementation.

As demonstrated before this structure is still a relational structure, because it holds all the relational properties as its “unordered” compound.

Chapter 11

Proposed New SQL Keywords

In this chapter, new keywords and their syntax in SQL notation are proposed.[7]

The DDL new keywords are:

CREATE/ALTER/DROP ORDER FUNCTION <name> (<parameter list>) ON (table/view) AS []

This keyword creates an ordering function over a relation (specified here as a table or view). The parameter list is a subset of the relation attributes that forms an UCD. The function will create an OUCD from the UCD assigning a rank to each row in the table/view.

CREATE/ALTER/DROP RELATION <name> ON <tablelist> WITH (<field> [NOT NULL], …) CONSTRAINT [UNIQUE] <fieldlist>, …

This keyword will be responsible to create a relation between the tables in table list (thus allowing a relation with a degree higher that two).

The syntax is as follows: the <name> parameter must be replaced by a unique name inside a database, the <tablelist> is a list of valid tables/views of the database that are about to be related. The WITH clause lists the fields from each table that will be related. This list is the set of fields that compose the foreign key that implements the connection to another relation.

The CONSTRAINT keyword is the set of constraints applicable to the relation such as uniqueness (UNIQUE keyword) on a specific field list (<fieldlist>)

[7] Definition follow the definitions in ISO/IEC 9075 (SQL).

The adapted SQL keywords are:

INSERT/UPDATE/DELETE statements should also include RELATIONS as objects where it is applicable, thus allowing that, for relational databases, relationships between tables can be established by rules instead of simple data relate with foreign keys. These rules can be used by object oriented programming language as part of the object's methods

The DML new keywords are:

ORDERFIRST(<field>)

This keyword will return the first value for a relation (does not have the same meaning as the FIRST() SQL aggregate function which returns the group's first value)

ORDERLAST(<field>)

Similar to the previous keyword, but returning the last value for a relation (also does not have the same meaning of the LAST() SQL aggregate function which returns the group's last value)

ORDERPREV(<field>)

This keyword returns the value for field <field> from the previous row, if any. For the first row will return a NULL value with the meaning of "unknown". Of course, this field must be scanned for NULL values as they might occur within data.

ORDERNEXT(<field>)

This keyword is similar to the previous one but returns the value for field <field> from the next row, if any, instead. For the last row will return a NULL value with the meaning of "unknown". Of course, this field must be scanned for NULL values as they might occur within data.

ACCUMUL(<field>)

This keyword is a convenient shorthand for the expression “<field> + ORDERPREV(<field>)” that return the running sum for that field. This keyword might be used without specifying the field in the SELECT clause.

The altered DML keywords are:

ORDER BY clause in SELECT statements (using an UDOF[8] as ordering function)

The syntax of the ORDER BY clause is to be extended accepting an Order Function, as defined earlier, as a construct element. This function will order the output according solely to its logic.

NOTE:

Actual FOREIGN KEY con be maintained for backward compatibility since there is no further use for it (it has been replaced by the RELATION keyword).

[8] UDOF – User defined ORDER FUNCTION, as defined earlier.

Chapter 12

Case Studies

There are some cases where ordering is crucial for a special activity. When such cases arrive, the usual approach is to prepare the database to support the data and see if any built in functions for some particular database engine can fit the purpose needed.

If it does not, then some kind of workaround is built in order to solve the question.

We will present three cases that needed a different approach from what was given to them

The first one is an application designed with database technologies that has as a fundamental request that the out coming data of a certain query should never be presented in the same sort order although the base query could be the same.

This is a real case and concerns a list of hotels with rooms to let. The request is that even if two different users choose the same criteria to search for a room, the result must be sorted differently. The goal is that equality of opportunity was given to each hotel in the list. How can we fulfil that request if sort order is not a result of a relational operation?

The second case, another real case either, concerns the scholarship for needed students with strict rules about ordering the candidates combined with a budget for all the grants given. The question is even after the candidates were sorted by some complex criteria involving income declarations, school approval, other private scholarships that they could have and even if they were benefited on the previous year (which allow them to continue), there was a budget to respect. Therefore, the running sum in the candidate order should not exceed the

amount of the budget. They must be ordered by a specific set of rules and then granted a year scholarship. However, the budget will end at some point, so the running sum should be equal or less than the global budget. That means that two students following each other, the first can be granted the money (and reach the limit of the budget) and the next one gets nothing (the budget is exhausted).The question here is how to control the budget by a simple SQL statement, if order is not a result of a relational operation. In this case, it is needed to compare the running sum with a specified amount in order to know when to stop giving the benefit.

The base table can only be updated (it cannot be by SQL means) taking as a condition that the running sum in a particular order should be less than or equal to a certain budget value.

The last case it is necessary to obtain the values of a budget with the running values along with it (in every row, not summarized at the end). How can we sum in each row the preceding values without breaking the database performance, meaning, without having to calculate totals for the running value for every row in the table? How can we do this if we have no order and we cannot know which row is before and after in the result set?

In the first case, the solution was found by means of an temporary table, with the results on it and with a spare column (OrderID) that was filled after the query had populated the temporary table.

After that, a routine that handled each row of that result set would set the OrderID by generating random numbers and assign to the rows by turn (and verifying that no duplicates were assigned).

At last, the results were presented sorting on that last field.

This solution, although fully functional, was far from simple, since it involved some coding in a high-level language. Besides that, involved row by row

processing outside the database engine instead of a declarative statement, which was desirable in the first place.

In the second case, there was no solution but dealing with a row-by-row basis, calculating the amount to be given to a candidate and the subtract it from the budget until it reaches zero (a procedural solution). Then the calculated amount was recorded in the candidate record (or zero if the candidate was beyond the budget) by means of a cursor

In this case, a particular sort order should be taken and functions that use that order (such as the “ACCUMUL” function defined in chapter 11).

The last case, a complex SQL statement did the job, but with a very large break in performance because the running sums were all calculated in every row, hence affecting database performance as a whole.

Even building a view with only the data to be treated in order to limit the number of rows treated would not solve the performance issue if we have a very large number of rows to treat.

The implementation of this extension gives the result set in its requested order with the running sum along with that and it can actually filter the base data by the running sum field. This allows that processing can be made by a declarative statement as desired without the use of a cursor.

The SQL statements that can “solve” and their accompanying results are listed in the annexes.

As these three samples, databases are filled with every type of requests such as “ordering the Employes table by his Job Ranking”. “Job Ranking” is not a field but a complex function where the admission date, present rank and some other elements are combined to produce a result, or requests like “When do our

income reaches £1,5M?" hence calculating a running sum until that value is reached.

As seen, our database engines cannot properly answer those questions in a simple way. Although not everything is simple, we think that these issues could have a much simpler way to be solved.

The first case is solved with the definition of a RANDOM() function and issuing an SQL statement with the "ORDER BY RANDOM()" clause, as listed in the annexes.

In order to implement a random order, some issues were raised. It is not possible to use one of the classical algorithms to sort random data. When we randomize there is no guarantees that the generated data prevail across comparisons. So "a" can be greater than "b" in the first pass, "b" can be greater that "c" on the second pass and finally "c" can be greater that "a" on the next pass. One possible solution to this (and this was the solution used in the hotels case) was to pre-number all the rows by means of an extra field. In this way, the rows could be numbered randomly and sorted after by this "extra" field. But some considerations must be made on this process. If we had a very large database with terabytes of information what would be the cost (in time and other resources) of adding an extra field, populate it with random values between 1 and n (where n could be a very large number), then order it to produce the output.

But "random" is a very special case of ordering since it doesn't rely on the values of the rows. Therefore, there is no need for any comparison (as opposite to sort methods) but only the need for output in random order. Furthermore, the values may not be persistent, but randomized each time we need that particular sort order.

We only need to generate a random number between 0 and n-1 (where n is the number of rows) and pick that row to output. Next the row is deleted and the

process is repeated for the remaining n-1 rows. Since this process is done in the “ORDER BY” clause, this all applies to the output queue thus not affecting “live” data. The deletion of a row is a very expensive operation in time (but not in space) since all the rows above that one have to move one place down, if we consider the output queue as an array of data. The time and space complexity analysis for this method is presented in the Annexes.

Can we support a so large number? The answer is yes because every database engine has support for row count. So if the table can accommodate that quantity of rows, they can be counted by means of as integer (may be a bigint, which is an integer of 64 bytes). This means there is always support for the row count. If there is not then the recordset cannot be generated.

Can it be sorted in memory? Most database engines have support for large recordsets. It is obvious that a very large dataset may not fit in memory. In such case, database engines spans all the data between RAM and virtual memory in disk. Although the access to a disk sector is not a genuine random access, (it is truly a sequential access), the access times are very low and it behaves “almost” like a random access. In a large recordset with n records where only k records are in RAM (so we have n-k records in disk) we have a mean access time for a record $(t_k \times k + t_j \times (n-k))/n$ where t_k is the mean time to access to a RAM record and t_j is the mean time to access a disk record. Therefore, the size of the recordset to be sorted depends on the database engine and its strategy for dealing with large amounts of data. From a theoretical point of view, RAM is infinite and any recordset can be accommodated inside it. From a practical point of view, the database management systems provide limitations either in row size and either in number of rows per table. They also define the minimum RAM amount to operate. These are indicators that one must have in mind when performing heavy operations in databases. The time and space complexity analysis helps to determine how much RAM is needed for a random sort.

Can we generate a random number between 0 and n-1? Depending on the programming language used, it might be possible. Modern languages such as

VB.net, C#, Java have random number generators with 64 bits. Even with the IEEE representation, we can generate about 2^{62} random numbers between 0 and 1. In the unlikely event of having a database larger than that (meaning that it would have more than 4 billion of billions of rows), probably the database engine would have support for that large numbers so the problem would be naturally solved.

An algorithm (technology independent) was developed and used for this particular case. This algorithm is shown in the Annexes. The implementation of the algorithm took some elements in account. First, can it be implemented? Yes, it can. C# has all the primitives to implement it. When a recordset is populated, the high-level language does not know whether a particular record is already on memory or must be fetched form the disk. The database engine deals with the low-level details of every query. Can it be fast? Every sort algorithm must have multiple passes on the recordset (if the recordset is already sorted then it will have a single pass, but this is an extreme condition) whereas the random algorithm only access once to each record (it does not need to compare anything). The only overhead is the deletion of the record, which causes a delay. In the implementation, when the database engine has to perform a non-optimized sort, it creates a temporary table with the data being sorted (generally on the fields or expressions on the ORDER BY clause). Deletions on that temporary table can be made as “marked for deletion” but not actually deleted so there is a little overhead time in marking the record as deleted instead of an actual deletion. The actual deletion would result in moving all the remaining records up one position with a large overhead of time. In the annexes, a table shows the result times in two different machines (a laptop and a workstation) for the random algorithm over a table with about 1,700,000 records against a sort over a non-indexed field. The results show that either the presentation of the first record or the completion of the query was faster on the random procedure than on the sort one.

Since deleting a record and moving all the following records one position down, is a very time consuming operation, three variants of the algorithm were

studied. The first one (scenario 1) actually deletes the random picked record and moves all the others down. In this scenario the space occupied by the data never changes because while one recordset is being emptied the other one is being filled at the same rate. On the other hand, it performs badly in time when the record number grows. The second variant (scenario 2) does not delete the record but, instead, marks it for deletion. The output queue is never packed but the whole input recordset is discarded when the procedure ends. In this scenario, although it performs faster with noticeable improvement for large recordsets, the space occupied in memory is the double of the initial recordset because near the end the procedure holds the entire input recordset and the result recordset as well. The third variant is a mix of the previous where an arbitrary number of records are marked for deletion before they are actually deleted. In this scenario, the space occupied is directly proportional to the number of records marked for deletion and the time used is faster than deleting the rows in the output queue one by one.

In the second case, the running sum of is used in the where clause to limit the candidates that would benefit from the scholarship. This sample only tries to show that a filter can be built with an extension function so the update statement (which was not implemented) could use a WHERE clause like the SELECT does.

The third is solved with an SQL statement where in the field list was defined the running sum as the sum of <value> with the previous running sum for that value, as in “SELECT Date, BudgetValue + ORDERPREV(rsum) as rsum FROM …” where BudgetValue is the database field that holds the value. In this case, and because of the prototype limitations, the “ACCUMUL” function was used as a useful shorthand for the expression “AcmField = <field>+ORDERPREV (<AcmField>)”, as listed in the annexes.

As this, there are many situations where order is involved and, as for textual data, database engines internally have defined their own “ordering functions” based mainly on the binary value of data. Nevertheless, although this approach

was reasonably good, some “loose ends” began to emerge. One of them is the variations on the same character (for instance “a”, “ã” or “n” and “ñ”) in the beginning of a name that would place it out of order. Then codepages were defined more as a workaround than a solution.

The last case is not an everyday situation, but it tries to show that a user defined sort order with an outside procedure can be useful. The goal was to sort images only by content disregarding any metadata it may have. Viana, a Portuguese researcher, in his PhD thesis[46] has built an application that would analyse images of skin marks and by analysis of the shape of the edge of those marks it would determinate how probable they can be a cancer one. Those images were stored in a database and the analysis algorithm was implemented as a sort function so that the images may be sorted by the result of the analysis. This procedure was not optimized since it will be object of further study by both of us in the sequence of this work.

This approach solved almost every case of “common ordering”. However, there are still issues on ordering data because the low-level binary representation is definitively not enough.

Associating data with an OUCD and defining a function that can play the role of translating data into order and back will cover the past functionality of the binary sort order and can add a completely new range of functionality allowing defining precisely what order should data have.

Chapter 13

Object Oriented Databases

Object oriented databases began to emerge on the middle of the decade of 1980 (1986, to be more precise) [49] as need to solve issues that relational databases could not solve. One of the big issues was dealing with complex objects that must be decomposed into a relational schema before being made persistent [47][48]. After that, they could be stored and retrieved.

To solve this problem, object-oriented languages should create and maintain, in code, all the mapping necessary to convert objects in relations and back [49][52]. That was not desirable.

The first generation of object oriented databases are dated from the late eighties (1986 ~ 1988) such as G-Base, GemStone. A second generation of object-oriented databases appeared in 1989 along with “The Object-Oriented Database System Manifesto”, which stands as a milestone in ODBMS. In this year (1989) ODBMS like Ontos, ObjectStore, Objectivity and Versant (ODBMS) appeared. [50]

After the Manifesto, a third generation of ODBMS showed up in order to implement the new conclusions pointed out by the Manifesto. Orion, O_2, and Zeitgeist, all of them classified as third generation ODBMS [50] made their appearance in 1990.

The ODBMS lacked the formal structure as the RDBMS had and an object query language [47]. So, along with the “pure” object oriented databases, a new approach was made on relational models creating the object-relational model, which was no more than a relational database engine with an object-oriented interface [48][51]. These models (adopted by some major manufacturers like Oracle, for instance) have what is called as impedance mismatch. Impedance mismatch is no more than all the mapping that is needed to be done to convert

objects into relations, when storing, and back to objects when retrieving [52]. The Object-Relational model tried to put impedance mismatch on the database side, instead of being done by programmers on code. [49]

However, even on the ORDBMS side, the impedance mismatch was creating some overhead on storing and retrieving data, so these databases could never perform as a “pure” relational one. [52]

On the other hand, a general lack of appropriate language was felt because the language in ORDBMS was SQL (because these DBMS were, in fact, relational ones) and a language to deal with objects that could allow specifying inheritance without complex joins did not exist.

In 1991, the Object Database Management Group was founded with the goal to standardise the ODBMS system on the market by means of defining standard that should be implemented by the major ODBMS vendors. To achieve this and to increase portability among different ODBMS systems, it was needed to define what would be the Object Model, the Object Definition Language (ODL), the Object Manipulation Language (OML) and how the programming languages like C++ and Java should bind to the ODBMS. [50]

The ODMG 1.0 standard was published in 1993. Working in further developments in Object Databases, two more standards were published: The ODMG 2.0 standard in 1997, and the ODMG 3.0 Standard in 1999. The ODM group disbanded after the former publication. Since the interest for Object Databases continued a new group – the Object Database Technology Working Group (ODBMS WG) was formed in 2005 and is actually working on the fourth version of the object database standard. [50]

Object Database appeared mainly because mapping objects to a flat tabular data, besides being an extra effort (~30%) whilst programming [50], it was also a layer where modification either in objects or in the tables below were a source of potential errors and costs. With Object Databases Management Systems no

mapping between the application defined objects and their persistent counterpart was needed. [48]

On the other hand, there was risk that one programmer would map an object in a different way than another programmer, thus provoking different behaviour on the very same object. [49]

There was also the risk that, with access to the relational data, objects integrity rules were bypassed by tools [49]. This risk is the very same that relational database had with flat files. The access, by tools, to the flat files could jeopardize the integrity of a relational database.

Therefore, to deal with large, complex objects the Object databases are preferred, including working with these objects for a large period [48]. Still, if a database need to have a very large number of ad-hoc queries the Object Relational Database Management Systems are better, because of the use of SQL in queries [48].

With further developments on Object Databases is expected that these ones can replace the Relational Database Management Systems because Object Databases tend to be simpler to use and to represent reality, which is, in the very end, the main goal of a Database Management System.

One general lack in actual Object Databases is the ability of treating a table as an object itself. A table is a collection of objects, all with the same set of properties and sharing common methods. However, the table itself does not provide the ability of defining methods for the collection it holds. Implementing the ability of defining methods over tables would allow custom order to be defined over the data they hold. Of course, that should be a method over the collection of objects, and this configures an extended relation, where the “relation” part can be replaced without loss of meaning with a collection of objects allowing the implementation of the present theory.

In short words, this theory can be applied on relational / object-relational databases and also in object databases without loss of generality.

Chapter 14

Prototype

After developing the theory, this was applied to the three concrete cases referred in a previous chapter to see if this model could solve them effectively.

The most obvious choice was to modify the source code of an Open Source database engine such as MySQL. However, modifying the SQL parser, the interpreter and the optimizer poses a great risk of jeopardizing of the whole database engine, requiring a fresh start that another approach was done.

Although commercial, Microsoft's SQL 2008 allows us to develop procedures and functions in any .Net language such as C#. Building a string interpreter and defining functions using that framework seems not only feasible for the purpose of demonstrating, but also presented no risk in jeopardizing the database engine, so this unusual approach seemed the best one to suit the purposes of this research.

A prototype for the extension was built in C# to integrate Microsoft's database engine SQL Server. It seems that an extended stored procedure that would deal with the SQL statement and defer the execution of the standard part to the real engine (which is running without any modifications) would be safer in terms of stability of the prototype. Of course, that the final implementation should be made on the core of the engine and not by means of external code.

Since a prototype is being made to show that this extension can actually work on real situations, it has been chosen to implement only the necessary functions that allow the samples to show that the real cases could be also solved. Therefore, a certain number of limitations were constraints on the prototype because of all the choices stated earlier.

To build the prototype through an extended stored procedure it was clear that that procedure should take an SQL string (with or without the extended keywords) as its sole argument and produce the corresponding data table as a result.

In order to achieve this, the procedure should take the string, break the SQL into separate clauses (e.g. “SELECT” clause, “FROM cause”, and so on) and treat each one of them. Next, a valid SQL statement is executed by the database engine. If a sorting function, such as “SqlRandom” was defined through means of another extended stored procedure. The stored procedure is executed instead of the direct statement. These functions have the limitation that an extended stored procedure that implements it should be placed without any arguments in the ORDER BY clause and the implementation should take an SQL string (which is assumed well formed) and return the data corresponding to that SQL statement ordered by its means. After obtaining a data table from the valid SQL statement, the results are first treated for the new keywords (such as “ORDERPREV” or “ORDERLAST” and finally filtered by means of any extended keyword on the WHERE clause. The result is then presented to the user.

The first limitation of the prototype is that only the SELECT statement was implemented. There is no implementation for the INSERT, UPDATE and DELETE statements. The main reason for this choice is that the SELECT statement can be used to “see” what the other statements are going to insert/update/delete.

There is no support for expressions involving the extended keywords (although aliasing is provided) in the SELECT clause. This means that an SQL statement like “SELECT Name, Number, ACCUMUL(Value) AS RSumValue” is supported but the SQL statement “SELECT Name, Number, value + ORDERPREV (value) as RSumValue” is not. Note that this last example is a valid SQL statement as defined by the extension. Nevertheless, it was not implemented in the prototype.

The SQL statement is then split into clauses. The ORDER BY clause is searched for the extended order functions. The second limitation of the prototype is that only ORDERPREV, ORDERNEXT and ACCUMUL were implemented since they are the only ones needed for the cases stated earlier.

The parser replaces the keywords with valid SQL expressions that allow a result to be returned by the database engine to be processed in the next stage.

There is no syntax checking for the standard SQL. If the query is badly formed then the engine will raise an error. In addition, there is no proactive error handling. Using any of the unsupported features will make the prototype to behave erratically.

The FROM clause, as well as the GROUP BY and HAVING clause will have no processing for the present. Although in the future, the extended keywords can be used over groups, this implementation does not consider them necessary to show the functionality of the cases stated in previous chapters, so no parsing at all was done to these clauses. It is not considered as a limitation because the prototype was never meant to deal with order inside groups, but a future implementation within a database engine is considered.

The treatment of the WHERE clause presents the next limitation of the prototype. To fully parse the WHERE clause, removing the extended keywords in very complex logical expressions may prove to be very time consuming to do within the scope of the prototype. Although a full parser could be created by software (using YACC, Bison, or in this case, C# Flex), the number of keywords to be parsed is reduced, therefore a small program to treat the extended syntax was written. Since the prototype is for demonstration purposes only, the extended keywords may appear only as an expression in the form of “<keyword> <relational operator> <constant/field>” with logical operators to connect to other expressions. With this approach, they can be safely removed before the database engine processes the statement and can be used afterwards to filter the data returned by the engine.

If an extended ORDER BY clause is present, the data is ordered by that clause. Since the extended order function must be implemented as an extended stored procedure, the SQL (without the extended keywords) is sent to that procedure and the data is returned to further processing. If not, the data is already been order by the engine. It also implements the extended syntax for “regular” fields as a function, in the form of “ORDER BY <field>()”

All the ordering functions were coded in separate classes just to demonstrate that they are independent from the parser and can be defined independently.

Now, a two pass procedure is done, one forward and another backwards to, in the first pass, populate the “ORDERPREV” and “ACCUMUL” fields and, in the second pass, populate the “ORDERNEXT” field.

At last, with the data table fully ordered, the WHERE clause is examined to find any extended keywords and, if any, the data table is filtered (this represents the implementation the “AND” operator in the WHERE clause). The final step is to give the NULL value in the first row for any “ORDERPREV” fields and on the last row for any “ORDERNEXT” fields.

Now the data table is ready so it is outputted to the end user who receives as the result of the extended SQL statement entered, as if the engine could make the whole interpretation of that statement.

If there is a syntax error trapped by the engine, this will be shown. Other errors, such as misuse of the extended keywords, are not fully trapped, so a general message is given.

The Random function (SqlRandom) was coded as a separate extended stored procedure that takes an SQL string and returns a dataset randomly ordered. Other ordering functions can be defined as an extended stored procedure and be used by the prototype. The only thing required is that it takes as an argument

an SQL string and produces the ordered dataset. This procedure (and all variants) was implemented following the algorithm defined in the Annexes.

It has, though, a major improvement since it is not needed to wait to fill the whole result recordset to output it. In a matter of fact, the output recordset is never constructed since the rows are being outputted after being randomly picked. They are deleted afterwards from the output queue according to the variant chosen. The database engine actually buffers the output rows before they are displayed on the user interface.

Times were measured in two machines for two events: time to first display and time to complete the query. The procedure was compared to a regular “ORDER BY” query on a non-indexed field. Since a permanent index cannot be built because each time that “SqlRandom” is called there must be a different order of presentation, the only possible compare for performance was with a query with an “ORDER BY” clause on a non-indexed field.

We took a log table from a production database with around 1,700,000 records with a record length of 1,299 bytes. Afterwards tests with a query “ORDER BY” on the LogDate non-indexed field against the SqlRandom function were taken. The times to execute those queries are recorded on the Annexes. Each query was run two times on each machine and the results recorded, as displayed on a table in the Annexes. No further analysis was done because all the results pointed out clearly that the SqlRandom function always performed better on the completion time than a regular ORDER BY query on a non-indexed field, having no loss of performance thus. On the second run, the regular query was already optimized by the engine on that connection, so the time to display the first results dramatically reduces. The SqlRandom function cannot be optimized because it presents the data in different order each time that is run, so no strategy can be made for subsequent calls.

