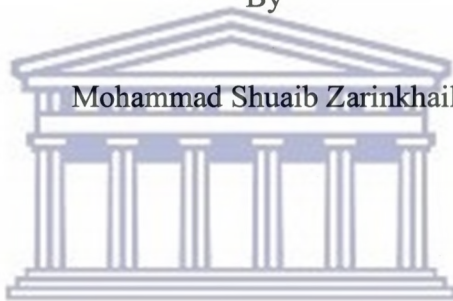


Static MySQL Error Checking

By

Mohammad Shuaib Zarinkhail



UNIVERSITY of the

WESTERN CAPE
A thesis submitted in fulfillment
of the requirements for the degree of

Magister Scientiae

Department of Computer Science

University of the Western Cape

July 2010

Supervisor: James Connan

Contents

Contents	i
List of Figures	v
List of Tables	vi
Glossary	viii
Keywords	x
Database	x
Abstract	xi
Declaration	xii
Acknowledgments.....	xiii
Chapter 1: Introduction	1
1.1 Background	1
1.2 Problem Statement	2
1.3 Research Objectives	3
1.4 Research Questions	4
1.5 Thesis Outline	4
Chapter 2: Database Systems.....	7
2.1 Data base System Cycle.....	7
2.1.1. Users.....	8
2.1.2. Database	9
2.1.3. The DBMS and Application Programs	9
2.2 The DBMS (In General).....	10
2.2.1. DBMS Functions.....	11
2.2.2. DBMS Types (by number of users).....	15
2.2.3. DBMS Types (by usage).....	16
2.3 RDBMSs	17
2.3.1. RDBMS Structure	17
2.4 Summary	19
Chapter 3: MySQL	20
3.1 Query Languages.....	21
3.2 SQL	21
3.3 SQL History	22
3.4 MySQL History.....	23
3.5 MySQL Functions	24
3.5.1. Control Flow Functions.....	24
3.5.2. String Comparison Functions.....	26
3.5.3. Case Sensitive String Comparison Functions	27
3.5.4. Using INSTR() for String Comparison.....	27
3.5.5. Using LIKE for String Comparison.....	29
3.5.6. Using LOCATE() for String Comparison.....	31
3.5.7. Using REPLACE() for String Comparison.....	34

3.5.8.	Using STRCMP() for String Comparison.....	36
3.5.9.	Using SUBSTRING_INDEX() for String Comparison.....	37
3.5.10.	Case Sensitivity in Regular Expressions.....	39
3.5.11.	Case Sensitivity in the Command Line.....	41
3.6	Summary	42
Chapter 4:	SQL Errors and Error Checkers	43
4.1	SQL Errors	43
4.1.1.	SQL Semantic Errors	44
4.1.2.	SQL Syntax Errors	44
4.1.3.	SQL Error Checking.....	45
4.1.4.	SQL Static Error Checking	45
4.1.5.	Semantic Integrity Support in ORDBMS.....	47
4.2	SQL Error Checkers	48
4.2.1.	eSQL.....	48
4.2.2.	AsseSQL.....	49
4.2.3.	SQLator	50
4.3	SQL Teaching and Error Checking Tools.....	52
4.3.1.	SQL-Tutor	52
4.3.2.	Acharya	53
4.3.3.	WinRDBI—A Relational Database Educational Tool.....	55
4.4	Other Error Checkers	58
4.4.1.	ESC/Java	58
4.4.2.	Lint	59
4.4.3.	LCLint.....	59
4.4.4.	Spec#.....	59
4.4.5.	ESC/Haskell.....	60
4.5	Summary	60
Chapter 5:	Common MySQL Errors	62
5.1	Errors-Case Sensitivity.....	62
5.2	Case Sensitivity Errors in Searches.....	63
5.3	Case Sensitivity Errors in User-defined Variable Names	64
5.4	Errors—MySQL Modes.....	66
5.4.1.	The ANSI_QUOTES Mode	67
5.4.2.	The IGNORE_SPACE Mode.....	68
5.4.3.	Other Modes for MySQL	70
5.4.4.	Combination of Different MySQL Modes Simultaneously	73
5.5	MySQL Error Classification	78
5.5.1.	Errors—SQL-DDL.....	78
5.5.2.	Errors—Database Definition.....	79
5.5.3.	CREATE DATABASE—using IF NOT EXISTS	79
5.5.4.	ALTER DATABASE ...—using IF EXISTS.....	81
5.5.5.	DROP DATABASE ...—using IF EXISTS	81
5.5.6.	Errors—Table Definition	83
5.5.7.	Errors—CREATE TABLE	83
5.5.8.	Errors—CREATE TABLE ... SELECT	84

5.5.9.	Errors—ALTER TABLE	85
5.5.10.	Errors—DROP TABLE	88
5.5.11.	Errors—SQL-DML	91
5.5.12.	Errors—INSERT Data.....	91
5.5.13.	Errors—INSERT NULL Values.....	92
5.5.14.	Errors—IGNORE used with INSERT	92
5.5.15.	Errors—Retrieve Data.....	93
5.5.16.	Errors—Subqueries	93
5.5.17.	Errors—Temporary Tables	96
5.5.18.	Errors—UPDATE Data	97
5.5.19.	Errors—IGNORE used with UPDATE	98
5.5.20.	Errors—DELETE Data.....	98
5.5.21.	Errors—IGNORE used with DELETE	99
5.6	Errors—Transaction Control and Locking Tables	102
5.6.1.	Errors—SAVEPOINTS	103
5.6.2.	Errors—LOCK TABLES	103
5.7	Errors—Using NULL Values.....	106
5.8	Summary	108
Chapter 6: Research Methodology.....		110
6.1	Evaluating Students' MySQL Code.....	111
6.1.1.	Students' Group Project.....	111
6.1.2.	Our Database - the 'HOSPITAL'	112
6.1.3.	Using the 'HOSPITAL' Database.....	113
6.2	Data Collection (What, How, Who).....	113
6.2.1.	The Collected Data.....	113
6.2.2.	The Collection of the Data	114
6.2.3.	The Collector of the Data	114
6.3	Summary	115
Chapter 7: Results.....		116
7.1	MySQL Error Types—Syntactic, Semantic and Logic.....	117
7.2	Student Errors—SQL-DDL.....	118
7.3	Student Errors—SQL-DML.....	122
7.4	Student Errors—Transaction Control and Locking Tables.....	129
7.5	Student Errors—Total Number of Errors in All Cases	132
7.6	Analysis of Results.....	134
7.7	Theoretical Analysis.....	134
7.8	Analysis of the Exact Results.....	135
7.9	Summary	137
Chapter 8: Conclusion and Future Work.....		138
8.1	Static Error Checker for SQL—Offline	139
8.2	Static Error Checking of SQL - Online.....	140
8.3	Concluding Notes.....	141
Appendix A		142
A-1	The 'HOSPITAL' database code in MySQL syntax.....	142
A-1.1.	Database definition commands	142

A-1.2	Table definition commands.....	142
A-1.4.	Relationship commands	145
A-1.4.	Data entry commands.....	147
Appendix B	151
B-1.	Sample MySQL Queries	151
Appendix C	153
C-1.	MySQL Versions.....	153
Appendix D	155
D-1	Raw Data	155
References	159



UNIVERSITY *of the*
WESTERN CAPE

List of Figures

Figure 2-1	Database System Cycle.....	7
Figure 2-2	RDBMS Structure.....	18
Figure 4-1	Trigger Execution and Constraint Checking	47
Figure 4-2	Architecture of Acharya.....	53
Figure 4-3	Acharya Interface Shot on Problem Solving	55
Figure 4-4	WinRDBI Architecture	56
Figure 4-5	WinRDBI User Interface	58
Figure 7-1	Analysis of Student Errors in create, update and delete Database and Table Commands	119
Figure 7-2	Analysis of Student Errors in alter Table Command.....	121
Figure 7-3	Analysis of Student Errors in Table and View related Commands	122
Figure 7-4	Analysis of Student Errors in data entry Commands.....	124
Figure 7-5	Analysis of Student Errors in data retrieve Commands – Part 1	125
Figure 7-6	Analysis of Student Errors in data retrieve Commands – Part 2	126
Figure 7-7	Analysis of Student Errors in data retrieve Commands – Part 3	127
Figure 7-8	Analysis of Student Errors in data retrieve, update, delete and truncate Commands.....	128
Figure 7-9	Analysis of Student Errors in transaction and rollback Commands	130
Figure 7-10	Analysis of Student Errors in rollback, savepoint	
	and rollback to savepoint Commands.....	131
Figure 7-11	Analysis of Student Errors in all Commands.....	133

List of Tables

Table 2-1	DBMS Examples and their Provider Names.....	11
Table 4-1	Percentages of agreed responses to Statements in the Online Test Evaluation Questionnaire	50
Table 5-1	MySQL Modes and their supported DBMS Versions	71
Table 5-2	The structure of 'tblOne' Table.....	86
Table 5-3	MySQL Data Definition Errors (SQL-DDL).....	90
Table 5-4	MySQL Data Manipulation Errors (SQL-DML)	101
Table 5-5	MySQL Transaction Control Errors (SQL-TCL)	106
Table 7-1	Percentages of Student Errors in create, update and delete Database and Table Commands.....	119
Table 7-2	Percentages of Student Errors in alter Table Command	120
Table 7-3	Percentages of Student Errors in Table and View related Commands ..	121
Table 7-4	Percentages of Student Errors in data entry Commands	123
Table 7-5	Percentages of Student Errors in data retrieve Commands – Part 1.....	124
Table 7-6	Percentages of Student Errors in data retrieve Commands – Part 2.....	126
Table 7-7	Percentages of Student Errors in data retrieve Commands – Part 3.....	127
Table 7-8	Percentages of Student Errors in data retrieve, update, delete and truncate Commands.....	128
Table 7-9	Percentages of Student errors in transaction and rollback Commands ..	130
Table 7-10	Percentages of Student Errors in rollback, savepoint and rollback to savepoint Commands	131
Table 7-11	Percentages of Student Errors Totals in all Commands.....	133
Table C-1	MySQL versions and features for different series	154
Table D-1	Student Errors in Data Definition Commands (SQL-DDL).....	156
Table D-2	Student Errors in Data Manipulation Commands (SQL-DML).....	157
Table D-3	Student Errors in Transaction Control Commands (SQL-TCL)	158
Table D-4	Student Error Totals in all commands.....	158

Glossary

AACE	Association for the Advancement of Computing in Education
Acharya	A tool for teaching SQL
ACID	Atomicity, Consistency, Isolation and Durability
ANSI	American National Standards Institute
API	Application Programming Interface
ASP	Active Server Page
AsseSQL	An online tool to check SQL
C	'C' programming language
C#	'C Sharp' programming language
CLI	Call Level Interface
DB2	Database 2
DBA	Database Administrator
DBMS	Database Management System
DML	Data Manipulation Language
DRC	Domain Relational Calculus
ERD	Entity Relationship Diagram
ESC/Haskell	Extended Static Checker for Haskell programming language
ESC/Java	Extended Static Checker for JAVA
eSQL	A tool for teaching SQL
GUI	Graphical User Interface
HTML	HyperText Markup Language
IDE	Interface Design Enhancement

IIS	Internet Information Server
ITS	Intelligent Tutoring System
LCLint	A tool for program development
Lint	A tool for checking C program code
mSQL	Mini SQL
ORDBMS	Object Relational Database Management System
RDBI	Relational Database Interface
RDBMS	Relational Database Management System
Spec#	A tool for checking C# program code
SQL	Structured Query Language
SQLator	An online workbench for teaching SQL
SQL DCL	SQL Data Control Language
SQL DDL	SQL Data Definition Language
SQL DML	SQL Data Manipulating Language
SQL-Tutor	A knowledge based system which is used to support learning SQL
Tcl	Tool command language
TRC	Tuple Relational Calculus
UoD	the Universe of Discourse
UWC	University of the Western Cape
VB	Visual Basic
WinRDBI	An educational tool for SQL

Keywords

Database

Structured Query Language (SQL)

MySQL

Static Checking

Coding Tools

Software Maintenance

Program Conventions

Error Checker Tools



Abstract

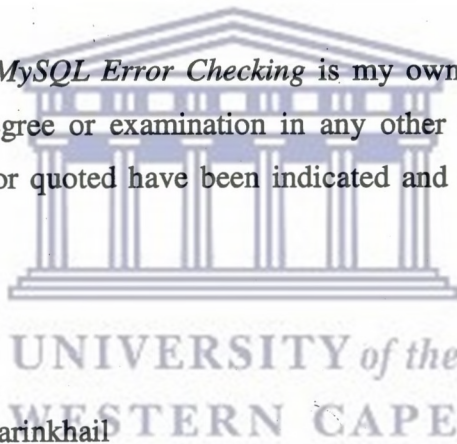
Coders of databases repeatedly face the problem of checking their Structured Query Language (SQL) code. Instructors face the difficulty of checking student projects and lab assignments in database courses.

We collect and categorize common MySQL programming errors into three groups: data definition errors, data manipulation errors, and transaction control errors. We build these into a comprehensive list of MySQL errors, which novices are inclined make during database programming. We collected our list of common MySQL errors both from the technical literature and directly by noting errors made in assignments handed in by students. In the results section of this research, we check and summarize occurrences of these errors based on three characteristics as semantics, syntax, and logic.

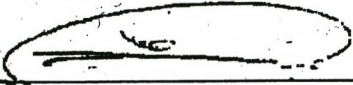
These data form the basis of a future *static MySQL checker* that will eventually assist database coders to correct their code automatically. These errors also form a useful checklist to guide students away from the mistakes that they are prone to make.

Declaration

I declare that *Static MySQL Error Checking* is my own work, that it has not been submitted for any degree or examination in any other university, and that all the sources I have used or quoted have been indicated and acknowledged by complete references.



Mohammad Shuaib Zarinkhail

Signed:  Date: October 2010

Acknowledgments

I would like to thank my supervisor and all the instructors and professors at the UWC who helped me to complete my research. This was difficult for the team at UWC to supervise our program from a very great distance. I am happy and grateful to Mrs Verna Connan, the postgraduate administrator at the UWC, for her kind and active guidance during our program.

I would like to thank my family members, especially my kind father, for their support. They let me complete this work. My father is really the greatest instructor in my life.

My special thanks to my colleagues here in Kabul University, who gave me enough time to complete my thesis.



Chapter 1

Introduction

Computer Science is a wide and important field of study. Almost all aspects of human life are highly influenced by computers and computerized systems [44]. Databases and their usage are important [36, p. 2]. Relational databases are widely used and are a crucial component of business and scientific systems [37] [45, p.133]. Databases are used in banking, airlines, credit card transactions, finance, human resource management, sales, and manufacturing. Databases are also used in education by universities, research academies, schools, and telecommunications departments. Massive search engines rely on rapid database access, i.e. Google, MSN, Yahoo, and so on. Electronic mail, registering users, and many other activities regarding email facilities are stated and used through databases.

1.1 Background

A database is a computerized collection of logically related records of persistent data and their definitions [14, p. 11]. According to a survey, there were ten million active databases in the world in 2004 [27]. Databases could be created by Database Management Systems (DBMSs) in large, medium and small sizes. Structured Query Language (SQL) is the most common query language used by Relational Database Management Systems (RDBMSs) [45, p. 135]. SQL is mostly taught in educational areas like schools, universities and academies. SQL programs are prone to errors. These errors could be semantic errors, syntax errors or logic errors.

Previous research in this field includes “Static error checking of C applications” [3], “Extended static checking for Java” [21], “How to write system-specific, static checkers in Metal” [10]. Some errors only manifest themselves at run time under specific conditions, or when the system reaches certain states. These errors are difficult to detect and are normally either semantic or logic errors. One technique for detecting run-time errors is to extract rules from commonly made errors. These rules can then be applied to code segments in order to determine whether they meet some rule-specific constraints [10]. SQL, like other coding languages, should have related rules. According to those rules, it should be possible to recognize and highlight errors for the user statically.

Most of DBMSs use SQL for their functionalities. MySQL is a DBMS that uses SQL commands. An early version of MySQL was developed in May 1995 [38, p. 6]. Students in database classes do MySQL programming. They also make MySQL queries. The students use generated queries to retrieve data from databases, and to insert and update data to database tables. Class work includes developing databases, using databases, inserting data into databases, updating data within databases, deleting data from databases, and finally retrieving data from databases. During class work, their programs may have errors. The errors that students make can either be semantic, syntax or logic errors.

1.2 Problem Statement

Teachers and instructors of database classes face problems when checking students’ work. The MySQL code generated by students may have errors. The error codes in students’ projects do not run on computers. Therefore, the MySQL code developed by students needs static checking.

During database course work, students often implement small and medium projects in MySQL. Teachers and instructors do not easily detect the mentioned kinds of SQL

errors, especially semantic errors. Finding these errors can also be very time consuming.

Another significant problem is that different students or student groups may submit the same copy of code for their assignments. When this happens, it reveals student dishonesty in classes. Kabul University, where we conducted this research, has large classes. The average number of students per class is 75. Finding code similarities in such big classes is a difficult and almost impossible task, which could be done by statically checking MySQL code.

1.3 Research Objectives

First, a general study of databases is needed where we explain the general terms including Database Management Systems (DBMSs), database files and database users. Then we do an in-depth study of SQL and MySQL where MySQL types, functionalities, functions and its proneness to errors are explained.

Finding common MySQL errors is another important part in this research. This project should come out with a comprehensive list of common MySQL errors. Similarly, common MySQL errors that database students make during class assignments are included in this document. Also, the errors are grouped and statistically analyzed.

This research focuses on methods, which can be used for designing a software-tool. This tool will find the common errors that students make while designing and implementing databases in MySQL. The software will analyze the code and highlight the errors to the users. Most of the users of this tool will be instructors and teachers in the educational and academic areas. In conclusion, this research will provide some background on error checking, investigate its relation to SQL and discuss the results of applying error checking to MySQL.

An outcome of this research is a categorized list of common errors that students make during MySQL programming. Also, methods for error checking which can be implemented in a software tool are proposed. This will enable users to quickly and easily mark projects and rectify errors. This project contributes to the field of databases by both providing a comprehensive list of common errors as well as providing methods for designing a tool to detect such errors.

1.4 **Research Questions**

The following are two main questions that state the research problem:

1. What errors do students make when programming MySQL in a database course?
2. How can these errors be classified?

Similarly, we have two supporting questions to the main questions as:

1. Do any patterns emerge?
2. What can be learned from this analysis?

1.5 **Thesis Outline**

In Chapter 1, an introduction to the research field is given and the problem and research questions are explained. This chapter includes a short background of the field of study and shows its relationship to the topic of this research. Problem statement, research objectives and research questions are included in this chapter. Chapter 1 also describes the structure of the thesis including each chapter and each appendix individually. A glance to the chapter topics and purpose of each chapter is explained here.

We give an overview of database management systems and include a broad study of the database field in Chapter 2. Individual topics including databases, database management systems, their types and their functionalities are added to this chapter. Chapter 2 is one of the introductory chapters in this thesis and its existence helped us to complete the chapters following it where some of them focus on the theoretical analysis of MySQL errors.

Chapter 3 explains SQL in general and MySQL in particular and starts by explaining query languages. This chapter includes a deeper study of SQL history and MySQL history compared to Chapter 2. MySQL functions, regular expressions, command line commands and their relation to error occurrences are explained in this chapter. Chapter 3 focuses on a specific DBMS and is used as a vehicle to explain specific points concerning MySQL.

Chapter 4 explains the common SQL errors and their types. It surveys some of the existing error checkers and discusses some of the SQL teaching tools that are employed. Semantic integrity support in object relational database management systems is another topic of Chapter 4. This chapter is added to our document as an advanced part of our work. The main reason for having Chapter 4 is to explain two different objectives of our research, MySQL common errors as well as error checkers. The error checkers are used to highlight, detect or prevent programming errors occurring in source code.

Chapter 5 focuses on common MySQL errors. This is a big chapter. The error types, which are explained in this chapter, depend on case sensitivity issues in searches and in user-defined variable names. MySQL modes and chances for error occurrence are discussed there. MySQL error classification, which is based on the types of commands, is described in this chapter. There is a big propensity for errors to occur in MySQL general commands. Chapter 5 includes many topics about specific errors.

