

A Comparative Evaluation and Investigation of MS SQL Server 2000 and Oracle 9i with respect to Integrity and SQL 2003 Standards

*Thesis submitted in partial fulfilment of the requirements of the
Bachelor of Science (Honours) in Computer Science Degree at Rhodes University*



RHODES UNIVERSITY
Where leaders learn

By

Paul Tarwireyi

Supervisor: John Ebdon

07 November 2005

Abstract

Database Management and transaction processing systems occupy a crucial position in our information technology-based world. It is imperative that these systems function correctly and reflect real world actions on the data that they store, manage and manipulate.

The constantly evolving nature of RDBMSs has lead to database wars among the various vendors in the market. This is evidenced by each vendor in the market making claims of the superiority of his product, hence making the task of choosing a RDBMS not an easy one for a DBA. Thus, the DBMS selection process requires consideration, knowledge and skills.

One of the major drives behind the development of RDBMSs is to ensure data consistency, yet this is one of those things that do not seem like an obvious topic for Database Administrators to address directly. Furthermore this has been totally ignored by database benchmarks.

Oracle and SQL Server are well established DBMSs, which are amongst the world's "Big three" DBMSs and are very strong rivals. This project gives an overview of a comparative evaluation of Oracle 9i and Microsoft SQL Server 2000 with respect to Integrity and conformity to the SQL 2003 standards. The results of testing and evaluating the current Database Management Systems help to highlight the problems found in each, hence allowing for improvements if necessary.

Experiments were carried out to test for integrity and also an investigation of their conformity to the SQL 2003 standards is made. On the integrity issue, it is found that both products come with the necessary tools and functionalities to implement and maintain data integrity, thus they were found to be equal. However on the standards conformance part, although both DBMSs are not SQL 2003 conformant, Oracle 9i support more standard features than SQL Server 2000, hence it is leading.

Acknowledgements

It is a pleasure to thank all the people who made this thesis possible. First and foremost, I would like to express my sincere gratitude to my supervisor John Ebden. With his inspiration, and great efforts to explain things, he helped me see the light in my project. Throughout my project, he provided encouragement and continued support. Without his supervision and guidance, the year would have been unbearable.

I would like to extend my gratitude to the managerial and technical staff of the Rhodes University Computer Science Department for all their support, specifically Carol Watkins, Jody Balarin, Jock Forrester and Chris Morley.

I must also acknowledge the financial support I received this year through the Andrew Mellon Foundation and Rhodes University.

To my friends, proof-readers and classmates, I also extend hearty thanks, without you guys the year would have been unbearable.

TABLE OF CONTENTS

Chapter 1: Background	10
1.1 Introduction.....	10
1.2 Aim.....	10
1.3 Motivation.....	11
1.4 Project Overview.....	11
1.4.1 The evaluation of data integrity	13
1.4.2 The evaluation of conformity to SQL 2003 standards.....	14
1.4.2.1 SQL Standards	14
1.4.2.2 Levels of conformance.....	17
1.4.2.3 The SQL 2003 standards.....	17
1.5 Database Management system selection criteria	21
1.6 Overview of Oracle.....	22
1.7 Overview of SQL Server.....	24
1.8 Summary of Chapter	24
Chapter 2: Design Considerations	25
2.1 Considerations for integrity tests	25
2.1.1 External factors	25
2.1.2 Operating System.....	25
2.1.3 Software	25
2.1.4 Sufficient tests.....	26
2.1.5 Accurate tests	26
2.2 Considerations for SQL 2003 standards.	26
2.3 Design of Integrity experiments.....	28
2.3.1 Choosing a dataset.	28
2.3.2 Hypothesis and Experiments design	28
2.3.3 Implementation of tests.....	29
2.3.4 Collection of results	29
2.3.5 Analysis of results.....	29
2.3.6 Drawing conclusions.....	29
2.4 Summary of chapter	29
Chapter 3: Integrity constraints experiments	30

Table of Contents

3.1 Entity integrity tests	30
3.1.1 PRIMARY KEY tests	30
3.1.2 UNIQUE KEY tests	32
3.1.2.1 Analysis of error messages	34
3.1.3 Identity property.....	34
3.1.4 Overall analysis of Entity integrity tests	34
3.2 Referential Integrity Tests.....	34
3.2.1 Summary of error messages.....	36
3.2.2 Analysis of error messages	37
3.2.3 Overall analysis of referential integrity tests	37
3.3 Domain Integrity tests.....	37
3.3.1 String or Character Tests (char, varchar and nchar)	38
3.3.1.1 Summary of error messages.....	38
3.3.1.2 Analysis of results.....	38
3.3.2 Numeric data type tests	39
3.3.2.1 Exact numeric data types	39
3.3.2.2 Approximate numeric data types	42
3.3.3 NOT NULL tests.....	44
3.3.4 Check constraints tests.....	44
3.3.4.1 Summary of error messages.....	44
3.3.5 DEFAULTS Tests.....	46
3.3.6 Overall analysis of domain integrity constraints	46
3.4 User - Defined Integrity	46
3.5 Summary of Chapter	47
Chapter 4: Transactions and concurrency control.....	48
4.1 Types of transactions	48
4.2 Transactions tests	48
4.3 ACID Properties.....	49
4.4 Atomicity Tests.....	49
4.5 Interactions and isolation levels.....	52
4.6 Concurrency in a nutshell	53
4.7 Summary Conclusion.....	54
Chapter 5: SQL Standards conformance	55

Table of Contents

5.1 SQL Standards investigation.....	55
5.2 Weighting of all the SQL 2003 results.....	62
5.3 Summary of Chapter	64
Chapter 6: Conclusions and Possible Extensions.....	65
6.1 Conclusions.....	65
6.2 Possible extensions	65
6.2.1 Evaluating the latest versions: SQL Server 2005 and Oracle 10g.....	65
6.2.2 Evaluating DBMSs with respect to Security	65
Appendix A: Prerequisite for Investigation and Implementation.....	66
1. Overview of PL/SQL	66
2. Overview of T-SQL	67
Appendix B: Integrity constraints	68
1.1 The motive behind the maintenance of data integrity.....	68
1.1.1 Protecting the data existence.....	69
1.1.2 Maintaining quality.....	69
1.1.3 Ensuring Confidentiality	69
1.2 Database Structure Integrity.....	69
1.3 Semantic Data Integrity constraints in SQL 2003	69
1.3.1 Entity Integrity constraints.....	71
1.3.1.1 Unique constraints.....	71
1.3.1.2 Primary constraints	72
1.3.2 Domain Integrity.....	72
1.3.2.1 Check constraints	72
1.3.3 Referential Integrity	73
1.3.3.1 Referential Actions	74
1.3.3.2 FOREIGN KEY constraints.....	75
1.3.3.3 Threats to Referential Integrity	76
1.3.4 User-defined integrity	76
1.3.4.1 Triggers	76
1.3.4.2 Stored procedures.....	78
1.3.4.3 Assertions.....	78
1.4 Advantages of Integrity constraints	78

Table of Contents

Appendix C: Error messages.	80
1 Summary of unique tests error messages.....	80
2 Summary of referential Integrity tests error messages	80
3 Summary of decimal data type tests error messages	81
4 Summary of small int tests error messages.....	82
5 Summary of float tests error messages	82
6 Summary of real tests error messages.....	82
7 Summary of check constraints tests error messages	83
Appendix D: SQL standards	84
1. Vendor lock in.....	84
2. SQL dialects.....	84
Appendix E: Tutorial and what is on the CD	89
1. Performing the tests in SQL Server 2000	89
1. Performing the tests on Oracle 9i.....	91
Appendix F: References.....	93

List of Tables

Table 1.1 - The milestones of the SQL Standards.....	15
Table 1.2 - SQL2003 statement classes	21
Table 3.1 - Primary key tests results	32
Table 3.2 – Unique tests results.....	34
Table 3.3 – Referential integrity tests results	36
Table 3.4 - SQL 2003 data types considered for tests	37
Table 3.5 – String or character data type tests results.	38
Table 3.6 – Decimal tests results	40
Table 3.7 - Integer and small int tests results	41
Table 3.8 - Float tests results.....	42
Table 3.9 – Real Tests results.....	43
Table 3.10 - Check constraints tests results.....	45
Table 4.1 - Interactions and isolation levels.....	53
Table 4.2 – Oracle and SQL Server isolation levels.....	53
Table B.1 – Oracle’s and SQL Server’s support of SQL2003 declarative integrity	71
Table D.1 - Schema commands	85
Table D.2 - SQL-data commands	86
Table D.3 - SQL-connection, session and transaction statements	87
Table D.4 - SQL 2003 rules for naming Identifiers	88

List of Figures

Figure 1.2 - Conformity to SQL 2003 standard	13
Figure 1.3 - Using SQL for database access.....	15
Figure 1.4 - SQL 2003 dataset hierarchy	19
Figure 1.3 - 30 years of Oracle innovation.....	23
Figure 2.1 - Mimer SQL-2003 Validator	27
Figure 2.2 - Mimer SQL-2003 Validator results	27
Figure 3.1 - Regions and territories relationship	35
Figure 3.2 – Self referencing foreign keys.....	35
Figure 4.1 – Example transaction.....	51
Figure 5.1 - Schema commands summary	56
Figure 5.2 - SQL-data commands summary	57
Figure 5.3 - SQL-connection, session and transaction statements summary	58
Figure 5.4 - SQL 2003 data types.....	59
Figure 5.5 - SQL 2003 rules for naming Identifiers.....	60
Figure 5.6 - SQL 2003 built-in functions support summary	61
Figure 5.7 - SQL 99 Core feature support summary.....	62
Figure 5.8 - SQL 2003 weighted results	63
Figure B.1- Domain Integrity: Datatypes, Not Null Constraints, and Check Constraints	73
Figure B.2 - Referential constraints	74
Figure B.3 – Referential action example.....	74
Figure B.4 - Self-referencing foreign key	75
Figure B.5 – Oracle triggers stored in the database separate from their associated tables.....	77
Figure B.6 - SQL 2003 triggers syntax.....	77
Figure E.1 starting Enterprise Manager.....	90
Figure E.2 Console Root node.....	91

List of Figures

Figure E.3 – Oracle login dialog box 92
Figure E.4- SQL* Plus Worksheet 92

Chapter 1: Background

1.1 Introduction

For a modern business endeavouring to drive competitiveness, data is the most valuable asset it has at its disposal. Making better use of their data will help businesses realise this goal. However, data is just bits and bytes on a file system and only a database management system (DBMS) can turn these bits and bytes into business information. This means choosing the right DBMS becomes one of the critical tasks that a business has to carry out. In most cases the choice of a DBMS has much to do with office politics, that is, what the Database Administrators, managers, and their friends already know or are familiar with, rather than objective facts. As more and more features are being introduced fierce competition continues to be a prominent feature in the DBMS market. This has been evidenced by database 'wars' which have been prevalent in the DBMS market for the past decade. However, this brutal competition has given businesses, relational database management systems (RDBMSs) that perform faster, efficiently, reliably and with a lower total cost of ownership (TCO) than ever before. Oracle has been one of the more dominating companies in the middle-to-large RDBMS market since the past decade. However Microsoft has also been on the rise, stiffening the competition for Oracle. Both DBMSs are amongst the “Big three” DBMSs in the world, ordered as IBM, Oracle and Microsoft respectively.

The Database Management Systems market is characterised by vendors making various claims about the superiority of their products, hence making the task of choosing a DBMS not an easy one for the DBAs, as they are often confronted with confusing masses of buzzwords and vendors' technological claims. This has resulted in an outcry for comparisons between different DBMSs.

1.2 Aim

As business organisations, vendors will always attempt to market their products as effectively as possible, even if this means misleading customers. There are several criteria which DBAs can use to objectively evaluate DBMSs and make informed decisions. These factors include platform support, price, ease of use, performance, security and many others. This project aims to make a comparative investigation and evaluation of Oracle 9i and Microsoft SQL Server 2000 with respect to the maintenance of data integrity and conformity to SQL 2003 standards. Proving which DBMS is the best has always been like a religious debate, thus this project does not aim to prove which DBMS is superior, but to establish which can be best implemented depending on the need. Nevertheless, each

of these DBMSs has its own advantages and disadvantages.

1.3 Motivation

Database wars have been a common feature in the DBMS market for more than a decade. This is mainly because, there are so many DBMSs available with all of them making various claims about the superiority of their products in their attempts to gain bigger market shares, thus making the selection process a difficult one for DBAs. Oracle 9i and MS SQL Server 2000 are amongst the leading DBMSs in the DBMS arena and there is a strong rivalry between them. It therefore becomes imperative to know what features are provided by each DBMS and which scenarios it is most suited for.

There is a wide checklist of features which can be used to evaluate these DBMSs. [Chigrik, A: 2000] made a comparison of these two DBMS with respect to features like performance, platform support and cost. Thus, it is also important to make evaluations with respect to other features such as Integrity and SQL 2003 standards.

[Türker, Gertz: 2000] mentioned that, the accuracy of the data managed by a DBMS is vital to any application using the data for business, research and decision making purposes. This means that DBMSs must be able to guard against erroneous data that do not reflect real world artefacts consumed and operated on by applications. The maintenance of data integrity was the one of the main motives behind the development of DBMSs, which means by failing to maintain integrity; this motive would have been defeated.

The buzzword in the world of computing nowadays is inter-operability and total ownership of costs. How far have these two leading DBMSs gone in developing non-vendor locking software? ANSI introduced SQL standards since 1986 with the hope of providing easier migration to third-party applications without the need to modify your SQL code, hence reducing vendor dependency. However, SQL dialects still continue to proliferate in bids that are meant to lock customers to specific vendors. Furthermore, knowing the current standards is crucial with the advent of open source database projects like MySQL and Postgress which are developed by teams [Kline, K: 2004].

1.4 Project Overview

The project is in two main parts, which are, the evaluation of integrity and the evaluation of conformity to the latest standards SQL: 2003. The two diagrams below are used to highlight these two main parts of the project.

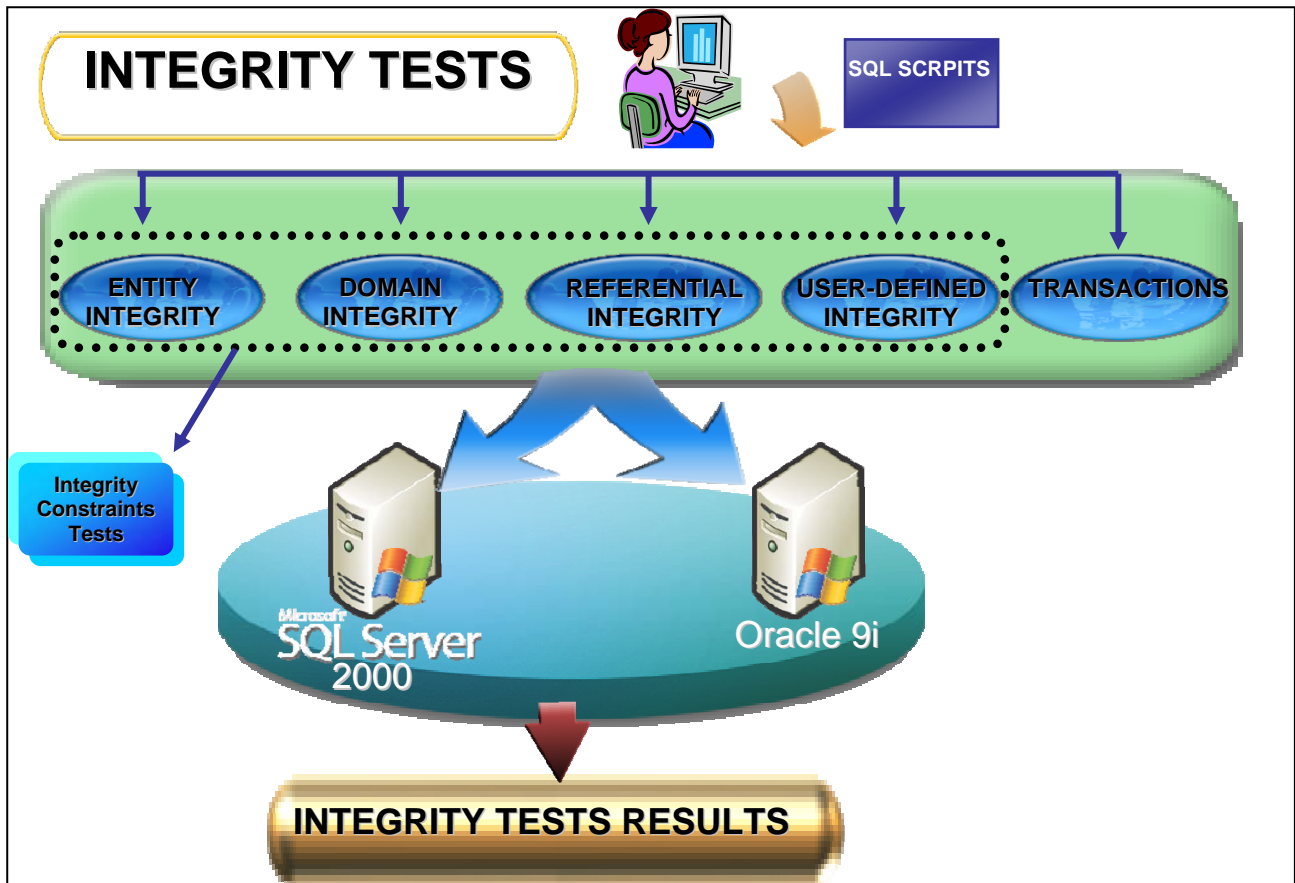


Figure 1.1 - Overview of Integrity experiments

Figure 1.1 shows that integrity tests were carried out using SQL scripts. Different queries were executed and the results were collected, analysed and then summarised.

Figure 1.2 shows the SQL 2003 part. This part involved an investigation of the conformity of Oracle and SQL Server to the SQL2003 standards. The results were also summarised.

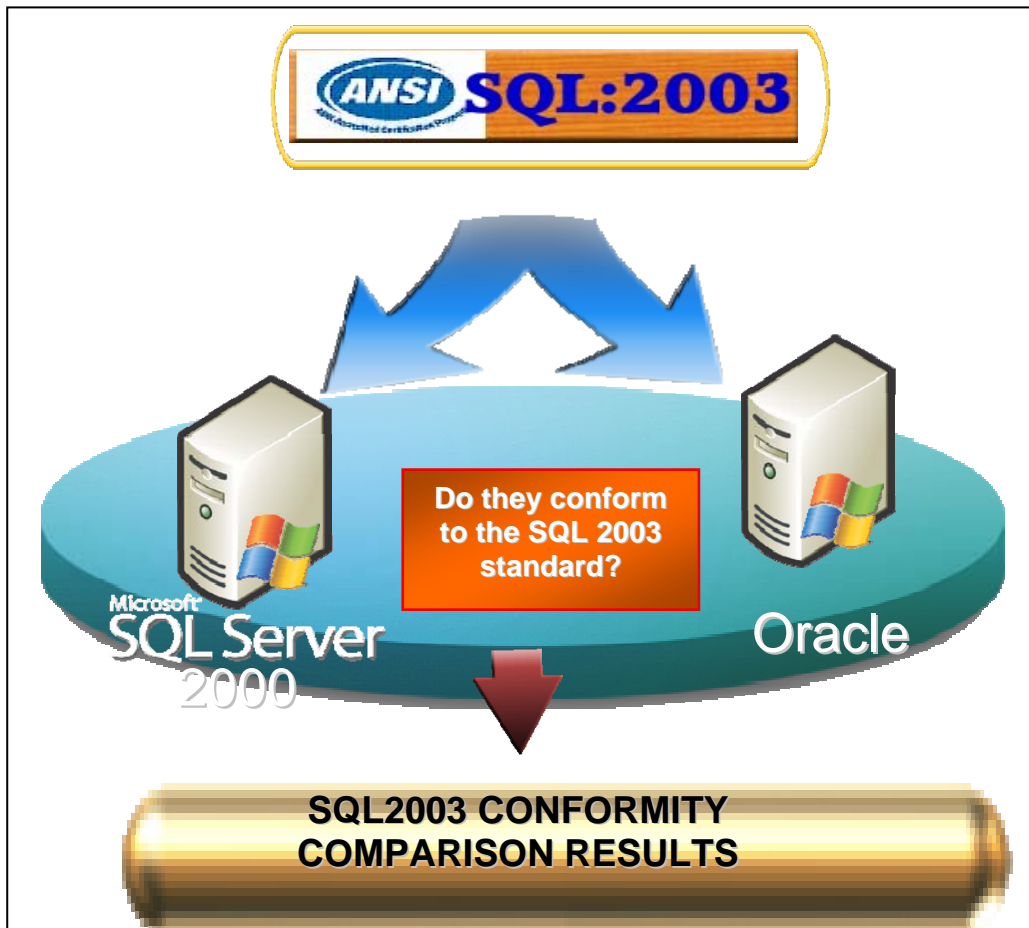


Figure 1.2 - Conformity to SQL 2003 standard

These parts are further elaborated below.

1.4.1 The evaluation of data integrity

The Integrity subsystem is responsible for maintaining the accuracy, correctness and validity of the data stored in a database, detecting and acting on integrity violations. It must exert deliberate control on every process that uses the data to ensure the continued correctness of the information.

The evaluation of data integrity was done by carrying out experiments to test the maintenance of data integrity in each DBMS. This was basically done by comparing simple integrity features such as primary keys and unique up to complex features like triggers, stored procedures, transactions, isolation levels and locking mechanism. Integrity tests were mainly grouped into Integrity constraints and transactions.

❖ Evaluation of Integrity constraints.

Integrity constraints are generally categorized into main categories, namely: Entity Integrity, domain Integrity, referential Integrity and user-defined Integrity which are shown in figure 1.1.

These are basically the four types of semantic integrity which have been implemented by each

DBMS to help maintain data integrity. Appendix B gives the detailed background of the different types of integrity that were considered. Essentially, it provides all the background information for the integrity part.

- ❖ Evaluation of transaction handling and concurrence control – even with sound integrity constraints, inept handling of transactions can still lead to severe data integrity problems. So this part investigated transaction handling capabilities and options, locking mechanisms and isolation levels provided by each DBMS to maintain data integrity.

1.4.2 The evaluation of conformity to SQL 2003 standards

This part was mainly of a research rather than experimental nature. That is, this part was mainly done by a collation of the literature available on the SQL standards. This was carried by making an investigation of the implementation of standard SQL by these DBMSs. This was mainly done through researching on the standards and to what extent are these DBMSs shunning the attitude of implementing their SQL in proprietary manners. Reference was also made to [Gulutzan. P: 2005], [Kline K: 2004] and [Troels A: 2005] who have written articles on DBMSs SQL 2003 support. In addition to the literature, a simple tool called “Mimer SQL Validator” was used to carry out some of the tests that were carried out. An overview of the standards follows below.

1.4.2.1 SQL Standards

According to [Rosenzweig, B, Silvestrova E. 2003], [Kline, K: 2004] and [Beaulieu A, Mishra S. 2002], in the early 1970s, the work of the IBM research fellow Dr. E. F. Codd of the application of the mathematical relational theory in databases, led to the emanation of a relational data model product called SEQUEL, or Structured English Query Language. SEQUEL ultimately became SQL, or Structured Query Language.