Just for an example, an ordering function that takes the first field on the resulting dataset and finds the words “first”, “second”, “third” and so on until “ninth”, and orders the data by that sequence is also presented.

For the latter there is no case associated with it. It only shows that any order can be achieved with this model.

The ability to order must be used against data that is not usually sorted directly. None of the problems stated needed a new theory to be solved. They needed a theory to be solved more efficiently. However, if we speak about sorting images something new must be done.

Generally, images are stored in a database along with some textual metadata about them. The metadata can also be provided by a header in some image file types. The database always store and retrieve the image but it does not have methods to work with it. If a method can be implemented over images stored in a database that configures an Ordering Function then we can order those images based on its content disregarding (or not) its metadata.

A sample of stored images of skin marks in a database was ordered based on a complex analysis for malignity. This analysis as developed by one of our colleagues as a windows application that was adapted to work within a database. This procedure can show that anything can truly be ordered if at least one ordering function can be defined over a particular set of data. In this particular case, the routine is not optimized at all. It just intends to show that it is possible to do so. However, this matter is to be followed because there is a great deal of interest in providing an automatic procedure that would analyse and compare several skin marks in order to determine the degree of malignity of each one. Besides the main routine optimization, the database implementation needs to be optimized.

These optimizations are one of the major future words (as referred in the proper chapter) because there is great interest in provide a fast method to analyse and compare several skin marks against malignity.

Chapter 15

Conclusions

The existence of a Unique Constraint Domain applied to a relation was shown. When this Domain is fully ordered then it is called an Ordered Unique Constraint Domain.

Also was shown that all candidate keys belong to irreducible Unique Constraint Domains. Moreover, all enlargements of Unique Constraint Domains will still be a Unique Constraint Domains.

These domains constitute the base for any index in a database. The uniqueness of the index can be achieved combining any attributes with any determinant. Those determinants can be either candidate keys or any surrogate order such as storage record order.

Since an index is meant to sort data, a function was added that could take an Ordered Unique Constraint Domain, that can associate a unique order to a unique key (and that does seem achievable), we can add some more meaning to the relation between two (or more) data tables. This function, defined to a specific relation composes a structure that we have called an “extended relation” and it is distinct to the same relation composed with another function.

It was have also shown that a Unique Constraint Domain can be composed with attributes from different domains (even from different relations) composing a cross-domain index.

Moreover, it was also shown that foreign keys are logical pointers to data, so they behave like indexes. All foreign keys can then be placed into “relationship tables” managed internally by database engines built as indexes. Since these relationships are created with cross-domain structures, the cross-domain indexes are the proper structure to implement them.

But since in every Ordered Unique Constraint Domain is possible to define an ordering function, these structures are now based in rules and not in data, although the rule can be as simple the lexicographical order of base data.

The meaning resides on the utility of such sort order. We can then define not only what the participants in a relation are, but also how they related among themselves and what its role in that relation is. The index can be built as a linked list so that a particular sort order could be followed if necessary.

All indexes on a database can be built with this logic, because all the “regular” indexes fall into this document definition of indexes.

The use of indexes in a general way of speaking is a good practice for ordering and finding data amongst large databanks. If data cannot be sorted, it might become very difficult or time consuming to find it. This is particularly relevant dealing with foreign keys because one must see if any integrity rule is being violated or not, and to do that data in the referred table must be found efficiently. This would be very hard to achieve if there were no indexes on a database.

There is still a step to take. How can this be implemented in a database engine? When a dataset is indexed, there is a declaration of a certain number of fields to index and if they are ordered ascending or descending. This is called the “natural” or “system” ordering (according to Ng [33]), since it only takes the binary value of data being sorted. And because it is needed more ways to sort data than this simple method, database engine builders created what they call “database sort-order” or “database code-page” which are no more than rules to sort out national characters in a way that matches the culture in which they are used.

For instance, the words “Abelardo”, “Álvaro” and “Berardo” should be sorted in this very order although the symbol “Á” has a higher binary code than the “B”

symbol. This is achieved by database engines by defining rules how to sort this kind of symbols.

The answer for our model resides on building functions (such as database sort order) that we can use for building indexes and sorting data.

The database engine should accommodate an area where sort orders could be defined and maintained so indexes could be created accordingly to the rules defined on such functions.

This, of course, would have to bring changes how the database language (SQL, primarily) [t11] accesses data, so an adjustment to the language should be made.

It should be possible to define any sort order, and access data in that order. It should be also possible to define a new function that allows the access data in a particular order (not just the first or last row but also next and previous rows).

Now with an index formally defined as a function that returns data in a particular order, more functions could be added to datasets.

With this definition, ordering can be provided by means of user-defined functions that can literally provide any desired order. An example of this is the definition of a function that can randomize a set of data, followed by its implementation.

The concept of first, last, previous and next can now be added and data can be accessed from the first row, last row, next row and previous row, without have to define a procedural access to that.

Some database engines implements a function called “row_number” based in an SQL ORDER BY clause, which is no more than an implementation of a singular function that ranks results based on a specified ORDER BY clause.

From the beginning of databases, indexes were considered an implementation option but not a conceptual issue. Over the times, the use has proved that indexes are one of the data access foundations. Some researchers began the study of Ordered Relations [33] but what was defined was the semantics of those domains. On the other hand relationships has been defined as a foundation to the relational model [8], but the relationships have been implemented by foreign keys which are no more than pointer to data, meaning that they are no more than indexes.

Putting together these two realities together, an index structure that can provide support for relationships amongst tables, provide the notion of role within a relationship and behaves like a function to return the raw data involved, given a key (whether is one table, a pair related or a multi-related junction) have been presented. Additionally, since these indexes can be made out of rules, we can also have a semantic meaning for each relation is a database.

Chapter 16

Future Research Work

The first step is build a real implementation from the prototype (which doesn't implement most of the functionalities, but only aims to show that the cases stated earlier could have a better solution if database engines could treat order not just from a simple view of data.

From now on, it might be possible to combine ordering with optical recognition of images, sound analysis, or any kind of objects, generally speaking, so that we can indeed search within a databank of images, or sounds or, generally, any kind of objects because order to find any kind of data has been defined.

Moreover, in Object Oriented Database Engines it could be possible, treating tables and views not only as a collection of objects, but also as objects themselves, to define methods that allow sorting of those collections. Since an Object Query Language is still needed, syntax to achieve this goal can yet be defined. Part of the future work consists also in contributing to Object Query Language wherever possible.

The industry and international organizations may now define new alphabets to search in databases, not only with human readable characters. Search and find something in its various forms, like, for instance, looking for the number of ads of a particular brand in the TV channels, recognizing the images and providing immediate feed back to the searcher without having to classify every piece of information (in a human readable way, that is) in a database.

Making data more searchable can also be useful for models that use relational database constructs as a foundation for higher constructs, as for example, some implementations of object-oriented databases, which in turn are relational databases with an object-oriented interface, can profit from an improvement on their basis. As far as we can see, ordering data sustained by a class will no

further implementations besides its definitions, or if a method defines a specific way of presenting data, that method should be programmed and included in the database engine as a base method for that kind of data as an ordering function.

The new SQL keywords will certainly improve database performance when used against very large databanks, since they can provide aggregate data that only could be obtained in a more resource expensive way.

In addition, at last, it can be used for further research, as we intend to do from this point forward. The main research steps forward this point is to define how to obtain and sort non-textual data and (if possible) to approach a semantic way to describe this type of data in such manner that can be used by future database engines, as referred earlier.

The researcher is aiming to develop, in conjunction with other colleagues, a model to efficiently store non-textual data (such as audio, video or other type of binary objects) and provide means of querying that data such as using a photo or a sound to perform a search in the database. In a matter of fact, the researcher is already working with a colleague to store images of skin marks and implementing a sorting algorithm to order according to malignity, which, in turn, is calculated through an algorithm that examines the image and produces a set of results that can be interpreted.

As a practical sample, since some metadata is stored together with multimedia data (as for instance the jpeg file format), this model allow the ordering for that metadata. This can be achieved without the metadata being actually copied into some field or fields in the database, gaining a non-redundant schema for accessing that type of data.

One goal to achieve with future work on Ordered Relations is the ability of analysing skin mark for early detection of skin cancer. Joining efforts with Viana, using an optimized code from his model of analysis of skin marks [46] it can be possible to implement a method that allows databases to analyse stored images

by content. This in contrast to analysis by metadata can produce a quick, but exact analysis of the skin marks giving an accurate classification of those skin marks, without the time needed to produce and store the metadata.

Is it known that multimedia databases generally use huge storage space so a scan through data can be an extreme time consuming operation. Improving how data can be order and searched can surely bring improvements to multimedia databases.

On the other hand, even in ordinary databases (meaning not designed to be a multimedia database) the use of multimedia data is increasing, so these techniques can also improve searches in those databases in the future.

Chapter 17

References

In first place, the researcher wants to clear that all readings and all study are relevant to create a thinking that allowed him to formulate his theory. This is so because someone else's knowledge, after being studied becomes part of him and acts as a new base to new knowledge. It is truly impossible to enumerate all the references that were actually used to build our research. However, it is certainly possible to enumerate the more important ones, those that, in a matter of fact, constitute the true base on this particular subject.

Below is the list of what the researcher thinks it was more relevant to this research:

[1] – Adbelguerfi, M. and Sood, A. K.: "Computational Complexity of Sorting and Joining Relations with Duplicates" in IEEE Transactions on knowledge and data engineering, vol. 3, no. 4, pages 496–503, (December 1991)

[2] – Alagic, S.: "Relational Database Technology", Springer-Verlay, New York, 1986

[3] – Atre, S.: "Data base: structured techniques for design, performance and management", Wiley, England, 1980

[4] – Braumann, P. B.: "Teoria da Medida e da Probabilidade – Parte I: Álgebra de Conjuntos", Fundação Calouste Gulbenkian, 1987 [9]

[9] Professor Pedro Braumann was Portuguese and this book was never translated into English language. The title means "Theory of measure and probability – Part I: Algebra of Sets"

[5] – Chaudhuri, S. and Shim, K.: “Optimizations of Queries with User-defined Predicates”, Proceedings 22nd International Conference on Very Large Data Bases, Mumbai (Bombay), India (September 1996)

[6] – Chen, P. P-S.: “The Entity-Relationship Model - Toward a Unified View of Data”, ACM, Transactions on Database Systems, 1976

[7] – Christment, C.: “Prática de Bases de Dados”, Ed. Presença, Lisboa, 1990 (Portuguese edition)

[8] – Codd, E. F.: “A Relational Model of Data for Large Shared Data Banks”, Comm. of the ACM 13, No. 6 (June 1970)

[9] – Codd, E. F.: “Data Models in Database Management”, Proc. Workshop on Data Abstraction, Databases and Conceptual Modeling, Pingree Park, Colo (June 1980)

[10] – Codd, E. F.: “Domains, Keys, and Referential Integrity on Relational Databases”, InfoDB3, Nº 1 (Spring 1988)

[11] – Codd, E. F.: “Is your DBMS Really Relational?" and "Does Your DBMS Run by Rules?”, Computerworld (October 14th and 21st, 1985)

[12] – Codd, E. F.: “The Relational Model for Database Management Version 2”, Addison-Wesley, 1990

[13] – Date, C. J. and Darwen, H. “Foundation for Object/Relational Databases: The Third Manifesto”, Reading, Mass.: Addison-Wesley, 1998

[14] – Date, C. J. and Darwen, H.: “A Critical Review of the Relational Model Version 2 (RM/V2)”, Relational Database Writings 1989-1991, Addison-Wesley, 1992

[15] – Date, C. J. and Darwen, H.: “The Duplicity of Duplicate Rows”, Relational Database Writings 1989-1991, Addison-Wesley, 1992

[16] – Date, C. J. and Darwen, H.: “What a Database really is: Predicates and Propositions”, Relational Database Writings 1994-1997, Addison-Wesley, 1998

[17] – Date, C. J.: “A Normalization Problem”, in Relational Database Writings 1991-1994. Reading, Mass. Addison-Wesley, 1995

[18] – Date, C. J.: “An Introduction to Database Systems - 8th Edition”, Addison-Wesley, 2003

[19] – Date, C. J.: “Referential Integrity”, Proc.7th Int. Conference on Very Large Data Banks, Cannes, France (September 1981)

[20] – Date, C. J.: “A Guide to the SQL standard”, Addison-Wesley Publishing Co., Massachusetts, 1987

[21] – Date, C. J.: “An Architecture for High-Level Language Database Extensions”, SIGMOD76, (December 1975)

[22] – Delobel, C., Adiba, M.: “Bases de données et systémes relationnels”, Dunod, Paris, 1982

[23] – Hahn, H. and Rosenthal, A.: “Set Functions”, University of New Mexico Press, Albuquerque, 1948

[24] – Hall, P. O. J. and Todd, S. J. P.: “Relations and Entities” in G. M. Nijssen (ed.) Modeling in Data Base Management Systems, Amsterdam, The Netherlands: North-Holland/New York, N. Y.: Elsevier Science, 1975

[25] – Inmon, W. H.: “Effective Data Base Design”, Prentice-Hall, New Jersey, 1981

[26] – Klug, A.: “Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions”, JACM 29, No 3 (July 1982)

[27] – Kung, H. T., Lehman, P. L.: “Systolic (VLSI) arrays for relational database operations”, Proceedings of the 1980 ACM SIGMOD international conference on Management of data, May 14-16, 1980, Santa Monica, California

[28] – Loomis, M. E. S.: “The Database Book”, Macmillan, New York, 1987

[29] – Maier, D.: “The Theory of Relational Databases”, Computer Science Press, 1983

[30] – Martin, J.: “Principles of Database Management”, Prentice-Hall (1976, 1989).

[31] – Martin, J: “Computer Data-base Organization” 2nd Edition, Prentice-Hall, New Jersey, 1977

[32] – Negri, M., Pelagatti, G., and Sbattella, L.: “Formal Semantics of SQL Queries”, ACM Transactions on Database Systems 16, nº 3 (September 1991)

[33] – Ng, W. K.: “An Extension of the Relational Database Model to Incorporate Ordered Domains”, ACM, Transactions on Database Systems, Vol. 26, No. 3, September 2001.

[34] – Oxborrow, E.: “Databases and Databases Systems: Concepts and Issues”, Chartwell-Bratt 2nd Edition, 1989

[35] – Paully, G. N. and Larson, P.: "Exploiting Uniqueness in Query Optimization" in Proceedings of the International Conference on Data Engineering, pages 68-79, 1994

[36] – Pereira, J. L.: "Tecnologia de Bases de Dados", FCA Editores 1998[10]

[37] – Ramakrishnan, R. et al.: "SRQL: Sorted Relation Query Language", Tenth International Conference on Scientific and Statistical Database Management, proceedings, pages 84-95, 1998.

[38] – Ramakrishnan, R. et al: "Sequence query processing" in Proceedings of the ACM SIGMOD Conference on Management of Data, pages 430–441 (May 1994)

[39] – Reed, P.: "The Unified Modeling Language Takes Shape", DBMS 11, Nº 8 (July 1998)

[40] – Rumbaugh, J., Blaha, M., Premerlani, W., Eddy F. and Lorensen, W.: "Object-oriented Modeling and Design", Prentice-Hall, 1991

[41] – Silberschatz, A., Stonebraker, M. and Ullman, J.: "Database Systems: Achievements and Opportunities", Communications of the ACM, 34, 10, 110-120, 1991

[42] – Stanczjk, S.: "Theory and Practice of relational databases", Pitman, London, 1990

[43] – Titchmarsh, E. C.: "The Theory of Functions", Oxford University Press, 1944

[10] José Luís Pereira is a Portuguese database researcher. He is a BSc in System Engineering and Computer Science

[44] – Tsichritzis, D. C., Lochovsky, F. H.: “Data Base Management Systems”, Academic Press, Inc (London) Ltd., London, 1977

[45] – Widom, J., and Ceri, S.: “Active Database Systems: Triggers and Rules for advanced Database Processing”, San Francisco, California, Morgan Kaufmann, 1996

[46] – Cunha Viana, J.; “Classification of Skin Tumours through the Analysis of Unconstrained Images”; De Montfort University, PhD Thesis, 2009

[47] – Hand, S., Chandler,J.; “Introduction to Object Oriented Databases”, September 1998 (PDF downloaded from www.odbms.org)

[48] – Chountas, P.: “RDBMS versus ORDBMS versus OODBMS”, University of Westminster, August 2005 (PDF downloaded from www.odbms.org)

[49] – Wade, A.: “Hitting the relational Wall”, Objectivity Inc., 2005 (PDF downloaded from www.odbms.org)

[50] – Signer, B.: “Introduction to Databases Object and Object-Relational Databases”, Vrije Universiteit Brussel, June 2010 (PDF downloaded from www.odbms.org)

[51] – Baumann, P.: “Object-Oriented or Object-Relational? An Experience Report from a High-Complexity, Long-Term Case Study”, Jacobs University, July 2010 (PDF downloaded from www.odbms.org)

[52] –Grossniklaus, M., Norrie, M.: “Object Oriented Databases - lecture series (Version 2009)”, ETH Zürich, 2009, (series of PDF downloaded from www.odbms.org)

Technical references:

[t01] – Ben-Gan, I. et al: “Inside Microsoft SQL Server 2008: T-SQL Querying”, Microsoft Press, (Mar 25, 2009)

[t02] – Brust, A. J. and Forte, S.: “Programming Microsoft SQL Server 2005”, Microsoft Press, (Jul 19, 2006)

[t03] – Computer Technology Research Corp: “Directions in Database Management Systems: Selection and Implementation”, 1991

[t04] – Delaney, K. et al: “Inside Microsoft SQL Server 2005: Query Tuning and Optimization”, Microsoft Press, (Sep 26, 2007)

[t05] – Delaney, K.: “Inside Microsoft SQL Server(TM) 2005: The Storage Engine”, Microsoft Press, (Nov 8, 2006)

[t06] – Dubois, P.: “MySQL (4th Edition)”, (Sep 8, 2008)

[t07] – Dubois, P.: “MySQL Cookbook”, (Jan 27, 2007)

[t08] – Harrison, G. and Feuerstein, S.: “MySQL Stored Procedure Programming”, (Mar 28, 2006)

[t09] – INFORMIX SOFTWARE: “Informix SQL Reference Library”, (Dec 27, 1999)

[t10] – ISO/IEC 8824-1: 1998, Information technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation, 1998

[t11] – ISO/IEC 9075-*: 2003, Information technology — Database Languages — SQL (2003~2006).

[t12] – Kipp, C.: “Programming Informix SQL/4GL: A Step-By-Step Approach”, 2nd Edition, (Nov 21, 1997)

[t13] – Knight, B. et al: “Professional SQL Server 2005 Integration Services (Programmer to Programmer)”, Microsoft Press (Jan 31, 2006)

[t14] – Melnyk, R. B. and Zikopoulos, P. C.: “DB2: The Complete Reference”, (Oct 2001)

[t15] – Mullins, C. S.: “DB2 Developer's Guide”, 5th Edition, (May 21, 2004)

[t16] – MYSQL AB: “MySQL Administrator's Guide and Language Reference (2nd Edition)”, (May 7, 2006)

[t17] – Pachev, S.: “Understanding MySQL Internals”, (April 10, 2007)

Online resources:

[i01] – http://www.microsoft.com

[i02] – http://www.ieee.org

[i03] – http://www.oracle.com

[i04] – http://www.postgresql.org

[i05] – http://www.mysql.com

[i06] – http://www.codeproject.com

[i07] – http://www.odbms.org

[i08] – http://www.gemstone.com

[i09] – http://www.service-arquitecture.com

[i10] – http://www.versant.com

[i11] - http://www.agiledata.org/essays/keys.html

[i12] – http://searchsqlserver.techtarget.com/feature/Why-use-surrogate-keys

[i13] – http://www.databasejournal.com/features/mssql/article.php/3922066/SQL-Server-Natural-Key-Verses-Surrogate-Key.htm

Appendix

The annexes presents the code needed for the hotel problem (written in Visual Basic 6.0), the code for the stored procedure that calculates the amount to give to each candidate (written in Transact SQL) and the complex SQL statement that displays a running sum. It also has a non-optimized sample of image sorting by direct image analysis.

The implementation in C# of the extended stored procedure for the prototype, written for Microsoft SQL Server 2008 follows the previous code. Along with the prototype the “SqlRandom()” function was implemented to be used in extended SQL statements. A program that shows a form that accepts as input an extended syntax SQL statement and displays the resulting data is also listed. Although the main code is for the extended stored procedures, this program implements a simple interface for the extended syntax. At last, the SQL expressions that implements the solution for the case studies, both in how should be directly parsed and as argument for the extended stored procedure “RMExtension”.

The procedure written to present rooms from hotels in random order in Visual Basic 6.

The part of the procedure that is actually responsible for randomizing the output is in bold typeface and highlighted.

```
Private Sub LblConsultar_Click()
On Error GoTo LblConsultar_ClickErr

  Dim RsCon As ADODB.Recordset, RsConHot As ADODB.Recordset
  Dim RsConBase As ADODB.Recordset, DtHoje As Date
  Dim aNumAl() As Byte, NumAl As Byte, i As Byte, j As Byte
  Dim bFound As Boolean
  Dim StrLocal As String

  If Not ValidaCampos() Then
    Exit Sub
  End If

  DtHoje = Date

  StrLocal = IIf(CboLocal = "Todos", "*", CboLocal)

  If DtaATL.rsConsulta.State <> adStateOpen Then
    DtaATL.Consulta
  End If
  If DtaATL.rsConsultaHotel.State <> adStateOpen Then
    DtaATL.ConsultaHotel
  End If

  If DtaATL.rsQryConsultaBase.State <> adStateClosed Then
    DtaATL.rsQryConsultaBase.Close
  End If

  DtaATL.QryConsultaBase CByte(TbxEstrelasMin), CByte(TbxEstrelasMax),
CCur(TbxPrecoMin), CCur(TbxPrecoMax)

  If StrLocal <> "*" Then
    DtaATL.rsQryConsultaBase.Filter = "Localizacao = '" & StrLocal &
"'"
  End If

  With DtaATL
    Set RsConBase = .rsQryConsultaBase
    Set RsCon = .rsConsulta
    Set RsConHot = .rsConsultaHotel
  End With

  If RsConBase.EOF Then  'Empty consulta
    MsgBox "Não existem hoteis nessas condições", vbInformation, "Sem
hoteis"
    Exit Sub
  Else
    RsCon.Filter = "NumPosto = " & NumPosto & " AND NumPos = " &
NumPos & " AND Data = #" & DtHoje & "#"

    If RsCon.EOF Then
```

```
      NOrdem = 1
    Else
      RsCon.MoveLast
      NOrdem = RsCon("Nordem") + 1
    End If
    RsCon.Filter = ""
    'RsConBase.MoveLast
    'RsConBase.MoveFirst

    With RsCon
      .AddNew
      .Fields("NumPosto") = NumPosto
      .Fields("NumPos") = NumPos
      .Fields("Data") = DtHoje
      .Fields("Nordem") = NOrdem
      .Fields("Localizacao") = CboLocal.Text
      .Fields("NEstrelasMax") = TbxEstrelasMax
      .Fields("NEstrelasMin") = TbxEstrelasMin
      .Fields("PrecoMin") = TbxPrecoMin
      .Fields("PrecoMax") = TbxPrecoMax
      .Fields("CodNacionalidade") =
CboNacPais.ItemData(CboNacPais.ListIndex)
      .Fields("CodProviniencia") =
CboProvPais.ItemData(CboProvPais.ListIndex)
      .Fields("CodDestino") = CboDestino.BoundText
      .Fields("CodMeioTransporte") = CboMeioTransporte.BoundText
      .Fields("NPessoas") = TbxNumPessoas
      .Update
    End With
    Randomize
    ReDim aNumAl(0 To RsConBase.RecordCount)
    i = 0
    Do While Not RsConBase.EOF
      Do
        bFound = False
        NumAl = Rnd * RsConBase.RecordCount
        For j = 0 To i
          bFound = bFound Or (aNumAl(j) = NumAl)
        Next
      Loop While bFound
      aNumAl(i) = NumAl
      i = i + 1
      With RsConHot
        .AddNew
        .Fields("NumPosto") = NumPosto
        .Fields("NumPos") = NumPos
        .Fields("Data") = DtHoje
        .Fields("Nordem") = NOrdem
        .Fields("CodHotel") = RsConBase("CodHotel")
        .Fields("CodQuarto") = RsConBase("CodQuarto")
        .Fields("NumAleatorio") = NumAl
        .Update
      End With
      RsConBase.MoveNext
    Loop
  End If
  Set RsConHot = Nothing
  Set RsCon = Nothing
  Set RsConBase = Nothing
```

```
  If DtaATL.rsQryListaHoteis.State <> adStateClosed Then
    DtaATL.rsQryListaHoteis.Close
  End If
  DtaATL.QryListaHoteis NumPosto, NumPos, DtHoje, NOrdem

  FrmListaHoteis.Show
  FrmConsulta.Hide

LblConsultar_ClickExit:
  Exit Sub

LblConsultar_ClickErr:
  MsgBox "Erro n°" & Err & vbCrLf & Err.Description, vbCritical,
"LblConsultar Click"
  Resume LblConsultar_ClickExit

End Sub
```