Chapter 6 includes our methodology for achieving the research goals and answering the research questions. This chapter explains what we have done in this project. Chapter 6 explains the breakdown of our work in three stages: study of database field and experiments, evaluating of students' generated code to statically count errors in the code, and the data collection explaining the (what, how, and who) terms.

Chapter 7 describes the results of our work. In this chapter we have analyzed and categorized the students' errors. The listed errors in this chapter are grouped and are shown in tables. Each table shows around 10 errors and a histogram is appended for each table. Chapter 7 is a compilation of the results of this research.

Finally, in Chapter 8 we give our conclusions, and discuss the future work in this field. We also propose a couple of methods for designing an error checker for SQL. As stated, the error checker can be designated for a single user and be used offline or it can be multi-user tool. The multi-user version may or may not be used online.

There are four appendixes. Appendix A includes the sample database code named the 'HOSPITAL'. This code is based on the MySQL syntax and students had to develop this database *ab initio*. Appendix A includes four groups of commands for the sample database 'HOSPITAL': i.e. database definition, table definition, relationships, and data entry commands. Appendix B includes the list of sample queries that are requested from students. The students had written those queries against the sample database called 'HOSPITAL' explained in Appendix A. In Appendix C the features of different versions of MySQL are contrasted. Appendix D tables student errors.

Chapter 2

Database Systems

The database systems chapter starts with some theory taken from C.J. Date's book titled "*An Introduction to Database Systems*" published in 2004 [14]. The theory specifies a cycle in which database system elements are divided into four groups: users, databases, application programs and DBMS. Each element of the database system cycle is explained separately. The relational database management system (RDBMS) is also discussed as a sub part of DBMS. The chapter concludes with a brief summary of the literature review and the chapter topics.

2.1 Data base System Cycle

A database system consists of four major parts: database, application programs, DBMS and users [14, p. 7]. We can call them the database system cycle. This cycle is shown below. Sometimes, application programs and DBMS are shown in one box and treated as a single entity in the database system cycle.

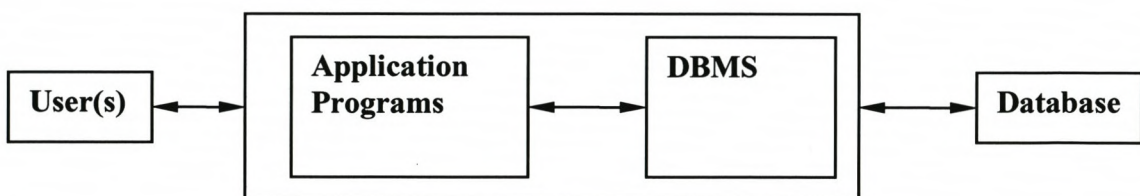


Figure 2-1 Database System Cycle

The above chart shows a database system cycle. According to the chart, DBMS is a software layer between databases and database users. Users can create, modify, edit or delete databases by DBMSs.

2.1.1. Users

Database users are categorized in four different types [45, p. 13]. They are differentiated by the way they expect to interact with the system.

- Application programmers are people who have enough technical knowledge of databases. Application programmers write application programs within a database internal structure (metadata). There are special types of programming languages, called fourth-generation languages. These languages combine necessary control structures like: *for* loops, *while* loops, and *if-then-else* statements, with the DML statements [45, p. 14]. Fourth-generation languages often include special features to facilitate the generation of forms and the display of data on the screen. A majority of commercial database systems use a fourth-generation language.
- Sophisticated users interact with the system and form their requests in a database query language like SQL. This category of users does not write database programs, they simply formulate queries in SQL. Sophisticated users submit each query to a query processor. The query processor then breaks down the DML statements into instructions that the storage manager understands. Data analysts, who submit queries to explore data in a system, belong to this category of users.
- Specialized users are also sophisticated users. This category writes specialized database applications while they are not in a traditional data-processing framework. Computer-aided design systems, knowledge base and expert systems, systems that store data with complex data types i.e. graphic data, audio data, and environment-modeling systems are examples of such specialized database applications.

- Naive users are end users with less knowledge of databases. They just use the system by invoking application programs or objects in a database system. Application programmers previously write those application programs and database objects.

2.1.2. Database

A database is an organized collection of logically related data, data definitions, relationships, indexes, views and other parts of computerized data. Basically, databases are created from relations. A relation is a two-dimensional data table consisting of records and fields. Each field in a relation has a unique name and a specified data type and domain. Each record in a relation contains data about only one entity instance. The order of fields and the order of records within a relation are irrelevant. Each cell can be NULL or can have only one value.

2.1.3. The DBMS and Application Programs

A DBMS is a collection of interrelated data and a set of programs to access the data [45, p. 1]. DBMS is software by which a database can be created; data to the database can be inserted, updated, retrieved, and deleted. Some DBMSs use application programs for these purposes. This topic is explained in more details.

Application programs are an important component in database management systems. DBMS developer companies create application programs and include them in their software products. These programs are treated as built-in objects in DBMSs. In the same way, a category of database users called application programmers are also able to write application programs to simplify database usage.

2.2 The DBMS (In General)

The DBMS is an important part in a database system [36, p. 2]. A DBMS is a layer of software between users and a database, which creates databases and stores, updates, deletes, and retrieves data in databases [14, p. 9]. Synonyms for DBMS are 'database manager', and 'database server'. DBMSs handle all requests for accessing data in a database. Adding, removing, and updating tables in a database, retrieving data from a database, and many other functions are implemented by DBMSs. A general function of the DBMS is 'the shielding of database users from hardware level details'. DBMSs provide users with a perception of the database that is elevated somewhat above the hardware level. It also supports user operations that are expressed in terms of that higher-level perception.

DBMSs prepare facilities for users in a database system to create and manipulate databases. Inserting data into a database is also handled by DBMSs. Deleting data without side effects, updating data in databases and retrieving data from databases are other available operations implemented by DBMSs. Many software companies produce DBMSs. The following lists some of the DBMSs with their provider company names.

DBMS Name	Provider Name
DB2	IBM Corporation
FileMaker	FileMaker Incorporation
Firebird	Firebird Project
Ingress	INGRESS Corporation
Microsoft Access	Microsoft Corporation
Microsoft SQL Server	Microsoft Corporation
MySQL Server	SUN Microsystems
Oracle	Oracle Corporation
PostgreSQL	PostgreSQL Global Development Group
SQLite	Dr Richard Hipp
Sybase Adaptive Server Enterprise	Sybase Corporation

Table 2-1 DBMS Examples and their Provider Names

Many DBMSs with different facilities and functionalities exist.

2.2.1 DBMS Functions

DBMSs implemented various functions. These functions generalize the usage of the DBMSs and facilitate their implementation. Examples of these functions are: data dictionary management, data storage management, optimization and execution, data transformation and presentation, security management, multi-user access control, backup and recovery management, data integrity management, database access language and Application Programming Interfaces (APIs), database communication interfaces, and performance.

1. **Data Dictionary Management:** DBMSs provide a data dictionary function. The data dictionary part is a database by itself. It contains data about data, also called 'metadata' or 'descriptors'. The definitions of the data elements and their relationships called metadata need to be stored in the data dictionary. DBMSs do this and manage the data dictionary. Examples of data dictionary management functions implemented by a DBMS are the immediately recording of any changes in a database structure or data abstraction and removing structural and data dependency from the system.
2. **Data Storage Management:** DBMSs accept data definitions in source and convert them to the appropriate object form. Therefore, a DBMS must include a DDL processor or a DDL compiler for each version of the data definition language (DDL). DBMSs must also understand and analyze the DDL statements, for example, to find an index through a database. DBMSs create complex structures through which data storage is performed. Regarding this functionality, users do not need to define and

program the physical data characteristics. A modern DBMS separately stores data and its related forms, such as screen definitions, report definitions, data validation rules, procedural codes, structures for video/picture formats.

3. **Data Manipulation:** DBMSs accept requests to retrieve, update, or delete existing data in a database or add new data to a database. Therefore, a DBMS must include a DML processor or a DML compiler to support the data manipulation language (DML). The DML processor prepares accessibility for implementing DML commands in a database.

DML commands are categorized as planned or unplanned requests:

- a. The physical structure of the database should be designed by the Database Administrator (DBA) to expect planned requests. Therefore, planned requests always guarantee good performance and come out with a complete result.
- b. An unplanned request is like an ad-hoc query, for which the need was unknown before. Unplanned requests are requested on demand. In such cases, the basic design of a database may or may not suit that kind of request. Unlike a planned request, the performance of an unplanned request in a database is not guaranteed.

Generally, planned requests are characteristic of operational or production applications while unplanned requests are more based on decision support applications.

4. **Optimization and Execution:** DBMSs process DML requests with an optimizer component. The optimizer determines an efficient way of

requesting implementation. A run-time manager is designed to control the execution of the optimized requests.

5. **Data Transformation and Presentation:** Storing data in a database implemented by a DBMS is maintained independently. Hence, DBMSs translate logical requests to physical commands, and store and retrieve data from databases. DBMS and/or database users do not need to distinguish between the logical and physical formats of data. DBMSs provide software independence and data abstraction application programs.
6. **Security Management:** DBMSs monitor user requests and reject attempts that violate the security and integrity constraints defined by the database administrators (DBAs). This functionality is implemented at compile time or at run-time. DBMSs have security systems that enforce user and data security; also, these systems determine data privacy within the database. The extracted rules could be:

- which users access the database
- which data items may be accessed by which user
- which operations, like read, add, update, or delete the user may perform

These terms are especially important in multi-user database systems.

7. **Multi-user Access Control:** Since multiple users use most of the large databases, it is important to keep up data integrity and data consistency. DBMSs provide this facility by using sophisticated algorithms. By using this method, DBMS guarantee the integrity of the databases.

8. **Backup and Recovery Management:** DBMSs must enforce certain recovery and control concurrency. This is done by a related software component called the transaction manager. This component is not part of the DBMS. DBMSs provide backup and data recovery facilities for databases. These aspects guarantee the data safety and data integrity within a database. Recovery management often occurs after abrupt events like power failure or a disk sector crashing.
9. **Data Integrity Management:** DBMSs usually enforce integrity rules for eliminating integrity related problems like data redundancy and data consistency. Solving the data integrity problem is generally important in transaction-oriented DBMSs.
10. **Database Access Language and Application Programming Interfaces:** DBMSs use a query language to access data in a database. A query language is a non-procedural language.¹ The query language used by the DBMS contains two components: DDL (Data Definition Language) and DML (Data Manipulating Language). One defines the structural part and the other allows users to extract data from a database. Programmers can also access the database using procedural languages like COBOL, Pascal, Visual BASIC and others.
11. **Database Communication Interfaces.** DBMSs can connect end-users to databases via networks both local and worldwide. DBMS can accept user requests in a computer network environment and prepare the data and information. DBMSs provide user access to databases through the Internet, using internet browsers for the front end. In this case communication can be prepared in different ways such as:

¹ In non-procedural languages users specify what must be done without specifying how it is to be done.

- The end user can fill in screen forms and refine his/her query.
- Automatically published data to the screen by the DBMS. So that any user can view it in a web browser.
- Connect the DBMS to a third party system and provide information by receiving emails and notes.

The overall purpose of DBMS is to present a user interface while hiding basic and necessary structures from the user.

2.2.2 DBMS Types (by number of users)

Different types of DBMSs depend on the number of users, the database site location(s), and the expected type and its usage extents [39, p. 21].

A DBMS could be a single-user or be a multi-user system. Single-user DBMSs are designed to create and manipulate small and medium size databases. Single-user DBMSs have different functions to create, use and manipulate a database by one user at a time. Some DBMSs are used as both single-user and multi-user DBMSs. Single-user DBMSs are mostly installed on personal computers or laptops and called 'Desktop DBMSs'.

Multi-user DBMSs are used for larger databases. Multi-user DBMSs have two types of database design. The first type is called 'Workgroup Databases' and they can be used by less than about fifty users at a time. Other types of multi-user DBMSs are designed for many more than fifty simultaneous users and they are known as 'Enterprise DBMSs'.

DBMSs are also categorized by site location(s). DBMSs could support a database only on a single site or they can support databases spread over separate sites. Again, it depends on the location, the usage, the number of users, and the capabilities of a database system. A database supported by a DBMS at a single site is called a centralized DBMS. A database distributed by a DBMS over many different sites is called a distributed DBMS.

DBMSs can be classified into different categories in terms their type of use and the extent of their use. Transactions like product or service sales and payments are examples of DBMSs that are used in day-to-day operations. In such cases the time is the most important aspect and the transactions must be recorded immediately. Such DBMSs are known as transactional DBMSs or product DBMSs.

Another category of DBMSs could be based on decision making. This category will require having accurate information stored in the database. Examples of such DBMSs are in sales forecasts, market positioning, and some other fields. These databases need large disk storage space to store and backup data. For this kind of DBMSs the term data warehouse is used. Data warehouses store huge amounts of information including the history of a field.

2.2.3 DBMS Types (by usage)

DBMSs are designed to create and use databases. The main functionalities of these DBMSs are the same. The usability and qualification of DBMSs are quite different. This difference is assigned in DBMS names. DBMS different types are:

- Relational Database Management System (RDBMS)
- Object Relational Database Management System (ORDBMS)

- Object Oriented Database Management System (OODBMS)

2.3 RDBMSs

The relational data model is the primary data model for commercial databases [46], [49]. This data model is simpler than the other data models such as the network data model or the hierarchical data model. The relational data model eases the use of databases for programmers.

The relational data model is implemented by a RDBMS. Arguably, the most important feature and advantage of the RDBMS is to let the user operate in a logical way [39, p. 24]. All the complex physical details of a database are managed by the RDBMSs.

A RDBMS is different from a regular DBMS. The main difference between a DBMS and a RDBMS is the usage of a set-oriented database language by the RDBMS [47, p. 8]. Set-oriented languages use most of the set operations. These operations include union, intersection, difference, selection, projection, divide, multiply, and join. Examples of set-oriented methods are relational-algebra and relational-calculus. Other types of set-oriented languages do exist and are used [11]. Most RDBMSs use SQL as a set-oriented or group-oriented database language.

2.3.1 RDBMS Structure

A relational data model consists of a collection of relations. Each relation has a unique name. Relations (also called tables) contain two dimensional data. Relations consist of attributes² and tuples³. Each attribute in a relation has a name and a data type. The name of each attribute within a relation is unique. The order of attributes

² Synonyms for attribute are column and field.

³ Synonyms for tuple are row and record.

and the order of tuples within a relation are irrelevant. A tuple in a relation represents a relationship among a set of values. Each cell in a relation can only contain one single value.

The structure of a typical RDBMS is shown below [36, p. 14]. Different layers, shown in Figure 2-1, are explained in more detail.

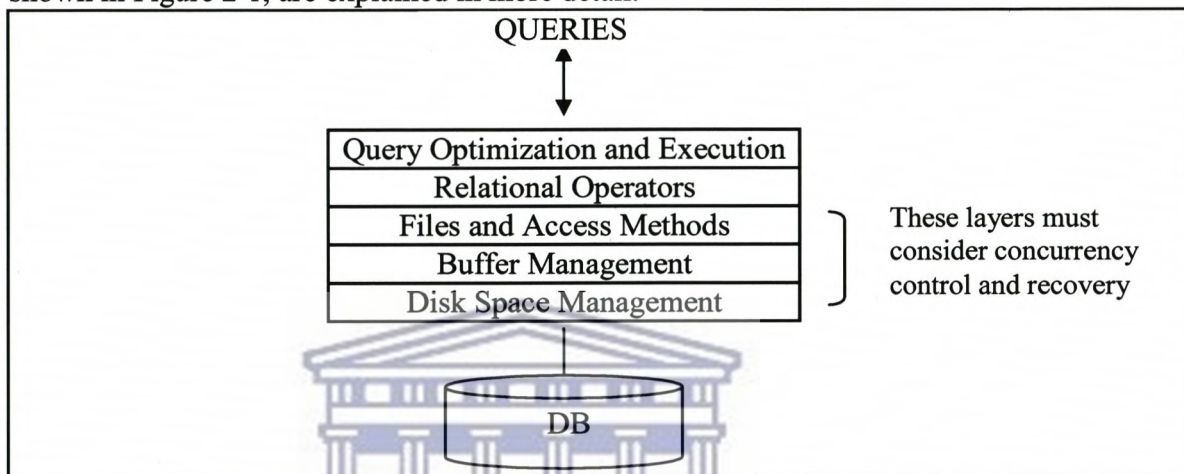


Figure 2-1 RDBMS Structure

The lowest layer, called disk space manager, is a place for storing data. Higher layers allocate, de-allocate, read, and write data and database structures. The next part is the buffer manager, which partitions the available main memory into pages or frames. The buffer manager brings pages and frames from memory to read and implement requests from transactions. The 'Files and access methods' layer holds software that supports the concept of a file in a database.

The 'Relational Operators' is another layer. Operators in this portion of evaluate queries against the data. They can retrieve and edit data stored in a specific database.

Query optimization and execution is done at the very top layer in a DBMS structure. ORDBMSs are an example of extending relational database systems to support

broader classes of applications and to provide a bridge between the relational and object-oriented tracks in the database environment [36, p. 631].

2.4 Summary

In this chapter we have reviewed database systems in general. Database system cycle including users, database, DBMS and application programs were explained. DBMS structure, functions and types were discussed, and we have explained relational database management systems (RDBMSs).



Chapter 3

MySQL

Our research focuses on finding common MySQL errors that students make in database courses. In this chapter we describe the MySQL DBMS and the database language SQL that is used by MySQL. We go into MySQL and SQL fairly deeply and describe their history shortly.

MySQL is a popular open source database management system [42], [43]. It is widely used in the educational arena for teaching databases. Besides education, MySQL software is used commercially. MySQL is ubiquitous. MySQL is usually employed as a backend and a separate user interface is designed to access the database in the front end. User interfaces are designed through websites and object oriented programming languages like Visual Basic, Java, C Sharp (C#), Python and so on.

MySQL Server is used by client-server systems [50]. In client-server systems, the database is placed on one machine. Its structures and data can be stored physically in one or many computers in a network. The database is accessible from other computers or terminals connected to that network.

MySQL is a relational database management system [52]. It can store database tables on different machines and relates those tables to each other. MySQL code can enter data into database tables, update data within database tables and retrieve data from database tables. MySQL uses a Structured Query Language (SQL) for implementing commands. SQL is a query language, which is used by most DBMSs [45, p. 135].

3.1 Query Languages

Query languages are used to request data from a database. Query languages are categorized in two groups: procedural query languages and nonprocedural query languages. In procedural languages all operations are programmed sequentially. On the other hand, nonprocedural languages do not need their operations in a logical sequence. Users can enter what they want with no need to describe any specific procedure. Most DBMSs offer a query language that includes both procedural and nonprocedural elements. SQL is a good example and is a very common and widely used query language. As stated earlier MySQL uses SQL statements for implementing its commands.

The relational algebra has procedural operations and its operators are used via SQL. The Tuple Relational Calculus (TRC) and the Domain Relational Calculus (DRC) are nonprocedural. These languages provide fundamental techniques for extracting data from a database.

3.2 SQL

SQL is a standard database language [47, p. 15]. It is most widely used by commercial databases and it is the most popular commercial query language to RDBMSs [36, p. 32], [43]. SQL is a data sub-language rather than being a complete programming language [27, p. 51]. SQL cannot be used independently. It does not create menus, special report forms, overlays, pop-ups, or any of other utilities and screen devices usually expected by end users. SQL makes database applications more powerful.

SQL works with any application that requires the manipulation of data stored in a relational database. Almost all relational software supports SQL coding. Its

statements are used with scripting languages like VBScript. Some other programming languages like JAVA and C# also can use SQL as a database language.

SQL is text oriented and can be run from a simple command line. SQL statements can be used to define database structure, query data from a database, update data in a database, delete data from a database, and finally control data in a database.