SQL is basically a language which allows programmers to manipulate the data stored in a database and get results. This is basically done by issuing commands to the DBMS, which will take your request perform the necessary operations and then return the appropriate information. SQL is a useful and powerful language which enables us to communicate and interact with the DBMS. This scenario it illustrated by the diagram below.

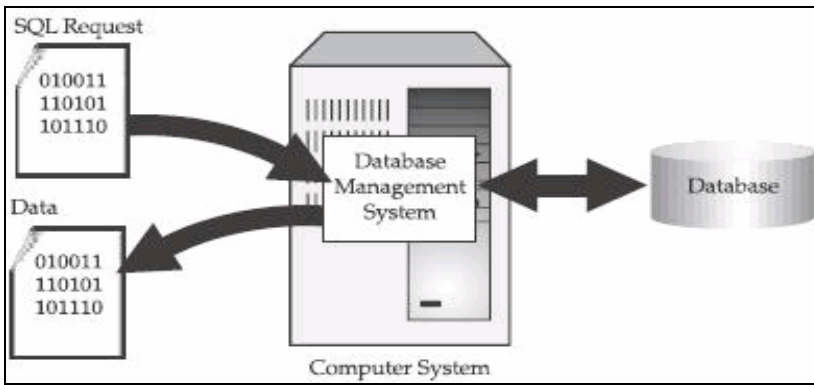


Figure 1.3 - Using SQL for database access¹

Due to the rapid increase in the number of vendors in the market, SQL dialects proliferated and over time, SQL proved popular enough in the marketplace to attract the attention of the American National Standards Institute (ANSI), which released the first standard for SQL in 1986. The table below is used to highlight the timeline of SQL standards since then.

Year	Name	Alias	Comments
1986	SQL-86	SQL-87	First published by ANSI. Ratified by ISO in 1987.
1989	SQL-89		Minor revision.
1992	SQL-92	SQL2	Major revision.
1999	SQL:1999	SQL3	Added regular expression matching, recursive queries, triggers, non-scalar types and some object-oriented features. (The last two are somewhat controversial and not yet widely supported.)
2003	SQL:2003		Introduced XML-related features, <i>window functions</i> , standardized sequences and columns with auto-generated values (including identity-columns).

Table 1.1 - The milestones of the SQL Standards²

ISO and IEC are the world’s standardisation body. Members of ISO or IEC participate in the development of international standard through technical committees which were set up. According to [Gulutzan P. [1]: 2005], there is an international committee working on the SQL standard (ISO/IEC JTC 1/SC 32/WG 3) as well as an American committee (ANSI TC NCITS H2).

After the first standard in 1986, a revised standard commonly known as SQL-89 or SQL1 was

¹ Adapted from [Groff and Weinberg: 2004]

² Adapted from [Wikipedia: 2005]

published in 1989. But due to partially conflicting interests among the commercial vendors, much of the SQL-89 standard was intentionally left incomplete, and many features were labelled implementer-defined. In order to strengthen the standard, the ANSI committee revised its previous work with the SQL-92 standard ratified in 1992 also called SQL2. This standard addressed several weaknesses in SQL-89 and set forth conceptual SQL features, which at the time exceeded the capabilities of any existing RDBMS implementation. In fact, SQL-92 standard was approximately six times the length of its predecessor. In 1999, the ANSI/ISO released the SQL-99 standard also called SQL3. This standard addresses some of the more advanced and previously ignored areas of modern SQL systems such as object-relational database concepts, call level interfaces, and *integrity management*. Recently ANSI/ISO released the SQL-2003 standard also called SQL-200n. The big SQL-2003 features are: more collection data types, cleaner object/relational specification, and references to new parts such as XML. The big missing SQL-2003 feature is the SQL-99 standard BIT data type [Daffodildb: 2004].

The SQL language is wide and deep. The fact that it is widely implemented in almost every DBMS that stores and manipulates data, partially explains the amount of effort that went into the theory and development of the standards.

According to the [Wikipedia: 2005], although the SQL standards are defined by both ISO/IEC and ANSI, there are many disparities in the versions of the language provided by these bodies. Oracle uses PL/SQL whilst Microsoft uses T-SQL. It is very common for these commercial implementations to omit support of basic features such as DATE and TIME, preferring their own variants. This means that even though there are standards, dialects still continue to persist. According to [Kevin E. Kline: 2004], this is mainly because the user community of a given database vendor often require capabilities in the database before the ANSI committee has created a standard and some of the earliest vendors from the 1980s have variances in the most elementary commands, such as SELECT, because their implementations predate the standards. Consequently, unlike ANSI C or ANSI Fortran, which can usually be ported from platform to platform without major structural changes, SQL code can rarely be ported between database systems without major modifications.

The [Wikipedia: 2005] also went on to highlight some of the reasons for this lack of portability as:

1. *The complexity and size of the SQL standard means that most databases do not implement the entire standard.*

- II. *The standard does not specify database behaviour in several important areas (e.g. indexes), leaving it up to implementations of the standard to decide how to behave.*
- III. *The SQL standard precisely specifies the syntax that a conformant database system must implement. However, the standard's specification of the semantics of language constructs is less well-defined, leading to areas of ambiguity.*
- IV. *Many database vendors have large existing customer bases; where the SQL standard conflicts with the prior behaviour of the vendor's database, the vendor may be unwilling to break backward compatibility.*
- V. *Some believe the lack of compatibility between database systems is intentional in order to ensure vendor lock-in.*

1.4.2.2 Levels of conformance

According to [Kline, K: 2004], “SQL92 first introduced levels of conformance by defining three categories: *Entry, Intermediate, and Full*. Vendors had to achieve at least Entry-level conformance to claim ANSI SQL compliance. Each higher level of the standard was a superset of the subordinate level, meaning that each higher level of the standard included all the features of the lower level of conformance”.

The introduction of SQL99 saw the base levels of conformance being altered. With SQL99, vendors had to implement all the features of the lowest level of conformance, (*this lowest level was called the Core SQL99,*) in order to claim (and publish) that they are SQL99 compliant. Core SQL99 included the old Entry SQL92 feature set, features from other SQL92 levels, and some brand new features. Vendors were also free to implement additional feature packages described in the SQL99 standard. The latest standard is SQL 2003 which was based on a similar premise as SQL 99. It also contains a Core part which is currently met by a few vendors. The relatively new part in this standard was SQL/XML; other parts were persisted from older versions with or without a few modifications.

[Kline K.:2004], however also states that, even if a DBMS conforms to the SQL99 standards, its commands may differ from other DBMSs because the SQL statements may be parsed, compiled, and executed differently, especially if differing binding styles are used.

1.4.2.3 The SQL 2003 standards

[Kline K.:2004] gives the SQL 2003 view of a DBMSs. A database language standard specifies the syntax and semantics of various components of a DBMS. In particular, it defines the structures and

operations of a data model implemented by the DBMS as shown below. It is upon this view that all the syntax and SQL constructs are based.

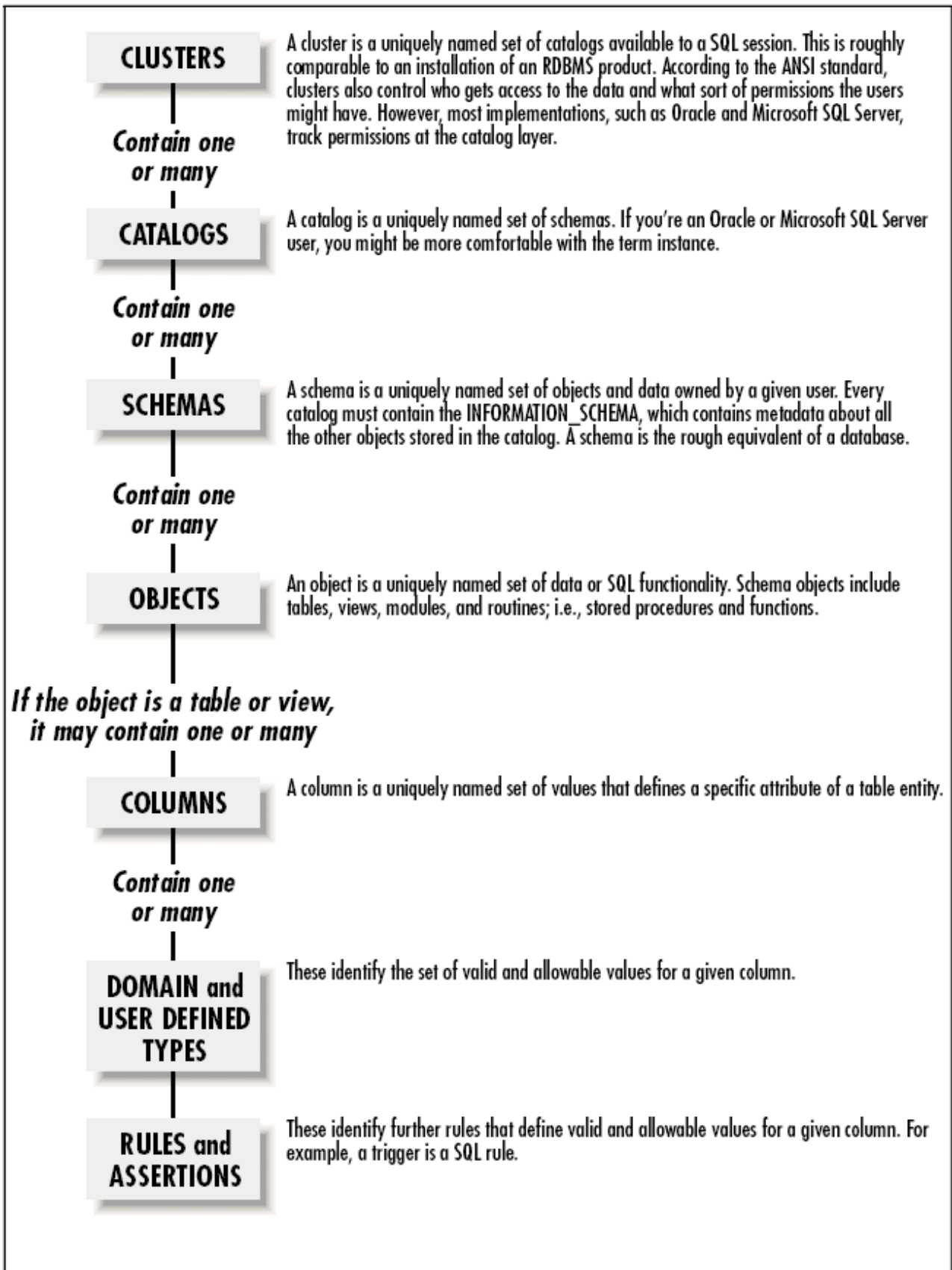


Figure 1.4 - SQL 2003 dataset hierarchy³

The basic structure of a relational model is a table composed of columns and rows, but in SQL 2003 *Clusters* contain sets of *catalogs*; *catalogs* contain sets of *schemas*; *schemas* contain sets of *objects*, such as *tables* and *views*; and *tables* are composed of sets of *columns* and *records*. It is upon this fundamental layout that most DBMSs start departing with the standards. Both Oracle's and SQL Server's architectures are different from this implementation. According to the SQL 2003, schema addressing should be in form *Catalog.schema.object* as shown in figure 1.4, but Oracle use *Schema.object* whilst MS SQL Server 2000 use *Server.database.schema.object* [Kline.K: 2004].

According to [Kline.K: 2004] this standard is made up of the following parts:

Part 1 - SQL/Framework

This part includes common definitions and concepts used throughout the standard. It defines the way in which the standard is structured and how the various parts relate to one another. It also describes the conformance requirements set out by the standards committee.

Part 2 - SQL/Foundation

This part includes the **Core** which is an augmentation of the SQL99 Core, and is the largest and most important part of the standard.

Part 3 - SQL/CLI (Call-Level Interface)

This part defines the call-level interface for dynamically invoking SQL statements from external application programs. SQL/CLI also includes over 60 routine specifications to facilitate the development of truly portable shrink-wrapped software.

Part 4 - SQL/PSM (Persistent Stored Modules)

Standardizes procedural language constructs similar to those found in database platform-specific SQL dialects like PL/SQL and Transact-SQL.

Part 9 - SQL/MED (Management of External Data)

Defines the management of data located outside of the database platform using data links and a wrapper interface.

Part 10 - SQL/OBJ (Object Language Binding)

³ Adapted from [Kline.K: 2004]

This part describes how to embed SQL statements in Java programs. It is closely related to JDBC, but offers a few advantages over JDBC. It is also very different from the traditional host language binding possible in early versions of the standard.

Part 11 - SQL/Schemata

Defines over 85 views (three more than in SQL99) used to describe the metadata of each database and stored in a special schema called INFORMATION_SCHEMA. A number of views that existed in SQL99 have been updated.

Part 12 - SQL/JRT (Java Routines and Types)

This part defines a number of SQL routines and types using the Java programming language. Features of Java, such as Java static methods and Java classes, are now supported.

Part 14 - SQL/XML

This is the new part. It adds a new type, called XML. New operators like XMLPARSE, XMLSERIALIZE, XMLROOT, and XMLCONCAT were introduced. It also includes rules for mapping SQL-related elements (like identifiers, schemas, and objects) to XML-related elements.

Parts 5, 6, 7, and 8 do not exist by design.

SQL 92 was the most popular standard which used to define statement classes as Data manipulation language, Data Definition language and Data Control language. However SQL2003 came with seven core categories, now called *classes*. These provide a general framework for the types of commands available in SQL. The table below identifies these classes and list some of the commands that are found in those classes.

Class	Description	Example commands
SQL connection statements	Start and end a client connection	<i>CONNECT, DISCONNECT</i>
SQL control statements	Control the execution of a set of SQL statements	<i>CALL, RETURN</i>
SQL data statements	May have a persistent and enduring effect upon data	<i>SELECT, INSERT, UPDATE, DELETE</i>
SQL diagnostic statements	Provide diagnostic information and raise exceptions and errors	<i>GET DIAGNOSTICS</i>

SQL schema statements	May have a persistent and enduring effect on a database schema and objects within that schema	<i>ALTER, CREATE, DROP</i>
SQL session statements	Control default behaviour and other parameters for a session	<i>SET statements like SET CONSTRAINT</i>
SQL transaction statements	Set the starting and ending point of a transaction	<i>COMMIT, ROLLBACK</i>

Table 1.2 - SQL2003 statement classes⁴

1.5 Database Management system selection criteria

As business applications become more and more complex, the DBMSs chosen to manage an organisation's data becomes critical to the organisation's overall success, which is to provide greater informational capabilities. Choosing the right DBMS involves more than making a tactical decision to solve an organisational immediate need, meaning the DBMS selection process can be so complex that it requires skill, knowledge and consideration. The wrong DBMS choice can lock the organisation into a technology that does not serve its interests well and will be expensive to change. Making the right decision in the first place can considerably alleviate the burden of maintaining and expanding the organisational information infrastructure. But, the question is, how can this be achieved? How does a user select the best DBMSs for his needs?

The answer is not an easy one. Given the fact that DBMSs are very complex pieces of software which are difficult to comprehend in their entirety, it becomes vital to build a checklist of criteria which can be used by DBAs in their selection processes. This basically helps to dissect these DBMSs into manageable criteria. Many database practitioners and authors including [Coronel C & Rob P. 2002] have written articles on possible DBMS selection criteria. Generally, the selection criteria revolve around the analysis of organisational needs and requisite DBMS features. Although the factors determining the selection process can vary from organisation to organisation some of the common factors are:

- ❖ Application requirements: These are basically the constraints that are put on the database by the application. For example, an application which will be used to handle multi-user request will definitely need to run on a database which supports transactions.
- ❖ DBMS features and tools. Gone are the days when DBAs had to hard code everything. Most DBMSs have developed sets of tools which automate most common laborious tasks. Basically these tools are there to facilitate the application development task. For example, the availability of report generators, query by example and so on.

⁴ Adapted from [Kline K: 2004]

- ❖ The underlying DBMS model. Is it Hierarchical, network, relational, object-oriented?
- ❖ Portability. How portable is it across platforms, systems, and languages
- ❖ DBMS hardware requirements. This includes things such as minimum processor speed, RAM capacity, disk space and so on.
- ❖ Cost, this generally refers to all the monetary costs of choosing a specific DBMS. These costs include: purchasing costs, maintenance costs, operational costs, license costs, installation costs, training costs and conversion costs.
- ❖ Maintaining data consistency: Without this feature, the motive of using a DBMS will be defeated. This can be regarded as the protection of the data in database from invalid alteration or destruction. This is basically enhanced by security, which has to do with the protection of data in the database against unauthorised disclosure, alteration or destruction. Data consistency is also ensured by making sure that backups are performed regularly to enable recovery.
- ❖ Response-Time requirements: The response time could be critical in certain cases, especially in web-oriented database whereas it is less important under different circumstances.

This project is mainly focused on the maintenance of data consistency and portability criteria.

1.6 Overview of Oracle

The Oracle RDBMS is more than two decades old. In 2006 it will be celebrating its 29th anniversary. It is the world's leading supplier of software for information management and the world's second largest independent software company. This is evidenced by the fact that, 9 of the top 10 automotive manufacturers use oracle, all 10 of the world's largest Web sites- from Amazon.com to Yahoo! - use Oracle and 65% of the Fortune 100 use Oracle for e-business .

[Oracle [1]: 2005].

“Today, the Oracle DBMS is supported on over 80 different operating environments, ranging from IBM mainframes, DEC VAX minicomputers, UNIX-based minicomputers, Windows NT and several proprietary hardware-operating system platforms, and is clearly the world's largest RDBMS vendor.” [Oracle [2]: 2005].

According [MCAD, 2005], the Oracle database grew by 14.5 percent on a yearly basis and increased its market share lead to 41.3 percent, whilst the worldwide market for relational database management systems grew by 11.6 percent in 2004. In 2003, Oracle Database posted 8.6 percent growth year over year and was the market share leader with 39.8 percent. However IBM and Microsoft followed Oracle with 30.6% and 13.4%, respectively.

Oracle is a powerful DBMS which basically started from humble beginnings as one of the DBMSs which were around in the early 70s to one of the leading DBMS in the world. According to [Mullins C.S [1]: 2005], “Oracle is assembling a juggernaut of packaged application software by its acquisition binge”. This is evidenced by its recent acquisitions of Siebel systems, PeopleSoft, Innobase and many other small companies this year. Some of Oracle’s bright moments and achievements are highlighted in the timeline shown below.

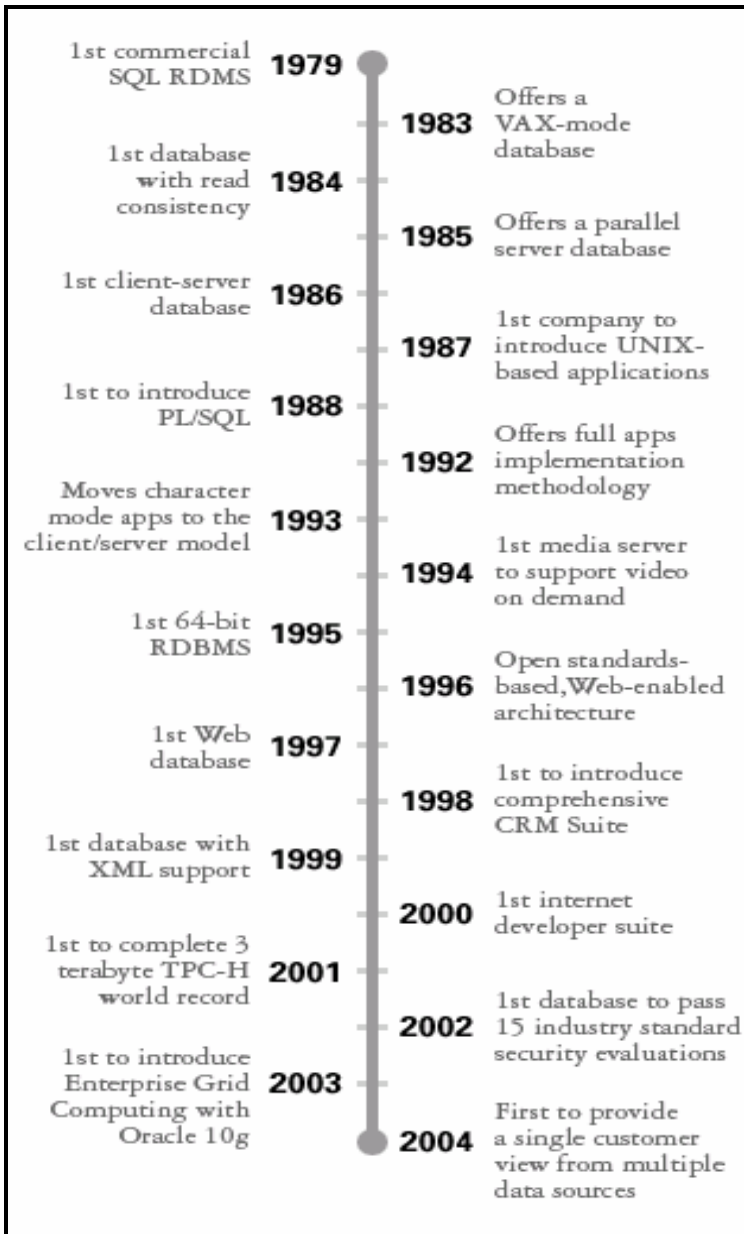


Figure 1.3 - 30 years of Oracle innovation⁵

⁵ <http://www.oracle.com/corporate/history.html>

1.7 Overview of SQL Server

According to the [Wikipedia [1]: 2005] in 1989, Microsoft, Sybase and Ashton-Tate teamed up to create and market the first version of SQL Server named SQL Server 4.2 for OS/2. The timeline of SQL Server releases can be highlighted as below.

- 1992 SQL Server 4.2
- 1993 SQL Server 4.21
- 1995 SQL Server 6.0, codenamed SQL95
- 1996 SQL Server 6.5, codenamed Hydra
- 1999 SQL Server 7.0, codenamed Sphinx and SQL Server 7.0 OLAP, codenamed Plato
- 2000 SQL Server 2000 32-bit, codenamed Shiloh
- 2003 SQL Server 2000 64-bit, codenamed Liberty
- 2005 SQL Server 2005, codenamed Yukon (November 7)

The current version, Microsoft SQL Server 2000, was released in August, 2000. Microsoft is beta testing its successor, SQL Server 2005.

“The success of SQL Server 2000 paved the way for the next two important milestones: leadership in the Windows database market and \$1 billion in annual sales. The first milestone was achieved before the end of 2000, according to Gartner Dataquest, which reported on May 23 that SQL Server accounted for 38 percent of new database license sales on the Windows Server platform that year, compared with 37 percent for Oracle.”[Microsoft [1]: 2001]

After years of massive investment in academia, Microsoft is beginning to realise its dividends. This is evidenced by the great strides it has made to be amongst the “Big three” DBMSs of the world. According to [MCAD, 2005], Microsoft showed the strongest growth of the top three vendors with a surge of 22% in 2004. SQL Server has also made great strides in such aspects as ease-of-use, manageability and support, where many say it is the leader.

1.8 Summary of Chapter

This chapter basically gave an overview of the problem being dealt with. It highlighted the possible DBMS selection criteria and why DBAs should bother about this process. An overview of the timelines for both Oracle and SQL Server was also given. This helps us know how these DBMSs have evolved.