The SQL Stored Procedure that calculates the scholarships
Comments are in Portuguese. The budget control is in bold typeface and highlighted

```
PROCEDURE [dbo].[CalcNormal]( @AnoProcesso int)

	SET NOCOUNT ON
	DECLARE @Bolsa Money, @LimOrc Money, @TotManual Money
	DECLARE @NumeroProcesso int, @Manual bit, @Capitacao money,
@PropinaAnual Money, @Deslocado bit, @NMeses smallint
	DECLARE @BolsasAuferidas Money, @PedidoComplemento Money,
@PedidoTransporte Money
	DECLARE @RCand CURSOR
	BEGIN TRAN
/* Se alterar o cursor, tenho de alterar estas procedures em
conformidade */
	EXECUTE QUpdClearBolsas @AnoProcesso
	EXECUTE QUpdClearRejeicao @AnoProcesso
	EXECUTE QUpdSetRejeicao @AnoProcesso
	SELECT @TotManual =  SUM(Total) FROM QGrpTotalBolsasManuais
WHERE AnoProcesso = @AnoProcesso
	SET @LimOrc = dbo.LimiteOrcamental(@AnoProcesso) - CASE WHEN
@TotManual IS NULL THEN 0 ELSE @TotManual END
/* Os não processados */
	SET @RCand = CURSOR FAST_FORWARD FOR SELECT NumeroProcesso,
Manual, Capitacao, PropinaAnual, Deslocado, NMeses, BolsasAuferidas,
PedidoComplemento,PedidoTransporte FROM QryCapitacaoParaProcessamento
WHERE EstadoProcesso < 2 AND AnoProcesso = @AnoProcesso
/* Todos */
/*    SET @RCand = CURSOR FAST_FORWARD FOR SELECT NumeroProcesso,
Manual, Capitacao, PropinaAnual, Deslocado, NMeses, BolsasAuferidas,
PedidoComplemento, PedidoTransporte FROM QryCapitacaoParaProcessamento
WHERE AnoProcesso = @AnoProcesso */
	OPEN @RCand
	FETCH NEXT FROM @RCand INTO @NumeroProcesso, @Manual,
@Capitacao, @PropinaAnual, @Deslocado, @NMeses, @BolsasAuferidas,
@PedidoComplemento, @PedidoTransporte
	WHILE (@@FETCH_STATUS = 0) AND @LimOrc > 0
	BEGIN
		IF  @Manual = 0
		BEGIN
			SET @PedidoComplemento = CASE WHEN @Deslocado <> 0
THEN
dbo.Menor(@PedidoComplemento,dbo.MaxComplementoBolsa(@AnoProcesso))
ELSE 0 END
			SET @PedidoTransporte = CASE WHEN @Deslocado <> 0
THEN dbo.Menor(@PedidoTransporte,dbo.MaxSubTrans(@AnoProcesso)) ELSE 0
END
			SET @Bolsa = dbo.CalcBolsa(@Capitacao, @PropinaAnual,
@NMeses,@AnoProcesso)
			SET @Bolsa = @Bolsa - dbo.Maior(@Bolsa +
@BolsasAuferidas / @NMeses - dbo.MaximaBolsa(@AnoProcesso),0)
			IF @Bolsa > 0
			BEGIN
				SET @Bolsa = ROUND((@Bolsa +
@PedidoComplemento) * 10 + 0.5,0)/10
				SET @LimOrc = @LimOrc - (@Bolsa * @NMeses) -
@PedidoTransporte
```

```
                        UPDATE ProcessoAnual SET BolsaCalculada =
@Bolsa WHERE NumeroProcesso = @NumeroProcesso AND AnoProcesso =
@AnoProcesso
                    END
                END
                FETCH NEXT FROM @RCand INTO @NumeroProcesso, @Manual,
@Capitacao, @PropinaAnual, @Deslocado, @NMeses, @BolsasAuferidas,
@PedidoComplemento, @PedidoTransporte
        END
        CLOSE @RCand
        DEALLOCATE @RCand
        /* Passar novos e tratados, sem excepção, a processados */
        UPDATE ProcessoAnual SET EstadoProcesso = 2 WHERE EstadoProcesso
< 2 AND AnoProcesso = @AnoProcesso
        /* Passar todos os que são falta de documentos, minutas, etc a
"Novos" para serem tratados nas instituições */
        --UPDATE ProcessoAnual SET EstadoProcesso = 0 WHERE AnoProcesso
= @AnoProcesso AND CodigoIndeferimento IN (2,19,22,23)
        COMMIT TRAN
        RETURN
```

The SQL for the running sum

```
SELECT B.DtBudget, B.MonthValue, Accumul = (SELECT SUM(MonthValue)
      FROM Budget BB
      WHERE BB.DtBudget BETWEEN '1-Jan-2010' AND B.DtBudget)
FROM Budget B
WHERE B.DtBudget BETWEEN '1-Jan-2010' AND '31-Dec-2010'
ORDER BY B.DtBudget
```

Listing of RMextensions.cs

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;

public partial class StoredProcedures
{

    [Microsoft.SqlServer.Server.SqlProcedure]
    public static void RMExtension(string strSQL)
    {
        // Test if we are under an SQL server connection
        if (!SqlContext.IsAvailable)
            return;

        // First split the string in statments to "remount" and
execute std SQL
        SqlConnection conn = new SqlConnection("context
connection=true");

        // save the original SQL in pieces
        string[] SQL = RMEUtils.SplitSQL(strSQL);

        string SQLText = "";
        string[] alias = { };
        string[] fnames = { };
        string whereclause = "";

        for (int i = 0; i < SQL.Length; i++)
        {
            string clause = SQL[i];
            // keep the where expression for further processing later
            if (clause.ToUpper().StartsWith("WHERE"))
                whereclause =
SQL[i].Substring(SQL[i].ToUpper().IndexOf("WHERE") +
"WHERE".Length).Trim();

            // get the field/aliases names for each field
            if (clause.ToUpper().StartsWith("SELECT"))
            {
                // keep the alias and the original field names
                string selstr =
SQL[i].Substring(SQL[i].ToUpper().IndexOf("SELECT") +
"SELECT".Length).Trim();
                alias = RMEUtils.GetFieldNames(selstr);
                fnames = selstr.Split(',');
            }

            // Search for ordering funcions inside the ORDER BY clause
            if (clause.ToUpper().StartsWith("ORDER BY"))
            {
                // we have a ordering function over here
                // Remove the Order By
                clause =
clause.Substring(clause.ToUpper().IndexOf("ORDER BY") + "ORDER
BY".Length).Trim();
                // explode the fields
```

```
                string[] fields = clause.Split(',');
                string[] fOrder = { };
                clause = "";
                // find any extended stored procedures with the same
name and fill the array with their names
                for (int j = 0; j < fields.Length; j++)
                {
                    string fld = fields[j].Trim() + ", ";
                    if (fields[j].ToUpper().Contains("(")) //found!
                    {
                        fld = fields[j].Trim().Substring(0,
fields[j].IndexOf("("));
                        if (SQLUtils.IsExtSP(fld, conn))
                        {
                            Array.Resize(ref fOrder, fOrder.Length +
1);
                            fOrder[fOrder.Length - 1] = fld;
                            fld = "";
                        }
                        else
                            fld += ", ";
                    }
                    clause += fld;
                }
                // remount the clause without the Order Function(s)
                if (clause != "")
                    clause = "ORDER BY " + clause.Substring(0,
clause.Length - 2);
                SQL[i] = clause;

                //prepare the EXEC statment for the order stored
procedures
                // Before the EXEC statments get the metadata for the
new prepared SQL
                for (int j = 0; j < fOrder.Length; j++)
                {
                    SQL[0] = SQL[0].Replace("'", "''");
                    SQL[SQL.Length - 1] = SQL[SQL.Length -
1].Replace("'", "''");
                    SQL[0] = "EXEC " + fOrder[j] + " '" + SQL[0];
                    SQL[SQL.Length - 1] += "'";
                }
            }
        }

        // Build the standard SQL to be runned by the engine
        SQLText = RMEUtils.remountSQL(SQL);

        // for the metadata to work fine, we need to remove the 'EXEC'
out of the select statement
        string SQLstd = SQLText;
        while (SQLstd.ToUpper().Contains("EXEC"))
        {
            SQLstd = SQLstd.Substring(SQLstd.IndexOf("'") + 1,
SQLstd.LastIndexOf("'") - SQLstd.IndexOf("'") - 1).Trim();
            SQLstd = SQLstd.Replace("''", "'");
        }

        // get the data out of the engine
```

```
        DataTable dt = null;
        try
        {
            dt = SQLUtils.lData(SQLText, conn);
        }
        catch (Exception e)
        {
            dt = null;
            SqlContext.Pipe.Send(e.Message);
        }
        if (dt != null)
        {

            // prepare the extension fields
            double[] runsum = new double[] { };
            for (int i = 0; i < fnames.Length; i++)
            {
                string field = fnames[i];
                if (field.ToUpper().Contains("ACCUMUL"))
                    Array.Resize(ref runsum, runsum.Length + 1);
            }

            for (int i = 0; i < runsum.Length; i++)
                runsum[i] = 0;

            // let us process the data
            // Evaluate the new keywords modifying the resulting
datatable
            for (int rc = 0; rc < dt.Rows.Count; rc++)
            {
                int i = 0;
                DataRow dr = dt.Rows[rc];
                foreach (DataColumn dc in dt.Columns)
                {
                    // find out what alias is it
                    int fpos = RMEUtils.GetFieldPos(alias, dc);
                    if (fpos != -1)
                    {
                        if
(fnames[fpos].ToUpper().Contains("ACCUMUL"))
                        {
                            dr.BeginEdit();
                            dr[dc.ColumnName] =
double.Parse(dr[dc.ColumnName].ToString()) + runsum[i];
                            dr.AcceptChanges();
                            dr.EndEdit();
                            runsum[i] =
double.Parse(dr[dc.ColumnName].ToString());
                            i++;
                        }
                        if
(fnames[fpos].ToUpper().Contains("ORDERNEXT"))
                        {
                            dr.BeginEdit();
                            if (rc != dt.Rows.Count - 1)
                                dr[dc.ColumnName] = dt.Rows[rc +
1][dc.ColumnName];
                            dr.AcceptChanges();
                            dr.EndEdit();
```

```
                        }
                    }
                }

            }

            //PREV must be treated upside down
            for (int rc = dt.Rows.Count - 1; rc >= 0; rc--)
            {
                DataRow dr = dt.Rows[rc];
                foreach (DataColumn dc in dt.Columns)
                {
                    int fpos = RMEUtils.GetFieldPos(alias, dc);
                    if (fpos != -1)
                        if
(fnames[fpos].ToUpper().Contains("ORDERPREV"))
                        {
                            dr.BeginEdit();
                            if (rc != 0)
                                dr[dc.ColumnName] = dt.Rows[rc -
1][dc.ColumnName];
                            dr.AcceptChanges();
                            dr.EndEdit();
                        }
                }
            }

            DataView dv = dt.DefaultView;

            // see if there are any WHERE Clause on the extension
fields
            dv.RowFilter = "";
            if (!String.IsNullOrEmpty(whereclause))
                dv.RowFilter = RMEUtils.BuidWhere(fnames, alias,
whereclause);

            DataTable dto = dv.ToTable();

            // treat last item of ORDERNEXT and first item of
ORDERPREV
            if (dto.Rows.Count > 0)
            {
                foreach (DataColumn dc in dto.Columns)
                {
                    int fpos = RMEUtils.GetFieldPos(alias, dc);
                    if (fpos != -1)
                    {
                        if
(fnames[fpos].ToUpper().Contains("ORDERNEXT"))
                        {
                            DataRow dr = dto.Rows[dto.Rows.Count - 1];
                            dr.BeginEdit();
                            dr[dc.ColumnName] = DBNull.Value;
                            dr.AcceptChanges();
                            dr.EndEdit();
                        }
                        if
(fnames[fpos].ToUpper().Contains("ORDERPREV"))
                        {
```

```
                                DataRow dr = dto.Rows[0];
                                dr.BeginEdit();
                                dr[dc.ColumnName] = DBNull.Value;
                                dr.AcceptChanges();
                                dr.EndEdit();
                            }
                        }
                    }
                }

                // get the fields sqltypes
                SqlMetaData[] flds = SQLUtils.GetMetaData(SQLstd, conn);

                // output the data to the user.
                SqlDataRecord rec = new SqlDataRecord(flds);
                SqlContext.Pipe.SendResultsStart(rec);
                foreach (DataRow dr in dto.Rows)
                {
                    for (int i = 0; i < dto.Columns.Count; i++)
                    {
                        rec.SetValue(i, dr[i]);
                    }
                    SqlContext.Pipe.SendResultsRow(rec);
                }
                SqlContext.Pipe.SendResultsEnd();
            }
        }
};
```

Listing of the RMEUtils.cs
It implements both RMEUtils Class and SQLUtils Class

```
using System;
using System.Data;
using System.Collections;
using System.Collections.Generic;
using System.Text;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;

class RMEUtils
{
    // keywords to be used inside this class
    private static readonly string[] keyword = { "SELECT", "FROM",
"WHERE", "GROUP BY", "HAVING", "ORDER BY" };
    private static readonly string[] keyext = { "ORDERPREV",
"ORDERNEXT", "ACCUMUL" }; //these are the only extensions implemented


    // Split the SQL string into a string array of clauses
    public static string[] SplitSQL(string strSQL)
    {
        string[] res = new string[] { };
        int i = 0;
        foreach (string k in keyword)
        {
            if (strSQL.ToUpper().Contains(k))
            {
                Array.Resize(ref res, res.Length + 1);
                res[i++] =
strSQL.Substring(strSQL.ToUpper().IndexOf(k)).Trim(); // fill the
string with the keyword forward
                strSQL = strSQL.Remove(0,
strSQL.ToUpper().IndexOf(k)).Trim(); // clean the begining of the
original string
                for (int j = 0; j < i - 1; j++) // clean the previous
substrings
                {
                    if (res[j].ToUpper().IndexOf(k) > -1)
                        res[j] =
res[j].Remove(res[j].ToUpper().IndexOf(k)).Trim();
                }
            }
        }
        return res;
    }

    // remount the SQL string, dealing with the extension fields
    // the result is a SQL string without any extended keywords so it
can be interpreted by the engine
    public static string remountSQL(string[] aSQL)
    {
        string res = "";
        foreach (string s in aSQL)
        {
            if (!String.IsNullOrEmpty(s.Trim()))
                res += ParseSQLComponent(s) + " ";
```

```
        }
        return res.Trim();
    }

    // parse every clause
    private static string ParseSQLComponent(string strSQLComp)
    {
        string res = strSQLComp;
        foreach (string k in keyword)
        {
            if (strSQLComp.ToUpper().Contains(k))
            {
                switch (k)
                {
                    case "SELECT":
                        // beware of EXEC statments
                        string start = strSQLComp.Substring(0,
strSQLComp.ToUpper().IndexOf(k));
                        // clean the SELECT clause
                        res =
CleanSelect(strSQLComp.Substring(strSQLComp.ToUpper().IndexOf(k) +
k.Length));
                        if (!String.IsNullOrEmpty(res))
                            res = start + k + " " + res;
                        break;
                    case "FROM":
                    case "WHERE":
                    case "GROUP BY":
                    case "HAVING":
                    case "ORDER BY":
                        // clean the field list from extended keywords
                        res =
CleanFieldList(strSQLComp.Substring(k.Length));
                        if (!String.IsNullOrEmpty(res))
                            res = k + " " + res;
                        break;
                    default:
                        break;

                }
                break;
            }
        }
        return res;
    }

    private static string CleanSelect(string strSQLComp)
    {
        // it is assumed that what was sent here was the SELECT clause
w/o the keyword
        // fields in the extension must be replaced by the original
field name
        string result = "";
        string s = strSQLComp.Trim();
        string[] s1 = s.Split(',');
        //get a array of field names
        string[] falias = GetFieldNames(strSQLComp);

        for (int i = 0; i < s1.Length; i++)
```

```
        {
            string sf = s1[i];
            string lsf = sf.Trim();
            string strFld = lsf;
            foreach (string k in keyext)
            {
                if (lsf.ToUpper().Contains(k))
                {
                    int p1Pos = lsf.IndexOf("(");
                    int p2Pos = lsf.IndexOf(")");
                    strFld = lsf.Substring(p1Pos + 1, p2Pos - p1Pos -
1).Trim();

                    // leave the field with its alias if he has one
                    // or create a suitable alias
                    if (p2Pos < lsf.Length - 1)
                        strFld += lsf.Substring(p2Pos + 1);
                    else
                        strFld += " AS " + falias[i];
                    break;
                }
            }
            result += strFld + ", ";
        }

        if (result.Contains(","))
            result = result.Substring(0,
result.LastIndexOf(",")).Trim();
        return result;
    }

    // clean the clause from any extended keywords
    private static string CleanFieldList(string strSQLComp)
    {
        // The string arrives without the clause keyword
        string result = "";
        string[] s1 = strSQLComp.Trim().Split(',');

        foreach (string sf in s1)
        {
            string lsf = sf.Trim();
            string strFld = lsf;
            foreach (string k in keyext)
            {
                if (lsf.ToUpper().Contains(k))
                {
                    //remove it
                    strFld = "";
                    break;
                }
            }
            if (!String.IsNullOrEmpty(strFld))
                result += strFld + ", ";
        }

        if (result.Contains(","))
            result = result.Substring(0,
result.LastIndexOf(",")).Trim();
        return result;
```

```
    }

    //get an array of suitable field names from a SELECT clause
    public static string[] GetFieldNames(string strSQL)
    {
        //only the select clause will enter here
        //gets the output field names from a select clause

        string s = strSQL.Trim();

        // let's determinate the fields names
        string[] s1 = s.Split(',');
        string[] res = new string[s1.Length];

        int i = 0;
        foreach (string sf in s1)
        {
            int spPos = 0;
            if (sf.Contains(" "))
            {
                spPos = sf.LastIndexOf(" ");
                if (sf.Contains("]"))
                    if (spPos > sf.IndexOf("]"))
                        res[i] = sf.Substring(sf.LastIndexOf(" "));
                    else
                        res[i] = sf.Substring(sf.IndexOf("[") + 1,
sf.IndexOf("]") - sf.IndexOf("[") - 1);
                else
                    res[i] = sf.Substring(spPos + 1);
            }
            else
                if (sf.Contains("["))
                    res[i] = sf.Substring(sf.IndexOf("[") + 1,
sf.IndexOf("]") - sf.IndexOf("[") - 1);
                else
                    res[i] = sf;
            //deal with the xxx(<fld>) fields  * in <fld> ==> record
e.g. COUNT(*) =>CountOfRecord
            if (res[i].Contains("("))
            {
                res[i] = res[i].Trim();
                string fname = res[i].Substring(res[i].IndexOf("(") +
1, res[i].IndexOf(")") - res[i].IndexOf("(") - 1).Replace("*",
"Record");
                res[i] = res[i].Substring(0, 1).ToUpper() +
res[i].Substring(1, res[i].IndexOf("(") - 1).Trim().ToLower() + "Of" +
fname.Substring(0, 1).ToUpper() + fname.Substring(1).ToLower();
            }
            i++;
        }

        return res;
    }

    //gets the index of a certain datacolumn inside an index of fields
    public static int GetFieldPos(string[] fields, DataColumn dc)
    {
        int res = -1;  //defaults to "not found"
        for (int i = 0; i < fields.Length; i++)
```

```
        {
            if (dc.ColumnName.ToUpper() == fields[i].ToUpper())
            {
                res = i;
                break;
            }
        }

        return res;
    }

    // build a WHERE string replacint the extensions with the actual
field name/alias
    public static string BuidWhere(string[] fields, string[] alias,
string wclause)
    {
        string res = wclause;
        for (int i = 0; i < keyext.Length; i++)
        {
            while (res.ToUpper().Contains(keyext[i].ToUpper()))
            {
               //find the string in the fields and replace it by their
alias
                int p1=res.ToUpper().IndexOf(keyext[i]);
                int p2 = res.IndexOf('(', p1);
                int p3 = res.IndexOf(')',p1);
                string toreplace = res.Substring(p1, p3 - p1 + 1);
                string field = res.Substring(p2 + 1, p3 - p2 -
1).Trim();
                for (int j = 0; j < fields.Length; j++)
                {
                    if (fields[j].ToUpper().Contains(field.ToUpper())
&& fields[j].ToUpper().Contains(keyext[i].ToUpper()))
                    {
                        res = res.Replace(toreplace, alias[j]);
                        break;
                    }
                }
            }
        }
        return res;
    }
}
```

```
class SQLUtils
{
    //gets the data from an SQL String. It is assumed that the SQL
string is well formed
    public static DataTable lData(string strSQL, SqlConnection conn)
    {
        // Execute the base sql string and get the data from it
        // assumes that the SQL String is well formed
        DataTable Res = new DataTable();
        try
        {
            conn.Open();
            SqlCommand cmd = new SqlCommand(strSQL, conn);
            SqlDataReader drd = cmd.ExecuteReader();
            Res.Load(drd);
        }
        finally
        {
            conn.Close();
        }

        return Res;
    }

    //finds out if a certain name correponds to an extended store
procedure
    //it is used to determinate if an Ordering Function is implemented
as a procedure
    public static bool IsExtSP(string spName, SqlConnection conn)
    {
        bool res = false;
        DataTable dt = new DataTable();
        conn.Open();
        SqlCommand cmd = new SqlCommand("SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID('" + spName + "') AND type in ('P','PC')",
conn);
        try
        {
            SqlDataReader drd = cmd.ExecuteReader();
            dt.Load(drd);
            res = (dt.Rows.Count > 0);
        }
        catch
        {
            res = false;
        }
        conn.Close();
        return res;
    }

    //This function was created from a sample located at

//http://www.java2s.com/Tutorial/CSharp/0560__ADO.Net/howtoreadatables
chema.htm
    //It's not the original. It has been fully adapted to work with
this prototype

    // gets an array of metadata from an SQl string
```

```
    public static SqlMetaData[] GetMetaData(string strSQL,
SqlConnection conn)
    {
        SqlMetaData[] res = new SqlMetaData[] { };
        conn.Open();
        SqlCommand cmd = new SqlCommand(strSQL, conn);
        SqlDataReader sdr =
cmd.ExecuteReader(CommandBehavior.SchemaOnly);
        DataTable dtStruc = sdr.GetSchemaTable();

        foreach (DataRow dr in dtStruc.Rows)
        {
            Array.Resize(ref res, res.Length + 1);
            SqlDbType dbt = (System.Data.SqlDbType)dr["ProviderType"];
            switch (dbt)
            {
                case SqlDbType.Binary:
                case SqlDbType.Char:
                case SqlDbType.NChar:
                case SqlDbType.NVarChar:
                case SqlDbType.VarBinary:
                case SqlDbType.VarChar:
                    res[res.Length -1] = new
SqlMetaData(dr["ColumnName"].ToString(), dbt,
long.Parse(dr["ColumnSize"].ToString()));
                    break;

                case SqlDbType.DateTime2:
                case SqlDbType.DateTimeOffset:
                case SqlDbType.Time:
                    res[res.Length - 1] = new
SqlMetaData(dr["ColumnName"].ToString(), dbt, 255,
byte.Parse(dr["NumericScale"].ToString()));
                    break;

                case SqlDbType.Decimal:
                    res[res.Length - 1] = new
SqlMetaData(dr["ColumnName"].ToString(), dbt,
byte.Parse(dr["NumericPrecision"].ToString()),
byte.Parse(dr["NumericScale"].ToString()));
                    break;

                default:
                    res[res.Length - 1] = new
SqlMetaData(dr["ColumnName"].ToString(), dbt);
                    break;
            }
        }
        sdr.Close();
        conn.Close();
        return res;
    }
}
```

Algorithm for the random sort (technology independent)

We assume that an array structure has these primitives:

- A.Add(e) – Adds element e to the array
- A.Delete(j) – Deletes the element in the j position from the array
- A.Length – returns the number of elements from the array. Returns 0 on an empty array.
- A[j] – Accesses the element of the array in the j position
- New Array() – creates an empty array

We assume also that the array is zero-based, i.e., it has n elements numbered from 0 to n-1.

We also assume that random number generator function exists

- Rnd() – Generates a random number in the interval [0..1[