SQL is a standard database language [25], [34]. SQL statements are used to manipulate and retrieve data from relational databases. A programmer or database administrator can do the following with SQL:

- Modify database structure
- Change system security settings
- Add user permissions to databases and tables within databases
- Query a database to retrieve information
- Update contents of databases
- And many other activities

3.3 SQL History

SQL was originally developed for IBM's DB2 product in San Jose, California [47, p. 7]. SQL was developed at IBM corporation in the SEQUEL_XRAM and System_R projects in 1970s [14, p. 85], [27, pp. 51, 52], [36, p. 181], [38, p. 23], [45, p. 135], [47, pp. 7, 8].

In 1980s, Oracle Corporation launched the world's first publicly available commercial SQL system [38, p. 24]. Other companies like Sybase, Informix, and Microsoft used implementations of SQL-based RDBMSs. At that time SQL was not a standard language.

SQL was approved by the American National Standards Institute (ANSI) in 1986. During 1989, ANSI approved another version of SQL called SQL-89 [47, p. 19]. The next version of SQL was SQL-92 and this version was approved by ANSI in 1992 [27, p. 52]. It is also called SQL-2. SQL-3 was released in the late 1990s. SQL-3 is more object-oriented but it did not become popular in the commercial area. SQL-2 is still widely used [27, p. 52].

SQL-2 (ANSI 1992) extended this language to an international standard. There are three levels of SQL compliances: entry, intermediate, and full [47, p. 20].

3.4 MySQL History

MySQL is an advanced form of Mini SQL (mSQL). mSQL had problems with speed. The speed of processing data in this DBMS was not convenient. Another problem with this software was its flexibility. mSQL was not flexible enough for use in database applications. Therefore, a new DBMS was developed. The Application Programming Interface (API) for MySQL is similar to the API of mSQL. MySQL covers all the functionalities of mSQL plus additional capabilities. Compared with mSQL, MySQL is more flexible.

Monty Widenius was the co-founder of MySQL. 'My' was his daughter's name. For this simple reason he named the new developed DBMS software MySQL. MySQL has many editions released in a series of different versions. These differences are highlighted in Appendix C. Newer versions usually enhance older versions.

3.5 MySQL Functions

MySQL uses functions to retrieve data from a database and to insert or update data. MySQL functions include built-in functions and user-defined [13] or stored functions [2], [22]. Some MySQL functions are prone to errors. In this part, we review the group of MySQL functions, which are treated as case sensitive and may cause errors in MySQL code.

3.5.1 Control Flow Functions

MySQL control flow functions are used in many ways. They are used to find and retrieve data, update data, delete data, and in some cases enter data into a database. This group of functions facilitates users to track data easily through MySQL databases. There are not too many of these functions, but they are very useful in databases. Some examples of their syntax and use follows:

- IF(expression1, expression2, expression3)

The result of this function is a comparison between the values given in expressions. If expression1 is true, the result of the function is expression2; and if expression1 is false, then the result is expression3.

- CASE value

```
WHEN compare_value1 THEN result1
```

```
WHEN compare_value2 THEN result2
```

```
...
```

```
ELSE result      END
```

When this function is called, the DBMS compares the given **value** in the command to **compare_value1**, if they match the **result1** displays; if they do not match, control tests the value after the next **WHEN** keyword. If **compare_value2** is true, then **result2** ensues; finally, if there was no match in compare values the **result** after the **ELSE** keyword is shown.

The **ELSE** keyword is optional, if there is no **ELSE** clause then the result is **NULL**. The **CASE**, **WHEN**, **THEN**, and **END** parts are required in this function. The number of **WHEN—THEN** clauses in **CASE** functions can be repeated, but there must be at least one such clause.

- **IFNULL(expression1, expression2)**

This function also compares values. Control flow functions can get values or expressions directly from the keyboard or they can get data from database tables or views. When expressions or values are set for these functions, then they compare values and yield results.

The **IFNULL** function compares **expression1** to **expression2** for **NULL** values. If **expression1** is **NULL**, the function shows **expression2** in the result. If **expression1** is not **NULL**, then the result is **expression1**.

- **NULLIF(expression1, expression2)**

This function also compares the two expressions or values given in parentheses. If they are equal, the result is **NULL**. If **expression1** is not equal to **expression2**, then the result of this function is **expression1**.

Generally, the control flow functions are not case sensitive. Any letter cases can be used for values and expressions in these functions. Normally, no case sensitive errors occur. However, starting from MySQL 3.23.51 in control flow functions, if one or

more of the expressions or values used in the function is case sensitive; then the result of the function is also case sensitive. In this case, if the result of the function is not treated as case sensitive, an error will occur.

3.5.2 String Comparison Functions

There are many string functions in MySQL. String functions are used for different purposes. They are used to find, update, delete, and sometimes retrieve data within database tables. Some functions are used for comparison of string values in databases.

The general MySQL functions are not case sensitive. MySQL string functions, as a subpart of MySQL general functions, are not case sensitive. But sometimes, MySQL string functions used for comparison of string values are case sensitive. Case sensitivity of these functions relates to their special uses in databases. We describe the case sensitivity issue of string comparison functions in Section 3.5.3.

The following list shows the group of string comparison functions, which treat values as case sensitive:

- INSTR(string, substring)
- LOCATE(substring, string)
- REPLACE(string, from-string, to-string)
- STRCMP(expression1, expression2)
- SUBSTRING_INDEX(string, delimiter, count)

These functions are explained briefly with examples in the next section.

3.5.3 Case Sensitive String Comparison Functions

If any expression in a string comparison is treated as case sensitive, the comparison by itself and its result is treated as case sensitive. If a comparison between two strings is executed; in general it is not case sensitive. However, if one or both of the operands are binary strings, then the result is evaluated as case sensitive. In such cases when binary strings are used and a positive result (TRUE) is expected, both operand strings should use exactly the same letter cases.

The following examples show results for comparison of case sensitive and case insensitive strings and their results as TRUE (1) or FALSE (0):

3.5.4 Using INSTR() for String Comparison

The INSTR() function is used for string comparison in MySQL. This function is case insensitive unless a binary string appears in one or both of its arguments. This function uses two arguments or values. These values are entered directly by users or can be extracted from database tables. Data may be directly entered by users or may be extracted from database tables. Strings from both of these methods may be used in string comparison functions. Other commands, which are used for string comparison, can also use strings from these two sources.

The general syntax of the INSTR() function is as follows:

- INSTR(string, substring)

The result of this function yields the position of **substring** in the main **string**. The arguments of INSTR() are in the opposite order of the arguments of LOCATE(). The LOCATE() function is explained later.

Some examples of this function follow. Using binary strings only causes case sensitive checks on strings. If they do not match exactly the result is erroneous.

```
-> SELECT INSTR('Kabul University', 'University');
```

- > 7

Means that the substring 'University' starts from the 7th character of 'Kabul University' .

```
-> SELECT INSTR('Kabul University', 'university');
```

- > 7

Means the substring 'university', all in lowercase letters, starts from the 7th position in 'Kabul University' main string.

```
-> SELECT INSTR('Kabul University', BINARY 'University');
```

- > 7

Means the 'University' substring starts from the 7th position in 'Kabul University' and it is true even when using a binary string (case sensitivity).

```
-> SELECT INSTR('Kabul University', BINARY 'university');
```

- > 0

Means the substring 'university' all in lowercase letters, does not match in 'Kabul University', because in this example the INSTR() treats values as case sensitive. The reason is the usage of binary check for comparing strings.

-> SELECT INSTR('Kabul University', BINARY 'U');

- > 7

The 'U' substring is the 7th character of 'Kabul University' and is true using a binary string, i.e. it is case sensitive.

-> SELECT INSTR('Kabul University', BINARY 'u');

- > 4

The 'u' substring in lowercase, located in the 4th location of 'Kabul University'. The result of the function in this example is case sensitive.

If any or both values in INSTR() function are NULL, the result of the function becomes NULL as well. This is illustrated in the following example.

-> SELECT INSTR('Kabul University', NULL);

- > NULL



One value or string is NULL the result is also NULL.

Shortly, if one or both components of the INSTR() function are used to check binary arguments, the result is case sensitive. This leads to errors. To prevent such errors, users need to be aware of this to make correct MySQL queries.

3.5.5 Using LIKE for String Comparison

The LIKE keyword is also used for string comparison in MySQL queries. Similar to other string comparison functions and commands; the LIKE does case insensitive

comparisons. Only if a binary string is used, the letters of LIKE argument values are treated as case sensitive.

The following examples show the usage of the LIKE keyword and its role regarding case sensitivity and preventing errors.

```
-> SELECT 'Kabul' LIKE 'KABUL';
```

```
-> 1          # means TRUE
```

```
-> SELECT 'Kabul' LIKE binary 'KABUL';
```

```
-> 0          # means FALSE
```

```
-> SELECT binary 'Kabul' LIKE 'KABUL';
```

```
-> 0          # means FALSE
```

```
-> SELECT binary 'Kabul' LIKE binary 'KABUL';
```

```
-> 0          # means FALSE
```

```
-> SELECT binary 'Kabul' LIKE binary 'Kabul';
```

```
-> 1          # means TRUE
```

If one or both operands in a comparison string operation are NULL, the result is NULL, independent of case sensitivity. The following examples show results of such cases:

```
-> SELECT binary 'Kabul' LIKE binary NULL;
```

- > NULL

-> SELECT NULL LIKE 'KABUL';

- > NULL

3.5.6 Using LOCATE() for String Comparison

The LOCATE() function also uses string comparison. This is a MySQL function. Use of this function is very similar to the INSTR() function described above. Most of the rules and characteristics which are applicable to INSTR() are exactly the same as for the LOCATE() function. One difference is in the order of strings, and another difference is in the setting of the position for the start of the comparison.

In LOCATE the substring comes first, the main string second. This is reversed in the INSTR() and is a source of errors.

The LOCATE() function can be called as follows.:

- LOCATE(substring, string)
- LOCATE(substring, string, position)

The result of this function yields the position of **substring** within **main string**. The **substring** argument appears prior to the **main string** argument. The first variation, shown above, does a normal comparison and finds a matching **substring** in the **main string**. The second variation of the syntax gives the option for the user to define the position of the character in the **string** from where the comparison starts. This is essential for finding further occurrences of a **substring** in the **main string**.

The following are examples of the LOCATE() function. Only binary checking of strings causes case sensitive errors.

```
-> SELECT LOCATE('University', 'Kabul University');
```

- > 7

Means the 'University' substring starts at the 7th position of 'Kabul University'.

```
-> SELECT LOCATE('U', 'Kabul University');
```

- > 4

Means the substring 'U' is located in the 4th position of 'Kabul University'.

```
-> SELECT LOCATE('U', 'Kabul University', 5);
```

- > 7

Means the 'U' letter as substring, checked from the 5th position, located in the 7th position in 'Kabul University'.

```
-> SELECT LOCATE('U', 'Kabul University', 9);
```

- > 0

Means the 'U' letter as substring, starting at the 9th position, is not matched in 'Kabul University'.

If a binary string or substring is included, these operations or comparisons are case sensitive. In such cases, if the BINARY keyword is missing, unexpected results and errors occur.

```
-> SELECT LOCATE(BINARY 'University', 'Kabul University');
```

```
-> 7
```

Means the 'University' substring starts at the 7th position of 'Kabul University'.

```
-> SELECT LOCATE(BINARY 'U', 'Kabul University');
```

```
-> 7
```

Means the letter 'U', as an uppercase letter and as a substring after binary checking or case sensitive checking is found at the 7th position of 'Kabul University'.

```
-> SELECT LOCATE(BINARY 'U', 'Kabul University', 5);
```

```
-> 7
```

Means the letter 'U' searching from the 5th position is found at 7th position in 'Kabul University'.

```
-> SELECT LOCATE(BINARY 'U', 'Kabul University', 9);
```

```
-> 0
```

Here, the letter 'U' starting from the 9th position, is not found in 'Kabul University'.

If any or both arguments of this function are NULL, its result becomes NULL. This is illustrated in the following example.

```
-> SELECT LOCATE('University', NULL);
```

```
-> NULL
```

The main string in this example as an argument is NULL the result is also NULL.

Briefly, if one or both arguments of the LOCATE() function are binary, the result is case sensitive. This leads to errors. To prevent such errors, users need to be careful and made aware of their danger in MySQL queries.

3.5.7 Using REPLACE() for String Comparison

REPLACE() can update strings in MySQL. This function is the only MySQL function that always performs case sensitive comparisons. It replaces old strings by new strings and all its checking is case sensitive.

If a user simply uses this function carelessly, i.e., he ignores case sensitivity, errors are inclined to happen. To prevent errors such as these, users should always be aware of letter cases. For this function, all the letters of a string should match exactly. Otherwise, the operation and value that a user expects will not result. From the viewpoint of error checking, this function has to be marked in red so that users are made aware to be careful with its usage.

The syntax of REPLACE() function follows:

- REPLACE(string, from-string, to-string)

This function first checks the main 'string' for occurrences of 'from-string', then it replaces all matched samples in the 'to-string'. The following are examples of REPLACE() functions where all results are case sensitive.

```
-> SELECT REPLACE('KBL University', 'KBL', 'Kabul');
```

```
- > Kabul University
```

In this example the from-string 'KBL' matches part of the main string, and changes the to-string as is expected.

```
-> SELECT REPLACE('KBL University', 'kbl', 'Kabul');
```

```
- > KBL University
```

Here the from-string 'kbl' does not match to any part of the main string; therefore, it does not change the resulting to-string giving an unexpected result.

Regarding the usage of NULL arguments in this function, it is important to note that if any string, main-string, from-string, or to-string is NULL, the result is also NULL. The following example illustrates this.

```
-> SELECT REPLACE('KBL University', 'KBL', NULL);
```

```
- > NULL
```

In this example, the to-string is NULL. Therefore, the result of this query is NULL.

In summary, the REPLACE() function always treats values and strings as case sensitive. For users, this function is error prone. These are not syntax errors, they are lexical errors which users should be warned to avoid. Prevention of this kind of errors has to be included in the software tool. The tool should take strings used in the REPLACE() function, and change the letter case of these strings to all possible forms, then return the result to the DBMS. The DBMS should use all matches in the REPLACE() function and return the expected results for users.

3.5.8 Using STRCMP() for String Comparison

The STRCMP() function was fully case sensitive before MySQL 4.0. In recent versions of MySQL, the usage of the STRCMP() function is not case sensitive. However, if we use binary strings in any version of MySQL, then the result returned is case sensitive. When using NULL values for any or both of strings within the function the result yields NULL.

The general syntax for the STRCMP() function is as follows:

- STRCMP(expression1, expression2)

It compares two strings. If both strings are the same, the result of the function is zero; if expression1 is less than expression2, the result is -1; otherwise, the result of this function is 1.

The following are examples of STRCMP() function:

-> SELECT STRCMP('abc', 'ABC');

-> 0 # means both strings are equal

-> SELECT STRCMP('abc', binary 'ABC');

-> 1 # means both strings are not equal

-> SELECT STRCMP('abc', 'abcd');

-> -1 # means 'abc' is smaller than 'abcd' string

-> SELECT STRCMP('abcd', 'abc');

- > 1 # means 'abcd' is neither smaller nor equal to 'abc' string

-> SELECT STRCMP ('abc', NULL);

- > NULL # means at least one string is NULL in the function

Based on the above paragraphs, MySQL errors occur easily when coding STRCMP functions. These errors relate especially to the use of binary strings in these functions. If a binary string is used in a string function, all letters that are directly typed or taken from columns of relations in a database, are case sensitive. If their letter-case is different, the result of a query will become incorrect and a logical error ensues.

3.5.9 Using SUBSTRING_INDEX() for String Comparison

The SUBSTRING_INDEX() searches for a delimiter in given data. When data matches the delimiter, one or more parts of a string are returned. The string can be entered directly by a user or can be taken from a database source (table).

All comparisons of this function are case sensitive. If case sensitivity is ignored while using the SUBSTRING_INDEX() function, then errors will occur. To prevent case sensitive errors in this function, users should avoid letters in different letter cases. The delimiter string letters should match exactly.

The following is the syntax of this function:

- SUBSTRING_INDEX(string, delimiter, count)

It returns a substring extracted from the main string and delimiter is looked up in string; count is an integer and can be positive or negative. The sign of the count

determines the direction of comparison and can be either from the end when count is negative or from the start when it is positive.

The following are some examples using this function:

```
-> SELECT SUBSTRING_INDEX ('John Black', ' ', 1) AS 'First Name';
```

- > John

Means: show one occurrence of 'John Black' that ends with space delimiter.

```
-> SELECT SUBSTRING_INDEX ('John Black', ' ', -1) AS 'Last Name';
```

- > Black

Means show one occurrence of 'John Black' that ends with space delimiter searching from the end.



```
-> SELECT SUBSTRING_INDEX ('Black, John', ';', 1) AS 'Last Name';
```

- > Black

This example uses a comma and a space instead of only a space.

```
-> SELECT SUBSTRING_INDEX ('Afghanistan', 'istan', 1) AS  
Country_Nation;
```

- > Afghan

Removes 'istan' from 'Afghanistan' giving 'Afghan'.

```
-> SELECT SUBSTRING_INDEX ('AFGHANISTAN', 'istan', 1) AS  
Country_Nation;
```

```
-> NULL
```

This yields a NULL because SUBSTRING is case sensitive.

Note: Usage of this function and other string comparison functions may be performed on different data sources that include more than one record. In this document, we only have shown these functions on single strings. The last example can be implemented on a table that includes country names like 'Afghanistan', 'Tajikistan', 'Uzbekistan', 'Turkmenistan', etc.

3.5.10 Case Sensitivity in Regular Expressions

The REGEXP operation is also used for pattern matching in MySQL. The RLIKE is a synonym for the REGEXP command. REGEXP and RLIKE are fully case sensitive prior to MySQL 3.23.4. In recent versions of MySQL, REGEXP and RLIKE are case sensitive only when a binary string is compared with another string. If a binary string is used as operand with the REGEXP command, it should match in case; otherwise, errors and unexpected results ensue.

The REGEXP / RLIKE syntax is:

- expression REGEXP pattern
- expression RLIKE pattern

The expression and pattern can be written directly in the command line or can be referenced in a data source. If any or both operands, expression or pattern is binary, the comparison will be case sensitive.

The following examples show the usage of REGEXP operation and its possible error occurring in MySQL code:

```
-> SELECT 'Kabul' REGEXP 'KABUL';
```

```
-> 1          # means TRUE
```

```
-> SELECT 'Kabul' REGEXP binary 'KABUL';
```

```
-> 0          # means FALSE
```

```
-> SELECT binary 'Kabul' REGEXP 'KABUL';
```

```
-> 0          # means FALSE
```

```
-> SELECT binary 'Kabul' REGEXP binary 'KABUL';
```

```
-> 0          # means FALSE
```

This operation can also be negated by using NOT. The negated REGEXP, or NOT RLIKE are shown below.

```
-> SELECT 'Kabul' NOT REGEXP 'KABUL';
```

```
-> 0          # yields FALSE
```

```
-> SELECT 'Kabul' NOT REGEXP binary 'KABUL';
```

```
-> 1          # yields TRUE
```

If one or both operands of REGEXP or RLIKE operators are NULL, the result is NULL, Independent of case sensitivity.

3.5.11 Case Sensitivity in the Command Line

Option names and letters used in the command line are case sensitive. For example, lowercase 'v' and uppercase 'V' that are used in the command line are different. The lowercase 'v' corresponds to 'Verbose' and the uppercase 'V' is used for 'Version'. These two cannot be swapped.

Each command has two forms: a long form and a short form. The short forms of the MySQL commands are case sensitive and the long forms of these commands are not case sensitive. For example, we can use the '\t' command to start a session file and use the '\T' to stop recording to the session file. These two commands can be written in long forms as 'tee' and / or 'notee'. The first two commands are case sensitive and have opposite effects. The second two are case insensitive. We can write them as 'TEE', 'Tee' and also 'NoTee' or 'NOTEET'.