Chapter 2: Design Considerations

As properly designed and executed experiments generate more precise results while using substantially fewer experiments, it was crucial to formulate a plan of action on how the experiments were going to be carried out. This chapter discusses the considerations and the steps taken in designing the integrity experiments. It also discusses the considerations made for the investigation for the conformity to SQL 2003 standards.

2.1 Considerations for integrity tests

There are important factors which influence experiments which need to be considered. These factors generally revolved around the general experiment design principles like, controlling other variables and replication. The issues considered in this project are discussed below as partly adopted from [Stakemire T.S, 2000].

2.1.1 External factors

In planning experiments it was necessary to limit any bias that might have been introduced by the experimental units or conditions. To improve the accuracy of the experiments, it was essential to keep external variables as constant as possible. Both Oracle and SQL Server are commercial RDBMSs which use the same formal language (SQL) for data definition and manipulation. So as not to introduce unnecessary external influence on the results, a pre-defined database was used. The SQL Server's Northwind database was mainly used. It was migrated to Oracle using 'AdventNet SwisSQL SQL Server to Oracle Edition Release 2.6'. In cases where the desired functionalities were not found in Northwind, custom tables and relationships were created. This basically ensured the standardisation of the experiments by using the same data and the same database design. Also the same operating systems were used.

2.1.2 Operating System

The choice of operating system was Windows Server 2003, standard edition. This choice was made mainly because SQL Server only supports the windows platform.

2.1.3 Software

There are so many DBMSs in the market to choose from. The main reasons for choosing Oracle 9i and SQL 2000 were:

- ❖ Both DBMSs are extensively used in the industry, many companies run Oracle and SQL

Server as their backend systems.

- ❖ Microsoft was still beta testing SQL Server 2005, so Oracle 10g would not be compared.

2.1.4 Sufficient tests

To be sure about the results obtained, a substantial number of tests were carried out for each experiment. Where appropriate, experiments were run several times. This was mainly to ensure that the same results were obtained for the same experiment. This was generally an application of the experimental design principle of replication, where the same test is carried out a number of times to have a higher degree of certainty in your results.

2.1.5 Accurate tests

It was of utmost importance that the experiment tested the correct operation. This was basically achieved by first clearly stating the problem that had to be addressed by the experiment, then control other unrelated variables such that they would not influence the results. For example, to use the ID column to perform data type tests, the primary key attribute had to be turned off as this was not of interest at that moment. This was augmented by the analysis of error messages to make sure that they were for the operation that was being performed. That is, error messages were supposed to be of the task being tested not other erroneous transactions.

2.2 Considerations for SQL 2003 standards.

The standards are too voluminous to be exhaustively examined against each of the documentations of these two DBMSs to check their conformity. Certain aspects of the standard were evaluated. In each case both DBMSs were evaluated against the same section at a time. For example. on data type tests, the same data types were chosen for each DBMS. The SQL-2003 standard specification's most important part is the Core SQL-2003. It is a subset of SQL-2003, which provides the minimal conformance level for SQL-2003. Core SQL-2003 includes: all of Entry SQL-92, much of Transitional and Intermediate SQL-92, some of Full SQL-92 and SQL-99 features, as well as new features from SQL-2003. The features that were investigated in this project were chosen from the Core listed at [Mimer Developer page: 2005].

Mimer SQL-2003 Validator was used in conjunction with documentations of both DBMSs and various other sources to test and evaluate the conformity of the various statements to the SQL 2003 standards. The procedure undertaken can be highlighted as shown below:

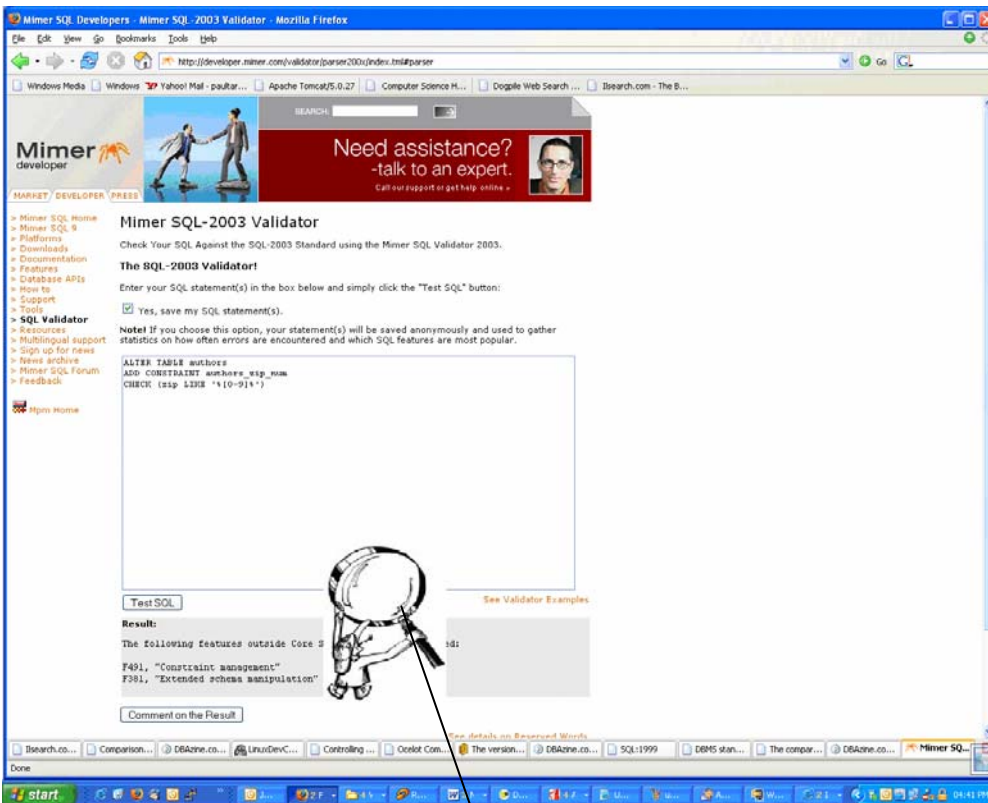


Figure 2.1 - Mimer SQL-2003 Validator

```
ALTER TABLE authors
ADD CONSTRAINT authors_zip_num
CHECK (zip LIKE '%[0-9]*')
```

[See Validator Examples](#)

Result:

The following features outside Core SQL-200x (draft) are used:

F491, "Constraint management"
F381, "Extended schema manipulation"

Figure 2.2 - Mimer SQL-2003 Validator results

Using Mimer SQL-2003 Validator, an SQL implementation of a particular command was typed into

the text area as shown above. This was then submitted to Mimer using the *Test SQL* button. Results were returned showing the support status of that statement. The picture above shows that the statement tested made use of non core features *F49I* and *F38I*, but still they were standard features. All those statements which were non standard returned error messages. Suggestions were given for the correct syntax.

2.3 Design of Integrity experiments

This process was involved in the planning and designing experiments. Integrity was broken down into two main parts which are mainly integrity constraints and transactions. These experiments were carried out in an iterative manner and the activities involved are highlighted below.

2.3.1 Choosing a dataset.

This was rather a prerequisite for the design and implementation of tests. This was basically concerned with choosing the dataset upon which the tests were to be carried out. Northwind was selected. It was only in those cases where a given functionality was not found in Northwind when custom tables had to be created. After the database had been chosen, the next step was to decide on the relations which would be used for a particular experiment. This decision was largely influenced by the kind and nature of the experiments to be carried out.

2.3.2 Hypothesis and Experiments design

This was concerned with the identification of the necessary operations that can be carried out on a particular constraint or feature, for example, INSERT UPDATE and DELETE. Control factors to be included and varied in each experiment were identified. This was particularly necessary to ensure that the correct operations were carried out for the correct tests. For instance, you had to ensure that you will not get a primary key violation error whilst you are carrying out data type tests. After deciding on the particular operations which had to be carried out, a general execution plan was drafted. This was followed in the implementation stage.

This part experimented with and evaluated the use of the four main types of semantic integrity mentioned above in each DBMS. This was essentially achieved using SQL scripts to carry out some form of *black box testing*. This means that small unit tests were developed and executed for each integrity constraint to check if there was any violation of integrity after implementing the constraints as specified by the DBMS's syntax.

2.3.3 Implementation of tests

After the execution plan was drawn, the tests were then implemented. This was basically achieved by writing custom SQL scripts to perform the desired operation as many times as was necessary to ensure that accurate results were obtained. This was mainly dependent on the table and fields being used. SQL scripts were designed for each of the DBMSs. However, the first DBMS to be tested was chosen randomly, the script was then adopted to work for the other DBMS. The scripts were loaded using the respective enterprise managers of each DBMS.

2.3.4 Collection of results

The results were collected and recorded. This was mainly done by observing the results of the operation which would have been carried out. Some of the results were collected from the enterprise manager and some through the verification of how the particular operation would have affected the database. For example, it was investigated whether a DBMS would not raise an error message but carries on to make the change in the database, for example, insert a duplicate in a primary key field.

2.3.5 Analysis of results

Before executing a unit-test, the expected outcome was already known, for example, we know that trying to insert a duplicate record should produce a violation of primary key constraint error. Failure to achieve this result mean that the test had failed otherwise it would have passed. So results were first analysed by checking if they were of that particular operation being tested or they were of some other erroneous operations that were out of the scope of the current experiment. After that, it was then verified if the expected outcome had been reached.

2.3.6 Drawing conclusions

The verification of whether or not outcomes conformed to expectations, led to the final stage of the test which was to conclude whether the test had passed or failed. An example would be a test to check whether the character 'a' is allowed in an Oracle *INT* field. If the character is allowed to go in, then the test would have failed since this is a violation of domain integrity.

2.4 Summary of chapter

This chapter gave an overview of how the experiments were designed, planned and implemented. It gave an outline of the basic steps that were followed in the design process. It also carried on explaining the criteria that was used to decide whether a test has either passed or failed.

Chapter 3: Integrity constraints experiments

Integrity is generally defined as a fast adherence to a given set of rules. This means that as long as these rules are maintained, integrity is always upheld. The general background on Integrity and the different classifications that were considered is given in appendix B. The SQL scripts which were used to carry out these tests can be found on the accompanying CD.

3.1 Entity integrity tests

This was mainly concerned with testing the maintenance of entity integrity. This was done by investigating the two classes of entity integrity which are PRIMARY KEYs and UNIQUE KEY. The entity integrity rule stipulates that every instance of an entity is uniquely identified or the value of the PRIMARY KEY must exist, be UNIQUE, and **cannot be null**. So tests passed as long as this rule was upheld.

3.1.1 PRIMARY KEY tests

To test primary keys the following commands were used:

INSERT

- ❖ Inserting a normal record. This was used for baseline purposes to show the expected behaviour when everything have been done in the proper way.
- ❖ Inserting a duplicate record.
- ❖ Inserting a record with the primary key value omitted from the query string.
- ❖ Inserting a record with a null value supplied for the primary key

UPDATE

- ❖ Update to another normal record, this was also a baseline test
- ❖ Update to a record to a duplicate record.
- ❖ Update the primary key of a record to null.

ALTER

- ❖ Drop the primary key.
- ❖ Add a new primary key whilst there is some normal (non-duplicated, non null) data in the primary of the table.
- ❖ Add a primary key whilst some of the data are duplicates/ nulls or just violate the primary key constraint.
- ❖ Rename the *PRIMARY KEY column*.

Tests were performed for both single and multi-column primary keys. The primary key constraint was satisfied if and only if it was UNIQUE and did not allow null values in the specified column(s).

The error messages, actions and the status of tests can be summarised in the table shown below.

SK: represents tests performed using single keys.

CK: represents tests performed using composite keys

Status



Means tests passed, that is, it performed as expected



Means tests failed that is, it did not perform as expected

The error messages that were generated for these tests can be summarised as follows:

I. **Server: Msg 2627, Level 14, State 1, Line 1**

Violation of PRIMARY KEY constraint 'PK_Region'. Cannot insert duplicate key in object 'Region'. The statement has been terminated

II. ORA-00001: unique constraint (PAUL.PK_REGION) violated

III. ORA-01400: cannot insert NULL into ("PAUL"."REGION"."REGIONID")

IV. **Server: Msg 515, Level 16, State 2, Line 1**

Cannot insert the value NULL into column 'RegionID', table 'Paulos.dbo.Region'; column does not allow nulls. INSERT fails. The statement has been terminated.

V. **Server: Msg 1505, Level 16, State 1, Line 1**

CREATE UNIQUE INDEX terminated because a duplicate key was found for index ID 1. Most significant primary key is '2'.

Server: Msg 1750, Level 16, State 1, Line 1

Could not create constraint. See previous errors.

The statement has been terminated.

VI. ORA-01407: cannot update ("PAUL"."REGION"."REGIONID") to NULL

VII. ORA-02437: cannot validate (PAUL.PK_REGION) - primary key violated

Example Test performed

SK/ CK: Insert Normal – Using single key (SK) or composite keys (CK), insert a normal record into the database. No errors were raised from both DBMSs and the row was successfully inserted.

SK/ CK are used to mean that the results obtained using either SKs or CKs were generally the same, so they are summarized into one row.

Table 3.1 provides a summary of the tests outcomes.

Tests performed	Oracle error messages	Oracle Action and status	SQL Server Error messages	SQL Server Action and status
SK/ CK: Insert Normal	None	Inserted ✓	None	Inserted ✓
SK/ CK: Insert Duplicate	II	Not Inserted ✓	I	Not Inserted ✓
SK/ CK: Insert with PK Omitted	III	Not Inserted ✓	IV	Not Inserted ✓
SK/ CK: Insert Null	III	Not Inserted ✓	IV	Not Inserted ✓
SK/ CK: Update to Normal	None	Updated ✓	None	Updated ✓
SK/ CK: Update to Duplicate	II	Not Updated ✓	I	Not Updated ✓
SK/ CK: Update to NULL	VI	Not Updated ✓	IV	Not Updated ✓
Drop Primary Key	None	Dropped ✓	None	Dropped ✓
Add PK when no duplicates	None	Added ✓	None	Added ✓
Add PK when there are duplicates	VII	Not Added ✓	V	Not Added ✓

Table 3.1 - Primary key tests results

3.1.1.1 Analysis of error messages (Primary Keys tests)

As can be seen from the table above, there were ticks throughout, meaning that all the DBMS maintained data integrity with respect to primary keys. This means that these DBMSs performed as expected in all tests. Appropriate error messages were raised and the corresponding actions were also executed. Both DBMSs enforced the primary key constraint correctly and ensured that at any point in time no two records or rows could be the same if a primary key is set for a table.

3.1.2 UNIQUE KEY tests

The following commands were used to carry out the *unique key* tests. Given the fact that unique keys can be set to allow or not allow nulls, it was imperative to set this property to a fixed state. In this case the *unique* was set to not null. Another variable that had to be considered was the fact that both DBMSs can allow you to use `IGNORE_DUP_KEY`, which will in-fact allow duplicates to be entered, so for the purposes of these tests, this option was not used. So this means that unique keys were satisfied if and only if no two rows in a table could have the same non-null values in their *UNIQUE* columns.

The operations which were carried out can be highlighted as follows:

INSERT

- ❖ Inserting a normal, record (baseline).
- ❖ Inserting a duplicate.
- ❖ Omitting the Unique key value

UPDATE

- ❖ Update to another normal record
- ❖ Update to a duplicate record

ALTER

- ❖ Drop the Unique key
- ❖ Add a new Unique key whilst the data is already there but with no duplicates.
- ❖ Add a primary key whilst some of the data are duplicates.

The **summary of the various error messages** generated for these tests can be found in appendix C: *summary of unique tests error messages.*

Example test performed

Insert with unique Key Omitted – means run a query to insert a record into the database but with unique key field omitted from the query string. Oracle and SQL Server gave error messages III and IV respectively. And they both did not insert that particular record.

Table 3.2 provides a summary of the tests outcomes.

Tests performed	Oracle error messages	Oracle Action and status	SQL Server Error messages	SQL Server Action and status
Insert Normal	None	Inserted ✔	None	Inserted ✔
Insert Duplicate	I	Not Inserted ✔	II	Not Inserted ✔
Insert with unique Key Omitted	III	Not Inserted ✔	IV	Not Inserted ✔
Update to Normal	None	Updated ✔	None	Updated ✔
Update to Duplicate	I	Not Updated ✔	II	Not Updated ✔
Drop unique Key	None	Dropped ✔	None	Dropped ✔
Add unique Key when there are no duplicates	None	Added ✔	None	Added ✔
Add Unique Key when there are duplicates	V	Not Added ✔	VI	Not Added ✔

Table 3.2 – Unique tests results

3.1.2.1 Analysis of error messages

The major objective of these tests was to investigate if any of these DBMSs would allow duplicates in its unique key field. As can be seen in the results obtained in the table above, in all cases both DBMSs ensured that integrity was maintained as specified or as expected. No duplicates were allowed in the unique key, hence all the tests passed.

3.1.3 Identity property

This is a new feature in the SQL 2003 standard, although it has been around for a long time in small databases like Access. But however it is not supported in Oracle, so it was not investigated.

3.1.4 Overall analysis of Entity integrity tests

As evidenced by the results, there were no problems with this type of integrity. It was upheld in all cases. As a result we conclude that both Oracle 9i and SQL Server 2000 implement and maintain entity integrity.

3.2 Referential Integrity Tests

As mentioned in appendix B, referential integrity is all about maintaining and synchronising the relationships between tables. Tests were carried out both for self-referencing foreign keys and multi-table foreign keys.

The tests were carried out by performing the following operations:

On the referencing table/column (child)

- ❖ Insert a value that does not exist in the referenced table/ column. This was mainly aimed at verifying whether or not an orphan record can be allowed.
- ❖ Update a value to a value that does not exist in the referenced table. Again this was aimed at trying to create an orphan.

On the referenced table/column (parent)

- ❖ Delete/ Update where rule was NO ACTION (*rules defined in the appendix*)
- ❖ Delete/ Update where rule was CASCADE
- ❖ Delete/ Update where rule was SET DEFAULT – aimed at setting all referenced columns to their default values.

- ❖ Delete/ Update where rule was SET NULL – here child records will be retained but the referencing ID should be set to NULL.
- ❖ Drop referenced column

Figure 3.1 gives an example of multi-table foreign key. A region is uniquely identified by a region ID, in a specific region there are territories. Each territory is in turn identified by a territory ID.

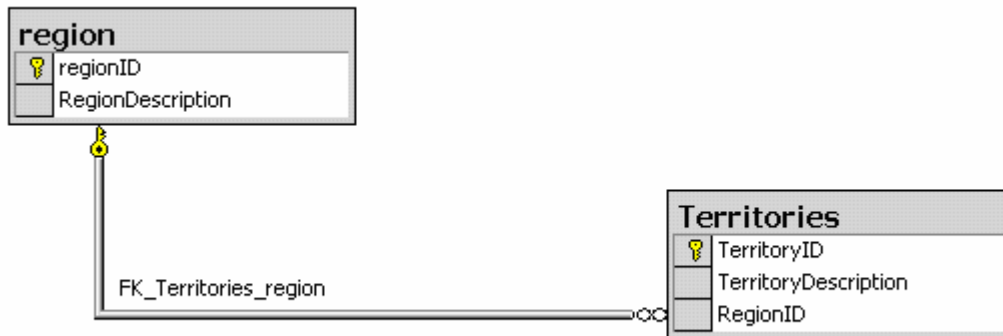


Figure 3.1 - Regions and territories relationship

Normally foreign keys reference primary keys in other tables, but at times they reference primary keys in the same table. This type of referencing is called self referencing. For this type of foreign keys, the students-student rep relationship was used. A class is made up of students and amongst those students there is a class rep who is also a student. Figure 3.2 is used to illustrate this.

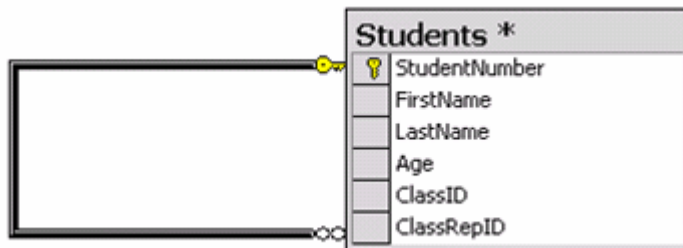


Figure 3.2 – Self referencing foreign keys

The results of the tests are summarised in table 3.3 shown below. The symbols used in this table are first defined.

Child - this refers to the referencing Table or Column

Parent - this refers to the referenced Table or Column

T – Means table

C – Means column

Referential action or rule set.

- NA – rule set to No Action
- CS – rule set to CASCADE
- SD – rule set to SET DEFAULT
- SN – rule set to SET NULL
- RS – rule set to RESTRICT

3.2.1 Summary of error messages

See appendix C: summary of referential integrity tests error messages

Example test performed

Insert orphan Child T/C rule not specified – means run a query to insert an orphan record into the child table (T) or column (C). Oracle and SQL Server generated error messages V and I respectively, whilst they did not insert the record.

Tests performed	Oracle error messages	Oracle Action and status	SQL Server Error messages	SQL Server Action and status
Insert orphan Child T/C rule not specified	V	Not Inserted ✓	I	Not Inserted ✓
Update to orphan Child T/C rule not specified	V	Not Inserted ✓	III	Not Inserted ✓
Delete/ Update Parent T/C where rule = NA	Not supported	Not supported	I	Not Updated/ Deleted ✓
Delete Parent T/C where rule = CS	none	Deleted ✓	none	Deleted ✓
Update Parent C where rule = CS	Not supported	Not supported	IV	Not Updated/ Deleted ✓
Delete/ Update Parent T/C where rule = RS	V	Not Updated/ Deleted ✓	Not supported	Not supported
Add Unique Key when there are duplicates	V	Not Added ✓	VI	Not Added ✓
Delete/ Update Parent T/C where rule = SD	Not supported	Not supported	Not supported	Not supported
Delete/ Update Parent T/C where rule = SN	none	Updated/ Deleted ✓	Not supported	Not supported
Drop referenced column	VI	Not dropped ✓	II	Not dropped ✓

Table 3.3 – Referential integrity tests results

3.2.2 Analysis of error messages

From the results it can be seen that both Oracle and SQL Server maintained referential integrity, there were slight differences in implementation here and there but the effect was generally the same. Oracle supported RESTRICT as its default, although it did not allow its explicit definition. SQL Server supported NO ACTION by default. These two actions are almost the same. The only difference is when the referential constraint is enforced. RESTRICT enforces the delete rule immediately; NO ACTION enforces the delete rule at the end of the statement (deferred).

The other difference is that Oracle did not support ON UPDATE CASCADE option natively; rather complex custom PL/SQL code has to be written to implement this functionality. The other difference was in implementing the delete cascade on self-referencing tables. Oracle deleted with no error, but SQL Server couldn't delete because it was citing cycles or multiple cascade paths (IV).

3.2.3 Overall analysis of referential integrity tests

As evidenced by the results, there were no problems with this type of integrity. It was upheld in all cases. Therefore we conclude that both Oracle 9i and SQL Server 2000 implements and maintains referential integrity. It is guaranteed that as long as you create the relationships correctly and enforce the correct referential actions, your relations will always be synchronised.

3.3 Domain Integrity tests

These DBMSs have lots of data types, and these include both standard and proprietary data types. For the comparison's sake, it was imperative to consider only the standard data types that were supported by both DBMSs. A selection of the data-types given in *table 3.4* was made.