```
Proc Random( Array A) returns Array
        R = new Array()
        While A.Length > 0 {
                Pos = Rnd() * A.Length
                R.Add(A[Pos])
                A.Delete(Pos)
        }
        Return R
End Proc
```

This algorithm is convergent since the condition in the while loop will be true at some point in time or if the input array is already empty. There is not any situation where Array.Length can be less than zero, by definition.

Space and time complexity analysis for the random sort algorithm

First, the algorithm will be analyzed considering the operation of deletion of the row (scenario 1). After, the array element will not be deleted but, instead, marked for deletion, improving the time albeit degrading the space (scenario 2). Finally, the scenario where an optimal number of elements are marked for deletion before the actual deletion it will be considered along with its impact in time and space (scenario 3).

This set of analysis is supposed to be done in ideal conditions. Therefore, it is assumed that there is infinite allocation space, all is done in memory and the random access is immediate. The input/output operations are considered to have constant times.

Space Complexity Analysis (scenario 1).

Since this is the machine independent algorithm, pointer allocation space was not taken in account since it was been considered irrelevant. Also it is only being studied the space occupied by the procedure itself, not considering nothing of the calling process. The results of this space analysis, therefore, must be added to the caller process whatever this process is.

The process receives an array with n elements of size s, so it takes ns space at start.

Next, a new variable (empty array) is initialized with no size at all.

Since two arrays are used and the length of each one is needed, it is assumed that two variables are needed for that effect. The size for this variable type will be “l”.

Then a cycle with n iterations is made. In this cycle we have a comparison, a calculation, a random access to one array and two other array operations.

The comparison doesn’t take any extra space because the result is used but never stored.

The calculation takes the result of the Rnd() function, multiplies by the length of the Array and stores the result in a variable. So we need for the intermediate calculation the size of the Rnd() function and also the final result of the calculation. Pos is assumed to in the range [0..n-1], so with a Rnd() function that generates real numbers in the range [0..1[will be suitable to multiply by the number of elements of the array. When n is multiplied by a number in the range [0..1[it will result in a real number between [0..n[. Converting to an integer will result in an integer on the range [0..n-1] which is the range for the position variable.

Next we need to access the input array at a certain position (although this operation takes time, this analysis only is concerned with the space used). This operation doesn't take any extra space.

Since all the variables must be defined beforehand (as most programming languages requires), the procedure must reserve space for the input array and for four variables to hold numbers (e.g. integers), resulting in the following space: ns + 4l.

The new empty (with no space used) is created and then the cycle begins.

At this point, in the k^{th} iteration we have n-k+1 elements in the input array, k -1 elements in the result array and one array element in a temporary variable with the length of s each.

(n-k+1)+(k-1) = n

The element is then appended to the result array resulting in n-k+1 in the input array, k in the result array and one array element in a temporary variable.

(n-k+1)+k = n+1

Then a deletion of the element of the input array frees the space of one element, before the end of the k^{th} iteration the input array will hold n-k elements. (n-k+k = n)

A new iteration begins.

When k reaches n, the last element is going to be copied thus having on the beginning of the iteration n-n+1 = 1 element on the input array and n-1 elements on the result array. The element is copied to the result and deleted from the input array and the cycle ends.

At the end the resulting array is returned. At this point the input array holds no elements and the resulting array holds n elements with (ns + 4l) of space occupied (the same as in the beginning)

Space occupied step by step:

Step	Space occupied
`Proc Random( Array A) returns Array`	ns + 4l (space is already reserved)
`R = new Array()`	ns + 0s + 4l
`While A.Length > 0 {`	(no changes)
`Pos = Rnd() * A.Length`	(n – k + 1)s + (k – 1)s + 4l (at the k^{th} iteration)
`R.Add(A[Pos])`	(n – k + 1)s + ks + 4l
`A.Delete(Pos)`	(n – k)s + ks + 4l
`}`	(no changes)
`Return R`	(no changes)
`End Proc`	(no changes)

With n as the number of the input array elements, k the iteration number, s the size of one array element and l the size of a variable to hold the length (two), the position being treated and for the generated random number.

So, the maximum value for the space occupied is given by the expression in step 5 [(n – k + 1)s + ks + 4l] than can be simplified as [(n + 1)s + 4l].

The occupation of space is, therefore, linear depending only on the number of the elements of the input array. The space occupied is given by the expression (n+1)s+4l.

For large arrays the term 4l can be neglected, so the expression for the space occupied by the procedure can be displayed as a function of the number of elements of the array and the size of each one.

The expression of the space occupied by this scenario is, therefore:

$$S \approx (n+1)s$$

Where n is the number of elements of the array and s is the size of each element.

Time Complexity Analysis (scenario 1)

In the time complexity analysis the most important and time consuming operations are the reading and writing the elements of the array, since on its implementation are in fact I/O operations. For this reason we can consider a residual time for other operations than those of I/O. Since some operations besides the I/O operation take place inside the loop the time used is a multiple of that residual operation time. Residual time is noted as “r”. For the purpose of simplification it is considered one residual operation for the calling/return of the procedure as well as another residual operation inside the cycle resuming all operations of test/jump/storing values other than array operations.

In the addition operation it is considered Tw as the time needed append an element to the array or to write an element in some other position within the array.

For this purpose we will consider the time to access and read an element is different from the time to write, although both are I/O operations, because it is considered to exists ideal random access. This time will be denoted as Tr.

In the deletion operation, it is considered that the array is “moved down” from the element that is being deleted up to the top of the array. Since the element is picked randomly we can assume that every element has the same probability of being chosen, so we will consider the uniform distribution as the distribution followed by the random function.

As assumed, the uniform distribution of n elements in the form of $U_{(0,n)}$ has the expected value of ½ * (n – 0) = n/2. Each element has 1/(n – 0) = 1/n probability of being chosen.

At iteration k where we have (n-k+1) elements in the input and one is deleted, the expected value of the position chosen is therefore (n-k+1)/2 and consequently (n-k+1)/2 elements will be moved downwards in the array using Tw time each.

Step	Time taken
`Proc Random( Array A) returns Array`	0
`R = new Array()`	r
`While A.Length > 0 {`	0
`Pos = Rnd() * A.Length`	r
`R.Add(A[Pos])`	Tw + Tr
`A.Delete(Pos)`	Tw(n – k + 1)/2
`}`	0
`Return R`	0
`End Proc`	0

The expression that gives us the time consumed by the algorithm (T_t) is therefore:

$$T_t = r + \sum_{k=1}^{n}(r + T_w + T_r + \frac{(n-k+1)}{2}T_w)$$

Simplifying the expression:

$$T_t = r + \sum_{k=1}^{n}(r + T_w + T_r + \frac{nT_w - kT_w + T_w}{2})$$

The constant terms relative to k are multiples of n

$$T_t = r + nr + nT_w + nT_r + \frac{n^2T_w}{2} + \frac{nT_w}{2} - \sum_{k=1}^{n}\left(\frac{kT_w}{2}\right)$$

$$T_t = r + nr + nT_w + nT_r + \frac{n^2T_w}{2} + \frac{nT_w}{2} - \frac{T_w}{2}\sum_{k=1}^{n}k$$

The expression $\sum_{k=1}^{n} k$ represent the sum of the first n integers and its result is given by the formula “n(n +1)/2” so the whole expression takes the form of:

$$T_t = r + nr + nT_w + nT_r + \frac{n^2 T_w}{2} + \frac{nT_w}{2} - \frac{T_w}{2}\frac{n(n+1)}{2}$$

Putting all the Tw in evidence and developing the expression

$$T_t = r + nr + nT_r + T_w(n + \frac{n^2}{2} + \frac{n}{2} - \frac{(n^2+n)}{4})$$

$$T_t = r + nr + nT_r + T_w \frac{n^2 + 5n}{4}$$

$$T_t = r + nr + nT_r + \frac{T_w(n^2 + 5n)}{4}$$

$$T_t = r(n+1) + nT_r + \frac{T_w(n^2 + 5n)}{4}$$

The term “r(2n + 1)” that represents all the residual time will be placed in the equation as R representing the full residual time took by the operations other than I/O. The final expression is, arranged to present the residual at the end is

$$T_t = \frac{T_w(n^2 + 5n)}{4} + nT_r + R$$

With T_t as total time, Tw as the time to write an element of the array, Tr the time to write an element of the array, R the whole residual time and n the number of array elements.

For a large numbers, the expression dependent on Tw grows with the square of n whilst the residual grows linearly we can ignore that term. The time to read can be expressed as the time to write (Tw) less some differential noted as δ. So

$$T_w = T_r + \delta$$

Replacing Tr with Tw-d the total time is represented as

$$T_t = \frac{T_w(n^2 + 5n)}{4} + n(T_w - \delta) + R$$

$$T_t = \frac{T_w(n^2 + 5n)}{4} + nT_w - n\delta + R$$

The term $n\delta$ will also be considered as residual for large arrays since the term $T_w n^2$ is much larger than the one being ignored. Therefore it is going to be included in the R term.

Putting, again, Tw in evidence, the expression results in:

$$T_t = T_w \frac{(n^2 + 9n)}{4} + R$$

Therefore, for large arrays the full execution time can be given as a function of the time to perform a write operation and the number as elements as:

$$T_t \approx T_w \frac{(n^2 + 9n)}{4}$$

Space Complexity Analysis (scenario 2)

In this scenario, the array element is not deleted but marked for deletion. Since all the elements are to be deleted from the source array, there is no need to proceed to an individual deletion but delete the whole array at the end.

All the assumptions from the previous scenario are applicable. The difference is in the space not released by the deletion.

After the beginning of the cycle, in the k^{th} iteration we have n elements in the input array and k-1 elements in the result array. (n+k-1)

Then the element is appended to the result array so now it holds k elements. The space occupied is now n+k elements

Then the source element is marked for deletion. This operation takes no space.

A new iteration begins.

At the end the resulting array is returned. At this point both the input array and the resulting array hold n elements.

Space occupied step by step:

Step	Space occupied
`Proc Random( Array A) returns Array`	ns + 4l
`R = new Array()`	ns + 0s + 4l
`While A.Length > 0 {`	(no changes)
`Pos = Rnd() * A.Length`	ns + (k-1)s + 4l (at the k^{th} iteration)
`R.Add(A[Pos])`	ns + ks + 4l
`A.Delete(Pos)`	(no changes)
`}`	(no changes)
`Return R`	(no changes)
`End Proc`	(no changes)

So, the maximum value for the space occupied is given by the expression in step 5 [ns +ks + 4l] than can be simplified as [(n + k)s + 4l].

Since n,s and l are constants the expression maximizes with k. Since is linearly dependent of k the expression for k=n (which is the maximum) is: 2ns + 4l

The occupation of space is, therefore, linear depending only on the double of the elements of the input array.

For large arrays the term 4l can be neglected, so the expression for the space occupied by the procedure can be displayed as a function of the number of elements of the array and the size of each one.

The expression of the space occupied by this scenario is, therefore:

$$S \approx 2ns$$

Where n is the number of elements of the array and s is the size of each element.

In this scenario the procedure will take about twice as space as in scenario 1. The relation (δS) between S_1 (Space used on scenario 1) and S_2 (space used in scenario 2) is as follows:

$$\delta S = \frac{S_1}{S_2} = \frac{(n+1)s}{2ns} = \frac{n+1}{2n} = \frac{n}{2n} + \frac{1}{2n} = \frac{1}{2} + \frac{1}{2n}$$

Therefore, when n tends to a large number the term 1/2n tends to 0 and the whole expression tends to ½, meaning that scenario 2 takes about the double of the space than scenario 1.

Time Complexity Analysis (scenario 2)

On this analysis the same assumption of the previous scenario are applicable. The difference is that the array does not need to be reorganized since the deletion does not actually take place.

We still consider the I/O time as Tw to append an element to an array, and “r” as the residual time for any other operation than I/O ones. We define a new time – Tu – as the time to update an array element (instead of the deletion) and assume that, although Tu is an I/O operation its value is considered to be less than or, at most, equal to Tw.

And again it is considered that the algorithm has ideal conditions and the time to randomly access on element may be ignored.

In this scenario, in the cycle the only three I/O happens. One I/O is for reading the array element, other for appending one element to the result array. The last I/O is to mark the element of the input array as “deleted” without actually deleting it.

Marking the record as deleted creates a small overhead to find the k^{th} not deleted element of the array whereas in scenario 1 there were no elements marked for deletion. The whole time needed to find the k^{th} non-deleted element is directly dependent on the k-1 array elements already marked for deletion, meaning that in the first iteration when no records are marked for deletion the access is residual and in the last iteration the whole array (is the worst scenario) must be read to find the non-deleted element.

In the worst scenario, as stated, in iteration k, k-1 marked for deletion array elements must be read. In the best scenario no elements marked for deletion must be read. For each cycle a random number between 0 and k-1 of array elements marked for deletion must be read. Since the uniform distribution was chosen for access the array elements and the elements marked for deletion were also accessed with that probability it expected that those elements have also a uniform distribution along the array.

With this assumption, in each iteration is expected to access (k-1)/2 elements with the individual Tr time.

So the time for each step can be represented as followed:

Step	Time taken
`Proc Random( Array A) returns Array`	0
`R = new Array()`	r
`While A.Length > 0 {`	0
`Pos = Rnd() * A.Length`	r

R.Add(A[Pos])	Tw + (k-1)/2Tr
A.Delete(Pos)	Tu
}	0
Return R	0
End Proc	0

The expression that gives us the time consumed by the algorithm (T_t) is therefore:

$$T_t = r + \sum_{k=1}^{n} (r + \frac{(k-1)}{2} T_r + T_w + T_u)$$

Simplifying the expression:

$$T_t = r + \sum_{k=1}^{n} (r + \frac{kT_r}{2} - \frac{T_r}{2} + T_w + T_u)$$

The constant terms relative to k are multiples of n

$$T_t = r + nr - \frac{nT_r}{2} + nT_w + nT_u + \frac{T_r}{2} \sum_{k=1}^{n} k$$

Again the expression $\sum_{k=1}^{n} k$ represent the sum of the first n integers and its result is given by the formula “n(n +1)/2” so the whole expression takes the form of:

$$T_t = r + nr - \frac{nT_r}{2} + nT_w + nT_u + \frac{n(n+1)T_r}{4}$$

Developing the expression:

$$T_t = r(n+1) + nT_w + nT_u - \frac{nT_r}{2} + \frac{nT_r}{4} + \frac{n^2 T_r}{4}$$

$$T_t = r(n+1) + n(T_w + T_u) - \frac{nT_r}{4} + \frac{n^2 T_r}{4}$$

Putting all Tr terms in evidence

$$T_t = r(n+1) + n(T_w + T_u) + T_r(\frac{n^2 - n}{4})$$

We can see that the time needed to complete the procedure now depends linearly on the writing times and has a polynomial dependency on the reading time.

Considering that both Tu and Tr are smaller than Tw we can rewrite those parameters as:

Tu = Tw – ε

Tr = Tw – δ

And the total time formula as:

$$T_t = r(n+1) + n(2T_w - \epsilon) + (T_w - \delta)(\frac{n^2 - n}{4})$$

$$T_t = r(n+1) + 2nT_w - n\epsilon + \frac{n^2 T_w}{4} - \frac{nT_w}{4} - \frac{n^2\delta}{4} + \frac{n\delta}{4}$$

Considering that ε can be very small but δ can represent a significant difference, the expression r(n+1) – nε is to be considered as residual and globally represented by R.

$$T_t = 2nT_w + \frac{n^2 T_w}{4} - \frac{nT_w}{4} - \frac{n^2\delta}{4} + \frac{n\delta}{4} + R$$

$$T_t = T_w\frac{7n}{4} + T_w\frac{n^2}{4} - \frac{\delta(n^2 - n)}{4} + R$$

$$T_t = T_w(\frac{n^2+7n}{4}) - \frac{\delta(n^2-n)}{4} + R$$

For large arrays, the term R and the term δn/4 can be ignored giving the final expression as:

$$T_t \approx T_w(\frac{n^2+7n}{4}) - \frac{\delta n^2}{4}$$

Now it will be compared with the total time expression in scenario 1 by determining what is the proportion (P) between the two times (T_{t1} for scenario 1 and T_{t2} for scenario 2).

$$P = \frac{T_{t2}}{T_{t1}} = \frac{T_w(\frac{n^2+7n}{4}) - \frac{\delta n^2}{4}}{T_w\frac{(n^2+9n)}{4}} = \frac{T_w(n^2+7n)}{T_w(n^2+9n)} - \frac{\delta n^2}{T_w(n^2+9n)}$$

$$= 1 - \frac{2n}{n^2+9n} - \frac{\delta n^2}{T_w(n^2+9n)} = 1 + \frac{2}{n+9} - \frac{\delta}{T_w}\frac{n}{n+9}$$

When n tends to infinity the expression evaluates as follows:

$$\lim_{n\to\infty}(1 + \frac{2}{n+9} - \frac{\delta}{T_w}\frac{n}{n+9}) = 1 + 0 + \frac{\delta}{T_w} = 1 + \frac{\delta}{T_w}$$

Since d expressed as d=Tw-Tr then the expression is:

$$1 + \frac{\delta}{T_w} = 1 + \frac{T_w - T_r}{T_w} = 2 - \frac{T_r}{T_w}$$

This proportion based solely on the reading and writing times shows that if the reading time for a record is the same as the writing time (Tr=Tw) the algorithm performs in this scenario as fast as in scenario 1. If the reading time is instantaneous (Tr=0) then the algorithm in this scenario performs twice as fast as in scenario 1.

It is then expectable that with a reading time lesser than a writing time and with the residuals that were ignored along the analysis, the total time is reduced in about half compared with the time analysis in scenario 1.

Space Complexity Analysis (scenario 3)

In scenario 1 the algorithm performs fairly according to space, but poorly according to time. In scenario two the algorithm performs poorly in space but increases its performance in time. To achieve an intermediate situation the algorithm must use more space to gain more time.

In this scenario the analysis will take in account that in a certain number of iterations array elements are marked for deletion and at the end of this cycle they will be actually deleted.

In this scenario an arbitrary number of array elements (j-1) are marked for deletion instead of being deleted and in the j^{th} cycle they will be actually deleted.

In this scenario the operation of deletion will occur m times being m=n/j. With this definition k can be rewritten as k=mj+i where “i” represents an integer between 0 and j-1. Every time that j reaches I it will be reset to 0 and m is incremented so that after the iteration k=(mj+i) where “i” takes the value of j-1, hence k=mj+(j-1), the next iteration will have the expression of k=mj+(j-1)+1, which, after simplification results in k=(m+1)j+0, meaning that the former “m” value has been incremented after j cycles.

But the input array is only packed every j cycles having a constant number of elements during each m cycles given by mj elements.

The expression in step five on scenario 1 can be rewritten as follows:

$$(n-k+1)s+ks+4l \Leftrightarrow (n-mj+1)s+(mj+i)s+4l$$

This expression simplifies to

$S = (n + i + 1)s + 4l$

Since n and l are constants the maximum value is when i reaches j, just before the deletion of the j array elements.

Therefore the maximum space occupied in this scenario and given an arbitrary number j of array elements to cycle is given by the expression:

$S = (n + j + 1)s + 4l$

Again, for large arrays the term 4l becomes residual, so the final expression is given by

$S \approx (n + j + 1)s$

When i is equal to 0, meaning that every element is deleted, the formula falls into the first case. When j is equal to n, meaning no element is ever deleted, the formula falls into the second scenario.

Since j belongs to the interval [0..n-1] it is easy to verify that the space occupied in this scenario is between the space in scenario 1 and the space in scenario 2.

When can now define that the space occupied by the random algorithm is given by:

$S \approx (n + j + 1)s$

Where n is the number of elements in the array, s the size of each element and j an arbitrary number of elements to mark for deletion before they are actually deleted. Since j can take values in the range [0..n[, this expression covers all scenarios.

Time Complexity Analysis (scenario 3)

In this scenario, during j-1 iterations elements will be marked for deletion whist only once the deletion of j records will occur. So there will be n/j cycles with j iterations each. The number of iterations at a certain point will be denominated as m and varies from 1 to n/j. For the purpose of this study it is not important if the value is not integer, since the cycle will perform an integer number of times. What may happen is the last cycle is shorter than the others, consequently a litter shorter in time than the others. It will be considered the worst situation where the value of n/j is as integer, meaning that the last cycle is a full cycle, thus m will always be integer, for the purpose of the analysis.

The difference between the j^{th} iteration and the previous j-1 is the way the operations in steps 5 and 6 (R.Add(A[Pos]) and A.Delete(Pos)) are performed.

The number of elements in the input array only changes after every j iterations. The number of elements at a certain moment is given by n-(m-1)j where m is the cycle number between 1 and n/j. This means the at the beginning of cycle 1 the number of elements are n-(1-1)j = n and in the beginning of the last cycle (n/j) it will hold n-(n/j-1)j = n-n+j = j elements.

In the i^{th} iteration, where the “i” stands for a number between 1 and j-1, for the same reasons in scenario 2, it will be expected to skip i-1 elements marked for deletion but not actually deleted.

On the j^{th} iteration, the input array holds (as seen) n-(m-1)j elements, and j elements are to be deleted and the all the elements above the lower one to be deleted will be moved down. Although they may move several positions down there is only one write operation for each element.

Since the uniform distribution is being used for this model, is expected that the j elements marked for deletion will be uniformly distributed along the array, dividing it in j+1 partitions equal in size. Every element above the first partition has to move. Each partition has (n-(m-1)j)/(j+1) elements being n-(m-1)j the

number of elements in the input array and j+1 the number of partitions. All partition except the first one will be affected, so the expected number of elements to be written somewhere else is given by j(n-(m-1)/(j+1).

Some operations such as writing the element to the result array or using the variables doesn't depend on the deletion operation and will occur j times during a single m cycle.

So, time for each step can be represented this way:

Step	Time taken (1 to j-1)	Time taken (j)
`Proc Random( Array A) returns Array`	0	
`R = new Array()`	r	
`While A.Length > 0 {`	0	
`Pos = Rnd() * A.Length`	r	
`R.Add(A[Pos])`	Tw + (i-1)/2Tr	Tw + Tr
`A.Delete(Pos)`	Tu	jTw(n-(m-1)j)/(j+1)
`}`	0	
`Return R`	0	
`End Proc`	0	

The expression that gives us the time consumed by the algorithm (T_t) is therefore:

$$T_t = r + \sum_{m=1}^{\frac{n}{j}} (jr + jT_w + \sum_{i=1}^{j-1} \left(T_r\left(\frac{i-1}{2}\right) + T_u\right) + jT_w(\frac{n-(m-1)j}{j+1}))$$

The inner sum can be solved as follows:

$$\sum_{i=1}^{j-1} \left(T_r\left(\frac{i-1}{2}\right) + T_u\right) = T_u(j-1) - \frac{T_r}{2}(j-1) + \frac{T_r}{2}\sum_{i=1}^{j-1} i$$

$$= T_u(j-1) - \frac{T_r}{2}(j-1) + \frac{T_r}{2}\frac{(j-1)j}{2} = \frac{(j-1)}{4}(T_r(j-2) + 4T_u)$$

This expression now will be replaced on the time formula

$$T_t = r + \sum_{m=1}^{\frac{n}{j}} (jr + jT_w + \frac{(j-1)}{4}(+T_r(j-2) + 4T_u) + jT_w(\frac{n-(m-1)j}{j+1}))$$

Since all the terms but the last does not depend on m they can be multiplied by n/m and removed from the sum as

$$T_t = r + nr + nT_w + \frac{n(j-1)}{4j}(+T_r(j-2) + 4T_u) - \frac{jT_w}{j+1}\sum_{m=1}^{\frac{n}{j}}(n-(m-1)j)$$

The sum can be solved as

$$\sum_{m=1}^{\frac{n}{j}}(n-(m-1)j) = \frac{n^2}{j} + n - j\sum_{m=1}^{\frac{n}{j}} m = \frac{n^2}{j} + n - \frac{j}{2}\left(\frac{n}{j}\right)\left(\frac{n}{j}+1\right)$$

$$= \frac{n^2}{j} + \frac{nj}{j} - \frac{nj}{2j}\left(\frac{n}{j}+1\right) = \frac{n}{j}\left(n + j - \frac{j}{2}\left(\frac{n}{j}+1\right)\right) = \frac{n}{j}\left(n + j - \frac{n}{2} - \frac{j}{2}\right)$$

$$= \frac{n}{j}\left(\frac{n}{2} + \frac{j}{2}\right) = \frac{n(n+j)}{2j}$$

The Total time expression without the sums is

$$T_t = r(n+1) + nT_w + nT_w\frac{n+j}{2(j+1)} + \frac{n(j-1)}{4j}(T_r(j-2) + 4T_u)$$

Considering that both Tu and Tr are smaller than Tw we can rewrite those parameters as:

Tu = Tw – ε

Tr = Tw – δ

And the total time formula as:

$$T_t = r(n+1) + nT_w + nT_w\frac{n+j}{2(j+1)} + \frac{n(j-1)}{4j}((T_w-\delta)(j-2) + 4(T_w-\varepsilon))$$

Simplifying the expression

$$T_t = r(n+1) + nT_w + nT_w\frac{n+j}{2(j+1)} + nT_w\frac{(j-1)(j-2)}{4j} - \frac{n\delta(j-1)(j-2)}{4j} + nT_w\frac{(j-1)}{j} - n\varepsilon\frac{(j-1)}{j}$$

The terms r(n+1) and the term in e are considered as residual and consolidate in a single term R. The formula can now be represented by

$$T_t = nT_w + nT_w\frac{n+j}{2(j+1)} + nT_w\frac{(j-1)(j-2)}{4j} - \frac{n\delta(j-1)(j-2)}{4j} + nT_w\frac{(j-1)}{j} + R$$

Putting all nTw terms in evidence it results in

$$T_t = nT_w\left(1 + \frac{n+j}{2(j+1)} + \frac{(j-1)(j-2)}{4j} + \frac{(j-1)}{j}\right) - \frac{n\delta(j-1)(j-2)}{4j} + R$$

$$T_t = nT_w(1 + \frac{n+j}{2(j+1)} + \frac{(j-1)(j-2)+4(j-1)}{4j}) - \frac{n\delta(j-1)(j-2)}{4j} + R$$

$$T_t = nT_w\left(1 + \frac{n+j}{2(j+1)} + \frac{(j-1)(j-2+4)}{4j}\right) - \frac{n\delta(j-1)(j-2)}{4j} + R$$

$$T_t = nT_w(1 + \frac{n+j}{2(j+1)} + \frac{(j-1)(j+2)}{4j}) - \frac{n\delta(j-1)(j-2)}{4j} + R$$

For large arrays, the term R giving the final expression as:

$$T_t \approx nT_w(1 + \frac{n+j}{2(j+1)} + \frac{(j-1)(j+2)}{4j}) - \frac{n\delta(j-1)(j-2)}{4j}$$

Comparing this formula in this scenario with the others from the other scenarios when j=1, meaning that every record is deleted the formula evaluates to

$$T_t \approx nT_w\left(1+\frac{n+1}{4}+0\right)-0 = nT_w\left(\frac{4+n+1}{4}\right) = T_w\left(\frac{n(n+5)}{4}\right)$$

This expression is similar to the expression in scenario 1, as expected.

On the other hand, if j=n meaning that no records are deleted but marked for deletion instead, the expression takes the following form

$$T_t \approx nT_w\left(1+\frac{n+n}{2(n+1)}+\frac{(n-1)(n+2)}{4n}\right)-\frac{n\delta(n-1)(n-2)}{4n}$$

$$T_t \approx nT_w\left(1+\frac{n}{(n+1)}+\frac{(n-1)(n+2)}{4n}\right)-\frac{\delta(n-1)(n-2)}{4}$$

$$T_t \approx nT_w\left(\frac{4n(n+1)+4n^2+(n-1)(n-2)(n+1)}{4n(n+1)}\right)-\frac{\delta(n-1)(n-2)}{4}$$

$$T_t \approx nT_w\left(\frac{4n(n+1)+4n(n+1)-4n+(n-1)(n-2)(n+1)}{4n(n+1)}\right)$$
$$-\frac{\delta(n-1)(n-2)}{4}$$

$$T_t \approx nT_w\left(\frac{4n+4n+(n-1)(n-2)}{4n}-\frac{1}{n+1}\right)-\frac{\delta(n-1)(n-2)}{4}$$

$$T_t \approx nT_w\left(\frac{4n+4n+n^2-3n+2}{4n}-\frac{1}{n+1}\right)-\frac{\delta(n-1)(n-2)}{4}$$

$$T_t \approx nT_w\left(\frac{4+4+n-3}{4}+\frac{1}{2n}-\frac{1}{n+1}\right)-\frac{\delta(n-1)(n-2)}{4}$$

$$T_t \approx nT_w\left(\frac{n+5}{4}+\frac{1}{2n}-\frac{1}{n+1}\right)-\frac{\delta(n-1)(n-2)}{4}$$

This expression tends to a similar expression than scenario 2 for arrays with large number of elements.

So the expression on this scenario takes a time between scenario 1 and scenario 2 depending on the value chosen for j.

Conclusions of the analysis

The expressions were found in ideal scenarios where particular conditions found in different machines were not considered. One particular machine may have large amount of memory and can accommodate a recordset entirely whilst some other may have to span the recordset to disk. One particular machine may have solid state disks with very small access time whilst other may have a very slow disk. One database engine optimizes the reading and writing of pages whilst another reads and writes each data record at a time.

How can the impact of this algorithm be measured? The formulae found here are all based on the average time of a single write operation. Once this value is found for a machine it can be used to estimate the time that this algorithm may take to run.

Also space, based on the available memory, the size of a record and the expected number of records can be estimated. Moreover, if the recordset is smaller than the amount of free memory a j number of records to be deleted can be estimated along the impact on the time consumed to perform it.

Listing of the SqlRandom Extended Stored Procedure that implements the ordering function “SqlRandom()” for all scenarios.
SqlRandomD implements the sort algorithm considering scenario 1, SqlRandomM implements it for scenario 2 and, finally, SqlRandom implements it for scenario 3 with and arbitrary number of 1000 rows marked before the actual deletion.