The following are used in the command line:

- \b means 'back space'
- \B means the B letter by itself
- \n means disable pager (nopager) end of line
- \N means NULL value
- \c clears the recent command
- \C switches to another charset
- \r reconnects to the Server

- \R reconfigures the MySQL prompt, its long form is `prompt`
- \g sends a command to the Server, its long form is `go`
- \G sends a command to the Server and displays results vertically,
its long form is `ego`

To prevent or reduce case sensitive errors in command line operations, users need to be familiar with the letter cases for short commands. Short options are included in computer programs to make coding easy for users. Computer users, especially in the database field, can use short options in command lines from user terminals. These options are as short as two characters.

Every alphabetic letter has two showcases, uppercase and lowercase. To have more options for command line operations, almost each letter may be used for two different options. One option is with the lowercase of a letter and the other option is with the uppercase of that letter. In general, users have to be serious with the usage of letter cases for short options in command line operations.

3.6 Summary

In this chapter we described MySQL and explained query languages. A quite general study of SQL as a widely used query language [43] was introduced and the history of SQL and MySQL were discussed. We also gave a complete view of MySQL functions, regular expressions and command line operations for MySQL. While MySQL functions were explained, we also described potential errors. Most error occurrences are related to case sensitivity issues in MySQL queries. The case sensitivity and error proneness of MySQL queries were clearly explained. Similarly, MySQL versions and differences between them were explained. We described some similarities, additions and omissions in MySQL versions.

Chapter 4

SQL Errors and Error Checkers

This chapter focuses on SQL errors and the existing error checkers for database languages. Errors may occur in any program. SQL code as any common programming language may contain errors. In this chapter, common SQL errors and types of errors are explained. Methods for SQL error checking at runtime (dynamic) and before runtime (static) are discussed. Then some of the SQL teaching and error checking tools are explained. The chapter continues discussion on error checkers for other programming languages and the chapter concludes with a summary.

4.1 SQL Errors

The general structure of an SQL statement has three parts: **SELECT**, **FROM**, and **WHERE**. Errors can occur in any of these parts at run time. These errors are grouped as data manipulation errors of MySQL. The errors are based on the six fundamental SQL select statement clauses, i.e. **SELECT**, **FROM**, **WHERE**, **GROUP BY**, **HAVING**, and **ORDER BY** [26]. The mentioned errors are checked as semantic errors, syntax errors, and logic errors.

A basic implementation of the SQL-Tutor system described in [34] discovers and analyzes syntax and semantic errors. As mentioned above, SQL errors are grouped into separate categories. These categories are based on the elements commonly found in the SQL select statement, i.e. table names, attributes, prefixes, symbols, and

aggregate functions. SQL can answer a task to be solved if it has a correct query format.

4.1.1 SQL Semantic Errors

Programs sometimes have errors. These errors could be semantic errors, syntax errors or logic errors. Semantic errors occur when a user creates a syntactically correct statement in a program, but the statement does not reflect the user's aims correctly [26]. Semantic errors can be categorized into cases where:

- the task is known in order to detect that the query is incorrect and
- there is enough evidence that the query is incorrect, no matter what the task is.

Semantic errors return an empty set for a query. Most DBMSs do not show error messages while semantic errors occur. Designing a tool for finding semantic errors in SQL-statements, like the program *lint* for C [28] would be useful. Such a tool would be invaluable for teaching and for application software development. A good error message could speed up the debugging process of code.

4.1.2 SQL Syntax Errors

Syntax errors are errors with the actual syntax of a language [26]. Misspelling a keyword is an example of such errors. Syntax and logic errors usually occur at runtime. Hence they are called potential run-time errors [51]. Recently, Minock defined a standard logical notation that is more restricted [32]. This method is closed under syntactic query difference.

Syntax errors and some semantic errors can be detected by static error checking methods. The current status of this field and related topics include static error checking methods and static error checking tools. Methods and tools for static error checking are used to prevent such errors. These methods and tools are explained in the forthcoming paragraphs.

4.1.3 SQL Error Checking

SQL statements are only checked for correctness at runtime [31]. If the code has errors, they are uncovered at runtime. Call level interfaces usually offer significant power of SQL [31]. These interfaces accept dynamically generated SQL statements, but they provide no static syntax or static type checking. Language extensions like SQLJ [1] provide static syntax and static type checking. Persistent object systems allow stored data to be treated as objects but they do not expose the full power of the database engine.

Using a database engine through a Call Level Interface (CLI) involves constructing SQL statements. These statements are built with string concatenation and replacement. This method lets the user develop dominant queries with enough flexibility. The resulting queries are checked for correctness only at runtime when they are sent to the database engine.

This method of creating SQL statements can cause many types of problems and errors [31]. Examples of some common errors are bad syntax such as data type variance, misspelled column names, and unknown tables.

4.1.4 SQL Static Error Checking

Static error checking can improve software quality and productivity [21]. Static code analysis can help a team's ability to deliver a high-quality product or program. It will

not change the face of software development, nor will it turn poor code into good code or weak programs into better programs. However, when database experts agree that a method and tool is useful, it can be used for doing code cleanup, then that work will guarantee the correctness of the code and will be able to show exactly where the problem is and suggest extremely localized fixes of the problem. Static error checking can discover errors and improve awareness of errors by raising the errors at an early phase of the coding.

Static code analysis is a critical part of the bug-finding process. Static code analysis tools are like smarter compilers, better language libraries, new-and-improved software methodologies, high-level dynamic languages, modern Interface Design Environments (IDEs), automated unit test runners, code generators, document tools and any number of other software tools that have shown up over the past few decades.

Most database courses in the educational field, especially computer science, use MySQL for implementing student projects [12], [30]. In these cases, instructors and teachers are responsible for checking the database code generated by students. This checking should be precise, accurate and efficient.

Student skills can be improved by having a competitive classroom environment. Students in a competitive classroom try to do better and be the best among their peers. In a competitive environment, students take advantage of their natural inclination of comparing themselves to others. Similarly, in competitive classrooms even high-performing students are stimulated to higher levels of learning and performance.

Instructors and teachers also need to uncover suspicious similarities between the code generated by students. This similarity in parallel assignments may be a result of student plagiarism.

4.1.5 Semantic Integrity Support in ORDBMS

Semantic integrity support in Object Relational Database Management Systems (ORDBMSs), explained in [49], surveys the state of the art of semantic integrity constraints, and provides an overview and comparison of semantic integrity support in the new SQL standards. Figure 4-1 shows the processing order of declarative constraints and triggers.

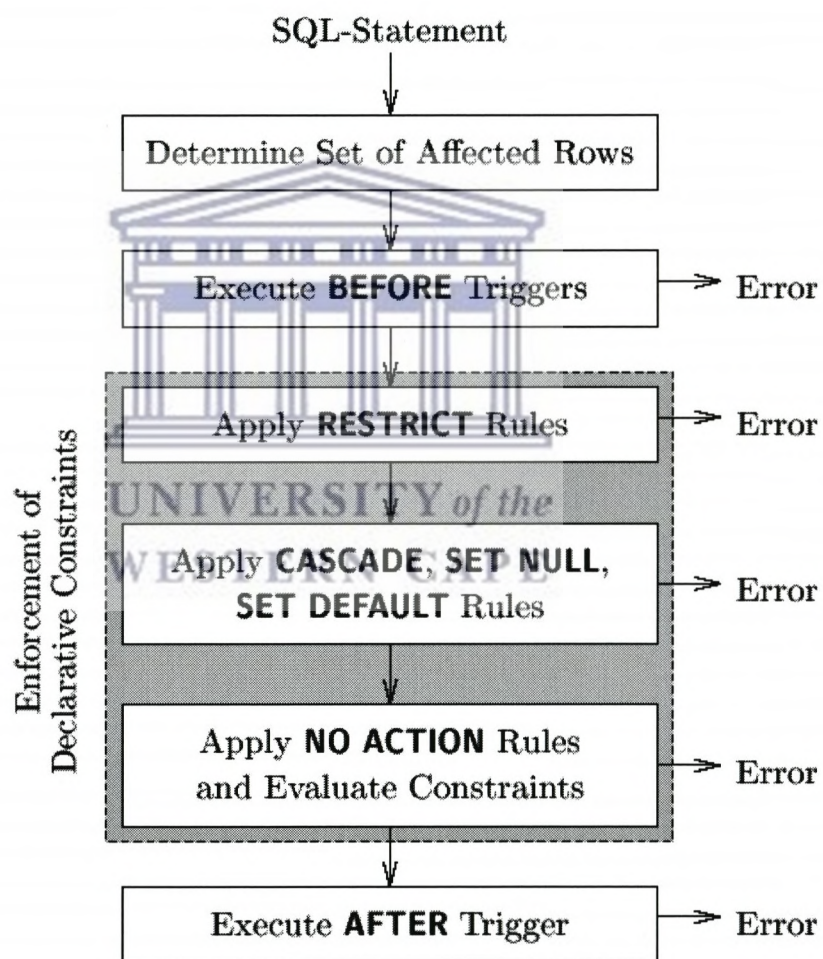


Figure 4-1 Trigger Execution and Constraint Checking

Static error checking is a good option for analyzing and detecting errors in program source code in object-oriented languages. Similar research for object oriented languages, like Java and the C family has already been completed [51]. Flanagan showed the possibility of applying an extended static checker (named ESC/Java) for Java [19], [21].

4.2 SQL Error Checkers

SQL queries are tested on database engines against data. If the query fails, no DBMS provides a clear feedback to the users [5]. Such feedback is important to clear up and give opportunities to the users to develop their skills in SQL queries. The following software is used for teaching SQL. Besides, the software can be used to provide basic ideas for prototyping SQL error checkers. Some of the tools provide feedback and notes to their users. The users of this software should be familiar with the methods that provide feedback. The same methods can be used for error detecting and may work well for the proposed SQL error checkers.

4.2.1 eSQL

eSQL is a tool for teaching SQL. This tool was proposed in 1997 [25]. eSQL is not used for query evaluation or consideration. eSQL does not show a query result, instead, it visualizes query implementation in series of images. It displays a step by step process of how a query outcome results. eSQL is designed for teaching databases rather than for developing them. This system has a simple data store and a parser, which are coded in C. The Graphical User Interface (GUI) of this system is designed in the tool command language Tcl/TK. It is mostly used for rapid prototyping, scripting applications, GUIs and testing. Tcl/TK is extensively used on embedded systems platforms [48].

eSQL has a text-editing area where commands and queries are typed or pasted. Further, the tool may process the code from this area to show its results. It also has a message area for the system. This area displays messages relating the execution status of the queries. Additionally, eSQL has menus, and a separate area to display the full path and names of the current schema and file of SQL commands.

4.2.2 AsseSQL

AsseSQL is an online tool, which provides feedback and notes [35]. This tool allows entry and execution of queries by users. The tool provides immediate response and feedback on users' queries. Then, it indicates to users how accurate their queries are and the correctness of the solutions provided by the users. The AsseSQL does not provide any comments or suggestions for improvement. Users can only guess and solve problems within their queries. This tool is mainly helpful for instructors and teachers to check queries and code generated by their students but it is not helpful for students to edit and find errors in their queries.

AsseSQL was evaluated by students in a survey and its usage was verified by this survey [35]. A structured questionnaire was prepared and issued to the students who participated in an online test via AsseSQL [35]. The results of this evaluation, including questions, are shown in Table 4-1. Ninety two percent of students, who took the test, completed the questionnaire.

Question	Statement	% Agreed
Q1	I was more motivated to practice SQL because of the online test than with a written assignment	85
Q2	I was more motivated to practice SQL because of the online test than with a written test.	85
Q3	Practicing SQL queries interactively and online helped me to improve my SQL query skills.	92
Q4	I preferred taking the online SQL test to taking a written SQL test.	88

Question	Statement	% Agreed
Q5	I preferred taking the online SQL test to submitting a written SQL Assignment.	84
Q6	I have an accurate idea of my ability to construct SQL queries after taking the online test.	78
Q7	The time given for the test was reasonable.	67
Q8	The Marking was consistent and fair.	87

Table 4-1 Percentages of agreed responses to Statements in the Online Test Evaluation Questionnaire

Julia and Raymond [35] mentioned that the next version of AsseSQL will be improved. It will make cheating difficult. The new model will respond to user queries in a second scenario database. This database is not be visible to the users. Answers on the second database will be exactly the same as the first database but its data will be slightly different. This system also will use binary marking on user answers. It will mark an answer either as 'correct' or 'incorrect'. No partial marks will be given to an answer or a query.

4.2.3 SQLator

SQLator was created at the University of Queensland, Australia in 2004 [41] functioning in a similar way to the previous tool AsseSQL. This tool is basically an online learning workbench [41]. SQL commands are categorized into three groups: Database Definition Language (SQL DDL), Data Manipulating Language (SQL DML), and Data Control Language (SQL DCL). SQLator focuses on the SELECT statement, which belongs to the SQL DML part of the SQL commands. Similar to AsseSQL, SQLator also does not provide comments or suggestions for improvement to the users. SQLator and AsseSQL both can only apply binary grading (correct / incorrect) to queries and code [35], [41]. Users' misunderstanding and further learning could not be corrected by the Binary grading methods.

SQLator provides several sample databases for learners to choose from and uses a fully generic Equivalence Engine [41]. This engine judges whether a proposed

solution in a query corresponding to the English statement is correct or not. SQLator system databases describe a business scenario and they contain hundreds of English statements. These statements describe the query requirements regarding SQL rules.

SQLator has different categories of users, i.e. teachers, students, and administrators. Since, SQLator is an online tool and is accessible via the Internet; its users can access it from anywhere. A username and password issued by the tool administration and the authentication for each category of user is also determined. SQLator has three major technology components [41]:

- Web Application – The web provides access for users. The tool uses Active Server Page (ASP) with static HyperText Markup Language (HTML) pages. The system is hosted by a Microsoft Internet Information Server 5.0 (IIS 5.0).
- SQLator Engine – The main part of the SQLator tool is the equivalence engine. This component is registered at the IIS server and implements the core functionalities of the system.
- SQLator Databases – This component contains the SQLator main database and sub databases. The databases are stored on Microsoft SQL Server 2000 DBMS. These databases record user data and hundreds of the English language statements used in comparisons within queries.

The above tools were primarily developed and described for the general Computer Science and Information Systems field. They were not specific to the Relational database theory, and none of them provide detailed implementation information.

4.3 SQL Teaching and Error Checking Tools

There are numerous software-packages used as teaching tools. They are designed specifically for SQL query formulation skills. Examples of such software are SQL-Tutor [33], Acharya [5], and WinRDBI [15]. SQL-Tutor is a problem solving environment. It supports acquisition of domain knowledge bases as feasible forms. SQL-Tutor lets users practice using these systems and learn from them. Some web sites are available for helping users to improve their query skills, e.g. www.sqlator.com, www.sqlcourse.com [35].

Existing SQL teaching methodologies only explain the conceptual part of the SQL for a limited group of examples [7], and do not have enough practice exercises.

4.3.1 SQL-Tutor

SQL-Tutor is a knowledge based system [33], [34]. This system is used to support users and students in learning SQL. SQL-Tutor is an intelligent teaching system. This system has been shown to a number of database teachers and they became very interested in it [34]. With this system, an important improvement in problem solving performance on the post-test of queries was achieved [7]. SQL-Tutor includes three types of learning: conceptual method, problem solving method, and Meta learning method.

SQL-Tutor demonstrates concepts and preliminaries of SQL to its users. This is also a problem solving tool. It supports acquisition of domain knowledge in a declarative form using constraints. Also, it provides support for strengthening practical knowledge. SQL-Tutor assists users and students to solve problems by arguing against errors. Finally, SQL-Tutor supports Meta learning on self-explanation by preparing error messages and sending corrected solutions back to users.

4.3.2 Acharya

Acharya is another system used for teaching and testing SQL statements. Acharya is an Intelligent Tutoring System (ITS) [5]. This ITS is composed of three main modules:

- The *Expert module* deals with the domain knowledge of the system.
- The *Student module* works with the knowledge base of students for a specific topic and updates the student models.
- The *Instructor module*, then, checks the Student modules and based on their outputs, finds the weak points of the teaching and selects new strategies for the future.

This Acharya system uses truth table processing by which SQL SELECT statements are analyzed and corrected. Users get feedback on their work from the system [26]. Acharya architecture is Internet based with support for SQL problem solving. This tool is used via an object oriented interface in the front-end and PostgreSQL in the back-end. Java servlets are used to implement queries entered by users. Figure 4-2 shows the main architecture of the Acharya tool.

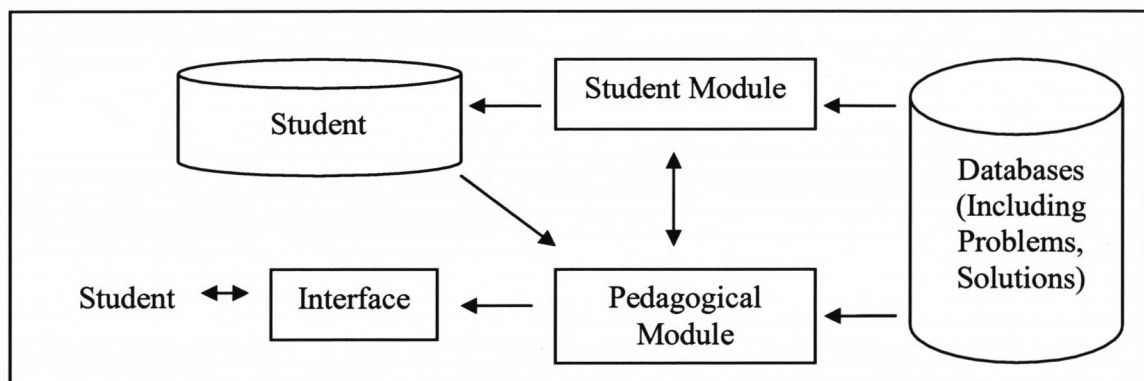


Figure 4-2 Architecture of Acharya

The Acharya user interface is accessible via a window with three areas that are always visible to its users. The upper part of the interface window shows the structure of the database including table-names, their attributes and possible links between tables. Similarly, text explaining the problem to be solved is displayed in this upper part of the window.

The middle part has all six SQL SELECT statement clauses, i.e. SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY [26]. These clauses are shown as labels with entry boxes. Users do not need to remember all query clauses and their order within a query. They just need to enter query parts as they solve the problem proposed by the system. Additionally, some navigation buttons are located in this part of the window. These buttons are labeled as 'Done', 'Clear', 'Hint', 'Knowledge Level', and 'Next Problem'.

The third part displays the results. The results are shown separately as 'Diagnosis result of problem', and 'Result of problem'. The second box in this area, 'Result of problem', only displays the result of the problem query if it is correct. In case of an error in the user's solution, a hyperlink to course materials is displayed.

Acharya has 70 diagnostic rules describing different possibilities between student solutions and expert solutions. This tool supports almost all SQL constructs, which are related to database querying. Only update constructs are not handled [5].

Acharya uses a manually created database. This database stores problems and their solutions provided by experts. The problems from the database are retrieved by users to practice and solve them. Manually tracking such a database system is difficult work. To automatically generate exercise problems, questions and answers, the Acharya system will be further developed [5].

Figure 4-3 is a screen shot from the interface window with a three table database, a problem statement, a solution by users, a diagnosis result of the problem, and a suggesting hyperlink to related course materials as explained.