SQL2003 data type	Oracle	MS SQL server
CHARACTER, CHAR	CHAR	CHAR
DECIMAL,DEC	DECIMAL	DECIMAL
FLOAT	FLOAT	FLOAT
INTEGER, INT	INTEGER	INT
REAL	REAL	REAL
SMALLINT	SMALLINT	SMALLINT
VARCHAR, CHAR VARYING, CHARACTER VARYING	VARCHAR2 ...	VARCHAR

Table 3.4 - SQL 2003 data types considered for tests

Experiments were aimed at investigating whether or not a given data type will restrict values to fall within the specified ranges. These experiments were mainly carried out using the commands:

INSERT, UPDATE, and ALTER.

Firstly, baseline tests were created to determine what the normal expected outcomes were. After that a series of experiments which violated a given domain integrity constraint were designed and implemented.

3.3.1 String or Character Tests (char, varchar and nchar)

Tests were carried out for data types char, varchar, and nchar. The results were generally the same, thus, these data types' results can be summarized as shown below

3.3.1.1 Summary of error messages.

- I. **Server: Msg 8152, Level 16, State 9, Line 1**
String or binary data would be truncated.
The statement has been terminated.
- II. ORA-01401: inserted value too large for column
- III. ORA-01439: column to be modified must be empty to change datatype

Example test performed

Insert more than specified chars – means run a query to insert a record having more than specified characters in the field being tested. Oracle and SQL Server raised error messages II and I respectively while not inserting the record.

Tests performed	Oracle error messages	Oracle Action and status	SQL Server Error messages	SQL Server Action and status
Insert normal values	none	Inserted ✓	none	Inserted ✓
Update to normal value	none	Updated ✓	none	Updated ✓
Insert more chars than specified	II	Not Inserted ✓	I	Not Inserted ✓
Update to more chars than specified	II	Not Updated ✓	I	Not Updated ✓
Alter to int	III	Not Altered ✓	None – as long as there were ints	Altered ✓

Table 3.5 – String or character data type tests results.

3.3.1.2 Analysis of results

As stressed in the design chapter, in these experiments, it was necessary to fix all other factors which were of no particular importance at that time. In these experiments features like, primary keys, unique, null were turn off so as to ensure that only the actions being tested would determine

the outcome of the experiments.

As can be noted from the results, both DBMSs performed as expected, no problem was discovered with their maintenance of data integrity using string data types, so these tests passed. However there were differences on their implementation of altering from the char type to numeric types. Oracle did not allow altering as long as there was data in the column. SQL Server permitted this operation. As all these were deliberate moves, we can say they all performed as expected. But each approach has its own advantages over the other. For example, one may say by allowing the change, SQL Server provides greater flexibility in case, a mistake was made in the initial design, unlike Oracle where you have to throw away all the data in that column to be able to make the change.

3.3.2 Numeric data type tests

Since there were many of them, several were tested. According to the SQL 2003 standard, there are two categories of numeric data types. These are namely: Exact numeric and approximate numeric. For the exact numeric DECIMAL, INTEGER and SMALLINT were investigated and for the approximate, FLOAT and REAL were investigated.

3.3.2.1 Exact numeric data types

These include the integer types and those types with a specified precision and scale. Every number has a precision, which is the number of digits. Moreover exact numeric types also have scales which are the digits that come after the radix point.

3.3.2.1.1 Decimal tests

Decimal supports numbers which are up to 38 digits in length.



- Means that the DBMS performed as expected, that is, it carried out the correct action and data integrity was not lost. But the error messages that were generated showed a technical implementation difference from the standards.

3.3.2.1.1.1 Summary of error messages

See appendix C: summary of decimal data type tests error messages

Example test performed

Insert an empty string – means run a query to insert a record having an empty string as the value for the decimal field being tested. Oracle and SQL Server generated error messages V and I respectively, whilst they did not insert the record.

The tests results and error messages are summarised in table 3.6 shown below.

Tests performed	Oracle error messages	Oracle Action and status	SQL Server Error messages	SQL Server Action and status
Insert normal values	None	Inserted	none	Inserted
Insert a string	III	Not Inserted	I	Not Inserted
Insert an empty string	IV	Not Inserted	I	Not Inserted
Insert more digits than specified	V	Not Inserted	II	Not Inserted
Insert a number with decimal places when precision is 0	None	Inserted rounded	none	Inserted rounded
Update to string	III	Not updated	I	Not updated
Update to an empty string	IV	Not Updated	I	Not Updated
Update to more digits than specified	V	Not Updated	II	Not updated
Update to a number with decimal places when the precision is set to zero	None	updated rounded	none	updated rounded

Table 3.6 – Decimal tests results

3.3.2.1.1.2 Analysis of results

As can be seen from the results, most of the tests passed. Both DBMSs did not allow the entry of strings in the Decimal data type field. SQL Server gave a very clear and concise error message, that there was an error converting to varchar but Oracle just issued an “invalid number” error message. There was a slight problem in the empty string tests with Oracle. It did not allow the entry of empty strings which was fine but its error message was:

ORA-01400: cannot insert NULL into ("PAUL"."TERRITORIES"."TERRITORYID")

This raised questions as we were not dealing with any NULL here. Upon further investigation it was found that Oracle treated empty strings as Nulls, and altering the column to accept nulls resulted in nulls being inserted. However, this was against the very definition of NULL. The SQL 2003 definition of NULL says, a NULL value is unknown and no two Nulls can be the same. So this means that there was something questionable about the way empty strings were handled in Oracle.

When inserting a number with decimal digits where the precision was set to zero, both DBMSs rounded the number without a warning, but in SQL Server there is an option to disallow this roundabout. It is disallowed by using the `SET NUMERIC_ROUNDABORT ON`. Although the commands are not directly apparent to programmer, maintaining data integrity becomes a matter of the programmer knowing his tools.

3.3.2.1.2 Integer and Small Int tests

The same operations that were performed on decimal data types were performed for integer tests and the error messages are summarised as below. Given the fact that the results obtained from both the Integer and small integer tests were basically the same, the word integer will be used in this context to mean both INT and SMALL INT tests.

3.3.2.1.2.1 Summary of error messages

See appendix C: summary of small int tests error messages

Example test performed

Insert more digits than specified – means run a query to insert a record having more than specified digits in the field being tested. Oracle and SQL Server generated error messages V and II respectively, whilst they did not insert the record.

Tests performed	Oracle error messages	Oracle Action and status	SQL Server Error messages	SQL Server Action and status
Insert normal values	None	Inserted	none	Inserted
Insert a string	III	Not Inserted	I	Not Inserted
Insert an empty string	IV	Not Inserted	none	Inserted a zero
Insert more digits than specified	V	Not Inserted	II	Not Inserted
Update to string	III	Not updated	I	Not updated
Update to an empty string	IV	Not Updated	none	Not Updated
Update to more digits than specified	V	Not Updated	II	Not updated

Table 3.7 - Integer and small int tests results

3.3.2.1.2.2 Analysis of error messages

As discovered earlier on, Oracle treated empty strings as Nulls for all its operations, SQL Server worked fine with the decimal data type but with the integer data type, there were problems.

Trying to insert an empty string into the database will result in a zero being actually inserted in the database. So from this action somewhere somehow, their zeros are equal to empty strings. Data integrity means the representing of real world scenarios as they are, since SQL Server represented the real world empty string " as something different in the database we say that data integrity was lost here. It is essential for Nulls to be treated correctly. For instance consider a product price tracking system. Having a product with a null value does not mean that, that product is free. It only means that the price is not known.

3.3.2.2 Approximate numeric data types

These basically include FLOAT, REAL, and DOUBLE PRECISION where the precision may optionally be specified.

3.3.2.2.1 Float tests

3.3.2.2.1.1 Summary of error messages

See appendix C: summary of float error tests error messages

Example test performed

Insert a number with too many decimal places than specified – means run a query to insert a record having too many decimal places than those specified for the field being tested. No error was generated by both DBMSs and the record was successfully inserted.

Tests performed	Oracle error messages	Oracle Action and status	SQL Server Error messages	SQL Server Action and status
Insert normal values	none	Inserted ✓	none	Inserted ✓
Insert a string	III	Not Inserted ✓	I	Not Inserted ✓
Insert an empty string	IV	Not Inserted ?	none	Inserted zero ✗
Insert more digits than specified	V	Not Inserted ✓	II	Not Inserted ✓
Insert a number with too many decimal places than specified	none	Inserted rounded ✓	none	Inserted rounded to 10 decimal places ✓
Update to string	III	Not updated ✓	I	Not updated ✓
Update to an empty string	IV	Not Updated ?	none	Updated ✗
Update to more digits than specified	V	Not Updated ✓	II	Not updated ✓
Update to a number with too many decimal places than specified	none	Inserted rounded ✓	none	Inserted rounded to 10 decimal places ✓

Table 3.8 - Float tests results

3.3.2.2.2 Analysis of results

As can be seen from the results the same problems which were imminent for some of the exact numeric data types are the one which were found here “the treatment of empty strings”. The other issue that was worth noting was the rounding of numbers with too many decimal places. This was regarded as correct because both DBMSs provide an option to disallow rounding; ‘SET NUMERIC_ROUNDABORT ON/OFF’. When this was set on, rounding was not allowed and an error message was raised.

3.3.2.2.2 REAL tests

3.3.2.2.2.1 Summary of error messages

See appendix C: summary of real tests error messages

Example test performed

Insert a number with too many decimal places than specified – means run a query to insert a record having too many decimal places than those specified for the field being tested. No error was generated by both DBMSs and the record was successfully inserted.

Tests performed	Oracle error messages	Oracle Action and status	SQL Server Error messages	SQL Server Action and status
Insert normal values	none	Inserted	none	Inserted
Insert a string	IV	Not Inserted	I	Not Inserted
Insert an empty string	V	Not Inserted	none	Inserted zero
Insert more digits than specified	VI	Not Inserted	III	Not Inserted
Insert a number with too many decimal places than specified	none	Inserted rounded	II	Not Inserted
Update to string	IV	Not updated	I	Not updated
Update to an empty string	V	Not Updated	none	Not Updated
Update to more digits than specified	VI	Not Updated	III	Not updated
Update to a number with too many decimal places than specified	none	Inserted rounded	II	Not updated

Table 3.9 – Real Tests results

3.3.2.2.2.2 Analysis of results

On the insertion of a number with too many decimal places than specified SQL Server produced a different action from the previous approximate data types which were investigated, error (II).

However, nothing much of a problem would be deduced from this so the test was declared passed.

3.3.3 NOT NULL tests

The presence of a NULL value indicates that the actual value of the column is unknown or not applicable [Wikipedia: 2005]. The tests implemented here, were mainly aimed at verifying whether or the DBMSs allow data to only fall within the specified limit, by trying to insert Nulls either by explicitly including them in the query string or omitting the non-null column from the query string. However it was found that in both Oracle and SQL Server this is well implemented. Appropriate error messages were generated and the correction actions were taken.

3.3.4 Check constraints tests

Although data types generally limit the values that a column can have, in most cases, we will not be wishing to use all the length of a specified data type. This is where we can make use of check constraints. Given a data type you would like the flexibility of choosing a custom length. For example, when designing column to hold student ages, we would like to be able to limit it say to two digits and so on. This enables the DBA to specify more robust data integrity rules directly into the database.

For these tests both column and table level check constraints were tested.

The tests that were carried out were:

- ❖ Those of a ceiling and floor nature (< or >) – here test were carried out to see if any violation can go unnoticed.
- ❖ Tests of enumerated types nature, for example, limiting the choice of gender to (M and F)
- ❖ The other set of tests was to break the table level constraints for example, setting constraints like (commission should be less than salary) and try to break them.

3.3.4.1 Summary of error messages

See appendix C: summary of check constraints tests error messages

Example test performed

Insert values outside the specified enumerated type – means run a query to insert a record having a value outside the specified enumerated supplied as the value of the column being tested. Error messages III and I were generated for Oracle and SQL Server respectively and the record was not inserted.











Tests performed	Oracle error messages	Oracle Action and status	SQL Server Error messages	SQL Server Action and status
Insert normal values	none	Inserted 	none	Inserted 
Insert values greater than the ceiling value	III	Not Inserted 	I	Not Inserted 
Insert values less than the ceiling value	III	Not Inserted 	I	Not Inserted 
Insert values outside the specified enumerated type	III	Not Inserted 	I	Not Inserted 
Insert values which violated the table level check constraints	III	Not Inserted 	II	Not Inserted 

Table 3.10 - Check constraints tests results

3.3.4.2 Analysis of error messages

Check constraints are a form of integrity enhancement facility. As long they were implemented properly there were no problems. This was evidenced by the fact that in all DBMSs there was no insulation against creating conflicting constraints. For instance one can easily declare one constraints CK₁ which requires all salaries to be less than R20 000 and another one CK₂ which requires all salaries to be greater than R30 000. However this becomes a matter of programming skills rather than a DBMS issue. We cannot blame DBMSs for our bad code.

Extreme caution had to be taken when working with check constraints as buggy programs could lead to serious problems. This means that DBAs need to be well versed with such thing as deferring checking and when to, and not to do certain operations. An example of such a situation is the father-son relationship shown below:

```
CREATE TABLE father (fatherID INT PRIMARY KEY,
                    SonID INT REFERENCES son (sonID));

CREATE TABLE son (sonID INT PRIMARY KEY,
                  FatherID INT REFERENCES father (fatherID));
```

Creating these tables as shown above would not be possible, rather they have to be created without specifying the foreign keys and then later use the ALTER table statement to include them as shown below.

```
ALTER TABLE father ADD CONSTRAINT fatherToson
FOREIGN KEY (sonID) REFERENCES son (sonID)
INITIALLY DEFERRED DEFERRABLE;
```

```
ALTER TABLE son ADD CONSTRAINT sonTofather
FOREIGN KEY (fatherID) REFERENCES father (fatherID)
INITIALLY DEFERRED DEFERRABLE;
```

Likewise a systematic approach should be used when deleting them. This should be possible by dropping the constraints first, then deleting afterwards. Other semantic problems may occur if two contradictory constraints are set on the same column.

3.3.5 DEFAULTS Tests

Both Oracle and SQL Server have the ability to let you specify defaults for columns. When a row is inserted and no value is specified for the column, the column will be set to the value defined as the default value. Simple tests were created to test if a default value is inserted when appropriate. No problems were found. This means that they functioned as expected. However there was still a risk of semantic problems. For example, both DBMSs allow the setting of Defaults which contradict with check constraints. One can set a default for the *employee type* column to be 'TEMP' yet the check constraints will be stating only values {TEMPORARY, PERMANENT} are allowed. At the end nothing will be able to be inserted into that field.

3.3.6 Overall analysis of domain integrity constraints

The general outcome of the experiments was that both DBMSs maintained data integrity in almost all cases. The only cases where problems were noted are those when SQL Server replaced empty strings for with zeros on almost all its numeric data types except, *DECIMAL*. This was the only difference which can set these two apart, otherwise things were almost equal. During the experiments it was noted that in some instances SQL Server flexible than Oracle, for example, in the case of altering a column to another data type. Oracle did not allow this unless the column was empty. With SQL Server as long as the data is compatible there was no problem in altering the column.

3.4 User - Defined Integrity

This part of the experimentation was based on general programming concepts. It was all about using the functionalities given by each DBMS, such as triggers, stored procedures and assertions to

define and implement business rules, for example, using these to convert a percentage mark to a grade like 2.1.

Given these tools it was possible to define different business rules. Both DBMSs provide triggers, stored procedures and functions to achieve this. SQL Server also provides assertions. So this means that the DBMSs have provided the necessary tools it all up to the DBA to produce good code which ensures that integrity is maintained.

3.5 Summary of Chapter

This chapter was about the different kinds of integrity constraints tests which were carried out and their outcomes. Generally both DBMSs maintained data integrity besides the exceptional case of empty strings. Although Oracle recognised empty strings as nulls, this did not result in the loss of data integrity but with SQL Server we can say data integrity was lost because an empty string is not a zero in real life. The other differences between these DBMSs were the support of certain standards features, Oracle does not support the Identity property which has recently been added to the standard and it also does not supported a common feature like update cascade. SQL Server also has its shortfalls, it does not support standard referential actions like set default and set null. Although they were differences here and there both DBMSs used the available features perfectly to maintain data integrity. Thus, given the available features, maintaining data integrity becomes a matter of writing good code.

Chapter 4: Transactions and concurrency control

Integrity constraints alone are not adequate to completely ensure integrity maintenance. One of the other important mechanisms in maintaining data integrity is the concept of a transaction. The ANSI/ISO SQL standard definition of a transaction is a logical unit of work that comprises all the executable SQL statements executed by a single user that ends when an explicit COMMIT or ROLLBACK statement is issued by the user [Microsoft : 2005].

Since it is the DBMS which is responsible for maintenance of integrity, it must guarantee the atomicity and durability of transactions, whilst accounting for current execution, multiple transactions, and various failure points.

4.1 Types of transactions

1. Non-Distributed transactions, which manipulate or query only a single database, which is the local database where the user is logged in.
2. Distributed transactions, which manipulate or query more than one node in a distributed database.
3. Remote transactions, which manipulate or query only a remotely located database.

For this project, only the non-distributed transactions were considered.

One of the important tests for reliability of a DBMS is the ACID test. ACID-compliant databases possess certain properties that offer greater protection to stored data in the event of an unexpected hardware or software failure, even if the database is being read from or written to at the time the failure occurs.

The ACID test alone does not guarantee reliability. Other factors such as the reliability of the host environment (both hardware and software components), a strictly observed backup policy, etc. are also crucial in maintaining any DBMS.

4.2 Transactions tests

Experiments to simulate possible failures whilst the data was being loaded into the database were carried out. These experiments were basically looking at Atomicity, Consistency, Isolation, and Durability (ACID) properties.

4.3 ACID Properties

- ❖ Atomicity. Either all operations of the transaction are committed in the database or none are.
- ❖ Consistency. A transaction should basically transform a database from one consistent state to another.
- ❖ Isolation. Although multiple transactions may execute concurrently, each transaction must only see, the changes that were made before it started. Intermediate transaction results must be hidden from other concurrently executed transactions.
- ❖ Durability. After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

4.4 Atomicity Tests

Assume that we want to transfer R2000 from the savings account 'YXZ', to the current account 'XYZ'. This should be carried as an atomic statement as shown below.

```
BEGIN TRAN
    UPDATE  accounts
    SET     Balance = Balance - 2000
    WHERE  Account_ID = 'YXZ'

    UPDATE  accounts
    SET     Balance = Balance + 2000
    WHERE  Account_ID = 'XYZ'

END TRAN
```

In carrying out these kinds of experiments, it was found that the two DBMSs had different implementations of transactions. For example, given a transaction T₁ shown below

```
BEGIN TRAN
    Write A
    ERROR Occur
    Write B
COMMIT TRAN
```

In these types of transactions, B was actually written in SQL Server. This was mainly because with SQL Server, transactions do not care if the statements run correctly or not. They only care if SQL Server, itself, failed in the middle. For example, the transaction will try to insert a duplicate entry into a primary key field but you get a primary key violation error message. A message will even tell you that the statement has been terminated. But the transaction will still be going. The UPDATE statement runs just fine and SQL Server then commits the transaction. So here there was a problem with the definition of a transaction in SQL server, even though one of the statements in the TRANS failed the transaction committed anyway. This means that other functionalities had to be included to check when these errors had occurred and rollback the transaction. An example of the workaround is shown below.

```
USE pubs
DECLARE @ErrorInt INT
BEGIN TRAN
    UPDATE Authors
    SET Phone = '213 354-8888'
    WHERE au_id = '586-60-5874'

    SELECT @ErrorInt = @@ERROR
    IF (@ErrorInt <> 0) GOTO EXCEPTION

    UPDATE Publishers
    SET city = 'Calcutta', country = 'India'
    WHERE pub_id = '9999'

    SELECT @ErrorInt = @@ERROR
    IF (@ErrorInt <> 0) GOTO EXCEPTION
COMMIT TRAN

EXCEPTION:
IF (@ErrorInt <> 0) BEGIN
PRINT 'AN ERROR OCCURRED'
    ROLLBACK TRAN
END
```

As can be seen above after every statement you need to have a,

```
SELECT @ErrorInt = @@ERROR
IF (@ErrorInt <> 0) GOTO EXCEPTION,
```

section which will check that no errors had occurred otherwise execution will be transferred to the EXCEPTION.

To start with, they have different transaction settings. SQL Server auto commits transactions by default whilst this functionality is off by default in Oracle. This means that to be able to carry out these tests, all DBMSs had to set the *AUTO COMMIT OFF* so that the DBA can explicitly choose when to commit or rollback the transactions.

To simulate a bigger transaction the looping capability of T-SQL and PL/SQL was used. My transaction generator was based on the funds transfer example but this time around it was on a large scale.

The government is paying R1000 000 000 000 into the unemployment fund which is going to be debited to, say 100 000 people.

Figure 4.1 – Example transaction

This means that there are 100 000 equal payments to these people. Using transactions, this is a funds transfer of some sort. It would make sense to record a payment of R1 billion from the government's account to the respective peoples' accounts. But as usual it should be done in such a way that the Accounts will balance. After all the transferred were finished the results were checked and they were consistent. This was then implemented using threading in visual studio.NET 2005.

Systems crashes were simulated by:

- Restarting the MSSQLSERVER service whilst the transaction was running

The Error message

[Microsoft][ODBC SQL Server Driver]Unspecified error occurred on SQL Server. Connection may have been terminated by the server. Connection Broken error message was return as the transaction stopped, upon checking the database it was still in a consistent state as all changes had been rolled back. Oracle also maintained data integrity.

- Restarting the machine whilst the transaction was running.
- Plugging off the machine from the power plug.
- Switching off the power.

All these operations were done whilst different transactions were running but not yet committed. In all cases data integrity was maintained. After these transaction transactions an investigation was made on the different types of locking mechanisms and isolation levels implemented these DBMSs.

4.5 Interactions and isolation levels

The ANSI/ISO SQL standard SQL92 defines three possible kinds of transaction interaction, and four levels of isolation that provide increasing protection against these interactions.

- ❖ **Dirty read** - a dirty read occurs when a transaction reads data that has not yet been committed. For example, suppose transaction 1 changes a row. Transaction 2 reads the changed row before transaction 1 commits the change. If transaction 1 aborts the change, transaction 2 will have read data that is considered to have never existed.
- ❖ **Non-repeatable read** — a non-repeatable read occurs when a transaction reads the same row twice but gets different data each time. For example, suppose transaction 1 reads a row. Transaction 2 changes or deletes that row and commits this change or deletion. If transaction 1 attempts to reread the row, it retrieves different row values or discovers that the row has been deleted.
- ❖ **Phantom** — a phantom is a row that matches the search criteria but is not initially seen. For example, suppose transaction 1 reads a set of rows that satisfy some search criteria. Transaction 2 inserts a new row that matches the search criteria for transaction 1. If transaction 1 re-executes the statement that read the rows, it gets a different set of rows.

The other problem which is part of these interactions is lost updates.

❖ Lost Updates

Lost updates occur if two transactions modify the same data at the same time, and the transaction that completes first is lost or overwritten. These are common with the READ UNCOMMITTED isolation level. Suppose that the beginning balance on my savings account is R5000. I deposit a R3000 at 08:30 a.m., and my brother withdraws R2000 from the ATM at 08:30 a.m. If all is well, my ending balance should be $R5000 + 3000 - 2000 = R6000$. However, if the transaction isolation level is set to READ UNCOMMITTED, and my brother's transaction is committed after mine, my ending balance at 08:32 a.m. will be $R5000 - 2000 = R3000$.