```
using System;
using System.Data;
using System.Collections;
using System.Data.SqlClient;
using Microsoft.SqlServer.Server;

public partial class StoredProcedures
{

    [Microsoft.SqlServer.Server.SqlProcedure]
    public static void SqlRandom(string strSQL)
    {
        const int MAXCYCLEROWS = 1000;
        const int NUMCYCLES = 10;
        // Test if we are under an SQL server connection
        if (!SqlContext.IsAvailable)
            return;

        SqlConnection conn = new SqlConnection("context
connection=true");

        // assume that the SQL is properly formed and
        // get the data out of the engine
        DataTable dt = null;
        try
        {
            dt = SQLUtils.lData(strSQL, conn);
        }
        catch (Exception e)
        {
            dt = null;
            SqlContext.Pipe.Send(e.Message);
        }
        if (dt != null)
        {
            Random rnd = new
Random(System.DateTime.UtcNow.TimeOfDay.Milliseconds);
            SqlMetaData[] flds = SQLUtils.GetMetaData(strSQL, conn);
            SqlDataRecord rec = new SqlDataRecord(flds);
            SqlContext.Pipe.SendResultsStart(rec);
            // send the results back to the server
            int i = dt.Rows.Count;
            int cycle = (i / NUMCYCLES > MAXCYCLEROWS) ? MAXCYCLEROWS
: i / NUMCYCLES;
            while (i > 0)
            {
                int nID = Convert.ToInt32(Math.Floor(rnd.NextDouble()
* i));
                //pick up a random row to output
                DataRow dr = dt.Rows[nID];
                rec.SetValues(dr.ItemArray);
```

```
                SqlContext.Pipe.SendResultsRow(rec);
                //delete that row
                dt.Rows.Remove(dr);
                //every «cycle» rows compact the memory file
                if ((i % cycle) == 0)
                    dt.AcceptChanges();
                //loop until we have sent all the rows away
                i = dt.Rows.Count;
            }
            SqlContext.Pipe.SendResultsEnd();
            dt = null;
            GC.Collect();
        }
    }

    [Microsoft.SqlServer.Server.SqlProcedure]
    public static void SQLRandomM(string strSQL)
    {
        // Test if we are under an SQL server connection
        if (!SqlContext.IsAvailable)
            return;

        SqlConnection conn = new SqlConnection("context
connection=true");

        // assume that the SQL is properly formed and
        // get the data out of the engine
        DataTable dt = null;
        try
        {
            dt = SQLUtils.lData(strSQL, conn);
        }
        catch (Exception e)
        {
            dt = null;
            SqlContext.Pipe.Send(e.Message);
        }
        if (dt != null)
        {
            Random rnd = new
Random(System.DateTime.UtcNow.TimeOfDay.Milliseconds);
            SqlMetaData[] flds = SQLUtils.GetMetaData(strSQL, conn);
            SqlDataRecord rec = new SqlDataRecord(flds);
            SqlContext.Pipe.SendResultsStart(rec);
            // send the results back to the server
            int i = dt.Rows.Count;
            while (i > 0)
            {
                int nID = Convert.ToInt32(Math.Floor(rnd.NextDouble()
* i));
                //pick up a random row to output
                DataRow dr = dt.Rows[nID];
                rec.SetValues(dr.ItemArray);
                SqlContext.Pipe.SendResultsRow(rec);
                //delete that row
                dt.Rows.Remove(dr);
                //loop until we have sent all the rows away
                i = dt.Rows.Count;
            }
```

```
            SqlContext.Pipe.SendResultsEnd();
            dt = null;
            GC.Collect();
        }
    }

    [Microsoft.SqlServer.Server.SqlProcedure]
    public static void SQLRandomD(string strSQL)
    {
        // Test if we are under an SQL server connection
        if (!SqlContext.IsAvailable)
            return;

        SqlConnection conn = new SqlConnection("context
connection=true");

        // assume that the SQL is properly formed and
        // get the data out of the engine
        DataTable dt = null;
        try
        {
            dt = SQLUtils.lData(strSQL, conn);
        }
        catch (Exception e)
        {
            dt = null;
            SqlContext.Pipe.Send(e.Message);
        }
        if (dt != null)
        {
            Random rnd = new
Random(System.DateTime.UtcNow.TimeOfDay.Milliseconds);
            SqlMetaData[] flds = SQLUtils.GetMetaData(strSQL, conn);
            SqlDataRecord rec = new SqlDataRecord(flds);
            SqlContext.Pipe.SendResultsStart(rec);
            // send the results back to the server
            int i = dt.Rows.Count;
            while (i > 0)
            {
                int nID = Convert.ToInt32(Math.Floor(rnd.NextDouble()
* i));
                //pick up a random row to output
                DataRow dr = dt.Rows[nID];
                rec.SetValues(dr.ItemArray);
                SqlContext.Pipe.SendResultsRow(rec);
                //delete that row
                dt.Rows.Remove(dr);
                //Pack the recordset
                dt.AcceptChanges();
                //loop until we have sent all the rows away
                i = dt.Rows.Count;
            }
            SqlContext.Pipe.SendResultsEnd();
            dt = null;
            GC.Collect();
        }
    }
};
```

Data recollected from tests done by executing the SqlRandom procedure, above, against a table with a row data size of 1299 bytes with 1,734,600 rows (scenario 1).

The first machine is a HP Z400 equipped with a Xeon W3520 processor @ 2.67 GHz with 4,00 GB of installed memory (RAM). The operating system is a 64-bit Windows 7 Professional, with a Windows Experience Index of 4.4. On this machine, the queries were run on a SQL Server 2008 R2, 64-bit Developer Edition, with Service Pack 1, with the version number 10.50.2500.

The second machine is an Acer Aspire M3910 equipped with Intel Core i5 processor @ 3.2 GHz with 4,00 GB of installed memory (RAM). The operating system is a 64-bit Windows 7 Ultimate, with a Windows Experience Index of 5.1. On this machine, the queries were run on a SQL Server 2008 R2, 64-bit Developer Edition, with Service Pack 1, with the version number 10.50.2500.

The regular query was “SELECT * FROM SiteLog ORDER BY DateTime”. The SqlRandom was also over the full dataset as “EXEC dbo.SqlRandom ‘SELECT * FROM SiteLog’”

On the HP machine the execution time for both runs of each query were as follows:

HP Z400			
Query	Run	First results	Query completion
Order By	1st	47”	2’55”
	2nd	14”	2’50”
SqlRandom	1st	51”	2’01”
	2nd	49”	2’24”

On the Acer machine the execution time for both runs of each query were as follows:

Acer Aspire M3910			
Query	Run	First results	Query completion
Order By	1st	1’19”	6’10”
	2nd	21”	4’14”
SqlRandom	1st	44”	1’52”
	2nd	42”	1’56”

Code for installing the System.Drawing.dll in order to use image functions in the SqlImage.cs extended stored procedures

```
ALTER DATABASE [RMExtension] SET TRUSTWORTHY ON

GO

CREATE ASSEMBLY [System.Drawing.dll]
FROM
'C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Drawing.dll'
WITH PERMISSION_SET = UNSAFE
```

Code for the SqlImage.cs that acts as an interface between the SQL Server ™ and the set of procedures that actually deals with image treatment.

```
using System;
using System.Data;
using System.Collections;
using System.Drawing;
using System.IO;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;
using System.ComponentModel;


public partial class StoredProcedures
{
    [Microsoft.SqlServer.Server.SqlProcedure]
    public static void SqlImage(string strSQL)
    {
        // Test if we are under an SQL server connection
        if (!SqlContext.IsAvailable)
            return;

        //create an array to store the numbers already used
        ArrayList ids = new ArrayList();

        SqlConnection conn = new SqlConnection("context
connection=true");

        // assume that the SQL is properly formed and
        // get the data out of the engine
        DataTable dt = null;
        try
        {
            dt = SQLUtils.lData(strSQL, conn);
        }
        catch (Exception e)
        {
            dt = null;
            SqlContext.Pipe.Send(e.Message);
        }
        if (dt != null)
        {
            DataTable dtout = new DataTable();

            //find a suitable name for the "extra" field
            string IDname = "ID";
            int i = 0;
            bool goodname = false;
            while (!goodname)
            {
                goodname = true;
                foreach (DataColumn dc in dt.Columns)
                {
                    if (dc.ColumnName == IDname)
                    {
                        IDname = "ID" + i.ToString("00");
                        ++i;
```

```
                    goodname = false;
                    break;
                }
            }
        }

        //now populate the columns collection of the output table
        foreach (DataColumn dc in dt.Columns)
        {
            dtout.Columns.Add(dc.ColumnName, dc.DataType);
        }
        // and add the ID column for classification rank
        dtout.Columns.Add(IDname, Type.GetType("System.Double"));

        //prepare the output
        SqlMetaData[] flds = SQLUtils.GetMetaData(strSQL, conn);
        SqlDataRecord rec = new SqlDataRecord(flds);
        SqlContext.Pipe.SendResultsStart(rec);

        //find out the first image column. This is a prototype
limitation.
        string fldImage = "";
        foreach (SqlMetaData fld in flds)
            if (fld.SqlDbType == SqlDbType.Image)
            {
                fldImage = fld.Name;
                break;
            }
        if (String.IsNullOrEmpty(fldImage))
        {
            //just output the original table
            foreach (DataRow dr in dt.Rows)
            {
                rec.SetValues(dr.ItemArray);
                SqlContext.Pipe.SendResultsRow(rec);
            }
        }
        else
        {
            foreach (DataRow dr in dt.Rows)
            {
                DataRow drout = dtout.NewRow();
                foreach (DataColumn dc in dt.Columns)
                {
                    drout[dc.ColumnName] = dr[dc.ColumnName];
                }
                // classify the image
                Double nID = ClassifyImage((byte[])dr[fldImage]);
                drout[IDname] = nID;
                dtout.Rows.Add(drout);
            }

            DataView dv = dtout.DefaultView;
            dv.Sort = IDname;

            // send the results back to the server
            i = 0;

            DataTable dto = dv.ToTable();
```

```
                foreach (DataRow dr in dto.Rows)
                {
                    for (i = 0; i < dtout.Columns.Count - 1; i++)
                    {
                        rec.SetValue(i, dr[i]);
                    }
                    SqlContext.Pipe.SendResultsRow(rec);
                }

                dto = null;
            }
            SqlContext.Pipe.SendResultsEnd();

            //clean the memory
            dt = null;
            dtout = null;
            GC.Collect();

        }
    }

    public static double ClassifyImage(byte[] pImage)
    {
        TypeConverter tc =
TypeDescriptor.GetConverter(typeof(Bitmap));
        Bitmap imgbmp = (Bitmap)tc.ConvertFrom(pImage);

        ImageUtils iu = new ImageUtils();
        return iu.process(imgbmp);
    }
};
```

Code for the ImageUtils.cs where the actual image treatment is done

```
using System;
using System.Data;
using System.Collections;
using System.Drawing;
using System.IO;

class ImageUtils
{
    /// <summary>
    /// Class to actually do the image analysis.
    /// These procedures were adapted from PhD thesis "Classification
of Skin Tumours through the Analysis of Unconstrained Images"
    /// Author:J. Cunha Viana
    /// </summary>
    ///
    private Bitmap imgbmp;
    private Bitmap imgbmp1;
    private Bitmap imgbmp2;
    private Bitmap imgbmp3;
    private Bitmap imgbmp4;
    private Bitmap imgbmp5;
    //private int detectType, enhanceType;

    //private string fn;
    //private string detectTypeDesc;
    //private string enhanceTypeDesc;
    //private Image img;
    private int centre_x, centre_y;
    private Color pixelColor;

    //private int row, col;
    //private int ROWS;
    //private int COLUMNS;
    //private int windowSize;
    //private string temp;

    private const int MAX_MASK_SIZE = 50;
    //private double sigma, sigma1, sigma2, sigmaSpread, sigmaC,
ratio;
    //private double lgau;
    //private int img_scale, fnslash, fndot, loadedImg = 0, numPoints;
    private int numPoints;
    //private double[,] data;
    //private double[,] data1;
    //private double[,] orig;
    //private int[,] edges;
    //private double[,] A;
    //private double[,] B;

    //private int colorTot, colorDifTot, finalLine, lesionPix;
    private int colorTot, lesionPix;
    private int distDifTot_X, distDifTot_Y, numEdgePoints;
    //private double meanColor, sigmaTot, meanDif, sigmaDif,
colorTot1;
    private double colorTot1;
    private double meanDifPoints_X, meanDifPoints_Y, sigmaDistDif_X,
sigmaDistDif_Y;
```

```
    //private int k, l, cN, cOld, rOld, nAllow, countDraw,
lesionPixels, altern;
    private int k, l, cN, countDraw, lesionPixels;
    private double sigmaCombi;
    private double var_R, var_G, var_B;
    private double X, Y, Z;
    private double var_X, var_Y, var_Z;
    private double CIE_L, CIE_a, CIE_b;
    //private int ind = 0, ind1 = 0, indMin = 0;
    private double ref_X = 95.047, ref_Y = 100.000, ref_Z = 108.883;
    private int[] cOpenPoint = new int[100];
    private int[] rOpenPoint = new int[100];
    private int[,] ptsLine;
    //private int[,] ptsFringe;
    //private double[] distFringe;
    private int[,] ptsL;
    private double[] WiGeeC;
    private double[] WiGeeR;
    private double[] WjGeeC;
    private double[] WjGeeR;
    private double[] GeeC;
    private double[] GeeR;
    private double[] c_RaG;
    private double[] r_RaG;
    private int[,] ptsDraw;

    private bool StepOK = false;


    public double process(Bitmap imgtrt)
    {
        imgbmp = imgtrt;
        double res = 0;
        combi();
        if (StepOK)
        {
            hough();
            colours();
            asymmetry();
            res = Math.Sqrt(Math.Pow(sigmaDistDif_X, 2) +
Math.Pow(sigmaDistDif_Y, 2));
        }

        return res;
    }


    public void hough()
    {
        lesionPix = 0;
        //sigma = 0;
        colorTot = 0;
        //sigmaTot = 0.0;
        //meanDif = 0.0;
        //sigmaDif = 0.0;
        //meanColor = 0.0;
        //colorDifTot = 0;
        //distDifTot_X = 0;
```

```
//meanDifPoints_X = 0.0;
//sigmaDistDif_X = 0.0;
//distDifTot_Y = 0;
//meanDifPoints_Y = 0.0;
//sigmaDistDif_Y = 0.0;
//numEdgePoints = 0;

numPoints = imgbmp.Width * imgbmp.Height;

//img_scale = 1;

int Rin, Gin, Bin;
int tmp;
int x_max = 0, x_min = imgbmp.Width;
int y_max = 0, y_min = imgbmp.Height;
int x_high, y_high;

int h_w = 500;
//Angle for 1 step
double theta_step = Math.PI / h_w;

tmp = Math.Max(imgbmp.Width, imgbmp.Height);
int h_h = (int)(Math.Sqrt(2) * tmp);

int[,] tmp_2d = new int[h_w, 2 * h_h];

//Find centre point coordinates
for (int i = 0; i < imgbmp.Width; i++)
{
    for (int j = 0; j < imgbmp.Height; j++)
    {
        pixelColor = imgbmp.GetPixel(i, j);
        Rin = pixelColor.R;
        Gin = pixelColor.G;
        Bin = pixelColor.B;
        if (Rin != 255 || Gin != 255 || Bin != 255) continue;
        //Background found
        else
        {
            numEdgePoints++;
            if (j < y_min) y_min = j;
            if (j > y_max) y_max = j;
            if (i < x_min) x_min = i;
            if (i > x_max) x_max = i;
        }
    }
}

centre_x = (x_min + x_max) / 2;
centre_y = (y_min + y_max) / 2;


for (int i = 0; i < h_w; i++)
{
    for (int j = 0; j < 2 * h_h; j++)
    {
        tmp_2d[i, j] = 0;
    }
}
```

```
        for (int i = 0; i < imgbmp.Width; i++)
        {
            for (int j = 0; j < imgbmp.Height; j++)
            {
                pixelColor = imgbmp.GetPixel(i, j);
                Rin = pixelColor.R;
                if (Rin != 255) continue;
                //Background found
                else
                {

                    // Edge pixel found
                    for (int k = 0; k < h_w; k++)
                    {

                        //Work out the r values for each theta step
                        tmp = (int)(((i - centre_x) * Math.Cos(k *
theta_step)) + ((j - centre_y) * Math.Sin(k * theta_step)));
                        //Move all values into positive range for
display purposes

                        tmp = tmp + h_h;
                        if (tmp < 0 || tmp >= h_h) continue;

                        //Increment hough array
                        tmp_2d[k, tmp]++;
                    }
                }
            }
        }

        int high = 0;
        for (int i = 0; i < h_w; i++)
        {
            for (int j = 0; j < h_h; j++)
            {
                //Find the max hough value
                if (tmp_2d[i, j] > high)
                {
                    x_high = i;
                    y_high = j;
                    high = tmp_2d[i, j];
                }
            }
        }

        int[] distEdgeCentre = new int[numEdgePoints];

        int pNumber = -1;
        int meanRay = 0;
        for (int i = 0; i < imgbmp.Width; i++)
        {
            for (int j = 0; j < imgbmp.Height; j++)
            {
                pixelColor = imgbmp.GetPixel(i, j);
                Rin = pixelColor.R;
                Gin = pixelColor.G;
                Bin = pixelColor.B;
                if (Rin != 255 || Gin != 255 || Bin != 255) continue;
```

```
                //Background found
                else
                {
                    // Edge pixel found
                    pNumber++;
                    distEdgeCentre[pNumber] =
(int)Math.Round(Math.Sqrt(Convert.ToDouble((i - centre_x) * (i -
centre_x) + (j - centre_y) * (j - centre_y))), 0);
                    meanRay += distEdgeCentre[pNumber];
                }
            }
        }

        meanRay /= numEdgePoints;

        double sumMeans = 0;
        for (int i = 0; i < numEdgePoints; i++)
        {
            sumMeans += Math.Pow(distEdgeCentre[i] - meanRay, 2.0);
        }
        double variance = sumMeans / numEdgePoints;
        double sigma_d = Math.Sqrt(variance);
    }

    public void colours()
    {
        int Rin, Gin, Bin, ROld, GOld, BOld, numPixels = 0;

        int colorDifTot = 0;
        colorTot = 0;
        for (int i = 0; i < this.imgbmp.Width; i++)
        {
            for (int j = 0; j < this.imgbmp.Height; j++)
            {
                pixelColor = this.imgbmp.GetPixel(i, j);
                Rin = this.pixelColor.R;
                if (Rin != 255) continue;
                //Background found
                else
                {
                    // Edge pixel found
                    // Find opposite edge
                    for (int k = this.imgbmp.Height - 1; k > j; k--)
                    {
                        pixelColor = this.imgbmp.GetPixel(i, k);
                        Rin = this.pixelColor.R;
                        if (Rin != 255) continue;
                        //Background found
                        else
                        {
                            // Edge pixel found
                            if ((k - j) <= 2)
                            {
                                k = j;
                                continue;
                            }
                            else
                            {
                                ROld = 0; GOld = 0; BOld = 0;
```