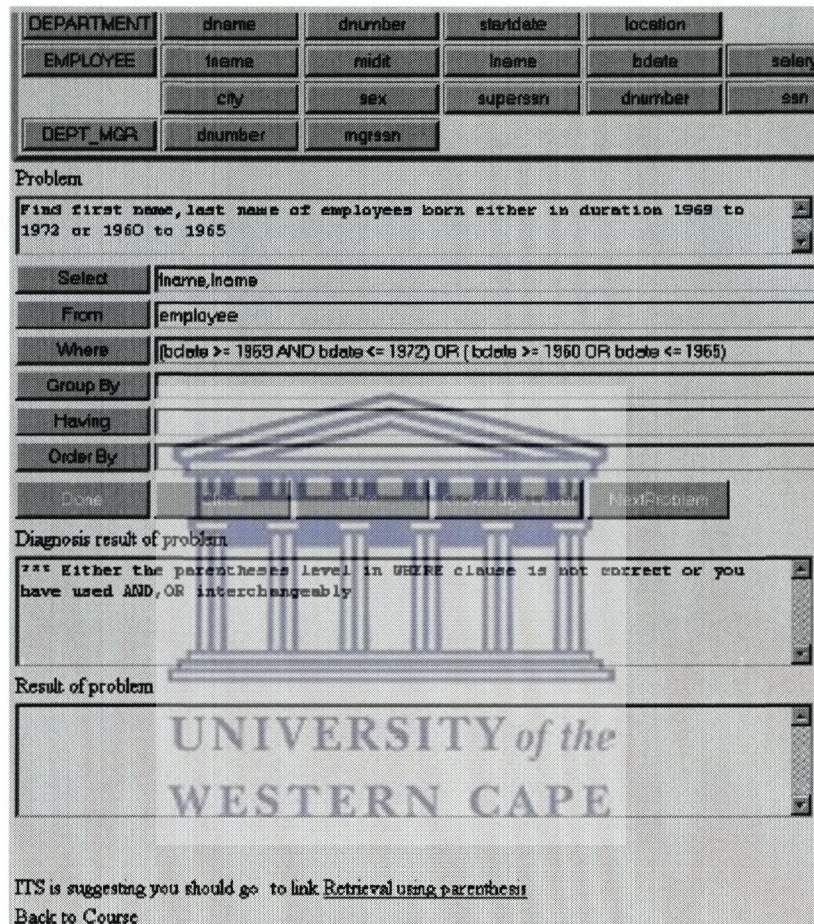


Figure 4-3 Acharya Interface Shot on Problem Solving

4.3.3 WinRDBI—A Relational Database Educational Tool

Similar to the previous tool, RDBI is an educational tool. It assists students to test their queries, which are written in formal relational query languages, i.e. Relational

Algebra, Domain Relational Calculus (DRC), and Tuple Relational Calculus (TRC) [15]. RDBI is a relational database interpreter, which prepares direct feedback for its users.

RDBI was initially developed for student projects, which linked databases to logic programming [15]. This tool is written in Quintus Prolog, an implementation of Prolog. SQL statements are parsed and converted to Prolog code and then implemented by the tool.

RDBI usage, as an educational tool, is limited to users who had Quintus Prolog installed in their machines. WinRDBI is a graphical version of RDBI for Windows [15] that does not require Quintus Prolog, and introduced a graphical interface.

The WinRDBI architecture is illustrated in Figure 4-4. The user interface of the tool is coded in Visual Basic 4.0, and the relational database is coded in Amzi! Prolog.

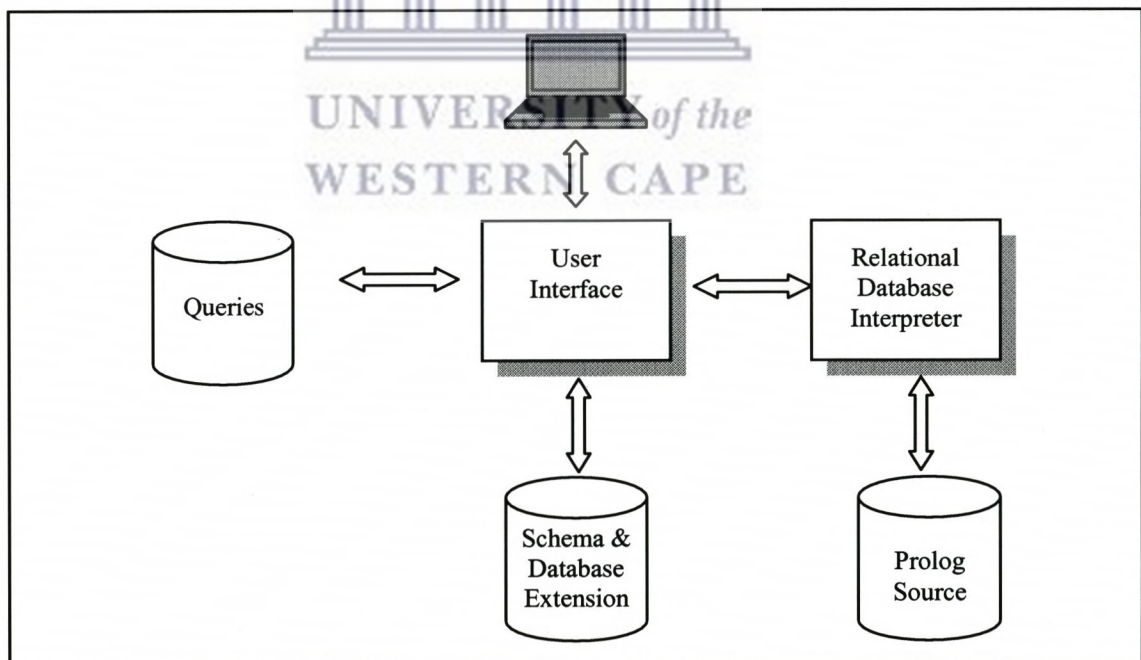


Figure 4-4 WinRDBI Architecture

The WinRDBI user interface has the following main components [15]:

- Query Definition

The Query Definition component accepts queries in one of the four supported languages: Relational Algebra, DRC, TRC, and SQL. Users can choose the language they want to use by selecting a radio button. An 'Execute', executes the entered query code. The results and feedback are displayed in the Query-Result component.

- Query Results

The Query Result component is used to view the results and can be navigated horizontally and vertically. Data shown by this component is read-only and cannot be edited.

- Relations

The list of the relations from the current session is displayed in the Relations component. This list is vertically scrollable. Users can view relation-instances and/or relation-definitions by single or double clicking on the relation name.

- Relation Instances

Relation instances and definitions are shown in the Relation-Instances component. The Relation-Instances component is horizontally and vertically scrollable and its data cannot be directly edited.

Figure 4-5 shows the user interface view of the WinRDBI tool.

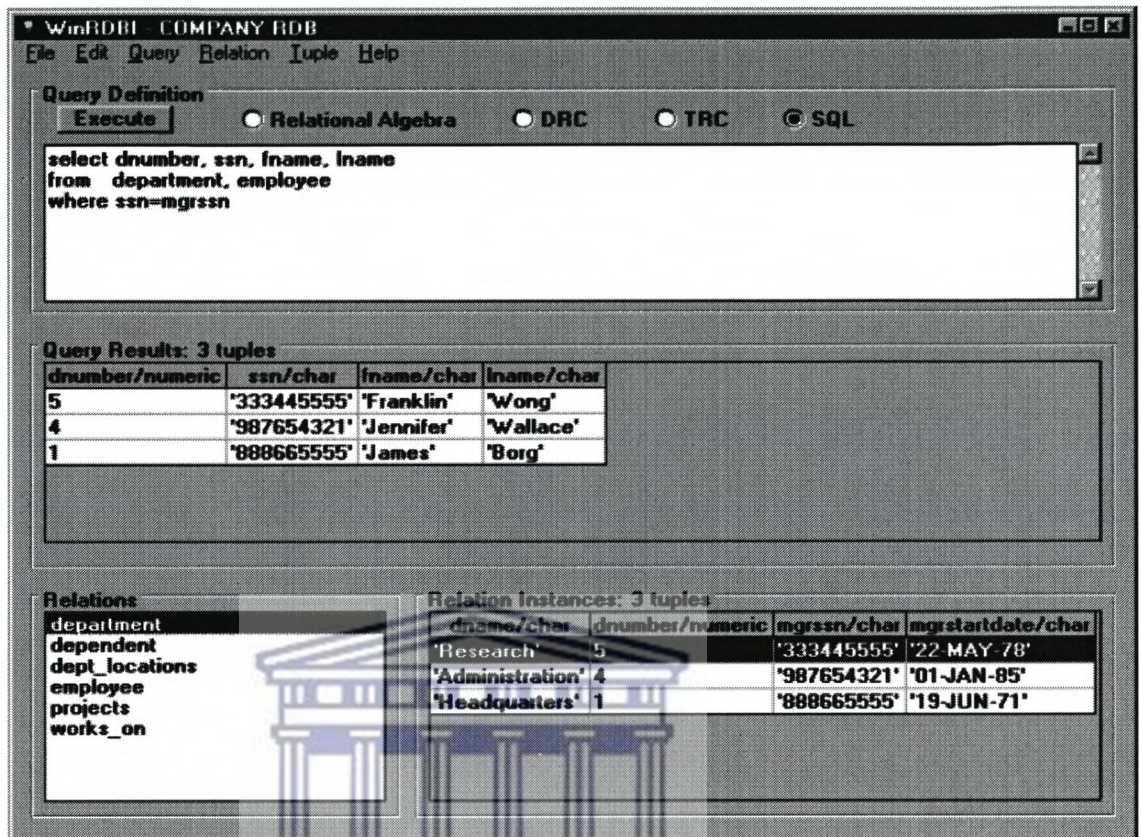


Figure 4-5 WinRDBI User Interface

4.4 Other Error Checkers

4.4.1 ESC/Java

ESC/Java is an experimental compile-time program checker, which finds common programming errors in Java programs. This tool provides a simple annotation language, which can express design decisions formally. ESC/Java also warns of potential runtime errors found in the code. ESC/Java's annotations are simple statements. Functional verification is not required [21]. Programmers can specify their own design decisions and it gives them the option to issue warnings if the program violates those design decisions. Houdini [20] is an example, which uses ESC/Java as a subroutine in inferring annotations [21].

4.4.2 Lint

Lint [28], as a specialized checker that finds broader classes of errors in C programs, such as stylistic errors, type errors and logic errors, but it is prone to making mistakes [9]. Lint detects features of the C program files which are likely to be bugs, or non-portable, or wasteful. It also checks the type usage of the program more strictly than C compilers. The usage of functions is checked, for example finding functions that return values in some places and not in others, functions called with varying numbers of arguments, and functions whose values are not used or whose values are used but none returned.

Flanagan also mentions that Lint, as a static error checker, only detects a limited class of errors [21].

4.4.3 LCLint

Besides these techniques, there are tools like LCLint [17], [18], which are useful in program development. Static error checking of C applications [3] is a similar method to static error checking in Java. Three other examples of static error checking methods are: symbolic execution [6], [8], abstract interpretation [12], and symbolic model checking [8].

Recent work by researchers from Stanford University shows that an effective technique for finding errors is a heuristic scan for irregularities in the program source code [16].

4.4.4 Spec#

Another similar system is Spec# and its automatic verifier Boogie [4] that is applicable to the C# language.

Some methods of error checking do not require complete verification [9]. The checkers using these methods are: LCLint, Aspect [23], extended static checkers [21], [40], [51], and the Monadic second-order logic checker [24].

4.4.5 ESC/Haskell

Previous work focuses primarily on object-oriented programming languages like Java, and C++. Haskell is the only functional programming language for which a static error checking method exists [51].

Recently, an extended static error checking method for Haskell, named ESC/Haskell, was implemented. Haskell is a strong functional programming language and this is the only error checker implemented on a functional programming language [51]. A functional programming language does not contain 'assignment' or 'go to' statements and a pure functional programming language has no side effects on the system on which it runs [29].

The mentioned static checkers work in a very similar way to what is already being done in many modern language compilers like the JAVA and the C# compilers. These tools implement basic semantic checking that verify the program sense. With this checking, source code demonstrates whether it works as intended or has errors. For example, program code is checked for possible security flaws, memory management leaks or synchronization issues like deadlock, access to shared data outside critical sections, etc.

4.5 Summary

This chapter has explained the general terms regarding SQL errors. It pointed to the error prone places within SQL code. Different types of SQL errors and possibilities for error occurrence are discussed. SQL error checking and different methods

including static error checking is explained. Semantic integrity support in ORDBMS was another topic in this chapter.

We have discussed some error checkers in the Computer Science and Information Systems fields. The mentioned error checkers are related to the database language and to other computer languages. SQL error checkers that are explained in this chapter, have limited options. However, error checkers for other languages have quite complete options to detect errors. Additionally, a couple of teaching tools for SQL were explained. Some of these tools have error an checking facility as well as teaching and/or visualizing SQL commands in screen shots.



Chapter 5

Common MySQL Errors

This chapter discusses the MySQL common errors. Case sensitivity errors in searches and in user-defined variable names are explained. Errors in various MySQL modes are included. Also, data definition, data manipulation and transaction control and locking table commands and their status for being error prone are discussed. All three categories of errors are summarized in tables. The chapter concludes with a summary and gives error totals for different classes of MySQL commands.

5.1 Errors-Case Sensitivity

Most MySQL commands are not case sensitive. This has a relationship with the operating system in which the MySQL DBMS is running. The operating systems, which use case sensitive file names, also use case sensitive identifiers and MySQL commands. The newer versions of MySQL are case insensitive even with identifiers and alias names. The previous versions of MySQL—before MySQL 4.1.1—have case sensitive database names, table names and their alias names.

On UNIX platforms and UNIX based operating systems, some of the MySQL commands and keywords are case sensitive. The following code shows a difference between using uppercase and lowercase letters in a single query:

```
SELECT * FROM tone WHERE TONE.id = 1
```

The above code generates an error while using MySQL 4.1.1 or its earlier versions in UNIX based operating systems. In this code, both the table reference and field

reference are written in two different letter cases. The table name is by itself an identifier. For the table reference, lowercase letters are used as 'tone' and for field reference, uppercase letters are used as 'TONE.id'.

As stated earlier, errors are often platform dependent. MySQL platforms are dependent on the version of the DBMS and the operating system. Some operating systems are case sensitive. This was explained in Chapter 3. Similarly, DBMSs can be used in different modes. Each mode has its own rules and characteristics. MySQL modes are explained later in this chapter. Another aspect of a platform is the hardware and storage media used for databases.

If users do not understand the platform they are using, error detection and correction becomes difficult.

5.2 Case Sensitivity Errors in Searches

By default, MySQL searches are not case sensitive in the Windows operating system platforms. In UNIX based operating systems, database identifiers and alias names are case sensitive. Database identifiers include database names and table names. It means that if a database name is 'dbone' written in lowercase letters, a database name with a different letter case should not be used for this database. Searching and referencing 'dbone', as 'DBone' or 'DBONE' expressions cannot be used. The keywords of MySQL code are not case sensitive. They can either be typed in lowercase, uppercase, title case, or mixed case.

When searching a database running on a UNIX platform to retrieve data and extract information from that database, errors can occur when referencing the identifier names. MySQL keywords are not case sensitive. Thus, the focus is to detect and prevent search errors in UNIX operating systems. The names of databases and the names of relations or tables within databases must be case consistent.

Since the alias names before MySQL 4.1.1 are case sensitive, the following query cannot run in older versions of MySQL DBMSs. The usage of alias names in different letter-cases causes this problem. Instead of lowercase 'a' only, both uppercase 'A' and lowercase 'a' can be used. The lowercase 'a' is an alias name for table 'tone' in the active database.

```
-> SELECT col1 FROM tone AS a WHERE a.id = 101 OR A.id = 102;
```

Column names and index names are not case sensitive. Similarly, alias names for regular columns and for indexed columns are also not case sensitive. Column names, index names and their alias names can be typed in any letter case.

In conclusion, to prevent case sensitivity errors, it is always better to use lower case letters for identifiers in MySQL databases. Other formats like using uppercase letters, title case methods, etc. can also be used. However, to use a standard method that can be ported to different platforms and will always work, a good idea is to use lowercase letters. Lowercase letters should be used for identifiers of database names and table names within databases. This should be standard practice in database development.

5.3 Case Sensitivity Errors in User-defined Variable Names

DBMS users can define variables in each session. Values can be stored in user defined variables. Then these values are accessible. User defined variables are temporary. At user exit, these variables are automatically dropped.

In a multi user database, one user cannot see variables defined by another user. User defined variables have been supported in MySQL from version 3.23.6. User defined variables are written as '@variable_name'. Before MySQL 5.0, user variable names

are case sensitive. Using these versions, case sensitivity for variable names is an issue. Users need to be take care to avoid case sensitivity errors in such platforms.

The following is the syntax for defining variables and setting values to them:

- SET @variable_name = expression [, @variable_name = expression] ...

Using the SET keyword for defining variable is one way to do this. There are other methods to define and use variables in a session. In this syntax, there is no space between '@' symbol and 'variable_name'.

The following example defines and accesses variables.

```
-> SET @Length = 4, @Width = 3;  
-> SELECT @length * @width AS Area;  
  
-> 12
```



This query takes values for its two arguments from defined variables, and then multiplies the values and shows their result in the **Area** column.

In new versions of MySQL, where variable names are not case sensitive, the result of the example query is correct. However, in older versions—before MySQL 5.0—the result of this operation is NULL. This is why the variable names in the given example may have different letter cases.

In general, if a string search or comparison is case sensitive and used in a query, that query is prone to errors. In such cases, query results should check to find and detect

errors. This can be done after running a query. To find the errors related to case sensitivity, the MySQL code needs to be checked while it is in text mode.

As a result of previous discussions, case sensitive errors mostly occur with binary searches, comparisons, and implementation of data. It is worth mentioning that in some cases the only way we can do an exact search in a database is by using binary arguments in queries. This includes searches and pattern matches. The results of these operations can further be used to update, delete or retrieve data from a database.

Based on the importance of binary operations in databases, we cannot ignore binary operations in database queries. Then, case sensitive errors in database queries result directly only when using binary operations. To find and prevent such errors, we should first check and find the binary operations in a query, then we can detect and prevent case sensitive errors in databases.

5.4 Errors—MySQL Modes

Different kinds of errors may occur while using identifier names in MySQL code. These errors may take place using database, table, index, column and alias name formats. Identifiers can be quoted or used unquoted in MySQL. If a MySQL reserved word is used for an identifier, or if an identifier includes some special characters or an identifier consists from more than one word, it should be quoted. Otherwise, using quotations for identifier names is optional.

Note: The identifier quotation character in MySQL is the back tick “`”.

MySQL uses different modes. Each mode has its own characteristics and methods for writing symbols and formats. For example, if the version of MySQL does not support double quotes, the following command will cause an error.

```
-> CREATE TABLE "TOne" (ColOneINT);
```

Quoting of identifiers started in MySQL from version 3.23.6. Before this version of MySQL, identifiers could not include special characters, reserved words or spaces.

Starting from MySQL 4.1, modes can be set for different users or clients individually. Each user can set the MySQL mode with which they like to work. From this version onwards, MySQL modes can be set by using the **SET** command. This can be done at run time. The syntax of the **SET** command follows:

```
- SET SQL_MODE = 'ModeName';
```

Any user can run this command at any time. The 'ModeName' can be any supported mode of MySQL. The default mode for MySQL is empty: while no mode is set or changed. Each user can change MySQL modes for their own use. This change does not affect other users or clients who are using the same database. If a user wants to set the default mode of MySQL after any changes, this is done by setting an empty string in the above command.

However, there are some options to be added to the 'SET' command. We can add **GLOBAL** or **SESSION** options to a SET command. Using the **GLOBAL** option sets changes to all users. Using the **SESSION** option sets changes to the specific user who issues this command. For using **GLOBAL** option in SET command, 'super' privileges are required.

The following topics describe some of the MySQL supported modes and their role in detecting / preventing errors.

5.4.1 The ANSI_QUOTES Mode

Setting this mode in MySQL causes the DBMS to accept double quotes (") and single quotes (') besides back ticks for identifiers. Normally, MySQL uses back ticks (`) for

quoting identifiers. If we use double quotes or single quotes for identifiers in MySQL, it causes errors. Similarly, if we use back ticks for identifiers while the `ANSI_QUOTES` mode is activated, MySQL still accepts commands and no error occurs.

Strings should not be quoted in double quotes in this mode. While the `ANSI_QUOTES` mode is active, strings are quoted using single quotations. Double quoted strings are not acceptable. Normally, MySQL supports both single quotes and double quotes for strings.

Aliased names are not affected by setting the `ANSI_QUOTES` mode. We can use either back ticks, single quotes or double quotes for alias names. This was introduced in MySQL version 4.0.0.



5.4.2 The `IGNORE_SPACE` Mode

This mode allows spaces between a `FUNCTION` name and the left parenthesis '(' that delimits the start of the function's arguments. Only MySQL built-in functions allow these spaces. The user-defined functions do not allow this space.