These interactions and isolation levels are:

Isolation Level	Dirty Read	Non-Repeatable Read	Phantom Read
READ UNCOMMITTED	Possible	Possible	Possible
READ COMMITTED	Not possible	Possible	Possible
REPEATABLE READ	Not possible	Not possible	Possible
SERIALIZABLE	Not possible	Not possible	Not possible

Table 4.1 - Interactions and isolation levels⁶

The behaviour of Oracle and Microsoft SQL Server 2000 is:

Isolation Level	Description
READ UNCOMMITTED	Oracle never permits "dirty reads." Although SQL Server use this undesirable technique to improve throughput, it is not required for high throughput with Oracle.
READ COMMITTED	Both SQL Server and Oracle meets the READ COMMITTED isolation standard.
REPEATABLE READ	Oracle does not normally support this isolation level, except as provided by SERIALIZABLE. Is supported in SQL Server
SERIALIZABLE	You can set this isolation level using the SET TRANSACTION command or the ALTER SESSION command. This is the highest level of isolation.

Table 4.2 – Oracle and SQL Server isolation levels⁷

This means that since READ COMMITTED is always the default for both SQL Server and Oracle problems non-repeatable and phantom reads possible [Akadia Information Technology: 2005]

4.6 Concurrency in a nutshell

With Oracle, readers do not block readers, readers do not block writers and writers do not block readers. This is called multi-version concurrency control. But with SQL Server readers block writers and writers block readers. With Oracle locks can be at less granular levels like rows whilst this is not possible with SQL Server. There are generally architectural implementation differences on transactions. Oracle uses redo log files and roll back segments to keep track of its transactions whilst SQL Server uses transactions log files. However this does not have a major influence on integrity but rather on the performance of the two DBMSs. Each approach has its own advantages.

⁶ Adapted from [Akadia Information Technology: 2005] http://www.akadia.com/services/ora_important_part_4.html

⁷ Adapted from [Akadia Information Technology: 2005] http://www.akadia.com/services/ora_important_part_4.html

4.7 Summary Conclusion

After this several tests were carried out using SQL scripts to tests things such as nested transactions and play around with different database settings, but the overall result was that, it was all about knowing what to use and when to use it in order to maintain integrity. Oracle was somehow superior to SQL Server in transaction handling by not allowing such things as dirty reads and its use of multi-version concurrency model, where readers do not block writers. A thorough investigation of locking mechanisms can be a project on its own, hence it was not considered in detail in this project. Probably that's the reason why Oracle is mostly the choice for big companies.

Chapter 5: SQL Standards conformance

5.1 SQL Standards investigation

Like SQL 99, in order to claim conformance to SQL 2003, there is a minimum level of conformance which had to be first met. This minimum level conformance can be defined as a claim to meet the conformance requirements of the two parts that make the core of this standard which are *Part 2 - SQL/Foundation* and *Part 11 - SQL/Schemata*. After the minimum has been met several claims on *parts*, *features* and *packages* can be made.

To be in a position to determine whether a DBMS conforms to this standard, it has to at least conform to the minimum level. So, it is this minimum level which was investigated in this project. This was done by investigating the features which are found in these two parts.

The findings for this part are in the form of summarised tables. In these tables the following conventions adapted from [Kline.K:2004] and [Gulutzan P: 2005] were used.

- *Supported (S)*
The vendor supports the SQL 2003 standard for the particular command.
- *Supported, with variations (SWV)*
The vendor supports the SQL 2003 standard for the particular command, using vendor-specific code or syntax.
- *Not supported (NS)*
The vendor does not support the particular command according to the SQL 2003 standard.

Mimer SQL-2003 Validator was used in conjunction with documentations of both DBMSs and various other sources to test the conformity of the various statements to the SQL 2003 standards.

As mentioned before, SQL 2003 defines seven types of SQL classes. Each of those classes has its own different commands. Using Mimer Validator, the following command classes adapted from [Kline.K:2004] were investigated and the results which were obtained can be summarised as shown below.

SQL schema commands

According to [Kline, K: 2004], these are commands that may have a persistent and enduring effect on a database schema and objects within that schema. Table D.1 in appendix D gives an overview of the support extent to which Oracle and SQL Server 2003 support this class of commands. The table can be represented graphically as shown in figure 5.1.

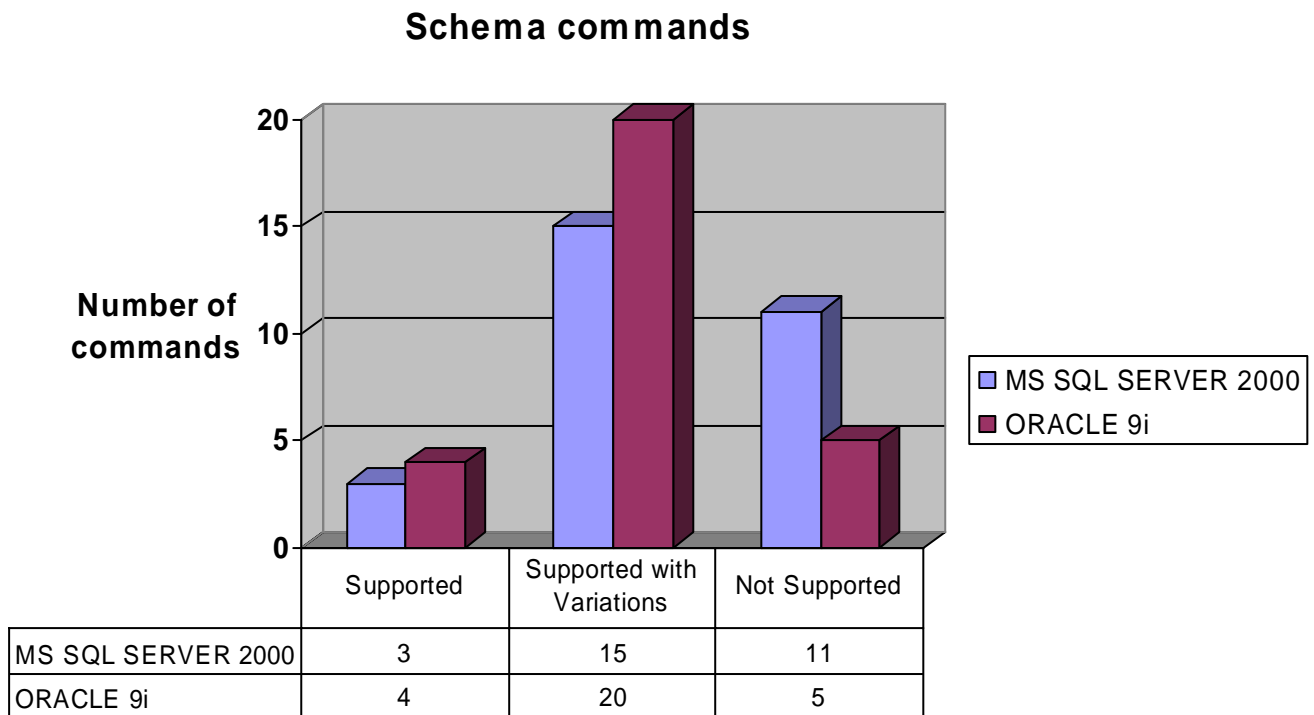


Figure 5.1 - Schema commands summary

This graph shows that there was a gap between Oracle and SQL Server, with Oracle leading. The fact that the number of features they *support* was about a fifth of those they *support with variations* shows how much these vendors are into product differentiation. However supporting with variations is better than not supporting the standard at all, thus Oracle was better off because it missed only 5 whilst SQL Server missed 11.

SQL-data commands

According to [Kline, K: 2004], these are statements that may have a persistent and enduring effect upon data. This is the class where the most common SQL statements like *Select*, *Insert*, *Update* and *Delete* are found. Table D.2 in appendix D gives an overview of the support extent to which Oracle and SQL Server support this class of commands. This can be graphically illustrated by figure 5.2.

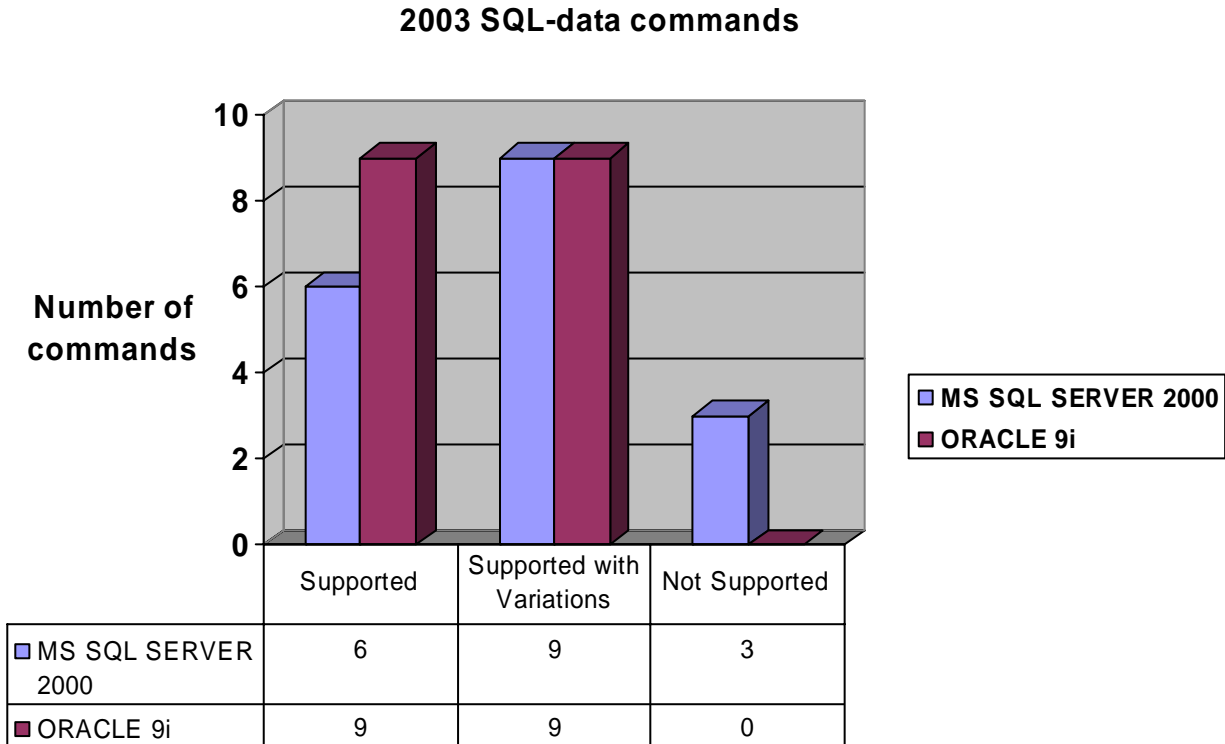


Figure 5.2 - SQL-data commands summary

On these type of commands, Oracle had the upper hand as it supported all the features considered whilst SQL Server missed 3.

SQL-connection, session and transaction statements

Table D.3 in appendix D gives highlights of the extent to which Oracle and SQL Server support these classes of commands. Again this can be diagrammatically represented by figure 5.3 shown on the next page.

SQL-Connection,Session and Transaction commands

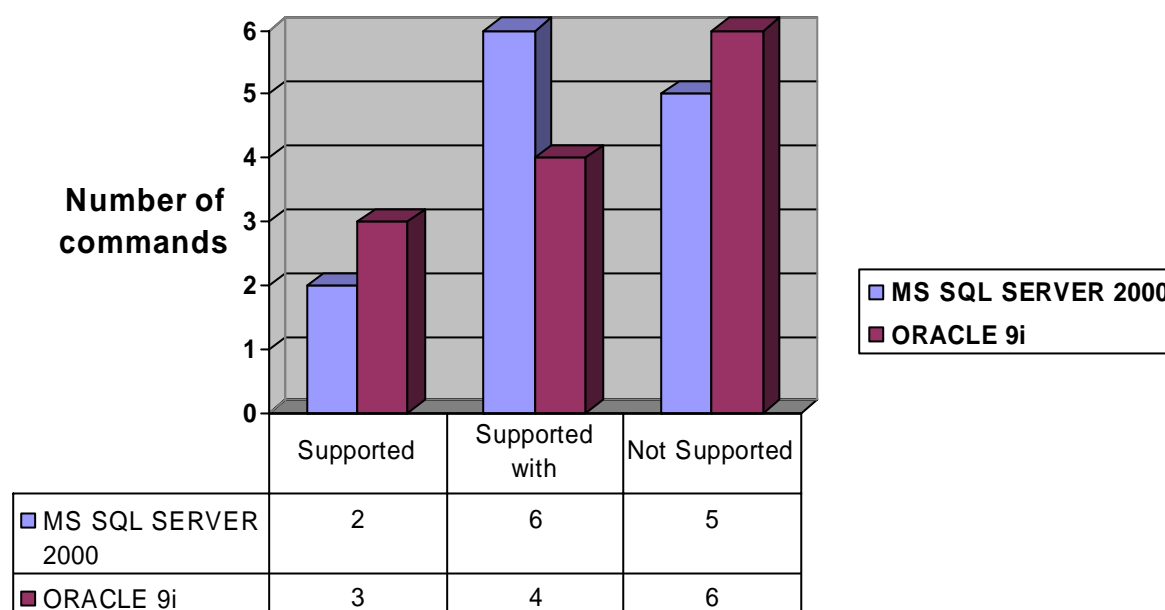


Figure 5.3 - SQL-connection, session and transaction statements summary

The results show that SQL Server supported 2 commands according to the standards whilst 6 with variations and missed 5. On the other hand Oracle missed 6 which was 1 more than SQL Server's, but supported 3 according to the standard and 4 with variations.

After the *command classes* an investigation was made on their support of SQL data types. Both Oracle 9i and SQL Server 2000 have a vast pool of data types. Some of them are proprietary whilst some are SQL 2003 data types. In this investigation only the SQL 2003 data types implemented by both DBMSs were considered.

SQL 2003 data types

For this part the *MySQL crash-me*⁸ toolkit in conjunction with other necessary documentation were used. With the *MySQL crash-me* tool, you choose any two or more DBMSs you want to compare. After you have made your selection a comparison is done. After verifying its results with the standards documents, it was found that it included some other data types like *bit*, which are no longer part of the standards.

The graph below is used to give a collation of the results of the data-types investigations.

⁸ The data types considered are found using the mysql crash me tool found at: <http://dev.mysql.com/tech-resources/crash-me.php>

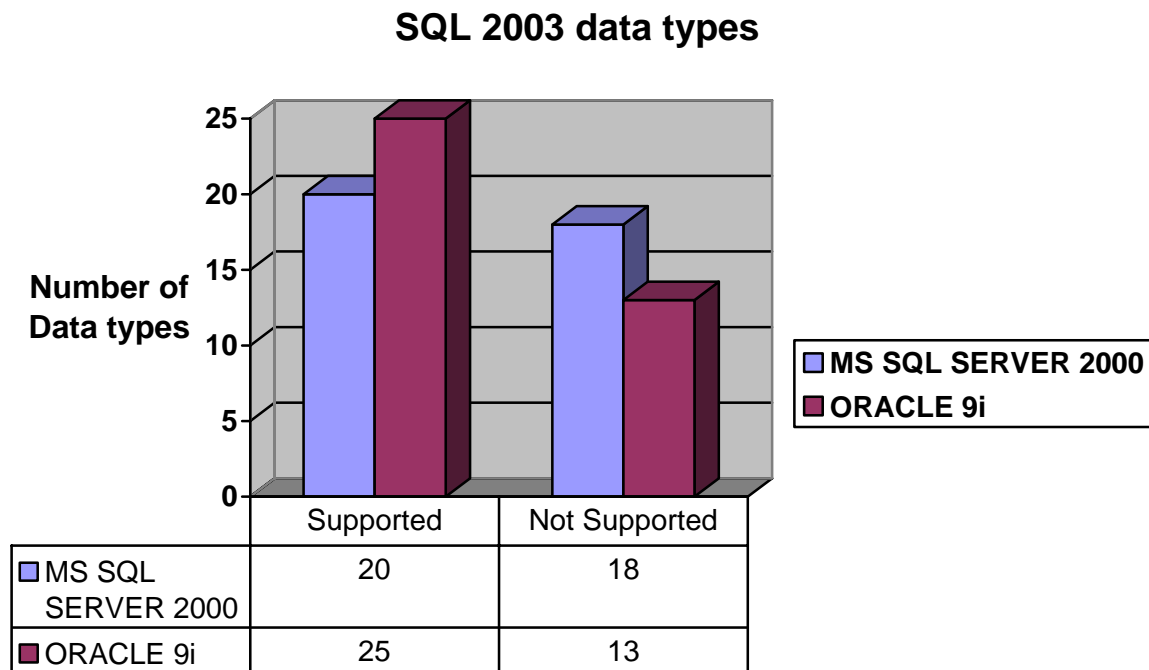


Figure 5.4 - SQL 2003 data types

Of the 38 SQL 2003 data types investigated Oracle supported more data types than SQL Server. Oracle support 25 whilst SQL Server supported 20. For example, SQL Server does not support types DATE and TIME; instead it uses its own dialect, DATETIME which is basically a time stamp. Another instance when both DBMSs fell short is in the support for the Boolean data type. They do not support this data type. After the data type investigation, the categories of syntax were investigated.

Categories of Syntax

Syntax falls into four categories which are identifiers, literals, operators and reserved words and key words. Identifiers were considered.

Identifiers – this describe a user or system supplied name for a database object. SQL 2003 provides rules for identifying these objects. The table below contrast the SQL 2003 rules with those of Oracle and SQL Server.

SQL 2003 rules for naming Identifiers

Figure D.4 in appendix D gives a comparison of the SQL 2003 rules of naming identifiers to those implemented in Oracle 9i and SQL Server 2000. This can be further refined to show the total numbers supported and not supported as shown in figure 5.5.

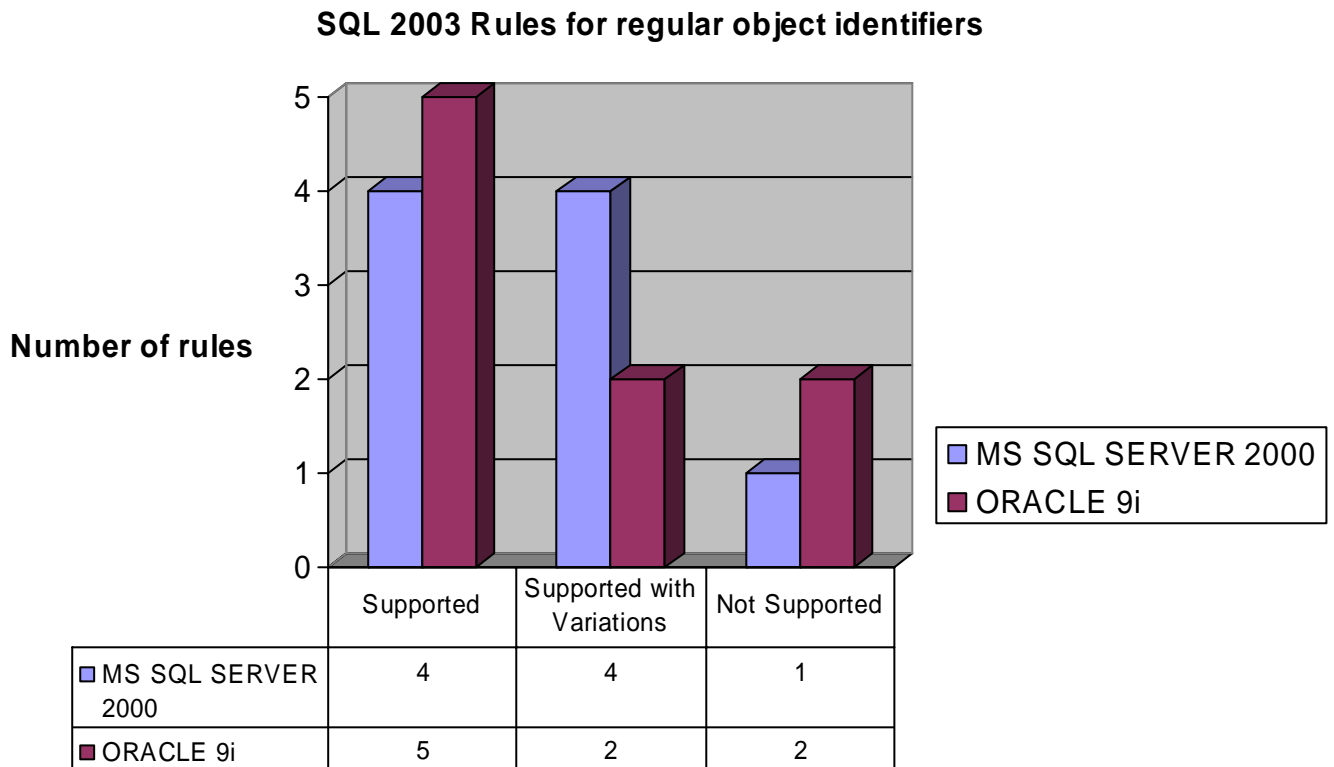


Figure 5.5 - SQL 2003 rules for naming Identifiers

There was a thin line between the two DBMS. Oracle supported 5 of the rules according to the standards, 2 with variations and missed 2. On the other hand SQL Server supported 4 as per the standards, 4 with variations and missed only 1.

After the identifier rules, the *MySQL crash-me* tool and Mime SQL 2003 Validator were used to evaluate the two DBMSs support for SQL 2003 built-in functions. The results are summarised as shown below.

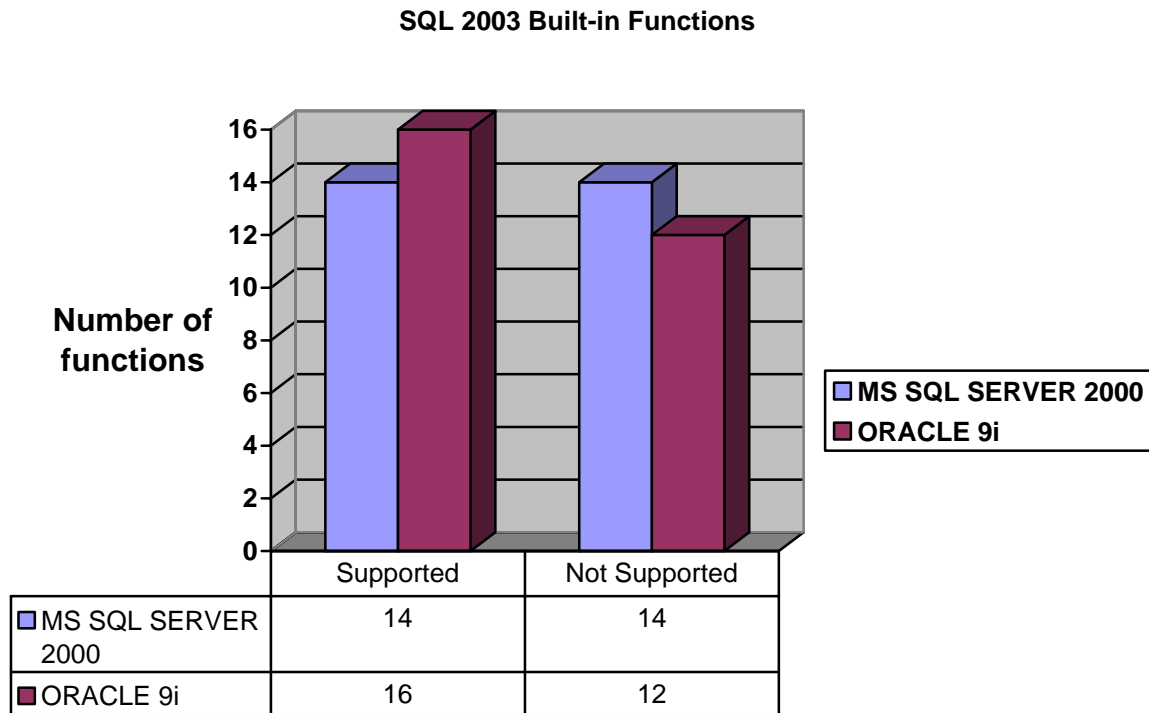


Figure 5.6 - SQL 2003 built-in functions support summary

As can be seen from the results Oracle again had a slight lead against SQL Server. It supported 16 and missed 12, whilst SQL Server supported 14 and missed 14.