```
                                for (int x = j + 1; x < k; x++)
                                {
                                    pixelColor =
this.imgbmp.GetPixel(i, x);
                                    Rin = this.pixelColor.R;
                                    Gin = this.pixelColor.G;
                                    Bin = this.pixelColor.B;
                                    if (Rin == 255) continue;
                                    colorTot += (Rin + Gin + Bin);
                                    if (ROld != 0 || GOld != 0 || BOld
!= 0) colorDifTot += (Math.Abs(Rin - ROld) + Math.Abs(Gin - GOld) +
Math.Abs(Bin - BOld));
                                    ROld = Rin;
                                    GOld = Gin;
                                    BOld = Bin;
                                    numPixels++;
                                }
                            }
                        }

                    }
                }
            }
        }

        double meanColor = colorTot / numPixels;
        double meanDif = colorDifTot / numPixels;
        int colorDifSquare;
        double colorTotSquare;
        colorTot1 = 0;
        colorDifTot = 0;
        numPixels = 0;
        for (int i = 0; i < this.imgbmp.Width; i++)
        {
            for (int j = 0; j < this.imgbmp.Height; j++)
            {
                pixelColor = this.imgbmp.GetPixel(i, j);
                Rin = this.pixelColor.R;
                if (Rin != 255) continue;
                //Background found
                else
                {
                    // Edge pixel found
                    // Find opposite edge
                    for (int k = this.imgbmp.Height - 1; k > j; k--)
                    {
                        pixelColor = this.imgbmp.GetPixel(i, k);
                        Rin = this.pixelColor.R;
                        if (Rin != 255) continue;
                        //Background found
                        else
                        {
                            // Edge pixel found
                            if ((k - j) <= 2)
                            {
                                k = j;
                                continue;
                            }
                            else
```

```
                                {
                                    ROld = 0; GOld = 0; BOld = 0;
                                    for (int x = j + 1; x < k; x++)
                                    {
                                        pixelColor =
this.imgbmp.GetPixel(i, x);
                                        Rin = this.pixelColor.R;
                                        Gin = this.pixelColor.G;
                                        Bin = this.pixelColor.B;
                                        if (Rin == 255) continue;
                                        colorTotSquare = ((Rin + Gin +
Bin) - meanColor) * ((Rin + Gin + Bin) - meanColor);
                                        colorTot1 += colorTotSquare;
                                        //Debug.Write ("  colorTot: ");
                                        //Debug.WriteLine (colorTot1);
                                        if (ROld != 0 || GOld != 0 || BOld
!= 0)
                                        {
                                            colorDifSquare = (Math.Abs(Rin
- ROld) + Math.Abs(Gin - GOld) + Math.Abs(Bin - BOld));
                                            colorDifSquare =
Convert.ToInt32(Math.Pow(colorDifSquare - meanDif, 2));
                                            colorDifTot += colorDifSquare;
                                            numPixels++;
                                        }
                                        ROld = Rin;
                                        GOld = Gin;
                                        BOld = Bin;
                                    }
                                }
                            }

                        }
                    }
                }
            }
        }
        double variance = Convert.ToDouble(colorDifTot / numPixels);
        double sigmaDif = Math.Sqrt(variance);

        variance = Convert.ToDouble(colorTot1 / (numPixels + 1));
        double sigmaTot = Math.Sqrt(variance);
    }


    public void asymmetry()
    {

        int Rin, numPoints_X = 0, numPoints_Y = 0, dist_X, dist_Y;

        int dist1, dist2;
        double distDifTotSquare_X = 0, distDifTotSquare_Y = 0;

        distDifTot_X = 0;
        distDifTot_Y = 0;

//=============================================================================
================================================
        //        Asymmetry along the X axis
```

```
//=================================================================================================

        for (int x = 0; x < this.centre_x; x++)
        {
            for (int y = 0; y < this.imgbmp.Height; y++)
            {
                pixelColor = this.imgbmp.GetPixel(x, y);
                Rin = this.pixelColor.R;
                if (Rin != 255) continue;
                //Background found
                else
                {
                    // Edge pixel found
                    dist1 = Math.Abs(centre_x - x);
                    // Find opposite edge
                    for (int k = this.imgbmp.Width - 1; k > x; k--)
                    {
                        pixelColor = this.imgbmp.GetPixel(k, y);
                        Rin = this.pixelColor.R;
                        if (Rin != 255) continue;
                        //Background found
                        else
                        {
                            // Edge pixel found
                            if ((k - x) <= 2)
                            {
                                k = x;
                                continue;
                            }
                            else
                            {
                                dist2 = Math.Abs(k - centre_x);
                                distDifTot_X += Math.Abs(dist1 -
dist2);

                                numPoints_X++;
                            }
                        }

                    }
                }
            }
        }

        meanDifPoints_X = distDifTot_X / numPoints_X;

        numPoints_X = 0;

        for (int x = 0; x < this.centre_x; x++)
        {
            for (int y = 0; y < this.imgbmp.Height; y++)
            {
                pixelColor = this.imgbmp.GetPixel(x, y);
                Rin = this.pixelColor.R;
                if (Rin != 255) continue;
                //Background found
                else
                {
```

```
                    // Edge pixel found
                    dist1 = Math.Abs(centre_x - x);
                    // Find opposite edge
                    for (int k = this.imgbmp.Width - 1; k > x; k--)
                    {
                        pixelColor = this.imgbmp.GetPixel(k, y);
                        Rin = this.pixelColor.R;
                        if (Rin != 255) continue;
                        //Background found
                        else
                        {
                            // Edge pixel found
                            if ((k - x) <= 2)
                            {
                                k = x;
                                continue;
                            }
                            else
                            {
                                dist2 = Math.Abs(k - centre_x);
                                dist_X = Math.Abs(dist1 - dist2);
                                numPoints_X++;
                                distDifTotSquare_X += (dist_X -
meanDifPoints_X) * (dist_X - meanDifPoints_X);
                            }
                        }

                    }
                }
            }
        }


        double variance = Convert.ToDouble(distDifTotSquare_X /
numPoints_X);
        sigmaDistDif_X = Math.Sqrt(variance);



//=============================================================================
================================================
        //        Asymmetry along the Y axis

//=============================================================================
================================================


        for (int y = 0; y < this.centre_y; y++)
        {
            for (int x = 0; x < this.imgbmp.Width; x++)
            {
                pixelColor = this.imgbmp.GetPixel(x, y);
                Rin = this.pixelColor.R;
                if (Rin != 255) continue;
                //Background found
                else
                {
                    // Edge pixel found
                    dist1 = Math.Abs(centre_y - y);
```

```
                    // Find opposite edge
                    for (int k = this.imgbmp.Height - 1; k > y; k--)
                    {
                        pixelColor = this.imgbmp.GetPixel(x, k);
                        Rin = this.pixelColor.R;
                        if (Rin != 255) continue;
                        //Background found
                        else
                        {
                            // Edge pixel found
                            if ((k - y) <= 2)
                            {
                                k = y;
                                continue;
                            }
                            else
                            {
                                dist2 = Math.Abs(k - centre_y);
                                distDifTot_Y += Math.Abs(dist1 -
dist2);

                                numPoints_Y++;
                            }
                        }

                    }
                }
            }
        }

        meanDifPoints_Y = distDifTot_Y / numPoints_Y;

        numPoints_Y = 0;

        for (int y = 0; y < this.centre_y; y++)
        {
            for (int x = 0; x < this.imgbmp.Width; x++)
            {
                pixelColor = this.imgbmp.GetPixel(x, y);
                Rin = this.pixelColor.R;
                if (Rin != 255) continue;
                //Background found
                else
                {
                    // Edge pixel found
                    dist1 = Math.Abs(centre_y - y);
                    // Find opposite edge
                    for (int k = this.imgbmp.Height - 1; k > y; k--)
                    {
                        pixelColor = this.imgbmp.GetPixel(x, k);
                        Rin = this.pixelColor.R;
                        if (Rin != 255) continue;
                        //Background found
                        else
                        {
                            // Edge pixel found
                            if ((k - y) <= 2)
                            {
                                k = y;
                                continue;
```

```
                            }
                            else
                            {
                                dist2 = Math.Abs(k - centre_y);
                                dist_Y = Math.Abs(dist1 - dist2);
                                numPoints_Y++;
                                distDifTotSquare_Y += (dist_Y -
meanDifPoints_Y) * (dist_Y - meanDifPoints_Y);
                            }
                        }

                    }
                }
            }
        }


        variance = Convert.ToDouble(distDifTotSquare_Y / numPoints_Y);
        sigmaDistDif_Y = Math.Sqrt(variance);
    }

    public void combi()
    {
        int c, r, count = 0, squareSide = 10;
        double CIE_LTot = 0, CIE_aTot = 0, CIE_bTot = 0;
        double CIE_LMed, CIE_aMed, CIE_bMed, geeX, efeX, combiT;
        imgbmp1 = new Bitmap(imgbmp);
        imgbmp2 = new Bitmap(imgbmp);
        imgbmp3 = new Bitmap(imgbmp);
        imgbmp4 = new Bitmap(imgbmp);
        imgbmp5 = new Bitmap(imgbmp);

        //int Rin, Rout, Gin, Bin, delta, multFac, colorTot = 0,
deltaTot = 0, deltaMax = 0;
        int Rin, Rout, Gin, Bin, delta, multFac, deltaMax = 0;
        int[] histIntens = new int[256];
        int[,] outVal = new int[imgbmp.Width, imgbmp.Height];
        int[] leftEdge = new int[imgbmp.Height];
        int[] rightEdge = new int[imgbmp.Height];
        int[] topEdge = new int[imgbmp.Width];
        int[] bottomEdge = new int[imgbmp.Width];

        //finalLine = 0;
        lesionPix = 0;


        // Eliminate white spots

        for (c = 0; c < imgbmp.Width; c++)
        {
            for (r = 0; r < imgbmp.Height; r++)
            {
                pixelColor = imgbmp.GetPixel(c, r);
                Rin = pixelColor.R;
                Gin = pixelColor.G;
                Bin = pixelColor.B;
                if (Rin != 255 && Gin != 255 && Bin != 255) continue;
                if (Rin == 255) Rin = 254;
                if (Gin == 255) Gin = 254;
```

```
                if (Bin == 255) Bin = 254;
                imgbmp.SetPixel(c, r, Color.FromArgb(Rin, Gin, Bin));
            }
        }

        for (c = 0; c < 256; c++) histIntens[c] = 0;
        for (c = 0; c < imgbmp.Height; c++)
        {
            leftEdge[c] = 0;
        }
        for (c = 0; c < imgbmp.Height; c++)
        {
            rightEdge[c] = 0;
        }
        for (c = 0; c < imgbmp.Width; c++)
        {
            topEdge[c] = 0;
        }

        for (c = 0; c < imgbmp.Width; c++)
        {
            bottomEdge[c] = 0;
        }

        //detectType = 7;

        //combiT = Convert.ToDouble(combiTIn.Text) / 100.0;
        combiT = 0.1; //defaults to 10%


//=========================================================================
====================
        //Calculate background color

//=========================================================================
====================

        //Top Left corner
        for (c = 0; c < squareSide; c++)
        {
            for (r = 0; r < squareSide; r++)
            {
                pixelColor = imgbmp.GetPixel(c, r);
                Rin = pixelColor.R;
                Gin = pixelColor.G;
                Bin = pixelColor.B;

                conv_CIE(Rin, Gin, Bin);

                CIE_LTot += CIE_L;
                CIE_aTot += CIE_a;
                CIE_bTot += CIE_b;
                count++;
            }
        }

        //Top Right corner
        for (c = imgbmp.Width - squareSide; c < imgbmp.Width; c++)
        {
```

```
            for (r = 0; r < squareSide; r++)
            {
                pixelColor = imgbmp.GetPixel(c, r);
                Rin = pixelColor.R;
                Gin = pixelColor.G;
                Bin = pixelColor.B;

                conv_CIE(Rin, Gin, Bin);

                CIE_LTot += CIE_L;
                CIE_aTot += CIE_a;
                CIE_bTot += CIE_b;
                count++;
            }
        }

        //Bottom Leftt corner
        for (c = 0; c < squareSide; c++)
        {
            for (r = imgbmp.Height - squareSide; r < imgbmp.Height;
r++)
            {
                pixelColor = imgbmp.GetPixel(c, r);
                Rin = pixelColor.R;
                Gin = pixelColor.G;
                Bin = pixelColor.B;

                conv_CIE(Rin, Gin, Bin);

                CIE_LTot += CIE_L;
                CIE_aTot += CIE_a;
                CIE_bTot += CIE_b;
                count++;
            }
        }

        //Bottom Right corner
        for (c = imgbmp.Width - squareSide; c < imgbmp.Width; c++)
        {
            for (r = imgbmp.Height - squareSide; r < imgbmp.Height;
r++)
            {
                pixelColor = imgbmp.GetPixel(c, r);
                Rin = pixelColor.R;
                Gin = pixelColor.G;
                Bin = pixelColor.B;

                conv_CIE(Rin, Gin, Bin);

                CIE_LTot += CIE_L;
                CIE_aTot += CIE_a;
                CIE_bTot += CIE_b;
                count++;
            }
        }

        CIE_LMed = CIE_LTot / count;
        CIE_aMed = CIE_aTot / count;
        CIE_bMed = CIE_bTot / count;
```

```
//=====================================================================
====================
        // Conversion from RGB to XYZ

//=====================================================================
====================

        for (c = 0; c < imgbmp.Width; c++)
        {
            for (r = 0; r < imgbmp.Height; r++)
            {
                pixelColor = imgbmp.GetPixel(c, r);
                Rin = pixelColor.R;
                Gin = pixelColor.G;
                Bin = pixelColor.B;

                conv_CIE(Rin, Gin, Bin);

                delta = Convert.ToInt32(Math.Sqrt((Math.Pow(CIE_LMed -
CIE_L, 2)) + (Math.Pow(CIE_aMed - CIE_a, 2)) + (Math.Pow(CIE_bMed -
CIE_b, 2))));
                if (delta > deltaMax) deltaMax = delta;
                imgbmp1.SetPixel(c, r, Color.FromArgb(delta, delta,
delta));

            }
        }

        //Calculation of the multiplication factor to use

        multFac = Convert.ToInt32(255 / deltaMax);
        if (multFac < 1) multFac = 1;

        for (c = 0; c < imgbmp.Width; c++)
        {
            for (r = 0; r < imgbmp.Height; r++)
            {
                pixelColor = imgbmp1.GetPixel(c, r);
                delta = multFac * pixelColor.R;
                imgbmp1.SetPixel(c, r, Color.FromArgb(delta, delta,
delta));
            }
        }


//=====================================================================
====================
        // Smoothing grey image

//=====================================================================
====================

        //int i = 0;
        CIE_LTot = 0;
        CIE_aTot = 0;
        CIE_bTot = 0;
        //double intens = 0;
```

```
        int intensMed;

//=================================================================================
==============================
        //Calculate background color Median intensity

//=================================================================================
==============================

        //Top Left corner
        for (c = 0; c < squareSide; c++)
        {
            for (r = 0; r < squareSide; r++)
            {
                pixelColor = this.imgbmp1.GetPixel(c, r);
                Rin = this.pixelColor.R;
                Gin = this.pixelColor.G;
                Bin = this.pixelColor.B;

                conv_CIE(Rin, Gin, Bin);

                CIE_LTot += CIE_L;
                CIE_aTot += CIE_a;
                CIE_bTot += CIE_b;
            }
        }

        //Top Right corner
        for (c = imgbmp.Width - squareSide; c < imgbmp.Width; c++)
        {
            for (r = 0; r < squareSide; r++)
            {
                pixelColor = this.imgbmp1.GetPixel(c, r);
                Rin = this.pixelColor.R;
                Gin = this.pixelColor.G;
                Bin = this.pixelColor.B;

                conv_CIE(Rin, Gin, Bin);

                CIE_LTot += CIE_L;
                CIE_aTot += CIE_a;
                CIE_bTot += CIE_b;
            }
        }

        //Bottom Leftt corner
        for (c = 0; c < squareSide; c++)
        {
            for (r = imgbmp.Height - squareSide; r < imgbmp.Height;
r++)
            {
                pixelColor = this.imgbmp1.GetPixel(c, r);
                Rin = this.pixelColor.R;
                Gin = this.pixelColor.G;
                Bin = this.pixelColor.B;

                conv_CIE(Rin, Gin, Bin);

                CIE_LTot += CIE_L;
```

```
                CIE_aTot += CIE_a;
                CIE_bTot += CIE_b;
            }
        }

        //Bottom Right corner
        for (c = imgbmp.Width - squareSide; c < imgbmp.Width; c++)
        {
            for (r = imgbmp.Height - squareSide; r < imgbmp.Height;
r++)
            {
                pixelColor = this.imgbmp1.GetPixel(c, r);
                Rin = this.pixelColor.R;
                Gin = this.pixelColor.G;
                Bin = this.pixelColor.B;

                conv_CIE(Rin, Gin, Bin);

                CIE_LTot += CIE_L;
                CIE_aTot += CIE_a;
                CIE_bTot += CIE_b;
            }
        }

        CIE_LMed = CIE_LTot / count;
        CIE_aMed = CIE_aTot / count;
        CIE_bMed = CIE_bTot / count;

        intensMed = Convert.ToInt32(Math.Sqrt((CIE_LMed * CIE_LMed) +
(CIE_aMed * CIE_aMed) + (CIE_bMed * CIE_bMed)));


        double delta_LTot = 0;
        double delta_aTot = 0;
        double delta_bTot = 0;


//================================================================
==============================
        //Calculate background color intensity variations

//================================================================
==============================

        //Top Left corner
        for (c = 0; c < squareSide; c++)
        {
            for (r = 0; r < squareSide; r++)
            {
                pixelColor = this.imgbmp1.GetPixel(c, r);
                Rin = this.pixelColor.R;
                Gin = this.pixelColor.G;
                Bin = this.pixelColor.B;

                conv_CIE(Rin, Gin, Bin);

                delta_LTot += Math.Pow(CIE_L - CIE_LMed, 2);
                delta_aTot += Math.Pow(CIE_a - CIE_aMed, 2);
                delta_bTot += Math.Pow(CIE_b - CIE_bMed, 2);
```

```
            }
        }

        //Top Right corner
        for (c = imgbmp.Width - squareSide; c < imgbmp.Width; c++)
        {
            for (r = 0; r < squareSide; r++)
            {
                pixelColor = this.imgbmp1.GetPixel(c, r);
                Rin = this.pixelColor.R;
                Gin = this.pixelColor.G;
                Bin = this.pixelColor.B;

                conv_CIE(Rin, Gin, Bin);

                delta_LTot += Math.Pow(CIE_L - CIE_LMed, 2);
                delta_aTot += Math.Pow(CIE_a - CIE_aMed, 2);
                delta_bTot += Math.Pow(CIE_b - CIE_bMed, 2);
            }
        }

        //Bottom Leftt corner
        for (c = 0; c < squareSide; c++)
        {
            for (r = imgbmp.Height - squareSide; r < imgbmp.Height;
r++)
            {
                pixelColor = this.imgbmp1.GetPixel(c, r);
                Rin = this.pixelColor.R;
                Gin = this.pixelColor.G;
                Bin = this.pixelColor.B;

                conv_CIE(Rin, Gin, Bin);

                delta_LTot += Math.Pow(CIE_L - CIE_LMed, 2);
                delta_aTot += Math.Pow(CIE_a - CIE_aMed, 2);
                delta_bTot += Math.Pow(CIE_b - CIE_bMed, 2);
            }
        }

        //Bottom Right corner
        for (c = imgbmp.Width - squareSide; c < imgbmp.Width; c++)
        {
            for (r = imgbmp.Height - squareSide; r < imgbmp.Height;
r++)
            {
                pixelColor = this.imgbmp1.GetPixel(c, r);
                Rin = this.pixelColor.R;
                Gin = this.pixelColor.G;
                Bin = this.pixelColor.B;

                conv_CIE(Rin, Gin, Bin);

                delta_LTot += Math.Pow(CIE_L - CIE_LMed, 2);
                delta_aTot += Math.Pow(CIE_a - CIE_aMed, 2);
                delta_bTot += Math.Pow(CIE_b - CIE_bMed, 2);
            }
        }
```

```
//=================================================================================
================================
        // Calculation of standard deviation (grey picture background)

//=================================================================================
================================

        sigmaCombi = Math.Sqrt(delta_LTot + delta_aTot + delta_bTot) /
count;


//=================================================================================
================================
        // Conversion from RGB to XYZ

//=================================================================================
================================

        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
            {
                pixelColor = this.imgbmp1.GetPixel(c, r);
                Rin = this.pixelColor.R;
                Gin = this.pixelColor.G;
                Bin = this.pixelColor.B;

                conv_CIE(Rin, Gin, Bin);

                delta = Convert.ToInt32(Math.Sqrt((CIE_LMed - CIE_L) *
(CIE_LMed - CIE_L)) + ((CIE_aMed - CIE_a) * (CIE_aMed - CIE_a)) +
((CIE_bMed - CIE_b) * (CIE_bMed - CIE_b)));
                geeX = 1.0 - (Math.Exp(-(delta * delta) / (2.0 *
sigmaCombi * sigmaCombi)));
                efeX = (1.0 / (Math.Sqrt(2.0 * Math.PI) * sigmaCombi))
* geeX;
                delta = Convert.ToInt32(delta * efeX);
                if (delta > deltaMax) deltaMax = delta;
                outVal[c, r] = delta;
            }
        }

        //Calculation of the multiplication factor to use

        multFac = Convert.ToInt32(255 / deltaMax);
        if (multFac < 1) multFac = 1;

        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
            {
                delta = multFac * outVal[c, r];
                if (delta > 255) delta = 255;
                this.imgbmp1.SetPixel(c, r, Color.FromArgb(delta,
delta, delta));
                this.imgbmp5.SetPixel(c, r, Color.FromArgb(delta,
delta, delta));
            }
```

```
        }


//=================================================================================
=================
        // Creating intensity histogram

//=================================================================================
=================

        //int intensMax = 0, totIntens = 0, countIntens = 0;
        int intensMax = 0;
        //int lesionCount = 0;
        //int T, T1 = 0, T2 = 255, ene = 0;
        int T, T1 = 0, T2 = 255;

        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
            {
                pixelColor = this.imgbmp1.GetPixel(c, r);
                Rin = this.pixelColor.R;
                if (Rin > intensMax) intensMax = Rin;
                histIntens[Convert.ToInt32(Rin)]++;
            }
        }


//=================================================================================
=================
        // Calculating bitmap thresholds

//=================================================================================
=================
        //lesionPix = Convert.ToInt32(combiT * this.numPoints);
        lesionPix = Convert.ToInt32(combiT * intensMax);

        int maxPoints = numPoints;

        //for (c = 255; c >= 0; c--)
        //  {
        //        if    (histIntens[c] == 0) continue;
        //        else
        //              {
        //                    T1 = c;
        //                    break;
        //              }
        //  }
        for (c = 255; c >= 0; c--)
        {
            if (histIntens[c] != 0)
            {
                T2 = c;
                break;
            }
        }
        T1 = Convert.ToInt32(T2 * combiT);
```

```
        //for (c = 255; c >= T1; c--)
        //  {
        //        if (histIntens[c] == 0) continue;
        //        else
        //              {
        //                    T2 = c;
        //                    break;
        //              }
        //  }

        T = Convert.ToInt32((T1 + T2) / 2.0);


//=================================================================
=================
        // Copying imgbmp1 to imgbmp2 and imgbmp3

//=================================================================
=================

        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
            {
                this.imgbmp3.SetPixel(c, r, this.imgbmp1.GetPixel(c,
r));
                this.imgbmp2.SetPixel(c, r, this.imgbmp1.GetPixel(c,
r));
            }
        }


//=================================================================
=================
        // Calculating edges and saving them on imgbmp2

//=================================================================
=================
        int x, y, n, m, k, kMax = 0;
        int p1, p2, p3, p4, p5, p6, p7, p8;

        for (y = 1; y < imgbmp1.Height - 1; y++)
            for (x = 1; x < imgbmp1.Width - 1; x++)
            {
                pixelColor = imgbmp.GetPixel(x + 1, y - 1);
                p1 = Convert.ToInt32(pixelColor.R);

                pixelColor = imgbmp.GetPixel(x + 1, y);
                p2 = Convert.ToInt32(pixelColor.R);

                pixelColor = imgbmp.GetPixel(x + 1, y + 1);
                p3 = Convert.ToInt32(pixelColor.R);

                pixelColor = imgbmp.GetPixel(x - 1, y - 1);
                p4 = Convert.ToInt32(pixelColor.R);

                pixelColor = imgbmp.GetPixel(x, y - 1);
                p5 = Convert.ToInt32(pixelColor.R);
```