When the `IGNORE_SPACE` mode is active, built-in function names are treated as MySQL reserved words. In this case, if we use the name of a built-in function for an identifier, it causes an error. When using any built-in function name as the name of a database, table, or column within a database, it should be quoted. If it is not quoted an error occurs.

The following examples show the usage of a MySQL built-in function name as an identifier in two states, normal mode and `IGNORE_SPACE` mode:

```
-> SET SQL_MODE = "";
```

-> Query OK, ...

The MySQL mode is equated to an empty string, means MySQL uses no mode. This is default.

-> CREATE TABLE PI (Col1 INT, Col2 CHAR(20));

-> Query OK ...

No error occurs. A table with the name of a MySQL built-in function PI() is created.

-> DROP TABLE PI;

-> Query OK ...

Removes the new created table named PI, from the database.

Now, we will try the same commands when the IGNORE_SPACE mode activated:

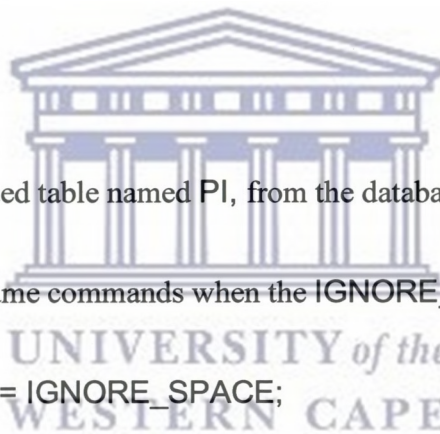
-> SET SQL_MODE = IGNORE_SPACE;

-> Query OK, ...

The MySQL mode is changed to the IGNORE_SPACE mode. It means that MySQL uses the built-in function names as reserved words and can not be used for identifiers.

-> CREATE TABLE PI (Col1 INT, Col2 CHAR(20));

-> Error ...



An error occurred: since the name `PI` is the name of a built-in function and is a reserved word in MySQL. Therefore, it cannot be used for table name. If we want to use this for identifiers, it must be quoted with the back tick characters.

To prevent this error and create the table by using `PI` word as identifier, we have to use the following code:

```
-> CREATE TABLE `PI` (Col1 INT, Col2 CHAR(20));
```

```
-> Query OK ...
```

Now, no error occurs. A table named as a MySQL built-in function name is created.

In summary, the `IGNORE_SPACE` mode can be activated in MySQL. In this mode, the MySQL built-in function names can be written separately from their parentheses while calling functions. These names become special words of MySQL and are treated as MySQL reserved words. If users forget this point and use reserved words as usual words, errors ensue. To prevent such errors, users should be aware of using quotations for these words. This was introduced in MySQL version 4.0.0.

5.4.3 Other Modes for MySQL

There are some other modes in MySQL that users can set. Setting a specific mode of MySQL can be done by using the `SET` command. The syntax of the `SET` command and some examples of activating a MySQL mode has been previously explained. The `GLOBAL` or `SESSION` options are activated with a `SET` command. Each option has its own usage and needs special rights for users. **Error! Reference source not found.** lists MySQL modes and their support from different versions of MySQL.

No.	Mode Name	From MySQL Version
1	ANSI_QUOTES	4.0.0
2	IGNORE_SPACE	4.0.0
3	NO_AUTO_VALUE_ON_ZERO	4.1.1
4	NO_DIR_IN_CREATE	4.1.1
5	NO_FIELD_OPTIONS	4.1.1
6	NO_KEY_OPTIONS	4.1.1
7	NO_TABLE_OPTIONS	4.1.1
8	NO_UNSIGNED_SUBTRACTION	4.0.2
9	ONLY_FULL_GROUP_BY	4.0.0
10	PIPES_AS_CONCAT	4.0.0
11	REAL_AS_FLOAT	4.0.0

Table 5-1 MySQL Modes and their supported DBMS Versions

Two important modes of MySQL, number 1 and 2 in the table that often cause errors, have been explained. We now briefly discuss the entire nine modes of MySQL, number 3 to 11 from **Error! Reference source not found.**, and their usage in database field.

3. The NO_AUTO_VALUE_ON_ZERO Mode

This mode affects the 'auto incremented' column in a table.

4. The NO_DIR_IN_CREATE Mode

This mode ignores all index and data directory directives.

5. The NO_FIELD_OPTIONS Mode

This mode does not allow printing the MySQL specific column options in the result of the `SHOW CREATE TABLE` command.

6. The `NO_KEY_OPTIONS` Mode

This mode does not allow printing the MySQL specific index options in the result of `SHOW CREATE TABLE` command.

7. The `NO_TABLE_OPTIONS` Mode

This mode does not allow printing the MySQL specific table options like `ENGINE` in the result of `SHOW CREATE TABLE` command.

8. The `NO_UNSIGNED_SUBTRACTION` Mode

This mode does not mark the result of an integer subtraction as unsigned. This happens when one of the operands is unsigned.

9. The `ONLY_FULL_GROUP_BY` Mode

This mode does not allow `SELECT` queries which have an aggregate function in their column reference list and one or more un-aggregated columns are not marked in the `GROUP BY` portion. The following example clears this mode's effects and the example query is invalid in this mode.

```
-> SELECT Name, Phone, MAX(Salary) FROM tblOne GROUP BY Name;
```

```
-> Error ...
```

In the `ONLY_FULL_GROUP_BY` mode, the above query results in an error. This is because all un-aggregated columns should be used after the `GROUP BY` portion.

To solve this problem, we have to define both the Name and Phone columns after the GROUP BY as follows:

```
-> SELECT Name, Phone, MAX(Salary) FROM TOne GROUP BY Name, Phone;
```

```
-> Query OK ...
```

The result of this query is now correct.

10. The PIPES_AS_CONCAT Mode

This mode treats the || symbol as a string concatenation operator. In normal mode this symbol is used as a synonym for 'OR'. This exactly works like the CONCAT() function in MySQL.

11. The REAL_AS_FLOAT Mode

This mode treats REAL number as FLOAT numbers. In general mode, REAL is treated as DOUBLE.

Setting any mode from the preceding list can cause errors in MySQL. Each mode has its own specifications and usage. When a specific mode is activated in a session, the user of that session has to be careful with the errors that are likely to occur. Similarly, if a specific mode is activated with the GLOBAL option, all users of the system have to be aware and be careful.

5.4.4 Combination of Different MySQL Modes Simultaneously

We can activate more than one mode of MySQL at a time and different combinations of the MySQL modes can be implemented during a specific session or globally on a

system. Similar to setting a single mode, combined modes can also be effective on a database system.

From time to time MySQL modes have been added to versions of MySQL. All modes are available from MySQL 4.1.1. Therefore, combination of more than one mode in one time is available since version of MySQL 4.1.1.. Each combination takes an identifier name and includes a set of modes.

The combined or unique modes in MySQL can be activated using the **SET** command. The syntax is exactly the same. For example we can activate the **ANSI** combination of modes as follows:

```
-> SET SQL_MODE = ANSI;
```

The following groups of different modes can be combined at the same time:

1. The ANSI Combination of MySQL Modes

ANSI is a combination of the following modes:

- REAL_AS_FLOAT
- PIPES_AS_CONCAT
- ANSI_QUOTES
- IGNORE_SPACE

The previous versions of MySQL—before MySQL 4.1.11—also allowed the **ONLY_FULL_GROUP_BY** mode in this combination but recent versions have only the listed modes.

2. The DB2 Combination of MySQL Modes

The DB2 is a combination of the following MySQL modes:

- PIPES_AS_CONCAT
- ANSI_QUOTES
- IGNORE_SPACE
- NO_KEY_OPTIONS
- NO_TABLE_OPTIONS
- NO_FIELD_OPTIONS

3. The MAXDB Combination of MySQL Modes

This combination group is similar to the previous one and to some other groups as well. It includes some modes found in other combined groups of MySQL. MAXDB is a combination of the following MySQL modes:

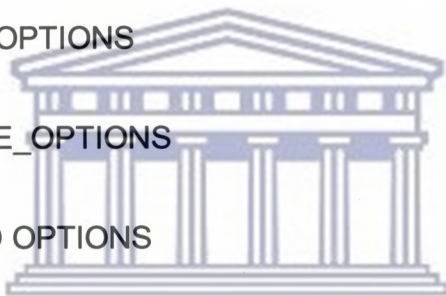
- PIPES_AS_CONCAT
- ANSI_QUOTES
- IGNORE_SPACE
- NO_KEY_OPTIONS
- NO_TABLE_OPTIONS

- NO_FIELD_OPTIONS

4. The MSSQL Combination of MySQL Modes

The MSSQL is a combination of the following MySQL modes:

- PIPES_AS_CONCAT
- ANSI_QUOTES
- IGNORE_SPACE
- NO_KEY_OPTIONS
- NO_TABLE_OPTIONS
- NO_FIELD_OPTIONS



5. The MYSQL323 Combination of MySQL Modes

This is equivalent to the NO_FIELD_OPTIONS mode.

6. The MYSQL40 Combination of MySQL Modes

Like the previous mode, this mode is also equivalent to NO_FIELD_OPTIONS.

7. The ORACLE Combination of MySQL Modes

The ORACLE is a combination of the following MySQL modes:

- PIPES_AS_CONCAT

- ANSI_QUOTES
- IGNORE_SPACE
- NO_KEY_OPTIONS
- NO_TABLE_OPTIONS
- NO_FIELD_OPTIONS

8. The POSTGRESQL Combination of MySQL Modes

The POSTGRESQL is a combination of the following MySQL modes:

- PIPES_AS_CONCAT
- ANSI_QUOTES
- IGNORE_SPACE
- NO_KEY_OPTIONS
- NO_TABLE_OPTIONS
- NO_FIELD_OPTIONS

Combining different modes of MySQL in one command and using them in a single session is more critical. Most of these combined forms include different modes. Each mode has different rules and uses different formats. When a specific group of combined modes is activated, users need to have enough knowledge of each mode included in the group. In such cases, different uncommon errors may occur in

MySQL code. To prevent those errors, users have to be prepared and check their code before running it.

5.5 MySQL Error Classification

SQL commands are divided into three groups: SQL-DDL (data definitions), SQL-DML (data manipulations), and SQL-DCL (data controls). Errors may occur in any of these groups. Database administrators use the data control part of the SQL commands. One of our research goals is to find and categorize students' and novices' errors in MySQL; therefore, we have focused on the first two groups i.e. SQL-DDL and SQL-DML. The SQL-DCL commands and their related errors are ignored. Additionally at the end of this chapter, we have explained transaction control statements and their proneness to errors.

We have categorized MySQL errors into three main groups: data definition errors, data manipulation errors, and transaction control errors. All the mentioned categories of errors are further checked as semantic, syntax, and logic forms of errors.

5.5.1 Errors—SQL-DDL

The data definition commands of MySQL are used to define, update and delete databases, database objects and their relationships within MySQL databases. The data definition commands also edit the structures of database tables including fields, data types, constraints, indexes and so on. The SQL-DDL part covers the following commands:

- CREATE ...
- ALTER ...
- DROP ...

- RENAME ...

We have summarized the data definition errors of MySQL in Table 5-1 at the end of this part. We used a unique reference to identify each error in the list. The identification for the listed errors in this part is combined of 'DD', which stands for 'data definition', and a two digit number started from '01'. Similarly, errors related to other parts have their own identification format. For example, the ID for the data manipulation errors consists from 'DM', stands for data 'manipulation', plus two digits started from '01'; and the ID for the transaction control errors consists from 'TC' and two digits. Each section errors are summarized in a table at the end of that section.

5.5.2 Errors—Database Definition

A database is a computerized collection of logically related data, data definitions, data relationships, and data retrieval methods. Databases are created via DBMSs. MySQL gives opportunity of defining / creating databases, altering databases, and dropping databases. These activities are implemented through CREATE DATABASE, ALTER DATABASE, and DROP DATABASE commands. In any of these controls errors are possible.

5.5.3 CREATE DATABASE—using IF NOT EXISTS

To prevent errors when creating a new database, we use the IF NOT EXISTS keywords prior to the database name in the command. This is implemented in MySQL 3.22 and later versions of this DBMS. Three commands are used for this purpose.

The first command just creates a database named dbOne. The second command again tries to create the same file dbOne that resulted with an error. The third

command first checks if any database with the same name exists. If there is no such database, then the dbOne database is created. The second command might be implemented if a database by the name dbOne exists in the system. However, this command will cause an error. The third command does not create a database, if a database with the name dbOne exists in the system. No error occurs.

-> CREATE DATABASE dbOne;

-> Query OK ...

With no error the dbOne database is created. This database was not in the system.

-> CREATE DATABASE dbOne;

-> ERROR ...

An error occurred. The dbOne database is already in the system. It was created by the previous command.

-> CREATE DATABASE IF NOT EXISTS dbOne;

-> Query OK ...

No error occurs even if 'dbOne' is in the system. The DBMS engine checks for the existence of dbOne. If it were in the system, the CREATE DATABASE operation is canceled and no error occurs.

In conclusion, we can add the IF NOT EXISTS keyword to most of the data definition commands of MySQL: i.e. SQL-DDL. This option prevents DBMS errors from occurring. This group of keywords is placed exactly between the main

command and the identifier object name within a database. Examples, of these forms have been explained.

5.5.4 ALTER DATABASE ...—using IF EXISTS

This command changes the overall characteristics of an existing database. If the database does not exist an error occurs. To prevent such error the IF EXISTS keyword is used. The IF EXISTS keyword is placed before the database name within the command.

The ALTER DATABASE command was added in MySQL version 4.1.1. Beginning from MySQL 4.1.8, a user can alter the active database. From this version onwards, the database name may be omitted from the ALTER DATABASE command without any error. In such cases, the ALTER command changes the default database.

5.5.5 DROP DATABASE ...—using IF EXISTS

Like other MySQL commands, users need privileges to use DROP DATABASE. If a database does not exist or if the command is mistyped, errors will occur. To prevent errors regarding the existence of a database, we use the IF EXISTS option before the database name in the command. This option was added in MySQL 3.22 and is in later versions. From MySQL 4.1.2, the DROP DATABASE command summarizes the number of tables removed within the deleted database.

```
-> CREATE DATABASE dbOne;
```

```
-> Query OK ... # The dbOne database is created
```

```
-> USE dbOne; # The dbOne database is activated / used
```

-> CREATE TABLE tOne (ID INT, Name VARCHAR(30));

-> Query OK ... # The tOne table is created

-> CREATE TABLE tTwo (Company Char(20), Address VARCHAR(20));

-> Query OK ... # The tTwo table is created

-> DROP DATABASE dbOne;

-> Query OK, 2 rows affected ...

No error occurs. The dbOne database is removed and the number of deleted tables, tOne and tTwo, is returned.

-> DROP DATABASE dbOne;

-> Error ... # The tOne table is created

An error occurs. The dbOne database was already removed by the previous command and it is not in the system. For preventing this error, the following command can work.

-> DROP DATABASE IF EXISTS dbOne;

-> Query OK ...

No error occurs. The dbOne database is not in the system; however, adding the IF EXISTS keywords before the database name prevents errors occurring.

The DROP DATABASE command deletes all structures in a database, i.e. relations, relationships, indexes, rules, userdata, and the database by itself.

5.5.6 Errors—Table Definition

Tables or relations are the basic structures within a database that store data in database files. Tables can be created by MySQL commands. Their structure can be updated or changed in MySQL. Similarly tables can be deleted from a database. Every operation on database tables, including 'CREATE TABLE ...', 'ALTER TABLE ...', and 'DROP TABLE ...' can cause errors.

5.5.7 Errors—CREATE TABLE ...

If a table already exists and the user tries to create a new table with the same name in current database, an error occurs. To prevent this kind of errors, we need to use the 'IF NOT EXISTS' option. See the following examples.

```
-> CREATE TABLE tblOne (ID INT);
```

```
-> Query OK ... # No error occurs.
```

```
-> CREATE TABLE tblOne (ID INT);
```

```
-> ERROR ...
```

An error occurs: the table already exists.

```
-> CREATE TABLE IF NOT EXISTS tblOne (ID INT);
```

```
-> Query OK ...
```

No error occurs. The DBMS first checks the existence of a table with the specified name in the command. If the table does not exist, it will be created, otherwise it will

not be created. The command will be ignored and a warning will be shown instead of an error message.

5.5.8 Errors—CREATE TABLE ... SELECT ...

The CREATE TABLE ... SELECT ... command creates a table within current database and directly enters data to this table. The data is retrieved from existing tables or views. The SELECT part of this command can be any query with all supported options. If any syntax or typing errors occur in this query, then the table will not be created.

The following two examples show two different statements for defining the table tblTwo, one is correct and the other is not:

```
-> CREATE TABLE tblTwo (Name CHAR(25), Value INT) SELECTT *  
FROM tblOne;
```

-> ERROR...

Table tblTwo will not be created. The problem is with the syntax of the query where the SELECTT keyword is mistyped.

This command has a typing error in the subsequent query and an error occurs. Therefore, table tblTwo is not created. In the following example, the syntax error in the subsequent query is solved. With no error, table 'tblTwo' will be created and will take values from the result of the query.

```
-> CREATE TABLE tblTwo (Name CHAR(25), Value INT)  
  
SELECT * FROM tblOne;
```

-> Query OK ... # Table tblTwo is created.

When referencing function or expression columns in the 'CREATE TABLE ... SELECT ...' statement, the function calls and expressions in referenced tables or views should be clearly aliased. If alias names are not used for the original columns, errors will occur and the tables will not be created or the command might cause undesirable results. The following is an example.

```
-> CREATE TABLE tblThree (Department CHAR(10), TotalStuds INT)
SELECT Department, COUNT(StuName) AS TotalStuds FROM tblTwo
GROUP BY Department;
```

-> Query OK ...

No error occurs and the table is created and gets data from another table as well.

The result of this command is correct. The 'tblThree' table is created. Data from the result of its subsequent query is entered to this table.



5.5.9 Errors—ALTER TABLE ...

The ALTER TABLE is one of the MySQL commands that enable users to edit the structure of an existing table within a database. This command can add or delete columns, set or drop indexes, change the type of a column, rename a column in a table. The ALTER TABLE command can even change the name of an existing table in a database. Under the circumstances described below, ALTER TABLE may cause errors.

- ALTER TABLE—ADD / DROP PRIMARY KEY ...

The ALTER TABLE command can be used to set or drop a primary key in an existing database table. Each table can have only one primary key. Multiple primary keys defined for a single table using this command cause errors to happen. The same command for setting a specific primary key may not be used twice. The following examples show this kind of error.

-> DESCRIBE tblOne;

-> Query OK ...

Table 5-2 shows the structure of table tblOne.

Field	Type	Null	Key	Default	Extra
ID	int(11)	No		0	
Name	char(10)	Yes		NULL	
Sal	decimal(10,0)	No		0	

Table 5-2 The structure of tblOne Table
UNIVERSITY of the
WESTERN CAPE

The tblOne does not have a primary key. We can use the following command to set the primary key for this table.

-> ALTER TABLE tblOne ADD PRIMARY KEY (ID);

-> Query OK ... # No error occurs

The previous command sets the ID field of table one as the primary key of that table. Running the following two commands causes errors.

-> ALTER TABLE tblOne ADD PRIMARY KEY (ID);

-> ERROR...

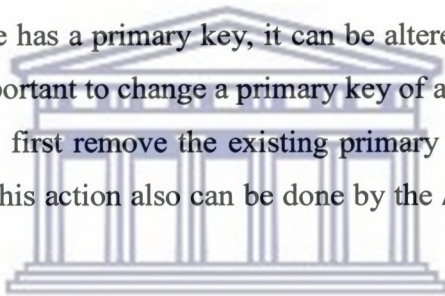
An error occurs. This table already has its primary key.

-> ALTER TABLE tblOne ADD PRIMARY KEY (Name);

-> ERROR...

An error occurs. This table already has its primary key.

Therefore, when a table has a primary key, it can be altered only by removing it and replacing it. If it is important to change a primary key of a table to another column or to a group of columns, first remove the existing primary key from the table, before adding the new key. This action also can be done by the ALTER TABLE command in MySQL.



The following example shows the usage of the ALTER TABLE for removing primary key of an existing table in a database.