As the standards conformance did not change very much from SQL 99. Conformance requirements also did not change, a DBMS which was SQL 99 conformant automatically became SQL 2003 conformant. [P. Gulutzan [1]: 2005] made a comparison of Oracle's and SQL Server's SQL 99 Core feature support. The outcome is summarised as below.

SQL 99 Core feature support

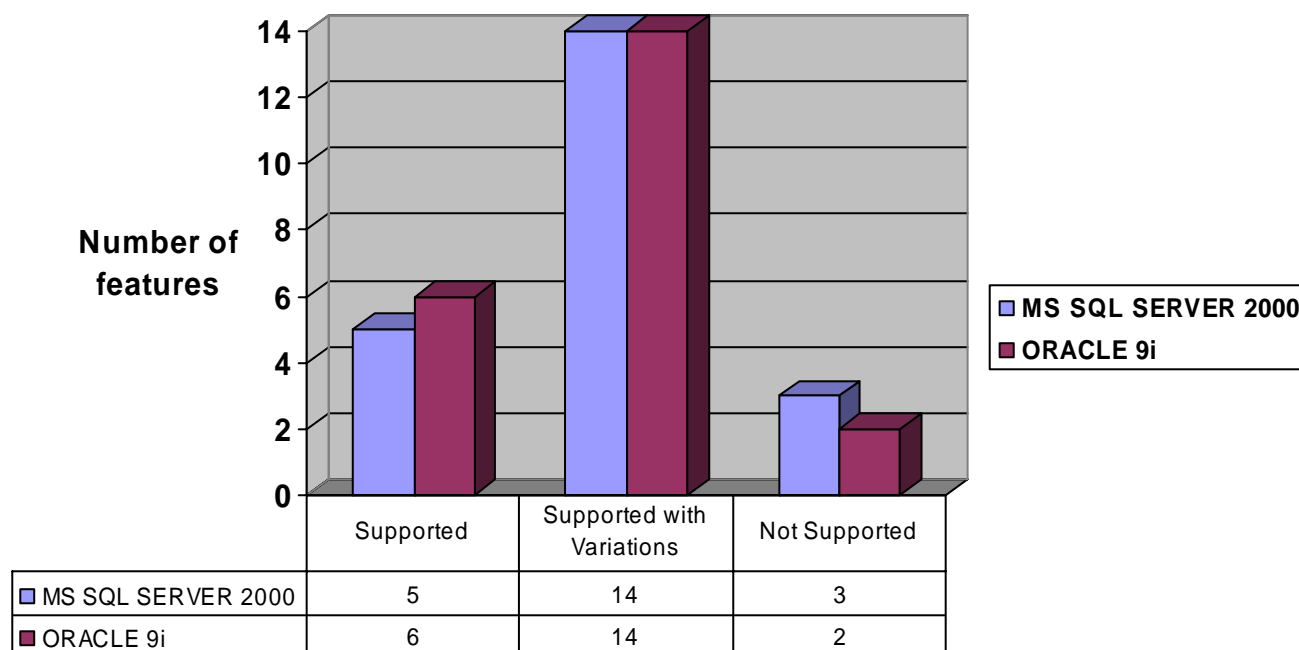


Figure 5.7 - SQL 99 Core feature support summary

Again from the results shown above there was a slight difference but Oracle was ahead. It supported 6 and supported 14 with variations whilst missing 2. On the other hand SQL Server supported 5, 14 with variations and missed 3.

5.2 Weighting of all the SQL 2003 results

After all these findings, it was decided to summarise all the results obtained from the different categories. This was essential as we needed to be in a position to conclusively say which one is better than the other. This was achieved formulating a weighting of these results. The process of devising the weighting criteria involved a disciplined subjective approach.

This approach involved first identifying the kind results that were obtained. As noted, these results are in three general categories namely: Supported, Supported with variations and Not supported.

The category having a chief importance in making the decision ahead was identified as *Supported* since this shows the act of exactly implementing the feature as specified by the standard. This was

followed by *Supported with variations*. Although this category shows that they support, it also cause the general problems of lack of portability due to the fact that features/ commands would have been implemented in a proprietary manner. This might also have advantages of adding some more powerful features which are not provided by the standard. The last which was the undesirable category was the not supported category. This category refers to all those features that are not supported, even in a proprietary manner.

Since these weights were to be used to determine the points each DBMSs will get, 10 points were given for each supported feature, 6 points given for each feature supported with variations and 3 points deducted for each feature not supported, meaning a weighting of 10, 6 and -3 was used. The weighted results can then be summarised and be shown in figure 5.8 below.

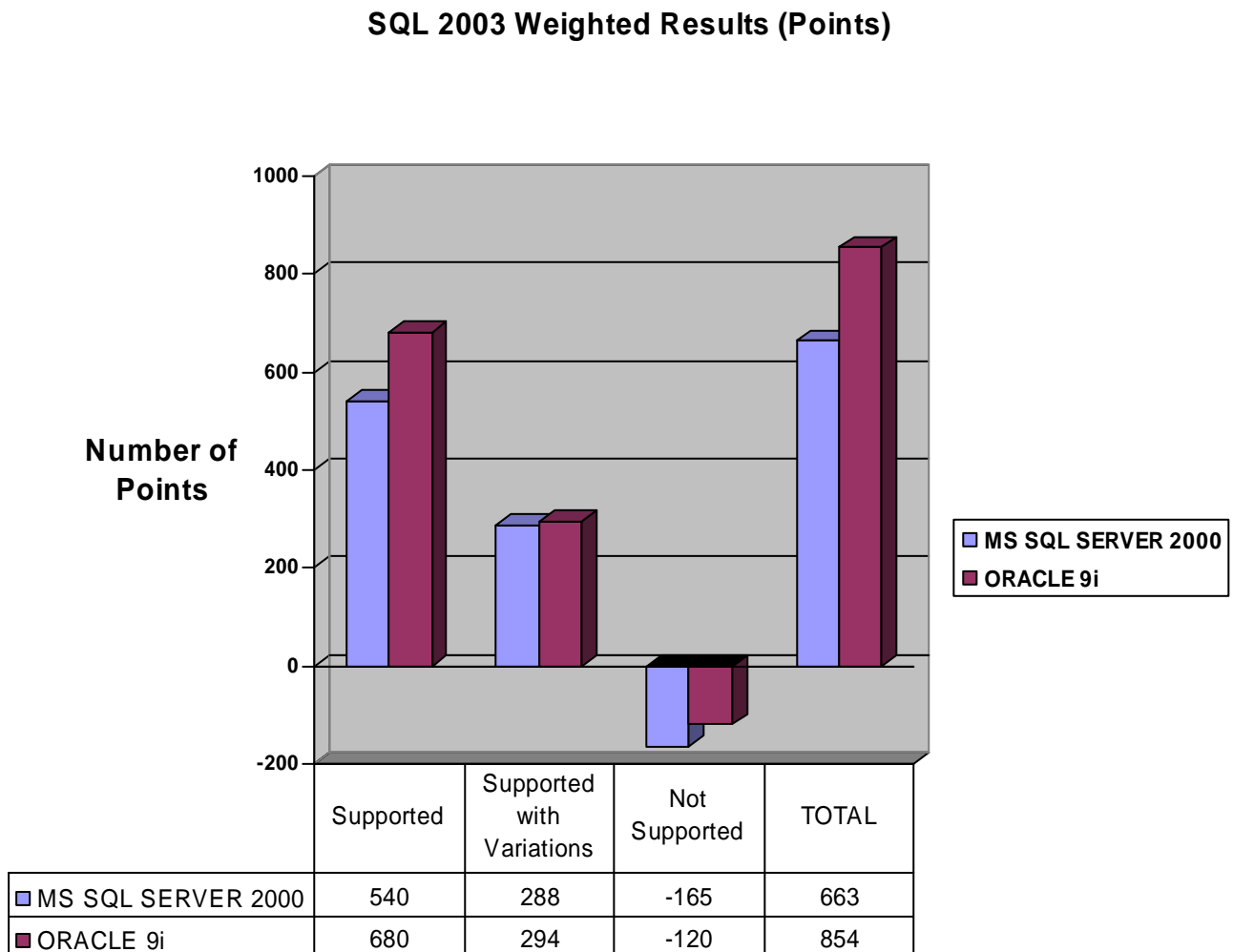


Figure 5.8 - SQL 2003 weighted results

As can be seen from the results in figure 5.8, it can be concluded that Oracle supports SQL 2003 better than SQL Server. Using the proposed weighting method, MS SQL Server had 540 points for supported features, 288 for those supported with variations and (-165) for those not supported. On the other hand Oracle have 680 for the supported features, 294 for those supported with variations and (-120) for those not supported. This shows that Oracle was ahead, although the gap was always narrow in the different categories considered.

5.3 Summary of Chapter

This chapter explored the SQL standards. This was done by first giving a general background on the standards that is their history and timeline. The important standards and their conformance requirements were highlighted. The SQL structure of and conformance requirements of SQL 2003 were also given after which evaluations were carried out. The two DBMSs were evaluated with respect to different standards features and components. The general result was that Oracle supported the SQL 2003 better than SQL Server.

Chapter 6: Conclusions and Possible Extensions

6.1 Conclusions

Integrity which is one of the main reasons for using DBMSs has gone through many years of refining in both Oracle and SQL Server. As a consequence, they now provide almost all the tools and functionalities which are necessary for the maintenance of integrity. Thus, as far as data integrity is concerned it is not true that SQL Server 2000 is better than Oracle 9i or vice versa. Both products can be used to build stable and reliable systems. And the stability and reliability of your applications and databases depend rather on the experience of the database developers and database administrator rather than on the database's provider. But however, SQL Server 2000 has some advantages in comparison with Oracle 9i and vice versa.

Although these DBMS have made tremendous efforts, they do not conform to the SQL2003 standards. But, as the first commercial implementation of SQL over 25 years ago, Oracle continues to lead SQL Server in implementing SQL standards. Surprising even SQL 2005 will not be SQL 99 compliant [Channel 9 forums: 2005]. However, all RDBMS platforms in the market are always behind the standards. Many times, as soon as vendors close in on the standard, the standards bodies update, refine, or otherwise change the benchmark. Conversely, the vendors often implement new features that are not yet part of the standard, probably because they are required by their customers before the standard is made [Kline, K 2004].

6.2 Possible extensions

6.2.1 Evaluating SQL Server 2005 and Oracle 10g

Since SQL Server 2005 is now released, future projects might carry out a state of the art comparison of this version with Oracle 10g.

6.2.2 Evaluating DBMSs with respect to Security

The maintenance of data integrity is also largely depended on the security mechanism or principles which a DBMS use. In recent years there have been a number of security attacks on various DBMS products. The attacks were mainly SQL injection and buffer-overflows. So it would be quite interesting to know which one is the most vulnerable.

Appendix A: Prerequisite for Investigation and Implementation

Before any kind of experiment or investigation could be done it was necessary to gain adequate knowledge of each of SQL languages used by each DBMS so as to be able to manipulate, interact and experiment with them effectively. Oracle uses PL/SQL whilst SQL Server uses T-SQL.

1. Overview of PL/SQL

PL/SQL stands for Procedural Language/SQL. PL/SQL extends SQL by adding constructs found in procedural languages, thus providing a structural language that is more powerful than SQL.

By adding procedural constructs to SQL, such as encapsulation, function overloading, information hiding, block structure, conditional statements, loop statements, variable types, structured data and customized error handling, the PL/SQL language takes on characteristics of object-oriented programming languages. PL/SQL's language syntax, structure and data types are similar to that of the ADA programming language. Integrated with a database server, PL/SQL does not exist as a standalone language. It typically is used to write data-centric programs to manipulate data in an Oracle database.

SQL does not readily provide "first row" and "rest of table" accessors, and it cannot easily perform some constructs such as loops. PL/SQL, however, as a Turing-complete procedural language which fills in these gaps, allows Oracle database developers to interface with the underlying relational database in an imperative manner [Wikipedia: 2005]

The basic unit in PL/SQL is a block. All PL/SQL programs are made up of blocks, which can be nested within each other. Typically, each block performs a logical action in the program. Blocks take the general form:

2. Overview of T-SQL

Just like oracle T-SQL is a proprietary extension of standard SQL to provide more flexibility and power.

T-SQL (Transact-SQL) is a set of programming extensions from Sybase and Microsoft that add several features to the Structured Query Language (SQL) including transaction control, exception and error handling, row processing, and declared variables. Microsoft SQL and Sybase both support T-SQL statements. Sybase refers to its statements as T-SQL while Microsoft refers to its statements as Transaction-SQL.

The name Transact-SQL isn't exactly self-explanatory, but it does imply the idea of "transactional" extensions to the SQL database programming language. Transact-SQL isn't a standalone product. You cannot use it to write applications in the same way you could with C++ or Java. Instead, Transact-SQL is the main enabler of programmatic functionality within the relational databases provided by Microsoft and Sybase. Transact-SQL is very closely integrated with SQL while adding programming capabilities not already standardized within the SQL database programming language. At the same time Transact-SQL extends SQL, it also integrates seamlessly with it [Kline K et al: 1999]

Appendix B: Integrity constraints

The Integrity subsystem is responsible for maintaining the accuracy, correctness and validity of the data stored in a database, detecting and acting on integrity violations. It must exert deliberate control on every process that uses the data to ensure the continued correctness of the information.

One of the major drives behind the development of RDBMSs was to ensure data consistency, yet this is one of those things that do not seem like an obvious topic for administrators to address directly and has been totally ignored by database benchmarks. According to [Mullins C S: 2002] a database is of little use if the data it contains is inaccurate. Integrity can be defined as a steadfast adherence to a strict moral or ethical code.

[Türker and Gertz: 2000] state that the accuracy of the data managed by a DBMS is vital to any application using the data for business, research and decision making purposes. Integrity basically deals with the extent to which the data managed by a DBMS reflect the real-world data or artefacts consumed and operated on by applications. Data integrity requirements are gathered from users, application developers, and business policies and then translated into integrity constraints. When these constraints have been defined in a DBMS language, they specify conditions which DBMS objects have to meet in order to correctly reflect real world data. This means that, they are translated into constraint enforcing mechanisms provided by the DBMS. Integrity constraints are predicates that specify what database states are admissible, that is, correctly reflect the real world semantics.

[Mullins C S: 2002] states that, integrity can be classified as Database structure integrity and Semantic data integrity. This section discusses a state-of-the-art overview and comparison of semantic integrity features provided by Oracle 9i and SQL Server 2000. We do so by considering the four different types of semantic integrity constraints which are namely: Entity Integrity, Referential Integrity, Domain Integrity, and User – Defined Integrity.

1.1 The motive behind the maintenance of data integrity

[Webopedia: 2005] states that data integrity can be compromised in a number of ways which include:

Human errors in entering data, errors that occur when data is transmitted from one computer to another, software bugs or viruses, hardware malfunctions, such as disk crashes and natural

disasters such as fires and floods.

After data has been loaded into a database, the question now would be why should we bother maintaining its integrity? There are several reasons for this, but the major ones can be identified as:

1.1.1 Protecting the data existence

After a customer has deposited his money into a bank account he expects it to be there the next time he comes to make a withdrawal, otherwise he would not have deposited it in the first place. This means that we expect all the data stored in a database to be available when needed, despite any problems which might have occurred. This includes safeguarding the data from catastrophes like fire and floods highlighted above.

1.1.2 Maintaining quality

This is all about ensuring that the data is accurate, current, consistent and complete. This is generally achieved by employing mechanisms of ensuring correct, consistent data entry, always having timely and up-to-date records and avoiding software bugs or viruses.

1.1.3 Ensuring Confidentiality

This is a responsibility of the security subsystem of the DBMS. The security subsystem should be to detect and disallow unauthorised access of data.

1.2 Database Structure Integrity

This refers to the architectural, internal structures and pointers used to keep database objects in the proper order. If these are disturbed in any way, database access will be compromised.

1.3 Semantic Data Integrity constraints in SQL 2003

This deals with the DBMS features and processes that can be used to ensure data consistency. The DBMS should exert deliberate control on every process that uses your data to ensure the continued correctness of the information. However the RDBMSs automatically enforce integrity up to a certain point, and from there DBAs have to ask themselves how best they could enforce data integrity, because the RDBMSs will not protect them from inept handling of transactions.


The current SQL standard, SQL 2003 provides support of semantic integrity constraints, both declaratively as well as procedurally using triggers. Semantic Integrity constraints define the valid

Appendix B: Integrity Constraints

states of SQL-data by constraining the value stored in the database. A constraint can either be a table constraint, a domain constraint or an assertion and is described by a constraint descriptor. This descriptor is composed of:

- A name - which makes sure that, an integrity constraint is uniquely identified by its name within a database schema. If a name is not specified explicitly, then the system will implicitly provide an implementation-dependent name.
- The initial checking mode – which is an indication whether this is set to deferred or immediate.
- A flag indicating whether or not the checking of the constraint can be deferred.

The declarative integrity support in each DBMS as compared to the SQL 2003 standard can be depicted in the table below.

 - means that this feature is supported

 - means that this feature is not supported

INTEGRITY FEATURES			SQL 2003	MS SQL 2000	Oracle 9i	
NOT NULL						
DEFAULT						
UNIQUE						
PRIMARY KEY						
FOREIGN KEY	MATCH	SIMPLE				
		PARTIAL				
		FULL				
	ON DELETE	NO ACTION				
			RESTRICT			
		CASCADE	SET NULL			
			SET DEFAULT			
	ON UPDATE	NO ACTION				
			RESTRICT			

		CASCADE	✓	✗	✓
		SET NULL	✓	✗	✗
		SET DEFAULT			
CHECK	Column – level		✓	✓	✓
	Row – level		✓	✓	✓
	Table – level		✓	✓	✓
	Database – level		✓	✗	✗
DOMAIN			✓	✓	✓
ASSERTION			✓	✓	✗

Table B.1 – Oracle’s and SQL Server’s support of SQL2003 declarative integrity⁹

As can be seen from the table above, both DBMSs have made big efforts in trying to implement all the standard features but they are still lagging behind the standards.

1.3.1 Entity Integrity constraints

Entity integrity is all about uniquely identifying each instance of an entity. The entity integrity rule stipulates that every instance of an entity is uniquely identified or the value of the PRIMARY KEY must exist, be UNIQUE, and **cannot be null** [SQL- 92]. In this case, each table represents an entity, and each row of a table represents an instance of that entity. For example, if **order** is an entity, the **orders table** represents the idea of an order and each row in the table represents a specific order. Thus experiments were carried out for UNIQUE and PRIMARY KEY constraints. The experiments were centred on testing if each of the two constraints performed as expected in each DBMS. Basic INSERT, UPDATE and ALTER command were used to create the scripts for these tests. SQL 2003 introduced a new entity integrity constraint called *IDENTITY*, however this was not considered as it was implemented in these DBMSs.

Both Oracle [Oracle: 2005] and Microsoft SQL Server [Microsoft: 2003] enforce uniqueness by automatically creating unique indexes whenever a PRIMARY KEY or UNIQUE constraint is defined. Additionally, primary key columns are automatically defined as NOT NULL.

1.3.1.1 Unique constraints

These constraints are satisfied if and only if no two rows in a table have the same non-null values in

their *UNIQUE* columns. [Türker, Gertz: 2000], states that:

A uniqueness constraint $UNIQUE(X_1, \dots, X_n)$ holds for a table R in a database if and only if there are no two rows r_1, r_2 in R such that the values of all their uniqueness columns X_i match and are not null.

1.3.1.2 Primary constraints

A primary key is a special unique constraint. A primary key constraint is satisfied if and only if it is *UNIQUE* and does not allow null values in the specified column(s). Entity integrity simply ensures that every row in a table is unique. In other word it makes sure that duplicate rows are not possible.

1.3.2 Domain Integrity

This basically operates at field level. It is all about the permissible entries that a column can have. And according to [Oracle: 2005] and [Microsoft: 2005], you can enforce domain integrity by restricting the type (through data types), the format (through *CHECK* constraints and rules), or the range of possible values (through *FOREIGN KEY* constraints, *CHECK* constraints, *DEFAULT* definitions, *NOT NULL* definitions, and rules). Oracle treats a default as a column property, and Microsoft SQL Server treats a default as a constraint. The SQL Server *DEFAULT* constraint can contain constant values, or *NULL*. It is also added that the syntax used to define *CHECK* constraints is the same in Oracle and SQL Server, and they create column constraints to enforce nullability. Their columns default to *NULL*, unless *NOT NULL* is specified in the *CREATE TABLE* or *ALTER TABLE* statements [Sheldon R, Wilansky E. 2001].

1.3.2.1 Check constraints

Check constraints place specific data value restrictions on the contents of a column and any attempt to modify the column data will cause the search condition to be evaluated. This ensures that values match specific conditions you would have set out.

The diagram below illustrated some of the domain integrity constraints.