```
                pixelColor = imgbmp.GetPixel(x + 1, y - 1);
                p6 = Convert.ToInt32(pixelColor.R);

                pixelColor = imgbmp.GetPixel(x, y - 1);
                p7 = Convert.ToInt32(pixelColor.R);

                pixelColor = imgbmp.GetPixel(x, y + 1);
                p8 = Convert.ToInt32(pixelColor.R);

                n = (p1 + 2 * p2 + p3) - (p4 + 2 * p5 + p6);
                //Debug.WriteLine(n);
                m = (p3 + 2 * p8 + p6) - (p1 + 2 * p7 + p4);
                k = (int)(System.Math.Sqrt((double)(n * n + m * m)) /
4.0);
                //Debug.WriteLine(k);
                if (k > 255) k = 255;
                if (k > kMax) kMax = k;
                imgbmp2.SetPixel(x, y, Color.FromArgb(k, k, k));
            }
        multFac = Convert.ToInt32(255 / kMax);

        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
            {
                pixelColor = this.imgbmp2.GetPixel(c, r);
                Rin = this.pixelColor.R;
                if (Rin * multFac > (T1 - T1 * combiT))
this.imgbmp2.SetPixel(c, r, Color.Black);
                else this.imgbmp2.SetPixel(c, r, Color.White);
            }
        }


//=================================================================
=================
        // Copying imgbmp2 to imgbmp1 (just to show a temporary image
of the Sobel filter result)

//=================================================================
=================

        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
            {
                this.imgbmp1.SetPixel(c, r, this.imgbmp2.GetPixel(c,
r));
            }
        }



//=================================================================
=================
        // Copying imgbmp3 to imgbmp1 (Resetting imgbmp1)
```

```
//=================================================================
=================

        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
            {
                this.imgbmp1.SetPixel(c, r, this.imgbmp3.GetPixel(c,
r));
            }
        }


//=================================================================
=================
        // Applying bitmap thresholds

//=================================================================
=================

        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
            {
                pixelColor = this.imgbmp1.GetPixel(c, r);
                Rin = this.pixelColor.R;
                if (Rin >= T1 && Rin <= T2) imgbmp1.SetPixel(c, r,
Color.Black);
                else imgbmp1.SetPixel(c, r, Color.White);
                if (Rin >= (T1 + ((T - T1) / 2))) imgbmp4.SetPixel(c,
r, Color.Black);
                else imgbmp4.SetPixel(c, r, Color.White);
            }
        }



//=================================================================
=================
        // Filling imgbmp4 (T thresholding result)

//=================================================================
=================

        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
            {
                pixelColor = this.imgbmp4.GetPixel(c, r);
                Rin = this.pixelColor.R;
                if (Rin == 0) continue;
                cN = 0;
                //Search West for a black pixel
                for (k = c - 1; k > 0; k--)
                {
                    pixelColor = this.imgbmp4.GetPixel(k, r);
                    Rout = this.pixelColor.R;
                    if (Rout == 0)
```

```
                    {
                        cN++;
                        break;
                    }
                }
                //Search East for a black pixel
                for (k = c + 1; k < this.imgbmp1.Width; k++)
                {
                    pixelColor = this.imgbmp4.GetPixel(k, r);
                    Rout = this.pixelColor.R;
                    if (Rout == 0)
                    {
                        cN++;
                        break;
                    }
                }
                //Search North for a black pixel
                for (k = r - 1; k > 0; k--)
                {
                    pixelColor = this.imgbmp4.GetPixel(c, k);
                    Rout = this.pixelColor.R;
                    if (Rout == 0)
                    {
                        cN++;
                        break;
                    }
                }
                //Search South for a black pixel
                for (k = r + 1; k < this.imgbmp1.Height; k++)
                {
                    pixelColor = this.imgbmp4.GetPixel(c, k);
                    Rout = this.pixelColor.R;
                    if (Rout == 0)
                    {
                        cN++;
                        break;
                    }
                }
                // If a white pixel is surrounded with 4 black pixels,
change it to black
                if (cN == 4) this.imgbmp4.SetPixel(c, r, Color.Black);
            }
        }

        lesionPixels = 0;
        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
            {
                pixelColor = this.imgbmp4.GetPixel(c, r);
                Rout = this.pixelColor.R;
                if (Rout == 0) lesionPixels++;
            }
        }
```

```
//=================================================================================
        // Copying imgbmp1 to imgbmp3 (save T1 and T2 thresholding
result)

//=================================================================================

        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
            {
                this.imgbmp3.SetPixel(c, r, this.imgbmp1.GetPixel(c,
r));
            }
        }


//=================================================================================
        // Copying imgbmp4 to imgbmp1 (just to show a temporary image
of the threshold T result)

//=================================================================================

        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
            {
                this.imgbmp1.SetPixel(c, r, this.imgbmp4.GetPixel(c,
r));
            }
        }



//=================================================================================
        // Merging imgbmp2 and imgbmp3 into imgbmp1 (Reduced edges)
        // Points of the segmentation process which are also points of
the Sobel edges

//=================================================================================

        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
            {
                pixelColor = this.imgbmp3.GetPixel(c, r);
                Rin = this.pixelColor.R;
                pixelColor = this.imgbmp2.GetPixel(c, r);
                Rout = this.pixelColor.R;
                if (Rin == 0 && Rout == 0) this.imgbmp1.SetPixel(c, r,
Color.Black);
                else this.imgbmp1.SetPixel(c, r, Color.White);
            }
```

```
        }


//=================================================================================
        // Initializing imgbmp3 to White

//=================================================================================
        //Cleaning imgbmp3 (Set every pixel to white)
        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
            {
                this.imgbmp3.SetPixel(c, r, Color.White);
            }
        }


//=================================================================================
        // Adjusting imgbmp1 (Reduced edges) using imgbmp4 (T
segmentation) and giving imgbmp3
        // Searching which point from the segmentation is closer to
the detected edge

//=================================================================================
        int dMin = 0, dist = 0, dirMin = 0, cNMin = 0;

        for (c = 1; c < this.imgbmp4.Width - 1; c++)
        {
            for (r = 1; r < this.imgbmp4.Height - 1; r++)
            {
                pixelColor = this.imgbmp4.GetPixel(c, r);
                Rin = this.pixelColor.R;
                cN = 0;
                if (Rin == 0)
                {
                    for (k = c - 1; k <= c + 1; k++)
                        for (l = r - 1; l <= r + 1; l++)
                        {
                            pixelColor = this.imgbmp4.GetPixel(k, l);
                            Rout = this.pixelColor.R;
                            if (Rout == 0) cN++;
                        }
                    if (cN < 9)
                    {
                        cN = 0;
                        dMin = 999;
                        //Search West direction
                        for (k = c - 1; k > 0; k--)
                        {
                            pixelColor = this.imgbmp2.GetPixel(k, r);
                            Rout = this.pixelColor.R;
                            if (Rout == 0) dist = c - k;
                            if (dist < dMin)
                            {
```

```
                                cNMin = dist;
                                dMin = dist;
                                dirMin = 1;
                            }
                        }
                        //Search East direction
                        for (k = c + 1; k < this.imgbmp1.Width; k++)
                        {
                            pixelColor = this.imgbmp2.GetPixel(k, r);
                            Rout = this.pixelColor.R;
                            if (Rout == 0) dist = k - c;
                            if (dist < dMin)
                            {
                                cNMin = dist;
                                dMin = dist;
                                dirMin = 2;
                            }
                        }
                        //Search North direction
                        for (k = r - 1; k > 0; k--)
                        {
                            pixelColor = this.imgbmp2.GetPixel(c, k);
                            Rout = this.pixelColor.R;
                            if (Rout == 0) dist = r - k;
                            if (dist < dMin)
                            {
                                cNMin = dist;
                                dMin = dist;
                                dirMin = 3;
                            }
                        }
                        //Search South direction
                        for (k = r + 1; k < this.imgbmp1.Height; k++)
                        {
                            pixelColor = this.imgbmp1.GetPixel(c, k);
                            Rout = this.pixelColor.R;
                            if (Rout == 0) dist = k - r;
                            if (dist < dMin)
                            {
                                cNMin = dist;
                                dMin = dist;
                                dirMin = 4;
                            }
                        }
                        //Search NW direction
                        cN = 0;
                        if (r < c)
                        {
                            for (k = r - 1; k > 0; k--)
                            {
                                cN++;
                                pixelColor = this.imgbmp1.GetPixel(c -
cN, k);
                                Rout = this.pixelColor.R;
                                if (Rout == 0) dist =
Convert.ToInt32(Math.Sqrt(2 * (cN * cN)));
                                if (dist < dMin)
                                {
                                    cNMin = cN;
```

```
                                    dMin = dist;
                                    dirMin = 5;
                                }
                            }
                        }
                        else
                        {
                            for (k = c - 1; k > 0; k--)
                            {
                                cN++;
                                pixelColor = this.imgbmp1.GetPixel(k,
r - cN);
                                Rout = this.pixelColor.R;
                                if (Rout == 0) dist =
Convert.ToInt32(Math.Sqrt(2 * (cN * cN)));
                                if (dist < dMin)
                                {
                                    cNMin = cN;
                                    dMin = dist;
                                    dirMin = 5;
                                }
                            }
                        }
                        //Search SW direction
                        cN = 0;
                        if (this.imgbmp1.Height - r < c)
                        {
                            for (k = r + 1; k < this.imgbmp1.Height -
1; k++)
                            {
                                cN++;
                                pixelColor = this.imgbmp1.GetPixel(c -
cN, k);
                                Rout = this.pixelColor.R;
                                if (Rout == 0) dist =
Convert.ToInt32(Math.Sqrt(2 * (cN * cN)));
                                if (dist < dMin)
                                {
                                    cNMin = cN;
                                    dMin = dist;
                                    dirMin = 6;
                                }
                            }
                        }
                        else
                        {
                            for (k = c - 1; k > 0; k--)
                            {
                                cN++;
                                pixelColor = this.imgbmp1.GetPixel(c,
r + cN);
                                Rout = this.pixelColor.R;
                                if (Rout == 0) dist =
Convert.ToInt32(Math.Sqrt(2 * (cN * cN)));
                                if (dist < dMin)
                                {
                                    cNMin = cN;
                                    dMin = dist;
                                    dirMin = 6;
```

```
                                }
                            }
                        }
                        //Search NE direction
                        cN = 0;
                        if (r < this.imgbmp1.Width - c)
                        {
                            for (k = r - 1; k > 0; k--)
                            {
                                cN++;
                                pixelColor = this.imgbmp1.GetPixel(c +
cN, k);
                                Rout = this.pixelColor.R;
                                if (Rout == 0) dist =
Convert.ToInt32(Math.Sqrt(2 * (cN * cN)));
                                if (dist < dMin)
                                {
                                    cNMin = cN;
                                    dMin = dist;
                                    dirMin = 7;
                                }
                            }
                        }
                        else
                        {
                            for (k = c + 1; k < this.imgbmp1.Width -
1; k++)
                            {
                                cN++;
                                pixelColor = this.imgbmp1.GetPixel(c,
r - cN);
                                Rout = this.pixelColor.R;
                                if (Rout == 0) dist =
Convert.ToInt32(Math.Sqrt(2 * (cN * cN)));
                                if (dist < dMin)
                                {
                                    cNMin = cN;
                                    dMin = dist;
                                    dirMin = 7;
                                }
                            }
                        }
                        //Search SE direction
                        cN = 0;
                        if (this.imgbmp1.Height - r <
this.imgbmp1.Width - c)
                        {
                            for (k = r + 1; k < this.imgbmp1.Height -
1; k++)
                            {
                                cN++;
                                pixelColor = this.imgbmp1.GetPixel(c +
cN, k);
                                Rout = this.pixelColor.R;
                                if (Rout == 0) dist =
Convert.ToInt32(Math.Sqrt(2 * (cN * cN)));
                                if (dist < dMin)
                                {
                                    cNMin = cN;
```

```
                                    dMin = dist;
                                    dirMin = 8;
                                }
                            }
                        }
                        else
                        {
                            for (k = c + 1; k < this.imgbmp1.Width -
1; k++)
                            {
                                cN++;
                                pixelColor = this.imgbmp1.GetPixel(c,
r + cN);
                                Rout = this.pixelColor.R;
                                if (Rout == 0) dist =
Convert.ToInt32(Math.Sqrt(2 * (cN * cN)));
                                if (dist < dMin)
                                {
                                    cNMin = cN;
                                    dMin = dist;
                                    dirMin = 8;
                                }
                            }
                        }
                        if (dirMin == 1 && c - cNMin > 0 && c - cNMin
< this.imgbmp1.Width - 1 && r > 0 && r < this.imgbmp1.Height - 1)
this.imgbmp3.SetPixel(c - cNMin, r, Color.Black);
                        else if (dirMin == 2 && c + cNMin > 0 && c +
cNMin < this.imgbmp1.Width - 1 && r > 0 && r < this.imgbmp1.Height -
1) this.imgbmp3.SetPixel(c + cNMin, r, Color.Black);
                        else if (dirMin == 3 && c > 0 && c <
this.imgbmp1.Width - 1 && r - cNMin > 0 && r - cNMin <
this.imgbmp1.Height - 1) this.imgbmp3.SetPixel(c, r - cNMin,
Color.Black);
                        else if (dirMin == 4 && c > 0 && c <
this.imgbmp1.Width - 1 && r + cNMin > 0 && r + cNMin <
this.imgbmp1.Height - 1) this.imgbmp3.SetPixel(c, r + cNMin,
Color.Black);
                        else if (dirMin == 5 && c - cNMin > 0 && c -
cNMin < this.imgbmp1.Width - 1 && r - cNMin > 0 && r - cNMin <
this.imgbmp1.Height - 1) this.imgbmp3.SetPixel(c - cNMin, r - cNMin,
Color.Black);
                        else if (dirMin == 6 && c - cNMin > 0 && c -
cNMin < this.imgbmp1.Width - 1 && r + cNMin > 0 && r + cNMin <
this.imgbmp1.Height - 1) this.imgbmp3.SetPixel(c - cNMin, r + cNMin,
Color.Black);
                        else if (dirMin == 7 && c + cNMin > 0 && c +
cNMin < this.imgbmp1.Width - 1 && r - cNMin > 0 && r - cNMin <
this.imgbmp1.Height - 1) this.imgbmp3.SetPixel(c + cNMin, r - cNMin,
Color.Black);
                        else if (dirMin == 8 && c + cNMin > 0 && c +
cNMin < this.imgbmp1.Width - 1 && r + cNMin > 0 && r + cNMin <
this.imgbmp1.Height - 1) this.imgbmp3.SetPixel(c + cNMin, r + cNMin,
Color.Black);

                    }
                }
            }
        }
```

```
//=================================================================================
        // Copying imgbmp3 to imgbmp1 (to show thin edges)

//=================================================================================

        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
            {
                this.imgbmp1.SetPixel(c, r, this.imgbmp3.GetPixel(c,
r));
            }
        }



//=================================================================================
        // Cleaning imgbmp1 (White)

//=================================================================================

        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
            {
                this.imgbmp1.SetPixel(c, r, Color.White);
            }
        }


//=================================================================================
        //Calculating average value for intensity within already
defined edge

//=================================================================================
        int RoutMax, cRoutMax, rRoutMax, RoutTot, RoutAvg, contRout;

        contRout = 0;
        RoutTot = 0;
        RoutMax = 0;
        for (c = 1; c < this.imgbmp3.Width - 1; c++)
        {
            for (r = 1; r < this.imgbmp3.Height - 1; r++)
            {
                pixelColor = this.imgbmp3.GetPixel(c, r);
                Rin = this.pixelColor.R;
                if (Rin == 0)
                {
                    pixelColor = this.imgbmp5.GetPixel(c, r);
                    Rout = this.pixelColor.R;
                    RoutTot += Rout;
                    contRout++;
```

```
                    if (Rout > RoutMax)
                    {
                        RoutMax = Rout;
                        cRoutMax = c;
                        rRoutMax = r;
                    }
                }
            }
        }
        if (RoutTot != 0) RoutAvg = Convert.ToInt32(RoutTot /
contRout);


//====================================================================
=================
        // Creating an array with points on the edge
        // Ind, 0 . Column
        // Ind, 1 . Row
        // Ind, 2 . Intensity gradient
        // Ind, 3 . Lit - 0/1 -  2 . Processed
        // Ind, 4 . |R(s)|
        // Ind, 5 . Number of neighbors

//====================================================================
=================
        ptsLine = new int[contRout * 2, 6];
        cN = create_ptsLine();



//====================================================================
=================
        // Cleaning imgbmp1 and imgbmp3(White)

//====================================================================
=================

        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
            {
                this.imgbmp3.SetPixel(c, r, Color.White);
                this.imgbmp1.SetPixel(c, r, Color.White);
            }
        }


//====================================================================
=================
        // Resetting imgbmp3(Copying array points to image)

//====================================================================
=================
        int ind, ind1, cRout;

        for (ind = 0; ind < this.ptsLine.Length / 6 - 1; ind++)
        {
            if (ptsLine[ind, 0] == 0 && ptsLine[ind, 1] == 0) break;
            if (ptsLine[ind, 3] == 0) continue;
```

```
            this.imgbmp3.SetPixel(ptsLine[ind, 0], ptsLine[ind, 1],
Color.Black);
        }


//=====================================================================
=================
        // Closing the edge (recreating ptsLine array)

//=====================================================================
=================


        //int cRmin, rRmin, gaps = 1;
        int cRmin, rRmin;

        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
            {
                this.imgbmp3.SetPixel(c, r, Color.White);
                this.imgbmp1.SetPixel(c, r, Color.White);
            }
        }

        for (ind = 0; ind < this.ptsLine.Length / 6 - 1; ind++)
        {
            if (ptsLine[ind, 0] == 0 && ptsLine[ind, 1] == 0) break;
            if (ptsLine[ind, 3] == 0 || ptsLine[ind, 3] == 2)
continue;
            this.imgbmp3.SetPixel(ptsLine[ind, 0], ptsLine[ind, 1],
Color.Black);
        }

        for (c = 1; c <= this.imgbmp1.Width - 1; c++)
        {
            for (r = 1; r <= this.imgbmp1.Height - 1; r++)
            {
                allowable(c, r);
            }
        }

        //Next array element where a pixel should be saved
        int ind2 = ind, indMin;
        double distMin;


        for (ind1 = 0; ind1 <= this.ptsLine.Length / 6 - 1; ind1++)
        {
            if (ptsLine[ind1, 0] == 0 && ptsLine[ind1, 1] == 0) break;
            if (ptsLine[ind1, 3] == 0) continue;
            this.imgbmp1.SetPixel(ptsLine[ind1, 0], ptsLine[ind1, 1],
Color.Black);
        }


        for (c = 0; c < this.imgbmp1.Width; c++)
        {
            for (r = 0; r < this.imgbmp1.Height; r++)
```

```
            {
                this.imgbmp3.SetPixel(c, r, this.imgbmp1.GetPixel(c,
r));
                this.imgbmp1.SetPixel(c, r, Color.White);
            }
        }



//=====================================================================
==================
        // Filling ptsL array elements

//=====================================================================
==================

        ptsL = new int[2 * contRout, 3];
        double[,] distPt = new double[2 * contRout, 2];


        cRout = 0;
        for (c = 1; c < this.imgbmp3.Width - 1; c++)
        {
            for (r = 1; r < this.imgbmp3.Height - 1; r++)
            {
                pixelColor = this.imgbmp3.GetPixel(c, r);
                Rin = this.pixelColor.R;
                if (Rin == 0)
                {
                    pixelColor = this.imgbmp5.GetPixel(c, r);
                    Rout = this.pixelColor.R;
                    ptsL[cRout, 0] = c;
                    ptsL[cRout, 1] = r;
                    ptsL[cRout, 2] = Rout;
                    cRout++;
                }
            }
        }


        clean_NotAllowable();


        WiGeeC = new double[cRout];
        WiGeeR = new double[cRout];
        WjGeeC = new double[cRout];
        WjGeeR = new double[cRout];
        GeeC = new double[cRout];
        GeeR = new double[cRout];
        c_RaG = new double[cRout];
        r_RaG = new double[cRout];


//=====================================================================
================
        // Calculating Wi*Gi(u) and Wj*Gj(u) for every point on the
edge
```

```
//================================================================
=================

        for (ind = 0; ind < cRout - 1; ind++)
        {
            WiGeeC[ind] = 0;
            WiGeeR[ind] = 0;
            WjGeeC[ind] = 0;
            WjGeeR[ind] = 0;
            GeeC[ind] = 0;
            GeeR[ind] = 0;
            for (ind1 = 0; ind1 < cRout - 1; ind1++)
            {
                if (ind != ind1)
                {
                    if (ptsL[ind, 0] == ptsL[ind1, 0]) dist =
Math.Abs(ptsL[ind1, 1] - ptsL[ind, 1]);
                    else if (ptsL[ind, 1] == ptsL[ind1, 1]) dist =
Math.Abs(ptsL[ind, 0] - ptsL[ind1, 0]);
                    else dist = Convert.ToInt32(Math.Sqrt(((ptsL[ind,
0] - ptsL[ind1, 0]) * (ptsL[ind, 0] - ptsL[ind1, 0])) + ((ptsL[ind, 1]
- ptsL[ind1, 1]) * (ptsL[ind, 1] - ptsL[ind1, 1]))));
                }
                if (dist <= 5)
                {
                    GeeC[ind] += Math.Exp(-Math.Pow((ptsL[ind, 0] -
(ptsL[ind1, 0] + dist)), 2) / 2 * sigmaCombi * sigmaCombi);
                    GeeR[ind] += Math.Exp(-Math.Pow((ptsL[ind, 1] -
(ptsL[ind1, 1] + dist)), 2) / 2 * sigmaCombi * sigmaCombi);
                }
            }
            WiGeeC[ind] += GeeC[ind] * ptsL[ind, 2];
            WiGeeR[ind] += GeeR[ind] * ptsL[ind, 2];
            WjGeeC[ind] += WiGeeC[ind];
            WjGeeR[ind] += WiGeeR[ind];
        }

        for (ind = 0; ind < cRout - 1; ind++)
        {
            if (WjGeeC[ind] > 0) c_RaG[ind] = ptsL[ind, 0] *
WiGeeC[ind] / WjGeeC[ind];
            if (WjGeeR[ind] > 0) r_RaG[ind] = ptsL[ind, 1] *
WiGeeR[ind] / WjGeeR[ind];
            ptsL[ind, 0] = 0;
            ptsL[ind, 1] = 0;
            ptsL[ind, 2] = 0;


//================================================================
=================
            // Drawing the RaG curve
            // filling the new line array

//================================================================
=================

            this.imgbmp1.SetPixel(Convert.ToInt32(c_RaG[ind]),
Convert.ToInt32(r_RaG[ind]), Color.Black);
```

```
            ptsL[ind, 0] = Convert.ToInt32(c_RaG[ind]);
            ptsL[ind, 1] = Convert.ToInt32(r_RaG[ind]);

        }


//=================================================================
=================
        // Creating a closed contour

//=================================================================
=================

        ptsDraw = new int[cRout, 4];

        for (int pp = 0; pp < countDraw; pp++)
        {
            ptsDraw[pp, 0] = 0;
            ptsDraw[pp, 1] = 0;
            ptsDraw[pp, 2] = 0;
            ptsDraw[pp, 3] = 0;
        }

        indMin = 0;
        ind = 0;
        countDraw = 0;
        distMin = Math.Max(this.imgbmp1.Width, this.imgbmp1.Height);
        for (ind1 = 1; ind1 < cRout - 1; ind1++)
        {
            dist = Convert.ToInt32(Math.Sqrt((ptsL[ind, 0] -
ptsL[ind1, 0]) * (ptsL[ind, 0] - ptsL[ind1, 0]) + (ptsL[ind, 1] -
ptsL[ind1, 1]) * (ptsL[ind, 1] - ptsL[ind1, 1])));
            if (dist < distMin && dist >= 1.0)
            {
                distMin = dist;
                indMin = ind1;
                cRmin = ptsL[ind1, 0];
                rRmin = ptsL[ind1, 1];
            }