-> ALTER TABLE tblOne DROP PRIMARY KEY;

-> Query OK ...

No error occurred; the primary key of tblOne is removed.

After implementing this command, tblOne does not have a primary key. If a table does not have a primary key and we run the MySQL command to drop its primary key, an error occurs. The following example shows this kind of error in MySQL.

-> ALTER TABLE tblOne DROP PRIMARY KEY;

-> ERROR ...

An error occurs because tblOne does not have primary key.

5.5.10 Errors—DROP TABLE ...

The DROP TABLE command is used to remove one or more tables from a database. The following is the syntax for this command

- DROP TABLE TableReferences

This command can be executed on one table at a time or on tables using a single command. If a table in the reference list does not exist, then the DBMS produces an error. The following example shows a simple use of this command:

-> DROP TABLE tblOne, tblTwo;

-> Query OK ...

Assume both tables exist in the system. No error occurs and both tables are removed.

-> DROP TABLE tblOne, tblThree;

-> ERROR ...

Assume tblOne does not exist (it is removed by the previous command). An error occurs in this command. The error arises because of a non-existent table i.e. tblOne in this command; other tables mentioned in the command will be removed from the database.

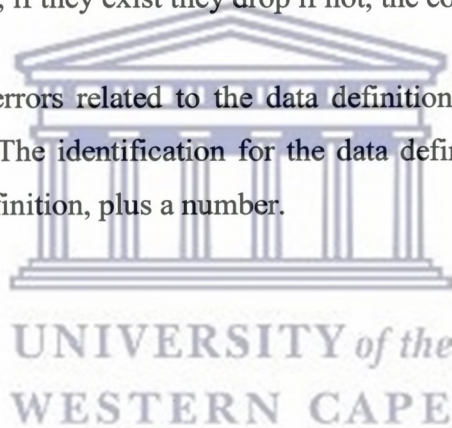
The DROP TABLE can also use the IF EXISTS option. If this option is added to the DROP TABLE command, no error will occur. The IF EXISTS option was added in MySQL version 3.22 and can be used from that version up to the newest version.

```
-> DROP TABLE IF EXISTS tblOne, tblFour;
```

```
-> Query OK ...
```

Assume `tblOne` does not exist; it was already removed by the first command. Still no errors happen. Using the IF EXISTS option, the DBMS checks for the table names typed in command line, if they exist they drop if not, the command is ignored.

The complete list of errors related to the data definition commands of MySQL is shown in Table 5-3. The identification for the data definition errors consists from DD, stands for data definition, plus a number.



ID	Error Name	Explanation
DD01	Create database	Database already exists
DD02	Alter database	Database does not exist
DD03	Drop database	Database does not exist
DD04	Use database	Database does not exist
DD05	Create table	Table already exists
DD06	Create table / column name	Duplicate column name (s)
DD07	Create table / data type	Does not support data type for one or more column
DD08	Create table / constraint	Does not support one or more constraint
DD09	Create table / reference table	The referenced table (s) does not exist
DD10	Create table / reference table / primary key or index key	The referenced table (s) does not have primary key nor indexed field(s)
DD11	Create table ... select	Any error in select query in the command
DD12	Alter table / add column	Column already exists
DD13	Alter table / add index	Column does not exist
DD14	Alter table / add constraint	Constraint does not supported
DD15	Alter table / add primary key	Primary key already exists
DD16	Alter table / primary key column (s)	The primary key column (s) does not exist
DD17	Alter table / add foreign key	The referenced table (s) does not exist
DD18	Alter table / foreign key / data type	Data types of foreign key in child table and primary key in parent table does not match
DD19	Alter table / foreign key data	The domain data does not match
DD20	Alter table / foreign key / primary key or index key	The referenced table (s) does not have primary key nor indexed field(s)
DD21	Alter table / change column	Column does not exist
DD22	Alter table / modify column	Column does not exist
DD23	Alter table / drop column	Column does not exist
DD24	Alter table / drop primary key	Primary key does not exist
DD25	Alter table / drop index	Index does not exist
DD26	Alter table / drop foreign key	Foreign key does not exist
DD27	Alter table / enable keys	Syntax
DD28	Alter table / disable keys	Syntax
DD29	Alter table order by column name	The referenced column does not exist
DD30	Alter table / rename	Syntax
DD31	Rename table	Table does not exist
DD32	Describe table	Table does not exist
DD33	Drop table	One or more tables referenced in the command does not exist
DD34	Create view	The view already exists
DD35	Create view / table, view	The referenced table (s) or the referenced view (s) does not exist
DD36	Create view / temporary table	The view references temporary table (s)
DD37	Alter view	The view does not exist
DD38	Drop view	The view does not exist

Table 5-3 MySQL Data Definition Errors (SQL-DDL)

5.5.11 Errors—SQL-DML

The data manipulating commands of MySQL are used to enter, update, retrieve, and delete user data within database tables. The SQL-DML statements are an important part on MySQL and cover the following commands:

- INSERT ...
- SELECT ...
- UPDATE ...
- DELETE ...
- TRUNCATE ...

Errors in SQL-DML commands are further classified into three parts: SELECT, FROM, and WHERE. Some groups also include errors in sub parts after the WHERE condition in a query, i.e. GROUP BY, HAVING, and ORDER BY.

We have summarized the data manipulation errors of MySQL in Table 5-4 at the end of this discussion. The unique ID for these errors is combined from 'DM', stands for 'data manipulation', and two digits started from '01'.

5.5.12 Errors—INSERT Data

The 'INSERT INTO ... SELECT ...' command in MySQL 3.23 always has IGNORE enabled. The newer versions of MySQL (4.0.1 and up) do not ignore errors. Instead, when errors arise in these versions the database stops inserting data. To solve problems in these new versions of MySQL, users need to use the IGNORE option in their queries.

If a value to a unique field or an index field is inserted for many times, errors occur. To prevent such kind of errors we can add the 'ON DUPLICATE KEY UPDATE ...' option to the INSERT command. This option can be used with the 'INSERT ... SELECT' command.

5.5.13 Errors—INSERT NULL Values

If a NULL value is inserted into a column that has 'NOT NULL' constraint, two different events ensue: first, if the INSERT command is for one record, an error occurs. Second, if the INSERT command is for multiple records or the 'INSERT INTO ... SELECT ...' command is used, the implicit default values are entered for each column. For example, a '0' zero is inserted for numeric columns, an ' ' empty string is inserted for string type columns and zero is inserted for date/time columns.

5.5.14 Errors—IGNORE used with INSERT

The IGNORE keyword can be added as an option to the INSERT command in MySQL. This option prevents errors and causes MySQL to ignore errors when inserting records of data into database tables. This option only ignores errors, which are related to inserting data. These errors are changed to warnings. Other kinds of errors, such as syntax errors and logic errors are not influenced by this option.

For example, if an INSERT command is implemented on data which duplicates a unique index field in a table and the IGNORE option is not used errors will occur and data will not be entered into the table. This is important for multi-line insert operations. In a multi-line insert if an error occurs, from that point no other data entry is done. However, if the IGNORE option is included, in a multi-line INSERT command, if values are mismatched, no errors will occur; and the values that are valid for entry, before and after the invalid values, will normally be inserted into the database tables.

5.5.15 Errors—Retrieve Data

The MySQL data retrieve command is the **SELECT** query. By the **SELECT** query, we can find and retrieve any part of data from databases. The database queries are consisted from three main clauses i.e. **SELECT**, **FROM** and **WHERE**. Errors can occur in any of these parts. In general, MySQL statements are influenced by the six clauses in queries. These clauses include the three parts of a MySQL command plus **GROUP BY**, **HAVING** and **ORDER BY** [26] keywords in a query. Data retrieval commands of MySQL can have errors in any of the six mentioned parts within a query. Each error occurrence can be semantic, syntactic, or logical.

5.5.16 Errors—Subqueries

Subqueries are used to retrieve data from database tables, queries, and views. Using subqueries, we can create complicated queries in an easy to understand format. Subqueries give more facilities to MySQL users to manipulate data within databases. Similar to most of other parts in MySQL, subqueries also can have errors.

There are some errors that apply only to subqueries in databases. These errors are shortly explained and exemplified below:

- Number of columns from subquery

If a subquery is used as an operand in an outer query, it should reference exactly one column with a specific value. The value has to be determined by the **WHERE** clause within the subquery. The following examples explain this:

```
-> SELECT (SELECT colOne, colTwo FROM tblTwo) FROM tblOne;
```

```
-> ERROR ...
```

In such cases an error occurs. Here, the result of the subquery should be a single value taken from one column.

```
-> SELECT (SELECT colOne FROM tblTwo) FROM tblOne;
```

-> ERROR ...

Again an error occurs. The result of the subquery is now limited to one column which is okay. The problem is with the resulting values of this query. If 'tblTwo' has one record, then there will be no error; otherwise the result should be limited to one value. This is explained in the following example.

```
-> SELECT (SELECT colOne FROM tblTwo WHERE colTwo = '112')  
FROM tblOne;
```

-> Query OK ...

The results of the main query and the subquery are both correct.

- Number of rows from a subquery

When a subquery is used as an operand in another query, the result of the subquery should be one row. If they are more than one rows, then an error will occur. Limiting the result of a query to one row also called **SELECTION** and can be done by using the **WHERE** clause in that query. This was illustrated in the previous example. We also have other methods that use the result of a subquery within another query.

The following examples show the use of results of a subquery with limited values:

```
-> SELECT * FROM tblOne WHERE colOne = (SELECT colOne FROM
tblTwo);
```

-> ERROR ...

An error occurs. Column one from table one is compared with the column one from table two. Here, one value or one row is needed to be compared with column one from table one's value. To make this query work and prevent an error, the example should be changed as follows:

```
-> SELECT * FROM tblOne WHERE colOne = ANY (SELECT colOne
FROM tblTwo );
```

-> Query OK ...

This query runs and no errors occur. In this query, the value of column one from table one is compared to any value from column of table two. If they match, the result of the main query is printed.

- Table use in subquery

Subqueries can be used to update, delete, and retrieve data. Data comparison can be done and data can be updated by subqueries. When a subquery is used for update, table reference is important. A table, which is used to be updated, can not be used in a subquery in the same command. If this is done an error occurs.

In the following example usage of the same table for update and subquery reference is prohibited. If this query runs, an error occurs.

```
-> UPDATE tblThree SET colOne = (SELECT MIN(colTwo) FROM
tblThree);
```

-> ERROR ...

An error results from this query. Table 3 is used to be updated and referenced in the subquery. This is not allowed.

5.5.17 Errors—Temporary Tables

Temporary tables can be created by users in a MySQL database. Syntax for creating temporary tables is exactly the same as for base tables. The only difference is with the addition of the **TEMPORARY** keyword in this command. The following example shows the syntax for creating a temporary table named 'tmpTable1' in the active database.

- CREATE TEMPORARY TABLE tmpTable1 (ID INT, Name CHAR(10));

Temporary tables are dropped when a session is closed or connection to the Server is lost.

Temporary tables store data. Data can be retrieved from these tables. We can use more than one temporary table in one query. Similarly, we can use a combination of base tables and temporary tables in a single query. However, we can not use a temporary table more than once in a single query. The following example causes an error in MySQL.

-> SELECT * FROM tmpTable1, tmpTable1 AS TableOne;

-> ERROR ...

An error occurs. The tmpTable1 is a temporary and can not be used twice in one query.

5.5.18 Errors—UPDATE Data

Errors occur when user data changes are made to a table. These changes can be through `INSERT` statements or by `UPDATE` commands. The changes might violate values of primary keys, unique keys, indexes, or foreign keys within database tables. If a transactional storage engine is used for a table, such errors will automatically roll back. In addition, if a non transactional storage engine is used, in case of any error or violation the update process terminates.

The `UPDATE` command may cause errors in some special cases. For example, the data update in the following command causes an error and will not be completed, the `ID` field is unique within `tblFive`:

```
-> UPDATE tblFive SET ID = ID + 1;
```

```
-> ERROR ...
```

This update will not take place while the `ID` field stores unique values.

To prevent these kinds of errors, we can set the updating order by using the `ORDER BY` option. This option can be expanded by using the `ASC` (ascending) or `DESC` (descending) options. `ASC` is the default order for this command and may be omitted. This option was added from MySQL 4.0.0. An `UPDATE` can be done in descending order by using `DESC`, as follows.

```
-> UPDATE tblFive SET ID = ID + 1 ORDER BY ID DESC;
```

```
-> Query OK ...
```

This update operation on table five will take place. Values of the `ID` field will be updated in reverse order.

5.5.19 Errors—IGNORE used with UPDATE

MySQL supports the IGNORE keyword to be used in combination with the UPDATE command. If this keyword is used, even if an error or violation happens with constraints, the update process will not stop. Only the error records will be ignored and the remaining records will be updated.

The row actually affected by update commands are shown after implementing each command. There are also some commands that return a number of warnings and errors. For example, in MySQL we can use 'SHOW ERRORS' and 'SHOW WARNINGS' commands. These commands were added and became applicable in recent releases of MySQL 4.1 and later.

5.5.20 Errors—DELETE Data

The DELETE command in MySQL is used to delete data from existed tables in databases. This command supports multiple options. The DELETE command works like a query in a database. Queries are used for different purposes in databases. Queries explicitly do read only operations on data stored in database tables. In contrast, the DELETE command removes data from database tables. We can use different conditions to use this command.

The following is the syntax and example of DELETE command in MySQL:

- DELETE FROM TableReferences WHERE conditions

-> DELETE FROM tblOne;

-> Query OK ... # All records from 'tblOne' are deleted.

DELETE supports some modifiers by which this command can be used in better ways. IGNORE is one of the modifiers for the DELETE command and can be added as an option to the DELETE command.

5.5.21 Errors—IGNORE used with DELETE

The 'IGNORE' option was added starting from MySQL 4.1.1 and uses as a modifier to the DELETE command in MySQL. This option prevents errors and causes MySQL to ignore errors when deleting rows of data from database tables. This option only ignores errors, which are related to deleting data. These errors are changed to warnings. Other kinds of errors, such as syntax errors and logic errors are not influenced by this option.

The complete list of errors related to the data manipulation commands of MySQL is shown in Table 5-3 and Table 5-4. The identification for the data manipulation errors consists from 'DM', stands for data definition, plus a number.

ID	Error Name	Explanation
DM01	Insert	Syntax
DM02	Insert / table	Table does not exist
DM03	Insert / column	Column order is not correct
DM04	Insert / column (s)	Column (s) does not match or does not exist
DM05	Insert / duplicate data	Data items for unique columns are repeated
DM06	Insert / delayed for table (s)	Table (s) are dropped prior to insertion
DM07	Insert / delayed for view (s)	Could not be implemented
DM08	Insert / select	Duplicate data items are inserted
DM09	Insert / NULL values	Insert NULL values into NOT NULL field (s)
DM10	Replace	Replace data into a table and select from the same table
DM11	Select	Syntax
DM12	Select *	Use of * wild card, while all columns are not expected
DM13	Select / column name (s) miss	No column (s) are referenced
DM14	Select / unknown column	The referenced column (s) within a table does not exist
DM15	Select / comma miss between column names	The comma character between the column reference is missed One column is treated as alias name for another
DM16	Select / column alias name (s)	Use of column alias name in a where condition

ID	Error Name	Explanation
DM17	Select / from	The referenced table (s) does not exist
DM18	Select / repeated table reference	One table is repeated more than one time in the reference list
DM19	Select / additional table reference	Additional table (s) is referenced, but a clear join or where condition does not set
DM20	Select / table name miss	Table name (s) is missing after the from clause
DM21	Select / comma miss between table names	The comma character between table names is missed One table name is treated as alias name for another
DM22	Select / where / conditions	No logical condition is set after the where clause
DM23	Group by / having / order by	The order of using these three clauses is not correct
DM24	Select / group by	Use a group function, but the group by is missed
DM25	Select / having	The having clause refers to an unknown column name
DM26	Select / order by	The order by clause is applied to non-existed column (s)
DM27	Select / subquery (s)	Syntax - all the mentioned errors for select queries
DM28	Select / subquery / table use	Table (s) does not exist
DM29	Select / from / subquery	The subquery after the from clause does not aliased
DM30	Select / from / subquery column reference	The subquery after the from clause has duplicate column references
DM31	Select / subquery / number of columns	Number of columns resulted from subquery are not expected
DM32	Select / subquery / number of rows	The result of the subquery shows more than one row, while one row is expected for its results
DM33	Select / temporary table (s)	Use of temporary table many times in one query
DM34	Select / union / select	Tables are not union compatible
DM35	Select / join / on condition (s)	Add conditions in the 'ON' part of a join rather than where
DM36	Select / join / alias table (s)	Set alias name for a table, but use original name of that table in join conditions
DM37	Select / where / table	The referenced table (s) in the where does not exist
DM38	Select / where / aggregate function (s)	Use of any aggregate function after the where condition
DM39	Select / computed columns	The result of the column operation does not match
DM40	Select / avg() function	No value matches, and returns NULL
DM41	Select / min() function	No value matches, and returns NULL
DM42	Select / max() function	No value matches, and returns NULL
DM43	Select / count() function	Implemented on a NULL column, un-expected results shown
DM44	Select / instr() / main string binary checked	The main string is binary checked and mismatched the sub string - different letter cases
DM45	Select / instr() / sub string binary checked	The sub string is binary checked and mismatched the main string - different letter cases
DM46	Select / instr() / both strings binary checked	Both strings / arguments of the function are binary checked and mismatched - different letter cases
DM47	Select / instr() / NULL values	Any of the argument strings of the function is NULL the result is NULL

ID	Error Name	Explanation
DM48	Select / locate() / sub string binary checked	The sub string is binary checked and mismatched the main string - different letter cases
DM49	Select / locate() / main string binary checked	The main string is binary checked and mismatched the sub string - different letter cases
DM50	Select / locate() / both strings binary checked	Both strings / arguments of the function are binary checked and mismatched - different letter cases
DM51	Select / locate() / NULL values	Any of the argument strings of the function is NULL the result is NULL
DM52	Select / replace() function	Any difference in letter cases between all three arguments: string, from string and to string
DM53	Select / strcmp() / binary check	Any or both of the two arguments of this function are binary checked - different letter cases
DM54	Select / strcmp() / Null values	Any of the argument strings of the function is NULL the result is NULL
DM55	Select / substring_index() function	Any difference in letter cases between the first two arguments: string and delimiter
DM56	Select / substring_index() / Null values	Any of the first two arguments of the function is NULL the result is NULL
DM57	Select / like / string comparison	One or more operands of the like comparison operator is binary checked - different letter cases
DM58	Select / like / NULL values	Any of the operand strings of the operation is NULL the result is NULL
DM59	Select / rlike / string comparison	One or more operands of the rlike comparison operator is binary checked - different letter cases
DM60	Select / rlike / NULL values	Any of the operand strings of the operation is NULL the result is NULL
DM61	Select / regexp / string comparison	One or more operands of the regexp comparison operator is binary checked - different letter cases
DM62	Select / regexp / NULL values	Any of the operand strings of the operation is NULL the result is NULL
DM63	Select / the same column names or aliases / order by	Use of similar column names with the order by clause in one query (ambiguous columns)
DM64	Select / limit missed argument	Use of limit condition with no argument in a query
DM65	Select / limit negative argument	Use of negative number as argument for the limit condition
DM66	Update table	The referenced column (s) within a table does not exist
DM67	Update / increment values	The indexed columns can not be incremented
DM68	Update / decrement values	The indexed columns can not be decremented
DM69	Update / subquery	The same table updated and used in subquery
DM70	Delete data from table (s)	Table (s) does not exist
DM71	Truncate table / not exist	Table (s) does not exist
DM72	Truncate table / locked	The truncated table is locked

Table 5-4 MySQL Data Manipulation Errors (SQL-DML)

5.6 Errors—Transaction Control and Locking Tables

In MySQL statements, local transactions and table locks can be used. These commands are used safely to update, delete or insert user data within database tables in a specific session. **TRANSACTION** or **LOCK TABLES** can be set by users individually. These commands can only be used for tables, which use transaction safe DBMS engines.