⁹ Adapted from [Kline et.al : 1999]

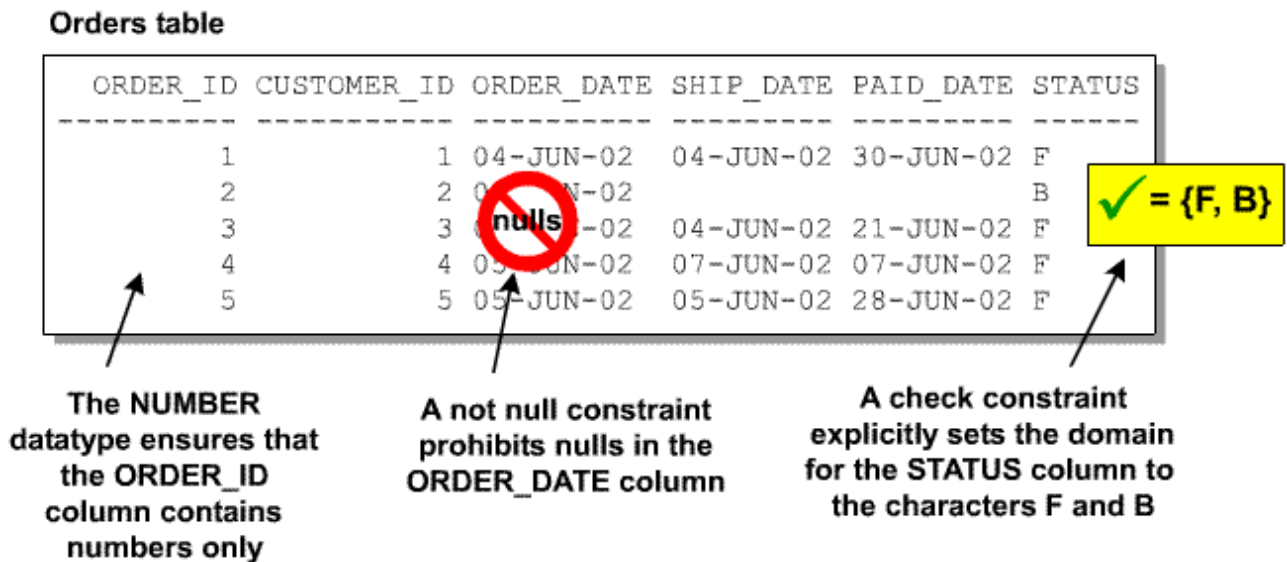


Figure B.1- Domain Integrity: Datatypes, Not Null Constraints, and Check Constraints¹⁰

Domain integrity ensures that each field value falls within a specified range. For example, in the diagram above, the ORDER_ID column only allows numbers, no string, no dates and the likes. The ORDER_DATE column only allows dates and in addition it cannot be NULL. Lastly the status column only permits a single character which is either F or B only.

1.3.3 Referential Integrity

Referential constraints are an important means to describe dependencies among (portions of) rows in tables. There are the referenced (or parent) and referencing (or child) tables where a subset f_i, \dots, f_k of the columns of the referencing table builds the foreign key and refers to the unique/primary key columns u_j, \dots, u_l of the referenced table [Türker, Gertz: 2000].

These are concerned with keeping the relationship between tables synchronized. In order for this type of integrity to be maintained, a *FOREIGN KEY (FK)* in a “child table” should only accept values if they exist in the “parent table”. In SQL Server 2000 and Oracle 9i referential integrity is based on relationships between foreign keys and primary keys or between foreign keys and unique keys (through FOREIGN KEY and CHECK constraints). This ensures that values are consistent across the tables. The diagram below depicts the relationships between Customers and their orders.

¹⁰ Adapted from [Animated Learning: 2002]

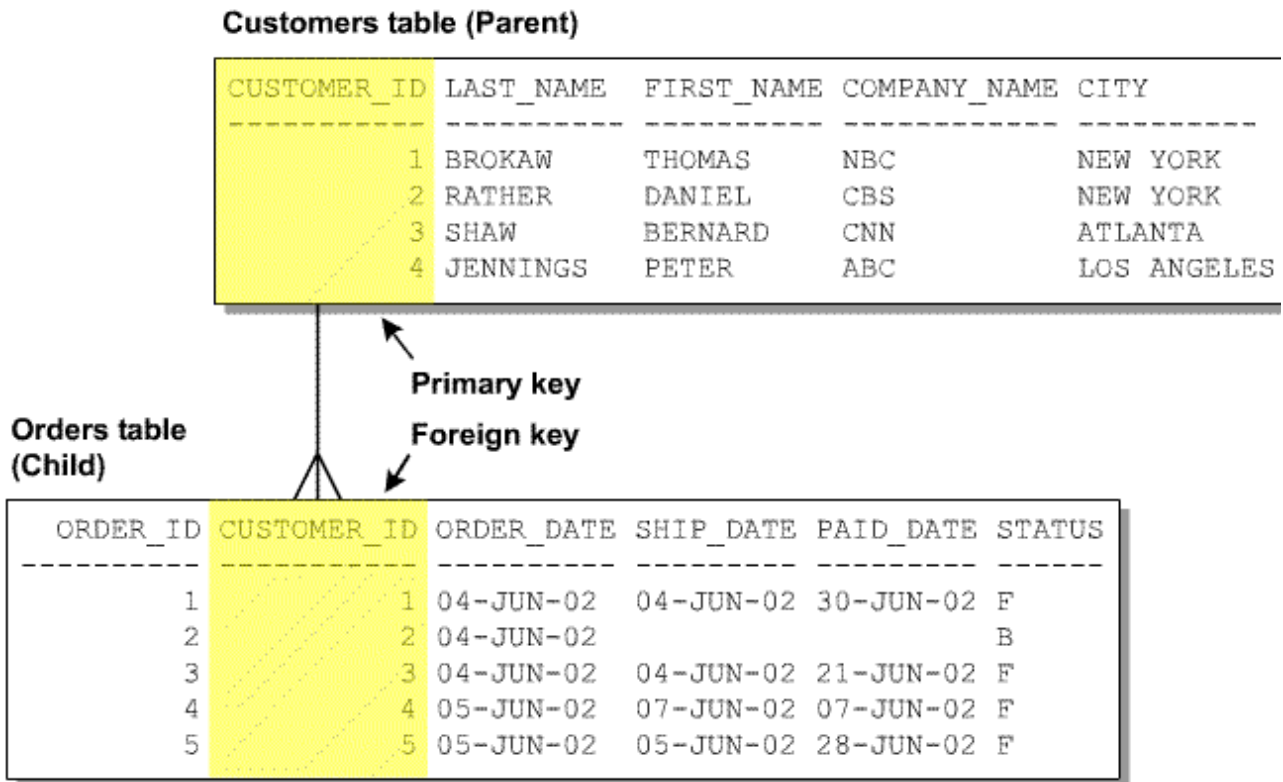


Figure B.2 - Referential constraints¹¹

A referential constraint is described by a referential descriptor which is composed of the following amongst other things.

- A list of the referencing columns
- The referenced table
- A list of the referenced columns
- Different match types and the referential actions.

1.3.3.1 Referential Actions

The ANSI SQL-2003 standard contains the concept of a *referential action*. Sometimes, instead of preventing a data-modification operation that would violate a foreign key reference, you might want the database system to perform another, compensating action that allows the modification and still honours the constraint. For example:

If you delete a row from the customers table that the Orders table references, you could instruct your DBMS to automatically delete all related Orders table rows (i.e., cascade the delete to Order table). That way, you can modify the customers table without violating the constraint.

Figure B.3 – Referential action example

¹¹ Adapted from [Animated Learning: 2002]

- The ANSI standard defines four possible referential actions that apply to deletes from or updates to the referenced table: NO ACTION, CASCADE, SET DEFAULT, and SET NULL.
- The NO ACTION option, which is the ANSI-standard default, prevents the modification if the row is referenced by another row in another table.
- CASCADE allows a delete or update of all matching rows in the referencing table.
- SET DEFAULT lets the delete or update occur but sets all foreign key values in the referencing table to a default value.
- SET NULL allows the delete or update and sets all foreign key values in the referencing table to NULL.

Referential constraints are generally satisfied differently depending on the match type selected.

1.3.3.2 FOREIGN KEY constraints

Foreign key constraints help *join*, establish and synchronise the relationships between tables. There are mainly two types of foreign keys which are: *self-referencing* and the ordinary foreign keys.

1.3.3.2.1 Self-referencing (same table foreign keys)

Normally foreign keys references primary keys in other tables, but at times they reference primary keys in the same table. This type of referencing is called self referencing. For example, in the students table as shown below, each student is associated to a class Rep who is also a student.

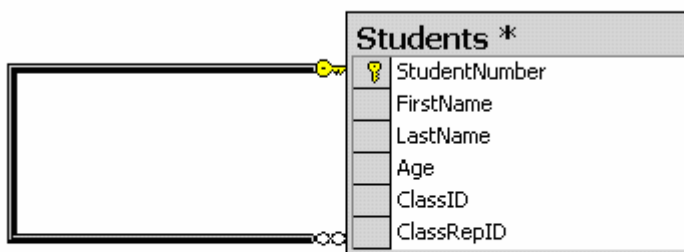


Figure B.4 - Self-referencing foreign key

The other type of referential integrity is based on different table foreign keys or just foreign keys. These are the most commonly used type of foreign keys.

1.3.3.3 Threats to Referential Integrity

1.3.3.3.1 Update threat

This can produce orphans when either the (*PK*) in the parent table or the (*FK*) in the child is updated without any synchronization mechanism. This is where the ON DELETE and ON UPDATE clauses are used with the FOREIGN KEY constraint [Microsoft: 2005].

1.3.3.3.2 Insert threat

This occurs when we allow insertion of records in the child table, with no associated records in the parent table.

1.3.3.3.3 Delete threat

This occurs when we delete records from the parent table and not do anything about the corresponding records in the child table.

1.3.4 User-defined integrity

This refers to specific business rules not covered by the types of integrity. Business rules may pertain to business calculations, for example, one of the implementation was how to convert a percentage mark (for example, 82%) into a grade. This is usually implemented using triggers, assertions and stored procedures. Also of utmost importance is the normalization of tables. The integrity rules will be useless unless your tables are normalized [Microsoft: 2005].

1.3.4.1 Triggers

Triggers are basically database objects that are *attached* to a table, and are only *fired* when an *INSERT*, *UPDATE* or *DELETE* occurs. This means that it specifies a particular action to take place whenever a given event takes place on a particular object. This idea can be diagrammatically presented as shown below.

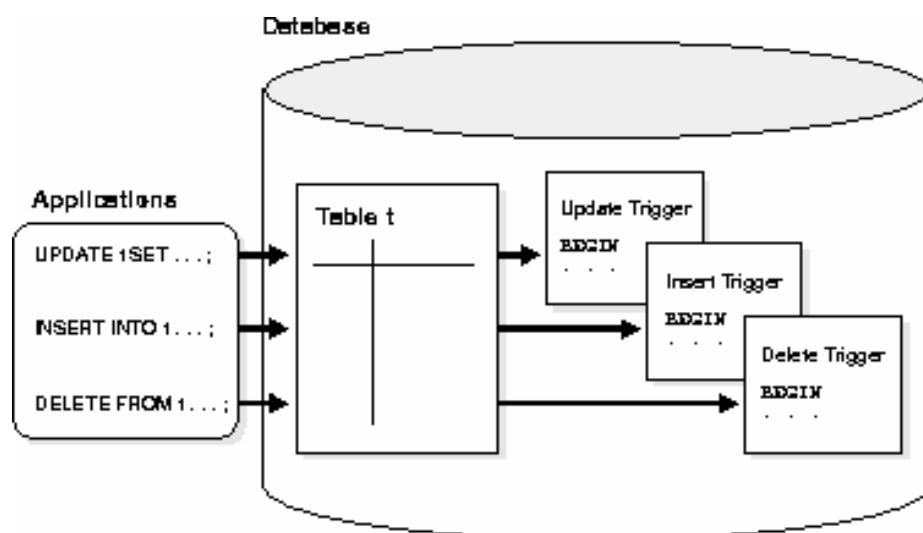


Figure B.5 – Oracle triggers stored in the database separate from their associated tables¹²

1.3.4.1.1 Uses of Triggers

- ❖ Maintaining integrity constraints
- ❖ Auditing of database actions
- ❖ Propagation of database modifications

To design a trigger, one has to specify

- ❖ The event and condition under which the trigger is to be executed, and
- ❖ The action(s) to be performed when the trigger executes

```

Create trigger <name>
  {before/after} <trigger event(s)>
  on <table> [referencing <transition table or variable list>]
  [for each {row | statement}]
  [when <condition>]
  <triggered SQL statement>
    
```

Figure B.6 - SQL 2003 triggers syntax

A trigger is fired if <trigger event(s)> occurred *before/after* an event in a transaction *immediate/deferred*). It is executed if <condition> evaluates to true

The important feature underlying triggers is that the DBMS keeps track of modifications done by a transaction using *transition tables* and use of special variables is made to make the data in the transition tables available to the triggered action.

1.3.4.2 Stored procedures

A Stored procedure is a name group of SQL statements that is previously created, compiled and stored in a database. It is processed as a unit that can be called from another SQL statement. It can accept input parameters and pass values to output parameters. Some of the advantages of using stored procedure over ad hoc queries as highlighted by [Henderson. K: 2002] include:

- ❖ Execution plan retention and reuse – they are beneficial to performance, since when you run a stored procedure for the first time. The query optimiser builds an execution plan so that it does not have to repeat parsing, optimization and other important stages during execution.
- ❖ Query auto parameterization
- ❖ Encapsulation of business rules and policies
- ❖ Application modularization
- ❖ Sharing of application logic between applications
- ❖ Access to database objects that is both secure and uniform
- ❖ Consistent and safe data modification
- ❖ Network bandwidth conservation
- ❖ Support for automatic execution at system start-up

1.3.4.3 Assertions

An assertion is a check constraint. It is a predicate expressing a condition that we want the database to satisfy.

An assertion is described by an assertion descriptor. In addition to the components of every constraint descriptor an assertion descriptor includes a *<search condition>*. An assertion is satisfied if and only if the specified *<search condition>* is not False. However Oracle does not support assertions.

1.4 Advantages of Integrity constraints

[Oracle: 2002], describes some of the advantages of using integrity constraints over other alternatives. These are illustrated below.

- ❖ **Declarative Ease**

They enable you to define integrity constraints using SQL statements. This means that no additional programming is required when you define or alter table. The DBMS will control and manage the functionality of integrity constraints for you.

¹² Adapted from Oracle reference : http://www-rohan.sdsu.edu/doc/oracle/server803/A54643_01/ch15.htm

❖ **Centralized Rules**

Since Integrity constraints are defined for tables and are stored in the data dictionary. There are always enforced no matter what application tries to access the database. Server implementation of business rules ensures that no erroneous data will make it to the database without being noticed. This centralises the maintenance of integrity and eases the amount of application logic.

❖ **Maximum Application Development Productivity**

This works in same manner as code re-factoring. Implementing business rules by an integrity constraint means that if these rules change, the administrator need only change that integrity constraint and all applications will automatically adopt. Unlike implementing it at the application level which means a change in the business rule will need multiple changes in all applications that use that rule.

❖ **Immediate User Feedback**

As information pertaining to each integrity constraint is usually stored in the data dictionary, constraint violations can be detected immediately and the feedback propagated to the user.

❖ **Superior Performance**

The semantics of integrity constraint declarations are clearly defined, and performance optimizations are implemented for each specific declarative rule. The query optimizer can use declarations to learn more about data to improve overall query performance.

❖ **Flexibility for Data Loads and Identification of Integrity Violations**

Even though integrity constraints are there to ensure that your data adhere to predefined steady fast rules, there is an overhead associated with them. This might be costly to performance when performing huge data loads. The good news is that they are flexible enough to be turn off when necessary. Checking of data which violates integrity constraints will then be initiated at a later stage when you have finished loading the data.

Appendix C: Error messages.

1 Summary of unique tests error messages

- I. ORA-00001: unique constraint (PAUL.UK_REGION) violated
- II. **Server: Msg 2627, Level 14, State 2, Line 1**
Violation of UNIQUE KEY constraint 'UK_Region'. Cannot insert duplicate key in object 'region'. The statement has been terminated.
- III. ORA-01400: cannot insert NULL into ("PAUL"."REGION"."REGIONID")
- IV. **Server: Msg 515, Level 16, State 2, Line 1**
Cannot insert the value NULL into column 'regionID', table 'Paulos.dbo.region'; column does not allow nulls. INSERT fails. The statement has been terminated
- V. ORA-02299: cannot validate (PAUL.UK_REGION) - duplicate keys found
- VI. **Server: Msg 1505, Level 16, State 1, Line 1**
CREATE UNIQUE INDEX terminated because a duplicate key was found for index ID 4. Most significant primary key is '2'.
Server: Msg 1750, Level 16, State 1, Line 1
Could not create constraint. See previous errors.
The statement has been terminated.

2 Summary of referential Integrity tests error messages

- I. **Server: Msg 547, Level 16, State 1, Line 1**
INSERT statement conflicted with COLUMN FOREIGN KEY constraint 'FK_Territories_region'. The conflict occurred in database 'Paulos', table 'region', column 'regionID'. The statement has been terminated.
- II. **Server: Msg 5074, Level 16, State 8, Line 1**
The object 'PK_Region' is dependent on column 'regionID'.
Server: Msg 5074, Level 16, State 1, Line 1
The object 'PK_Region' is dependent on column 'regionID'.
Server: Msg 5074, Level 16, State 1, Line 1

The object 'FK_Territories_region' is dependent on column 'regionID'.

Server: Msg 4922, Level 16, State 1, Line 1

ALTER TABLE DROP COLUMN regionID failed because one or more objects access this column

III. **Server: Msg 547, Level 16, State 1, Line 1**

UPDATE statement conflicted with COLUMN SAME TABLE REFERENCE constraint 'FK_Students_Students'. The conflict occurred in database 'Paulos', table 'Students', column 'ClassRepID'. The statement has been terminated

IV. **Server: Msg 1785, Level 16, State 1, Line 1**

Introducing FOREIGN KEY constraint 'FK_Students_Students' on table 'Students' may cause cycles or multiple cascade paths. Specify ON DELETE NO ACTION or ON UPDATE NO ACTION, or modify other FOREIGN KEY constraints.

Server: Msg 1750, Level 16, State 1, Line 1

Could not create constraint. See previous errors

V. ORA-02291: integrity constraint (PAUL.FK_REGION_TERRITORIES) violated - parent key not found

VI. ORA-12992: cannot drop parent key column

3 Summary of decimal data type tests error messages

I. **Server: Msg 8114, Level 16, State 5, Line 1** Error converting data type varchar to numeric.

II. **Server: Msg 1007, Level 15, State 1, Line 3**

The number '211564155656456465565645645645645645454565645645' is out of the range for numeric representation (maximum precision 38).

III. ORA-01722: invalid number

IV. ORA-01400: cannot insert NULL into ("PAUL"."TERRITORIES"."TERRITORYID").

V. ORA-01401: inserted value too large for column

4 Summary of small int tests error messages

- I. **Server: Msg 245, Level 16, State 1, Line 1**
Syntax error converting the varchar value 'test' to a column of data type int.
- II. **Server: Msg 8115, Level 16, State 2, Line 1**
Arithmetic overflow error converting expression to data type int.
The statement has been terminated.
- III. ORA-01722: invalid number
- IV. ORA-01400: cannot insert NULL into
("PAUL"."TERRITORIES"."TERRITORYID")
- V. ORA-01401: inserted value too large for column

5 Summary of float tests error messages

- I. **Server: Msg 8114, Level 16, State 5, Line 1**
Error converting data type varchar to float.
- II. **Server: Msg 168, Level 15, State 1, Line 3**
The floating point value '9.87987567273373E+308' is out of the range of computer representation (8 bytes).
- III. ORA-01722: invalid number
- IV. ORA-01400: cannot insert NULL into
("PAUL"."TERRITORIES"."TERRITORYID").
- V. ORA-01426: numeric overflow

6 Summary of real tests error messages

- I. **Server: Msg 8114, Level 16, State 5, Line 1**
Error converting data type varchar to float
- II. **Server: Msg 1007, Level 15, State 1, Line 3**
The number
'2.5345648646548486484864864684848345484848484212315645' is out of the range for numeric representation (maximum precision 38).
- III. **Server: Msg 1007, Level 15, State 1, Line 3**
The number
'25345648646548486484864864684848345484848484212315645' is out

of the range for numeric representation (maximum precision 38).

- IV. ORA-01722: invalid number
- V. ORA-01400: cannot insert NULL into
("PAUL"."TERRITORIES"."TERRITORYID").
- VI. ORA-01426: numeric overflow

7 Summary of check constraints tests error messages

- I. **Server: Msg 547, Level 16, State 1, Line 1**
INSERT statement conflicted with COLUMN CHECK constraint
'CK_Salary'. The conflict occurred in database 'Paulos', table
'Employee', column 'Salary'.
The statement has been terminated
- II. **Server: Msg 547, Level 16, State 1, Line 1**
INSERT statement conflicted with TABLE CHECK constraint
'CK_Sal_Comm'. The conflict occurred in database 'Paulos', table
'Employee'.
The statement has been terminated.
- III. ORA-02290: check constraint (PAUL.CK_SALARY) violated

Appendix D: SQL standards

1. Vendor lock in

[Wikipedia: 2005] states that this is a situation where a customer becomes so dependent on vendor's products that it will be difficult to switch to another vendor without substantial switching costs. This will act as a barrier to market entry for other products and if it is great enough, a monopoly might even arise. Microsoft software was also cited as carrying a highest level of vendor lock-in, based on its extensive set of proprietary APIs. As a solution to this problem in the 80s and 90s public, *royal free standards* were proposed. But this solution did not seem to work effectively. Ever since the late 90s, Free/Open Source software is being pushed as the way to go.

SQL is an open standard, not owned by any company, thus only ANSI-SQL is considered pure SQL. But, because the spirit of *product differentiation* is very strong among the different vendors in their endeavours to gain more customers, you could find that not only does a product support the standard SQL, but it also offers proprietary extra features, enhancements or extensions, and consequently, dialects are continuing to proliferate.

2. SQL dialects

According to [Kline.K:2001], the constantly evolving nature of the SQL standards has given rise to the number of dialects in the market. This is because the user community of a given database vendor required capabilities before the ANSI had set up the standard for that functionality. In some cases new features are produced by the research and academic communities.

These dialects have introduced procedural commands to support the functionality of a much more complete programming language. However, even if a DBMS conforms to the SQL99 standards, its commands may differ from other DBMSs because the SQL statements may be parsed, compiled, and executed differently, especially if different binding styles are used [Kline.K :2001].