        }
        for (int ind3 = 0; ind3 < cRout - 1; ind3++)
        {
            if (ind3 == indMin) continue;
            dist = Convert.ToInt32(Math.Sqrt((ptsL[ind, 0] -
ptsL[ind3, 0]) * (ptsL[ind, 0] - ptsL[ind3, 0]) + (ptsL[ind, 1] -
ptsL[ind3, 1]) * (ptsL[ind, 1] - ptsL[ind3, 1])));
            if (dist == distMin && dist < Math.Max(this.imgbmp1.Width,
this.imgbmp1.Height))
            {
                ptsL[ind3, 0] = 0;
                ptsL[ind3, 1] = 0;
                ptsL[ind3, 2] = 9999;
            }
        }

        int indOld = ind;
        ptsDraw[0, 0] = ptsL[ind, 0];
```

```
            ptsDraw[0, 1] = ptsL[ind, 1];
            ptsDraw[0, 2] = ptsL[indMin, 0];
            ptsDraw[0, 3] = ptsL[indMin, 1];
            ind = indMin;

            //int count1 = 1;
            countDraw++;
            do
            {
                if (ind >= ptsL.Length / 3) break;
                if (ptsL[ind, 0] == 0 && ptsL[ind, 1] == 0)
                {
                    ind++;
                    continue;
                }
                if (ptsL[ind, 2] != 0)
                {
                    ind++;
                    continue;
                }
                distMin = Math.Max(this.imgbmp1.Width,
this.imgbmp1.Height);
                cRmin = this.imgbmp1.Width;
                rRmin = this.imgbmp1.Height;
                for (ind1 = 0; ind1 < cRout - 1; ind1++)
                {
                    if (ind1 == 0 && countDraw < cRout * 2 / 3) continue;
                    if (ind == ind1) continue;
                    if (ind1 == indOld) continue;
                    if (ptsL[ind1, 0] == 0 && ptsL[ind1, 1] == 0)
continue;
                    if (ptsL[ind1, 2] != 0) continue;
                    dist = Convert.ToInt32(Math.Sqrt((ptsL[ind, 0] -
ptsL[ind1, 0]) * (ptsL[ind, 0] - ptsL[ind1, 0]) + (ptsL[ind, 1] -
ptsL[ind1, 1]) * (ptsL[ind, 1] - ptsL[ind1, 1])));
                    if (dist >= 1.0 && dist < distMin)
                    {
                        distMin = dist;
                        indMin = ind1;
                        cRmin = ptsL[ind1, 0];
                        rRmin = ptsL[ind1, 1];
                    }
                }
                if (ind == indMin || distMin > 8.0) break;
                ptsL[ind, 2] = indMin;
                ptsDraw[countDraw, 0] = ptsL[ind, 0];
                ptsDraw[countDraw, 1] = ptsL[ind, 1];
                ptsDraw[countDraw, 2] = ptsL[indMin, 0];
                ptsDraw[countDraw, 3] = ptsL[indMin, 1];
                for (int ind3 = 0; ind3 < cRout - 1; ind3++)
                {
                    if (ind3 == indMin) continue;
                    dist = Convert.ToInt32(Math.Sqrt((ptsL[ind, 0] -
ptsL[ind3, 0]) * (ptsL[ind, 0] - ptsL[ind3, 0]) + (ptsL[ind, 1] -
ptsL[ind3, 1]) * (ptsL[ind, 1] - ptsL[ind3, 1])));
                    if (ind3 != 0 && dist == distMin && dist <
Math.Max(this.imgbmp1.Width, this.imgbmp1.Height))
                    {
                        ptsL[ind3, 0] = 0;
```

```
                    ptsL[ind3, 1] = 0;
                    ptsL[ind3, 2] = 9999;
                }
            }

            indOld = ind;
            ind = indMin;
            countDraw++;
        }
        while (ind != 0);



        if (ind != 0)
        {
            //MsgBox.Show("Sorry! I could not define enough points to
build an edge.\nPlease try again with another
threshold","Warning",MessageBoxButtons.OK, MessageBoxIcon.Warning);
            // return an error
            return;
        }

//=====================================================================
=================
        // plot edge to imgbmp

//=====================================================================
=================


        Graphics g5 = Graphics.FromImage((Image)imgbmp);
        Pen pen1 = new Pen(Color.White);
        for (int pp = 0; pp < ptsDraw.Length / 4; pp++)
        {
            if (ptsDraw[pp, 0] == 0 && ptsDraw[pp, 1] == 0) break;
            g5.DrawLine(pen1, ptsDraw[pp, 0], ptsDraw[pp, 1],
ptsDraw[pp, 2], ptsDraw[pp, 3]);
        }

        StepOK = true;
        return;

    }

    private void conv_CIE(int pixel_R, int pixel_G, int pixel_B)
    {

        var_R = Convert.ToDouble(pixel_R / 255.0);
        //Where R = 0 ÷ 255
        var_G = Convert.ToDouble(pixel_G / 255.0);
        //Where G = 0 ÷ 255
        var_B = Convert.ToDouble(pixel_B / 255.0);
        //Where B = 0 ÷ 255
        if (var_R > 0.03928) var_R = Math.Pow((var_R + 0.055) / 1.055,
2.4);
        else var_R = var_R / 12.92;
        if (var_G > 0.03928) var_G = Math.Pow((var_G + 0.055) / 1.055,
2.4);
        else var_G = var_G / 12.92;
```

```
        if (var_B > 0.03928) var_B = Math.Pow((var_B + 0.055) / 1.055,
2.4);
        else var_B = var_B / 12.92;

        var_R = var_R * 100.0;
        var_G = var_G * 100.0;
        var_B = var_B * 100.0;

        //Observer. = 2°, Illuminant = D65
        X = var_R * 0.4124 + var_G * 0.3576 + var_B * 0.1805;
        Y = var_R * 0.2126 + var_G * 0.7152 + var_B * 0.0722;
        Z = var_R * 0.0193 + var_G * 0.1192 + var_B * 0.9505;

        // Conversion from XYZ to CIEL*ab

        var_X = X / ref_X;
        //ref_X =  95.047  Observer= 2°, Illuminant= D65
        var_Y = Y / ref_Y;
        //ref_Y = 100.000
        var_Z = Z / ref_Z;
        //ref_Z = 108.883

        if (var_X > 0.008856) var_X = Math.Pow(var_X, 1.0 / 3.0);
        else var_X = (7.787 * var_X) + (16.0 / 116.0);
        if (var_Y > 0.008856) var_Y = Math.Pow(var_Y, 1.0 / 3.0);
        else var_Y = (7.787 * var_Y) + (16.0 / 116.0);
        if (var_Z > 0.008856) var_Z = Math.Pow(var_Z, 1.0 / 3.0);
        else var_Z = (7.787 * var_Z) + (16.0 / 116.0);
        CIE_L = (116.0 * var_Y) - 16.0;
        CIE_a = 500.0 * (var_X - var_Y);
        CIE_b = 200.0 * (var_Y - var_Z);
    }

    private int create_ptsLine()
    {
        int c, r, Rin, Rout;
        cN = 0;
        for (c = 1; c < this.imgbmp3.Width - 1; c++)
        {
            for (r = 1; r < this.imgbmp3.Height - 1; r++)
            {
                pixelColor = this.imgbmp3.GetPixel(c, r);
                Rin = this.pixelColor.R;
                if (Rin == 0)
                {
                    pixelColor = this.imgbmp2.GetPixel(c, r);
                    Rout = this.pixelColor.R;
                    if (Rout == 255) continue;
                    ptsLine[cN, 0] = c;
                    ptsLine[cN, 1] = r;
                    ptsLine[cN, 2] = Rout;
                    ptsLine[cN, 3] = 1;
                    ptsLine[cN, 4] = 0;
                    ptsLine[cN, 5] = 0;
                    for (k = c - 1; k <= c + 1; k++)
                    {
                        for (l = r - 1; l <= r + 1; l++)
                        {
                            pixelColor = this.imgbmp3.GetPixel(k, l);
```

```
                            Rout = this.pixelColor.R;
                            if (Rout == 0)
                            {
                                ptsLine[cN, 5]++;
                                if (k == c - 1 && l == r ||
                                    k == c + 1 && l == r ||
                                    k == c && l == r - 1 ||
                                    k == c && l == r + 1) ptsLine[cN,
4]++;
                            }

                        }
                    }
                    cN++;
                }
            }
        }
        return cN;
    }

    private void clean_NotAllowable()
    {


//=====================================================================
=================
        // Unlight lit not allowable sites

//=====================================================================
=================

        int ind, x, y, Rin, Rout, countB;
        for (ind = 0; ind < this.ptsL.Length / 3; ind++)
        {
            pixelColor = this.imgbmp3.GetPixel(ptsL[ind, 0], ptsL[ind,
1]);
            Rin = this.pixelColor.R;
            if (Rin == 0)
            {
                countB = 0;
                for (x = ptsL[ind, 0] - 1; x <= ptsL[ind, 0] + 1; x++)
                {
                    for (y = ptsL[ind, 1] - 1; y <= ptsL[ind, 1] + 1;
y++)
                    {
                        pixelColor = this.imgbmp3.GetPixel(x, y);
                        Rout = this.pixelColor.R;
                        if (Rout == 0) countB++;
                    }
                }
                if (countB < 2)
                {
                    this.imgbmp3.SetPixel(ptsL[ind, 0], ptsL[ind, 1],
Color.White);
                    this.ptsL[ind, 0] = 0;
                    this.ptsL[ind, 1] = 0;
                    this.ptsL[ind, 2] = 9999;
                }
            }
```

```
        }
    }

    private void allowable(int c, int r)
    {


//=================================================================
=================
        // Unlight lit sites with |R(s)| = 2 and configurations like:
        //    *       x *       *     * x
        // A  x *   B  *    C  * x    D  *

//=================================================================
=================
        int Rin, Rout, ind;

        if (c == 0 && r == 0) return;
        for (ind = 0; ind <= this.ptsLine.Length / 6 - 1; ind++)
        {
            if (ptsLine[ind, 0] != c || ptsLine[ind, 1] != r)
continue;
            if (ptsLine[ind, 3] == 0) return;
            if (ptsLine[ind, 4] == 2)
            {
                //Situation A
                pixelColor = this.imgbmp3.GetPixel(c + 1, r);
                Rin = pixelColor.R;
                pixelColor = this.imgbmp3.GetPixel(c, r - 1);
                Rout = pixelColor.R;
                if (Rin == 0 && Rout == 0)
                {
                    this.ptsLine[ind, 3] = 0;
                    this.imgbmp3.SetPixel(c, r, Color.White);
                    //nAllow = 1;
                    return;
                }
                //Situation B
                pixelColor = this.imgbmp3.GetPixel(c + 1, r);
                Rin = pixelColor.R;
                pixelColor = this.imgbmp3.GetPixel(c, r + 1);
                Rout = pixelColor.R;
                if (Rin == 0 && Rout == 0)
                {
                    this.ptsLine[ind, 3] = 0;
                    this.imgbmp3.SetPixel(c, r, Color.White);
                    //nAllow = 1;
                    return;
                }
                //Situation C
                pixelColor = this.imgbmp3.GetPixel(c - 1, r);
                Rin = pixelColor.R;
                pixelColor = this.imgbmp3.GetPixel(c, r - 1);
                Rout = pixelColor.R;
                if (Rin == 0 && Rout == 0)
                {
                    this.ptsLine[ind, 3] = 0;
                    this.imgbmp3.SetPixel(c, r, Color.White);
                    //nAllow = 1;
```

```
                    return;
                }
                //Situation D
                pixelColor = this.imgbmp3.GetPixel(c - 1, r);
                Rin = pixelColor.R;
                pixelColor = this.imgbmp3.GetPixel(c, r + 1);
                Rout = pixelColor.R;
                if (Rin == 0 && Rout == 0)
                {
                    this.ptsLine[ind, 3] = 0;
                    this.imgbmp3.SetPixel(c, r, Color.White);
                    //nAllow = 1;
                    return;
                }
            }
        }
    }

}
```

Code for a program that acts as an interface and the database engine.
He receives as an input an extended syntax SQL and shows the resulting dataset.
Form layout:

```
namespace TestForm
{
    partial class frmTestSQLStr
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should
be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.txtSQLOri = new System.Windows.Forms.TextBox();
            this.lblSQLOri = new System.Windows.Forms.Label();
            this.OFDSqlOri = new
System.Windows.Forms.OpenFileDialog();
            this.btnSQLOri = new System.Windows.Forms.Button();
            this.lblSQLRes = new System.Windows.Forms.Label();
            this.btnClear = new System.Windows.Forms.Button();
            this.btnProc = new System.Windows.Forms.Button();
            this.btnExit = new System.Windows.Forms.Button();
            this.DgResults = new System.Windows.Forms.DataGridView();
            this.BtnSave = new System.Windows.Forms.Button();
            this.SFDSqlOri = new
System.Windows.Forms.SaveFileDialog();

((System.ComponentModel.ISupportInitialize)(this.DgResults)).BeginInit
();
            this.SuspendLayout();
            //
            // txtSQLOri
            //
            this.txtSQLOri.AcceptsReturn = true;
            this.txtSQLOri.Location = new System.Drawing.Point(12,
28);
```

```
            this.txtSQLOri.Multiline = true;
            this.txtSQLOri.Name = "txtSQLOri";
            this.txtSQLOri.Size = new System.Drawing.Size(506, 182);
            this.txtSQLOri.TabIndex = 1;
            //
            // lblSQLOri
            //
            this.lblSQLOri.AutoSize = true;
            this.lblSQLOri.Location = new System.Drawing.Point(12, 9);
            this.lblSQLOri.Name = "lblSQLOri";
            this.lblSQLOri.Size = new System.Drawing.Size(76, 13);
            this.lblSQLOri.TabIndex = 0;
            this.lblSQLOri.Text = "Extended SQL";
            //
            // OFDSqlOri
            //
            this.OFDSqlOri.Title = "Choose a SQL File";
            //
            // btnSQLOri
            //
            this.btnSQLOri.Location = new System.Drawing.Point(406,
216);
            this.btnSQLOri.Name = "btnSQLOri";
            this.btnSQLOri.Size = new System.Drawing.Size(51, 23);
            this.btnSQLOri.TabIndex = 2;
            this.btnSQLOri.Text = "Open";
            this.btnSQLOri.UseVisualStyleBackColor = true;
            this.btnSQLOri.Click += new
System.EventHandler(this.btnSQLOri_Click);
            //
            // lblSQLRes
            //
            this.lblSQLRes.AutoSize = true;
            this.lblSQLRes.Location = new System.Drawing.Point(13,
240);
            this.lblSQLRes.Name = "lblSQLRes";
            this.lblSQLRes.Size = new System.Drawing.Size(75, 13);
            this.lblSQLRes.TabIndex = 4;
            this.lblSQLRes.Text = "Resulting data";
            //
            // btnClear
            //
            this.btnClear.Location = new System.Drawing.Point(306,
485);
            this.btnClear.Name = "btnClear";
            this.btnClear.Size = new System.Drawing.Size(77, 30);
            this.btnClear.TabIndex = 6;
            this.btnClear.Text = "Clear Panes";
            this.btnClear.UseVisualStyleBackColor = true;
            this.btnClear.Click += new
System.EventHandler(this.btnClear_Click);
            //
            // btnProc
            //
            this.btnProc.Location = new System.Drawing.Point(389,
485);
            this.btnProc.Name = "btnProc";
            this.btnProc.Size = new System.Drawing.Size(68, 30);
            this.btnProc.TabIndex = 7;
```

```
            this.btnProc.Text = "Process";
            this.btnProc.UseVisualStyleBackColor = true;
            this.btnProc.Click += new
System.EventHandler(this.btnProc_Click);
            //
            // btnExit
            //
            this.btnExit.Location = new System.Drawing.Point(463,
486);
            this.btnExit.Name = "btnExit";
            this.btnExit.Size = new System.Drawing.Size(55, 29);
            this.btnExit.TabIndex = 8;
            this.btnExit.Text = "Exit";
            this.btnExit.UseVisualStyleBackColor = true;
            this.btnExit.Click += new
System.EventHandler(this.btnExit_Click);
            //
            // DgResults
            //
            this.DgResults.ColumnHeadersHeightSizeMode =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize;
            this.DgResults.Location = new System.Drawing.Point(12,
256);
            this.DgResults.Name = "DgResults";
            this.DgResults.Size = new System.Drawing.Size(506, 223);
            this.DgResults.TabIndex = 5;
            //
            // BtnSave
            //
            this.BtnSave.Location = new System.Drawing.Point(463,
217);
            this.BtnSave.Name = "BtnSave";
            this.BtnSave.Size = new System.Drawing.Size(55, 22);
            this.BtnSave.TabIndex = 3;
            this.BtnSave.Text = "Save";
            this.BtnSave.UseVisualStyleBackColor = true;
            this.BtnSave.Click += new
System.EventHandler(this.BtnSave_Click);
            //
            // SFDSqlOri
            //
            this.SFDSqlOri.Title = "Choose a destination";
            //
            // frmTestSQLStr
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F,
13F);
            this.AutoScaleMode =
System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(531, 528);
            this.Controls.Add(this.BtnSave);
            this.Controls.Add(this.DgResults);
            this.Controls.Add(this.btnExit);
            this.Controls.Add(this.btnProc);
            this.Controls.Add(this.btnClear);
            this.Controls.Add(this.lblSQLRes);
            this.Controls.Add(this.btnSQLOri);
            this.Controls.Add(this.lblSQLOri);
            this.Controls.Add(this.txtSQLOri);
```

```
            this.Name = "frmTestSQLStr";
            this.Text = "Extended SQL Test Pane";

((System.ComponentModel.ISupportInitialize)(this.DgResults)).EndInit()
;
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.TextBox txtSQLOri;
        private System.Windows.Forms.Label lblSQLOri;
        private System.Windows.Forms.OpenFileDialog OFDSqlOri;
        private System.Windows.Forms.Button btnSQLOri;
        private System.Windows.Forms.Label lblSQLRes;
        private System.Windows.Forms.Button btnClear;
        private System.Windows.Forms.Button btnProc;
        private System.Windows.Forms.Button btnExit;
        private System.Windows.Forms.DataGridView DgResults;
        private System.Windows.Forms.Button BtnSave;
        private System.Windows.Forms.SaveFileDialog SFDSqlOri;
    }
}
```

Form code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace TestForm
{
    public partial class frmTestSQLStr : Form
    {
        SqlConnection conn = new
SqlConnection("server=localhost;database=RMExtension;Trusted_Connectio
n=True");

        public frmTestSQLStr()
        {
            InitializeComponent();
        }

        private void btnSQLOri_Click(object sender, EventArgs e)
        {
            OFDSqlOri.AddExtension = true;
            OFDSqlOri.DefaultExt = "sql";
            OFDSqlOri.RestoreDirectory = true;
            OFDSqlOri.ShowDialog(this);
            if (!String.IsNullOrEmpty(OFDSqlOri.FileName))
                txtSQLOri.Text = new
System.IO.StreamReader(OFDSqlOri.FileName).ReadToEnd();
        }

        private void BtnSave_Click(object sender, EventArgs e)
        {
            if (String.IsNullOrEmpty(txtSQLOri.Text))
            {
                MessageBox.Show("You must have something to save",
"Empty Window");
            }
            else
            {
                SFDSqlOri.AddExtension = true;
                SFDSqlOri.DefaultExt = "sql";
                SFDSqlOri.RestoreDirectory = true;
                SFDSqlOri.ShowDialog(this);
                if (!String.IsNullOrEmpty(SFDSqlOri.FileName))
                {
                    System.IO.StreamWriter sr = new
System.IO.StreamWriter(SFDSqlOri.FileName);
                    sr.Write(txtSQLOri.Text);
                    sr.Flush();
                    sr.Close();
                }
            }
```

```
        }

        private void btnClear_Click(object sender, EventArgs e)
        {
            txtSQLOri.Text = "";
            DgResults.DataSource = "";
        }

        private void btnExit_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void btnProc_Click(object sender, EventArgs e)
        {

            DataTable dto = new DataTable();
            try
            {
                conn.Open();
                string SQL = "EXEC RMExtension '" + txtSQLOri.Text +
"'";
                SqlDataAdapter da = new SqlDataAdapter(SQL, conn);
                da.Fill(dto);
                conn.Close();
            }
            catch (Exception ex)
            {
                dto = new DataTable();
                DataColumn dc = new DataColumn("Info",
Type.GetType("System.String"));
                dto.Columns.Add(dc);
                DataRow dr = dto.NewRow();
                dr["Info"] = ex.Message;
                dto.Rows.Add(dr);
            }
            DgResults.DataSource = dto;
        }

    }

}
```

Listing of Alfa.cs

It implements the ordering function “SqlAlfa()” that implements sorting with pre-defined order words

```
using System;
using System.Collections;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;


public partial class StoredProcedures
{
    //private static readonly string[] sNumbers = { "ONE", "TWO",
"THREE", "FOUR", "FIVE", "SIX", "SEVEN", "EIGHT", "NINE" };
    private static readonly string[] sNumbers = { "FIRST", "SECOND",
"THIRD", "FOURTH", "FIFTH", "SIXTH", "SEVENTH", "EIGHTH", "NINTH" };

    [Microsoft.SqlServer.Server.SqlProcedure]
    public static void SqlAlfa(string strSQL)
    {
        // Test if we are under an SQL server connection
        if (!SqlContext.IsAvailable)
            return;

        SqlConnection conn = new SqlConnection("context
connection=true");

        // assume that the SQL is properly formed and
        // get the data out of the engine
        DataTable dt = null;
        try
        {
            dt = SQLUtils.lData(strSQL, conn);
        }
        catch (Exception e)
        {
            dt = null;
            SqlContext.Pipe.Send(e.Message);
        }
        if (dt != null)
        {

            //create the output table
            DataTable dtout = new DataTable();

            //find a suitable name for the "extra" field
            string IDname = "ID";
            int i = 0;
            bool goodname = false;
            while (!goodname)
            {
                goodname = true;
                foreach (DataColumn dc in dt.Columns)
                {
                    if (dc.ColumnName == IDname)
```

```
                    {
                        IDname = "ID" + i.ToString("00");
                        ++i;
                        goodname = false;
                        break;
                    }
                }
            }

            //now populate the columns collection of the output table
            foreach (DataColumn dc in dt.Columns)
            {
                dtout.Columns.Add(dc.ColumnName, dc.DataType);
            }
            // and add the ID column for the random number
            dtout.Columns.Add(IDname, Type.GetType("System.Int32"));


            // do all the sorting between dt and dtout
            foreach (DataRow dr in dt.Rows)
            {
                //populate the output table with data
                DataRow drout = dtout.NewRow();
                foreach (DataColumn dc in dt.Columns)
                {
                    drout[dc.ColumnName] = dr[dc.ColumnName];
                }
                int nID = 0;
                string cval =
dr[dt.Columns[0].ColumnName].ToString().ToUpper();

                for (i = 0; i < sNumbers.Length; i++)
                {
                    nID = 0;
                    string n = sNumbers[i];
                    if (cval.ToUpper().Contains(" " + n) ||
cval.ToUpper().Contains(n + " ") || cval == n)
                    {
                        nID = i + 1;
                        break;
                    }
                }
                drout[IDname] = nID;
                dtout.Rows.Add(drout);
            }


            DataView dv = dtout.DefaultView;
            dv.Sort = IDname;

            // send the results back to the server
            i = 0;
            SqlMetaData[] flds = SQLUtils.GetMetaData(strSQL, conn);

            SqlDataRecord rec = new SqlDataRecord(flds);
            SqlContext.Pipe.SendResultsStart(rec);
            DataTable dto = dv.ToTable();
            foreach (DataRow dr in dto.Rows)
            {
```

```
                for (i = 0; i < dtout.Columns.Count - 1; i++)
                {
                    rec.SetValue(i, dr[i]);
                }
                SqlContext.Pipe.SendResultsRow(rec);
            }
            SqlContext.Pipe.SendResultsEnd();
        }
    }
};
```

Solutions of the Case Studies.

The random list:
SELECT * FROM Hotels ORDER BY SqlRandom()
(EXEC RMExtension “SELECT * FROM Hotels ORDER BY SqlRandom()”, if done directly on the SQL Server Management Studio)

The Budget selection:
SELECT *,ACCUMUL(MonthValue) FROM Budget
(EXEC RMExtension “SELECT *,ACCUMUL(MonthValue) FROM Budget”, if done directly on the SQL Server Management Studio)

The Running Sum query:
SELECT CandidateName, CandidateBenefit, ACCUMUL(CandidateBenefit) AS RSum FROM Candidates WHERE ACCUMUL(CandidateBenefit) < 500
(EXEC RMExtension “SELECT CandidateName, CandidateBenefit, ACCUMUL(CandidateBenefit) AS RSum FROM Candidates WHERE ACCUMUL(CandidateBenefit) < 500”, if done directly on the SQL Server Management Studio)

In all these solutions, the data could be immediately used.