SQL2003 schema Command	SQL2003 class	Oracle 9i	SQL Server 2000
<i>ALTER DATABASE</i>	SQL-schema	SWV	SWV
<i>ALTER DOMAIN</i>	SQL-schema	NS	NS
<i>ALTER FUNCTION</i>	SQL-schema	SWV	SWV

Appendix D: SQL Standards

<i>ALTER METHOD</i>	SQL-schema	NS	NS
<i>ALTER PROCEDURE</i>	SQL-schema	SWV	SWV
<i>ALTER TABLE</i>	SQL-schema	SWV	SWV
<i>ALTER TYPE</i>	SQL-schema	SWV	NS
<i>CREATE DOMAIN</i>	SQL-schema	NS	NS
<i>CREATE FUNCTION</i>	SQL-schema	SWV	SWV
<i>CREATE METHOD</i>	SQL-schema	NS	NS
<i>CREATE PROCEDURE</i>	SQL-schema	S	S
<i>CREATE ROLE</i>	SQL-schema	SWV	NS
<i>CREATE SCHEMA</i>	SQL-schema	SWV	SWV
<i>CREATE TABLE</i>	SQL-schema	SWV	SWV
<i>CREATE TRIGGER</i>	SQL-schema	SWV	SWV
<i>CREATE TYPE</i>	SQL-schema	SWV	NS
<i>CREATE VIEW</i>	SQL-schema	SWV	SWV
<i>DROP DOMAIN</i>	SQL-schema	NS	NS
<i>DROP FUNCTION</i>	SQL-schema	SWV	SWV
<i>DROP METHOD</i>	SQL-schema	SWV	NS
<i>DROP PROCEDURE</i>	SQL-schema	S	S
<i>DROP ROLE</i>	SQL-schema	SWV	NS
<i>DROP TABLE</i>	SQL-schema	SWV	SWV
<i>DROP TYPE</i>	SQL-schema	S	NS
<i>DROP TRIGGER</i>	SQL-schema	SWV	SWV
<i>DROP VIEW</i>	SQL-schema	S	S
<i>GRANT</i>	SQL-schema	SWV	SWV
<i>OPERATORS</i>	SQL-schema	SWV	SWV
<i>REVOKE</i>	SQL-schema	SWV	SWV

Table D.1 - Schema commands¹³

¹³ Adapted from [Kline.K : 2004]

SQL2003 data Command	SQL2003 class	Oracle 9i	SQL Server 2000
<i>ALL/ANY/SOME</i>	SQL-data	S	S
<i>BETWEEN</i>	SQL-data	S	S
<i>CLOSE CURSOR</i>	SQL-data	S	SWV
<i>DECLARE CURSOR</i>	SQL-data	SWV	SWV
<i>DELETE</i>	SQL-data	SWV	SWV
<i>EXCEPT</i>	SQL-data	SWV	NS
<i>EXISTS</i>	SQL-data	S	S
<i>FETCH</i>	SQL-data	SWV	SWV
<i>IN</i>	SQL-data	SWV	S
<i>INSERT</i>	SQL-data	SWV	SWV
<i>INTERSECT</i>	SQL-data	SWV	NS
<i>IS</i>	SQL-data	S	S
<i>JOIN subclause</i>	SQL-data	S	SWV
<i>LIKE</i>	SQL-data	S	SWV
<i>MERGE</i>	SQL-data	S	NS
<i>OPEN</i>	SQL-data	S	S
<i>ORDER BY</i>	SQL-data	SWV	SWV
<i>SELECT</i>	SQL-data	SWV (ANSI joins supported)	SWV (ANSI joins supported)

Table D.2 - SQL-data commands¹⁴

¹⁴ Adapted from [Kline.K: 2004]

Appendix D: SQL Standards

SQL2003 Command	SQL2003 class	Oracle 9i	SQL Server 2000
<i>CONNECT</i>	SQL-connection	S	SWV
<i>DISCONNECT</i>	SQL-connection	SWV	SWV
<i>SET CONNECTION</i>	SQL-connection	NS	SWV
<i>SET CONSTRAINT</i>	SQL-connection	SWV	NS
<i>RETURN</i>	SQL-control	S	S
<i>SET</i>	SQL-session	NS	S
<i>SET CATALOG</i>	SQL-session	NS	NS
<i>SET COLLATION</i>	SQL-session	NS	NS
<i>SET DESCRIPTOR</i>	SQL-session	NS	NS
<i>COMMIT</i>	SQL-transaction	SWV	SWV
<i>RELEASE SAVEPOINT</i>	SQL-transaction	NS	NS
<i>ROLLBACK</i>	SQL-transaction	SWV	SWV
<i>SAVEPOINT</i>	SQL-transaction	S	SWV

Table D.3 - SQL-connection, session and transaction statements¹⁵

Characteristic	Platform	Specification
Identifier size	SQL2003	128 characters
	Oracle	30 bytes (number of characters depends on the character set); database names are limited to 8 bytes
	SQL Server	128 characters (temp tables are limited to 116 characters)
Identifier may contain	SQL2003	Any number, character, or underscore
	Oracle	Any number, character, and the underscore (_), pound (#), and dollar (\$) symbols
	SQL Server	Any number, character, and the underscore (_), at sign (@), pound (#), and dollar (\$) symbols
Identifier must begin with	SQL2003	A letter
	Oracle	A letter
	SQL Server	A letter, underscore (_), at sign (@), or pound (#)
Identifier cannot contain	SQL2003	Spaces or special characters
	Oracle	Spaces, double-quotes ("), or special characters
	SQL Server	Spaces or special characters
Allows quoted	SQL2003	Yes

¹⁵ Adapted from [Kline.K : 2004]

Appendix D: SQL Standards

identifiers		
	Oracle	Yes
	SQL Server	Yes
Quoted identifier symbol	SQL2003	Double-quote (")
	Oracle	Double-quote (")
	SQL Server	Double-quote (") or brackets ([]); brackets are preferred
Identifier may be reserved	SQL2003	No, unless as a quoted identifier
	Oracle	No, unless as a quoted identifier
	SQL Server	No, unless as a quoted identifier
Schema addressing	SQL2003	<i>Catalog.schema.object</i>
	Oracle	Schema.object
	SQL Server	Server.database.schema.object
Identifier must be unique	SQL2003	Yes
	Oracle	Yes
	SQL Server	Yes

Table D.4 - SQL 2003 rules for naming Identifiers¹⁶

¹⁶ Adapted from [Kline.K : 2004]

Appendix E: Tutorial and what is on the CD

On the CD there are the tools that were used for the project, tools like “DB Tools Version 5 for Oracle”, were mainly used as editors for SQL scripts.

On the CD there is also a working draft of the SQL 2003 standards which was used. The overview of the CD contents is given by the picture below.

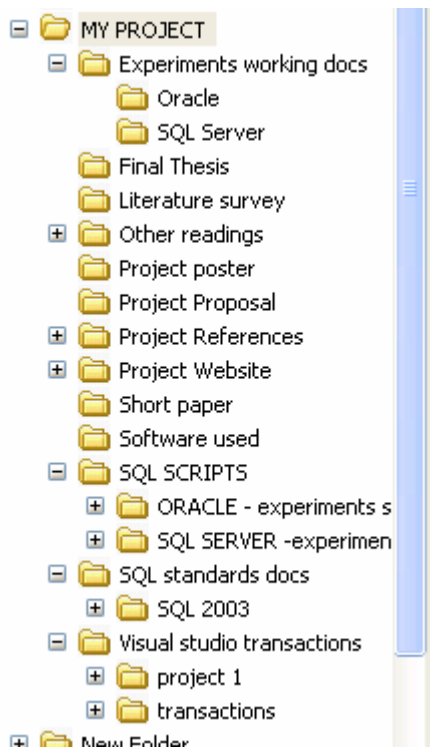


Figure E.0 – CD- contents

1. Performing the tests in SQL Server 2000

To start working with SQL Server 2000 you need to have administrative privileges on the machine.

In addition you need to have OLAP administrative powers.

The simplest way to work with SQL Server is through the Enterprise Manager, which found by following the steps shown below.

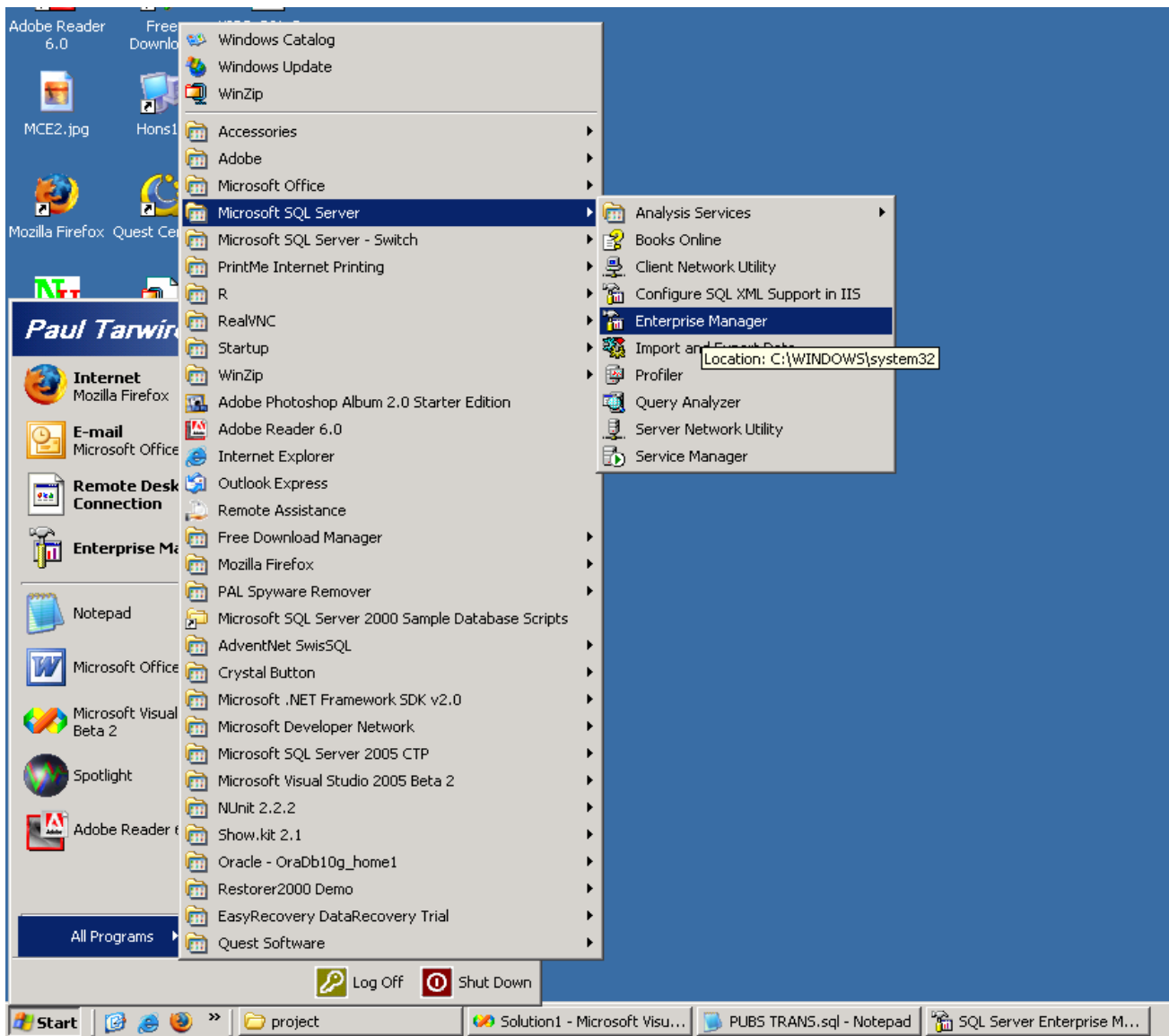


Figure E.1 starting Enterprise Manager.

Expand items under the *Console Root* node until you can access the databases as shown below

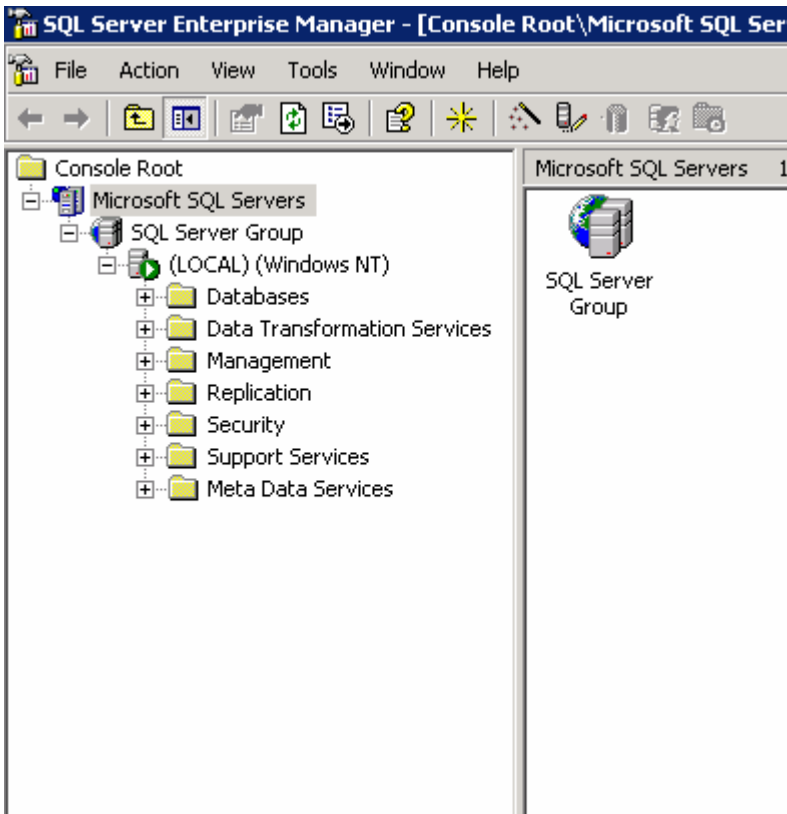


Figure E.2 Console Root node.

From there click on tools -> SQL Query analyzer

You then have to login using the sa password or windows authentication.

After this load and execute the script called Paulos_SQL SERVER.sql which is on the CD. This script will create most of the base tables needed for the experiments.

After that the tests can be carried out by running the different integrity scripts on the CD in any logical order.

1. Performing the tests on Oracle 9i

First of all you have to login to the Oracle enterprise manager using username paul: and password: smirage in the dialogue box shown figure E.3

After you are connected, you have to now choose the SQL*Plus Worksheet which will display as shown in figure E.4.

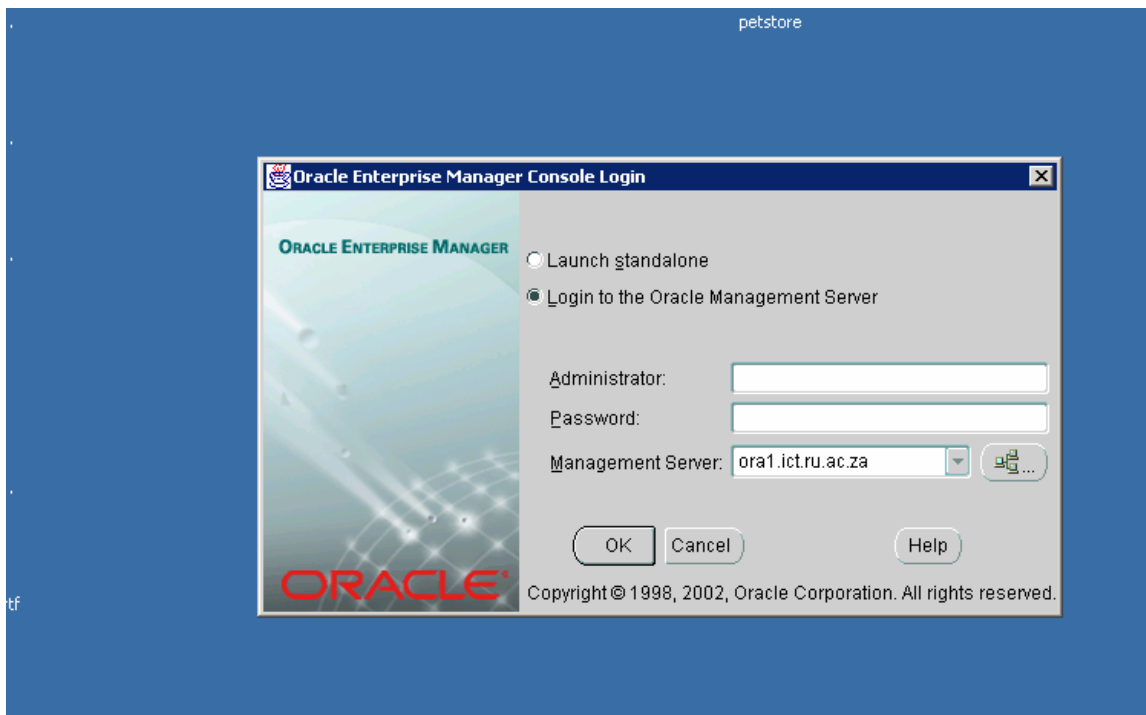


Figure E.3 – Oracle login dialog box

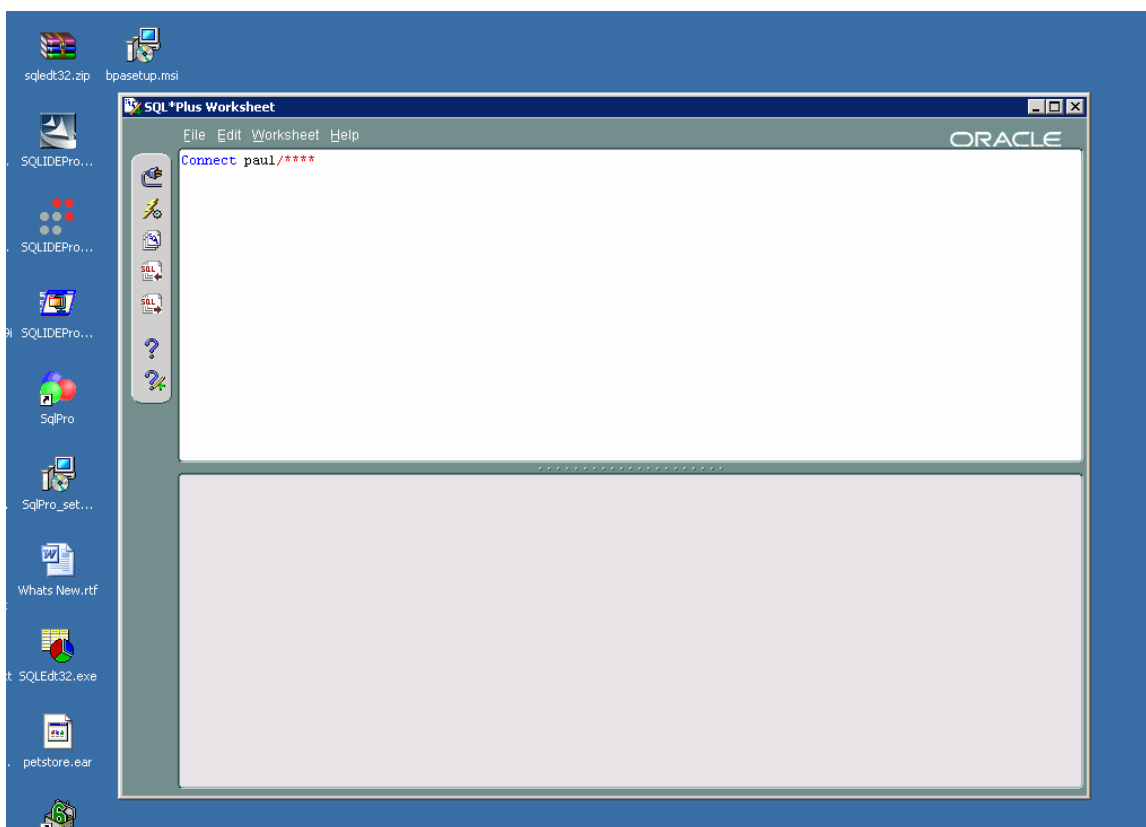


Figure E.4- SQL* Plus Worksheet

After you have connected to SQL*Plus Worksheet, you have load the PAUL_Oracle.SQL script to create the base tables. After that the experiments can be carried out following the structure given in the tests *performed* column of each integrity test in chapter 3.

Appendix F: References

[Akadia Information Technology: 2005]	Akadia Information Technology: 2005, <i>Oracle Tips and Tricks of the Week Part 4</i> http://www.akadia.com/services/ora_important_part_4.html
[Beaulieu A, Mishra S. 2002]	Beaulieu A, Mishra S. 2002, <i>Mastering Oracle SQL</i> , O'Reilly, 2002
[Channel 9 forums: 2005].	Channel 9 forums: 2005, <i>The version of SQL after version 2005</i>
[Coronel C & Rob P. 2002]	Coronel C & Rob P. 2002, <i>Database systems design, implementation and management 5 th Ed</i> , Course Technology , 2002
[Groff and Weinberg: 2004]	Groff-Weinberg, 2004, <i>SQL: The Complete Reference, Second Edition</i> , McGraw-Hill Osborne, 2004
[Gulutzan. P [1]: 2005]	Peter Gulutzan: 2005, <i>Standard SQL: Do IBM, Microsoft, and Oracle support the SQL: 1999 standard? And will they support the SQL: 2003 standard as well?</i> http://www.dbazine.com/db2/db2-disarticles/gulutzan3
[Gulutzan. P [2]: 2005]	Peter Gulutzan: 2005, <i>SQL Naming Conventions</i> http://www.dbazine.com/db2/db2-disarticles/gulutzan5
[Hawthorne R. 2002]	Rob Hawthorne. 2002, <i>The Battle of the RDBMSs: SQL Server 2000 Versus Oracle 9i (Part 1)</i> http://www.informit.com/articles/article.asp?p=27317&rl=1
[Kline K E. 2004]	Kline Kevin. E. 2004, <i>SQL in a Nutshell, 2nd Edition</i> , O'Reilly, 2004.
[Kline K et al: 1999]	Kevin Kline with Daniel Kline, 1999, <i>SQL IN A NUTSHELL A Desktop Quick Reference</i>

Appendix F: References

[Kline.K :2001],	Kline.K :2001, <i>SQL IN A NUTSHELL A Desktop Quick Reference, 2001</i>
[MCAD, 2005]	MCAD Article Management. 2005, <i>Oracle Recognized As Leader In Worldwide Database Market Share, According To Leading Research Firm</i> http://www10.mcadcafe.com/nbc/articles/index.php?section=CorpNews&articleid=169782
[Microsoft [1]: 2001]	REDMOND, Wash., June 28, 2001, <i>Building the Billion Dollar Database: Microsoft SQL Server Climbs to New Heights</i>
[Microsoft: 2005]	Microsoft: 2005, <i>Migrating Oracle Databases to SQL Server 2000</i> http://www.microsoft.com/technet/prodtechnol/sql/2000/deploy/sqloracle.msp
[Mimer Developer page: 2005].	Mimer Developer page: 2005, <i>Mimer SQL-2003 Validator</i> http://developer.mimer.com/validator/parser200x/index.tml
[Mullins C S. 2002]	Mullins C S. 2002, <i>Database Administration: The Complete Guide to Practices and Procedures</i> , Addison Wesley, 2002
[Mullins. C.S [1]: 2005]	Mullins. C.S [1]: 2005, <i>The Data Administration Newsletter (TDAN.com): The Database Report - October 2005 Reporting on Database Industry News from July through September, 2005</i> http://www.tdan.com/dbreport_issue34.htm
[Oracle [1]: 2005]	Oracle [1]: 2005, <i>Oracle Company Profile Page</i> http://company.monsterindia.com/oraclein/
[Oracle [2]: 2005]	Oracle [2]: 2005, <i>Oracle FAQ Page</i> http://www.orafaq.com/faqora.htm
[Rosenzweig B, Silvestrova E. 2003]	Rosenzweig B, Silvestrova E. 2003, <i>Oracle® PL/SQL by</i>

Appendix F: References

	<i>Example</i> , Third Edition, Prentice Hall PTR, 2003
[Stakemire T.S, 2000].	Stakemire T.S, 2000, <i>An evaluation of the integrity of MySQL and Oracle database management systems</i> http://hobbes.ict.ru.ac.za/csae/research/dbgroup/final_writeup.doc
[Troels A: 2005]	Troels A: 2005, <i>Comparison of different SQL implementations</i> http://troels.arvin.dk/db/rdbms/
[Türker, Gertz: 2000]	Türker, Gertz: 2000, <i>Semantic Integrity Support in SQL-99 and Commercial (Object-)Relational Database Management Systems</i> , VLDB Journal 10(4): 241-269, 2000
[Webopedia [1]: 2005]	Webopedia: 2005, <i>Data integrity</i> http://www.pcwebopaedia.com/TERM/d/data_integrity.html
[Wikipedia [1]: 2005]	Wikipedia [1]: 2005, <i>Microsoft SQL Sever</i> http://en.wikipedia.org/wiki/Microsoft_SQL_Server
[Wikipedia [2]: 2005]	Wikipedia : 2005, <i>PL/SQL</i> http://en.wikipedia.org/wiki/PL/SQL