The transaction control commands of MySQL are used to keep safe the data update, insert and delete processes. The SQL-TCL covers the following commands:

- START TRANSACTION ...
- BEGIN ...
- ROLLBACK
- COMMIT
- SAVEPOINT ...
- ROLLBACK TO SAVEPOINT ...
- RELEASE SAVEPOINT ...



We have summarized the transaction control errors of MySQL in Table 5-5 at the end of this discussion. The unique ID for these errors is combined from TC, stands for 'transaction control', and two digits started from 01.

In each transaction, one or more **SAVEPOINTS** can be set. Users can commit their changes, or they can roll back their changes up to a specific savepoint.

5.6.1 Errors—SAVEPOINTS

Using savepoints for transaction safe tables and using the InnoDB in MySQL DBMS, started from MySQL versions 4.0.14 and 4.1.1. Syntax for setting savepoints in MySQL follows:

- SAVEPOINT Identifier

ROLLBACK TO SAVEPOINT Identifier

In one session, more than one savepoint can be set. A user can roll back from each stage to any savepoint that has been set. As a general rule, the COMMIT keyword stores and implements all changes given by users. If a ROLLBACK is done to a SAVEPOINT that is not defined, then an error occurs. The ROLLBACK command without arguments never produces errors. Errors occur only if the ROLLBACK refers to an unknown savepoint identifier as its argument.

5.6.2 Errors—LOCK TABLES

The LOCK TABLES command is used explicitly for keeping transactions within database tables. This command also speeds up the update process on MySQL tables. The following is the syntax for the LOCK TABLES command:

- LOCK TABLES table_reference READ, table_reference WRITE ...
- UNLOCK TABLES # Releases tables to their normal stages

If a table is locked with a specific right and a user wants to access that table, errors occur. The following examples show the potential errors when using the LOCK TABLES command.

-> LOCK TABLES tblOne READ;

-> Query OK...

This command locks all other tables, only the **tblOne** can be read in the current session.

-> SELECT * FROM tblOne;

-> Query OK...

Since **tblOne** is released and can be read, the command does not cause any errors.

-> SELECT * FROM tblTwo;

-> ERROR ...

An error occurs. Table **tblTwo** is locked for reading.

When tables are locked, one table can not be used twice in one query. Tables are accessible only through their names or their alias names. The following examples show these cases and causes of potential errors in these commands:

-> LOCK TABLES tblOne WRITE, tblOne AS t1 READ;

-> Query OK...

This command locks all other tables, only **tblOne** accepts data entry and the same table can be read by its alias name **t1** in the current session.

-> INSERT INTO tblOne SELECT * FROM tblOne;

-> ERROR ...

An error occurs. Table tblOne is locked for reading through its own name.

-> INSERT INTO tblOne SELECT * FROM tblOne AS t1;

-> Query OK...

No error occurs. Table tblOne is locked for data entry by its own name and this table is locked for reading by its alias name 't1'.

The complete list of errors related to the transaction control commands of MySQL is shown in Table 5-5. The identification for transaction control errors is 'TC' followed by a two digit number.

ID	Error Name	Explanation
TC01	Start transaction / begin	Syntax
TC02	Start transaction / begin / option	Use of incorrect or not supported option (s)
TC03	Start transaction / db engine	Implement on nontransaction safe table (s)
TC04	Start transaction / rollback	Issue rollback after changes made to nontransactional tables
TC05	Start transaction / database mixed engines	Implement transaction on table (s) that use both transaction safe and nontransaction safe database engines
TC06	Rollback / create database	Try to rollback the create database command (s)
TC07	Rollback / alter database	Try to rollback the alter database command (s)
TC08	Rollback / drop database	Try to rollback the drop database command (s)
TC09	Rollback / create table	Try to rollback the create table command (s)
TC10	Rollback / alter table	Try to rollback the alter table command (s)
TC11	Rollback / drop table	Try to rollback the drop table command (s)
TC12	Rollback / rename table	Try to rollback the rename table operation
TC13	Rollback / create view	Try to rollback the create view command (s)
TC14	Rollback / alter view	Try to rollback the alter view command (s)
TC15	Rollback / drop view	Try to rollback the drop view command (s)
TC16	Rollback / truncate table	Try to rollback the truncate table operation (s)
TC17	Rollback / DDL commands	Try to rollback after any data definition command is run SQL DDL commands could not be rolled back; additionally, these commands do implicit commit Any changes before these commands are committed and are not rolled back

ID	Error Name	Explanation
TC18	Rollback / transaction control and locking statements	Try to rollback after any transaction control or locking statements are run Any changes before these commands are committed and are not rolled back
TC19	Set autocommit = 1 or 0	Setting autocommit to a non supported digit
TC20	Begin ... end	Syntax
TC21	Savepoint / rollback to savepoint	Syntax
TC22	Rollback to savepoint	The savepoint does not exist or released

Table 5-5 MySQL Transaction Control Errors (SQL-TCL)

5.7 Errors—Using NULL Values

In MySQL numeric functions and arithmetic operations, if incorrect values are used errors occur. In most of these cases, NULL values are shown in the result line of those queries. The following cases happen with the incorrect use of numeric functions and arithmetic operations in MySQL.

Division by zero produces NULL values in results. The following examples show this result:

```
-> SELECT 400 / 0;
```

```
-> NULL # Returns NULL
```

```
-> SELECT 400 / (1-1);
```

```
-> NULL # Returns NULL
```

```
-> SELECT 400 DIV 0;
```

```
-> NULL # Returns NULL
```

All mathematical functions in MySQL return NULL when any error occurs.

-> SELECT MOD(1, 0);

-> NULL # Returns NULL

As stated in the arc sine and cosine of X, if the X range is not between -1 and 1 an error occurs and NULL is shown in the result:

-> SELECT ASIN(1.1);

-> NULL # Returns NULL

-> SELECT ASIN(1);

-> 1.5707963267949 # Returns the expected result (no error)

-> SELECT ACOS(1.1);

-> NULL # Returns NULL

-> SELECT ACOS(0);

-> 1.5707963267949 # Returns the expected result (no error)

Similarly, if unexpected values are requested as cotangent for an angle, errors occur.

-> SELECT COT(0);

-> NULL # Returns NULL

-> SELECT COT(-8);

-> 0.14706506394948 # Returns the expected result (no error)

If any argument in the CONV() function is NULL, the result will be NULL.

```
-> SELECT CONV(12, 16, NULL);
```

```
-> NULL # Returns NULL
```

```
-> SELECT CONV(12, NULL, 2);
```

```
-> NULL # Returns NULL
```

```
-> SELECT CONV(12, 16, 2);
```

```
-> 10010 # Returns the expected result
```

The square root of any negative number causes NULL values in the results.

```
-> SELECT SQRT(-4);
```

```
-> NULL # Returns NULL
```

```
-> SELECT SQRT(4);
```

```
-> 2 # Returns the expected result (no error)
```

5.8 Summary

In this chapter we described common MySQL errors and discussed case sensitivity in searches and in MySQL modes. We have also classified and explained MySQL common errors in three groups: data definition, data manipulation and transaction control errors. Examples for each group are included in this chapter. A list of possible errors is attached at the end of each part. The data definition errors list

includes 38 errors; the data manipulation errors list shows 72 errors, and the transaction control errors list has 22 errors. The total number of MySQL common errors in this research exceeding 132. This chapter concludes by the usage of NULL values and their proneness to errors.



Chapter 6

Research Methodology

In this chapter we explain the methods by which we aim to achieve our goals and answer the research questions. There are three main activities in this research. First, an extensive study of the database field is done and MySQL software is studied. Case sensitivity in MySQL and its proneness to errors in MySQL commands is analyzed and evaluated under various MySQL modes. Second, we analyze portions of MySQL code that are generated by students. As a result of these two activities, the third main activity is to collect, analyze and group common MySQL errors and the frequency of those errors in a fixed number of cases.

SQL and MySQL have been studied quite intensively. MySQL functions and possibility of error occurrence in those functions are explained in Chapter 3. SQL DDL commands including database definition commands and table definition commands and the likelihood of errors occurring is another topic in the theoretical part of our work.

Chapter 5 has highlighted the kinds of errors that can be made by database developers using MySQL. In order to investigate which of these errors novice developers are prone to make and how frequently these errors are made the following was done. We want to know the exact occurrence and the frequency of the studied errors of MySQL in real life. For this we have evaluated students' generated SQL code that is run on different sample queries.

6.1 Evaluating Students' MySQL Code

Our undergraduate students do two database courses. The first course covers database concepts and the second course applies MySQL. These courses are mandatory at Kabul University in Afghanistan and are attended by about 60 students in each course. We have been teaching these two courses for the last five years. Students are divided into groups of 2 to 3 and each group designs and implements a database project. To complete this research we have asked different queries from these students. Then, the produced queries are checked for occurrence of common errors and mistakes that students make.

6.1.1 Students' Group Project

A medium sized database, which has around 20 tables, indexes, relationships and views, is developed by these students annually. During the first course (Database Concepts), they start developing this database from scratch. Their instructor gives the introductory information to students regarding the field, for which the database is planned. Students write the Universe of Discourse (UoD) for that topic and develop business rules as well. Based on those business rules, the Entity Relationship Diagram (ERD) for the database is designed. Then, initially the proposed database is implemented in the Microsoft Access database management system.

The MySQL is taught in the second database course at Kabul University. In the MySQL course, students continue their project from the previous database course. The pilot test of the project has been passed and the database has been implemented in Access. In this course, student groups implement the database using MySQL. This includes:

- defining database and tables include data types for each field

- setting primary keys, indexed fields and constraints
- creating relationships between tables
- defining the Referential Integrity Constraints (RICs) and cardinality, based on the business rules determined in the first stages of the project
- inserting sample data records into database tables

6.1.2 Our Database - the 'HOSPITAL'

One of the goals of this research is to find and record common MySQL errors that students make. The HOSPITAL database is chosen for student work in database courses. This database is developed by our students during their class assignments over two semesters.

MySQL code for the final copy of the HOSPITAL database may be found in Appendix A. All the database definition and table definition commands in MySQL syntax are written there. Primary keys, indexes, constraints and relationships between tables are noted. The data entry commands, which include sample data for the database, are also shown in Appendix A.

These relationships are based on business rules that students develop practically during their work in the 'Database Concepts' course. We teach them and help them to do this work exactly like the Access HOSPITAL database that is explained in the Appendix A.

6.1.3 Using the 'HOSPITAL' Database

Students in two separate classes implement the HOSPITAL database. The total number of students in these classes exceeds one hundred. A set of MySQL commands is requested from students. Each student wrote the requested queries and submitted their results individually. Furthermore, the code segments generated by students are used and analyzed for potential errors. After analyzing the errors, their final results are frequently classified into groups and recorded. This classification is based on the type of the errors and the commands where the errors occur.

6.2 Data Collection (What, How, Who)

Data on MySQL code is extracted from students' practical work in database courses. The collected data is analyzed to determine potential errors that may occur at run time. The results of analyzing that data is further classified into groups. The grouped data is checked for three types of errors: syntax, semantic and logic. The final results are shown in Chapter 7 of this document.

6.2.1 The Collected Data

We have collected data from students' practical work in database courses at Kabul University. Tens of different queries are requested from students. The query requests were in simple text and should be implemented on the HOSPITAL sample database. In response to the requested queries, students were writing SQL code. This activity continued for one semester, which is around 16 weeks of lecture.

There are two options for implementing queries in our courses. One option lets students run their queries before submitting the final answer sheets. The other option is to use a text editor or plain paper and type or write the SQL code. For this research, we encouraged our students to use the second option and submit their work

before running the code in MySQL. Therefore, most of the data items are in hard copies and some are in soft copies written in a text editor.

6.2.2 The Collection of the Data

The data needed for this research is SQL code that produces errors. One of the main objectives for this research was to find common errors and probable mistakes that students make during a SQL course. Therefore, data was collected for this purpose. We used different methods that include physical collection of errors harvested from written/printed papers and collected copies of students' SQL software.

The instructor gives query requests to students during lab hours. Students were responsible for writing SQL code to reflect the requested queries. The query requests are based on the HOSPITAL database, which they developed themselves. At the end of the lab hour, students handed over their work written or typed on paper. Then, the collected data was analyzed and student mistakes were found by hand and SQL errors were collected for this research. The frequencies of errors that occurred were recorded for each typical coding error.

The instructor(s) distribute query requests on the HOSPITAL database to students. Students work on the requested queries, develop queries and prepare a soft copy of their work. The soft copies are submitted to their instructors(s) through network or by email before a specific deadline. These two were the methods of collecting data used in this research.

6.2.3 The Collector of the Data

The author of this document collected most the data for this research. He is an Assistant Professor in the Department of Computer Science in Kabul University. Database courses are taught in that department to 3rd year students. For the last five

years, this researcher taught the database courses at Kabul University. For a period of two years code generated by students for a number of queries was collected, analyzed and checked for mistakes.

6.3 Summary

We have explained the methods used to find common MySQL errors in this chapter. The chapter started with a study of databases and MySQL and also covered how we compiled a list of common MySQL errors. It also included experiments in MySQL. The MySQL code generated by students and the probability of making an error in that code is another part of this research. The methodology for finding errors was explained. We had also discussed the terms that: what the students were asked to do, what data was collected, how the data was collected and who was the collector of the data for this research.



Chapter 7

Results

This chapter starts by recapping MySQL error types as syntactic, semantic and logic errors. Student errors are checked and recorded in different scripts. The recorded errors are shown in Appendix D as raw data. The errors are described using the percentages of students that made the error and they are classified by types of error. The errors are also charted. The analysis is based on the percentage of matching errors. Student errors are marked by an identity code, which shows the type of an error. The total number of student errors, as a result of this research, is explained at the end of this chapter. In addition, an analytical chart explaining the total number of errors is included there. The chapter concludes with a summary of its contents.

Our research focuses on the MySQL errors that students are inclined to make in their class work. During this project, two classes of Kabul University were chosen. According to the lists of errors for each category explained in Chapter 5, we have checked and analyzed most of those errors in the work of students. All this was done statically. The sample queries are attached in Appendix B.

We chose 70 samples of code for occurrences of each error. For a couple of errors, which are more general, we have used 140 samples. The total number of checked samples reaches 6510. Some of the checks are implemented on an individual script and some other checks are grouped and each group is checked on a single command. The number of occurrences is recorded in tables for each error. Each occurrence is checked for syntactic errors, semantic errors and for logical errors. We have analyzed each error based on their three statuses, i.e. syntax, semantic and logic. A total

number of checks and error occurrence for each section is recorded in three tables: Table D-1, Table D-2, and Table D-3. The grand total for all the error occurrences errors based on three different statuses is recorded in Table D-4. A percentage of error occurrences, based on the raw data, is analyzed and shown in a series of tables indicating error occurrences in all three types of commands are shown in section 7.5. Accordingly, the analyzed report charts for the errors are shown in a series of figures in this chapter.

7.1 MySQL Error Types—Syntactic, Semantic and Logic

MySQL code like other languages in the Computer Science and Information Systems fields is error prone. Each error can be checked for syntactic, semantic or logic form. Syntactic errors in a command or code portion are deviations from the required syntax of that command in a language [26]. Missing a keyword, missing part of a keyword, even missing a letter from a keyword in a command are cases where syntax errors occur. Misspelling of commands is another issue for occurrence of syntax errors.

Semantic errors are another form of error in MySQL programming. These errors relate directly to the meaning of a command or to the meaning of the requested data by a command. Sometimes, a user types a syntactically correct sentence, but the program does not yield the expected results. This happens when the sentence or the command has a semantic problem. Semantic errors may occur in two forms in a command. First, the requested task seems correct but the query by itself is incorrect. Second, the query is correct but still there is no confidence in the results.

Logical errors are another type of error. Like the previous two forms, an error in MySQL code could be logical. Logical errors are usually seen at runtime. Before running a command, recognizing such errors is not an easy task. Statically analyzing

MySQL code to find logical errors seems a little tricky. Humans tracing the code, however, find this possible.

MySQL commands are generally categorized as data definition, data manipulation and data control commands. In the following, we have checked MySQL commands under each of the three categories and analyzed errors as syntax, semantic or logic forms. Results for each individual command are recorded and tabled.

7.2 Student Errors—SQL-DDL

The MySQL data definition commands can have errors. In Chapter 5, we have explained this category of MySQL errors. In this part, we have checked this group of errors against students' work. In this survey we encounter error occurrences for all three states: syntactic, semantic and logic. Percentages of data definition errors are shown in the following three tables. Some of the listed errors from Chapter 5, which rarely occur, are ignored for checking students' work. The error identifier shows the selected errors in our survey.

Table 7-1 shows the percentage of errors occurring in create, update and delete database and table commands in MySQL. The raw data for these cases is recorded in Table D-1, Appendix D. In this table we used 6 scripts for 10 cases. The first 3 errors are checked individually on different scripts. Four cases: DD05, DD06, DD07 and DD08 are checked on one script. The next two are checked on another script and the last case is checked on a different script.

ErrorID	Error Name	% Scripts with Errors	Error Distribution (%)		
			Syntax	Semantic	Logic
DD01	Create database	17	17	0	83
DD03	Drop database	9	67	0	33
DD04	Use database	11	100	0	0
DD05	Create table	13	67	33	0
DD06	Create table / column name	9	17	83	0
DD07	Create table / data type	4	67	33	0
DD08	Create table / constraint	4	0	100	0
DD09	Create table / reference table	6	50	50	0
DD10	Create table / reference table / primary key or index key	17	8	25	67
DD11	Create table ... select	60	40	26	33

Table 7-1 Percentages of Student Errors in create, update and delete Database and Table Commands

The above table is analyzed and its analytical chart is shown in **Figure 7-1**.

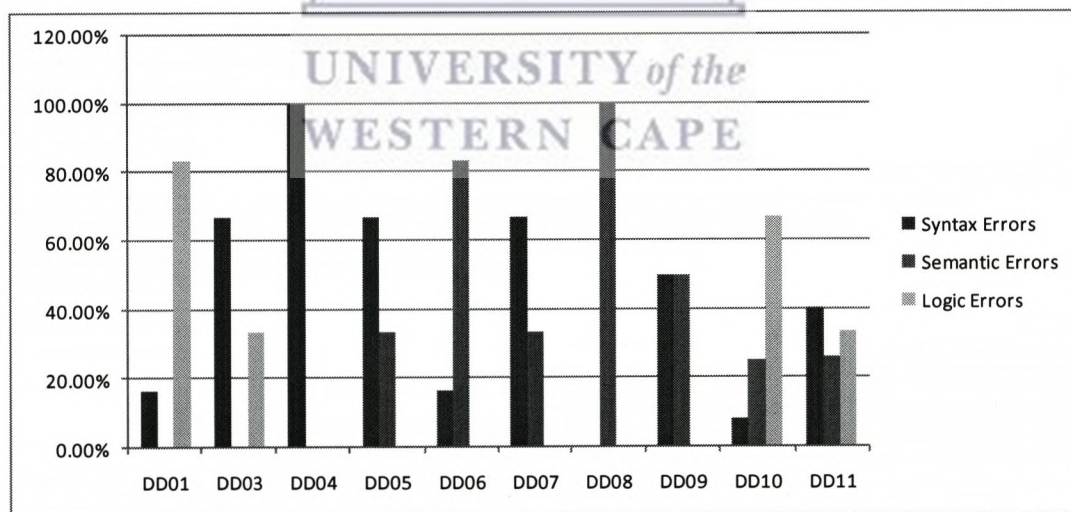


Figure 7-1 Analysis of Student Errors in create, update and delete Database and Table Commands

Another part of data definition errors are shown in Table 7-2. Similar to the previous table, the raw data for these cases is recorded in Table D-1, Appendix D. In this table we used 8 scripts for 11 cases. The first 2 cases are checked individually on different scripts. Four cases: DD17, DD18, DD19 and DD20 are checked on one script. The remaining 5 cases are checked on 5 individual scripts.

ErrorID	Error Name	% Scripts with Errors	Error Distribution (%)		
			Syntax	Semantic	Logic
DD12	Alter table / add column	11	13	63	25
DD15	Alter table / add primary key	24	24	12	65
DD17	Alter table / add foreign key	27	42	11	47
DD18	Alter table / foreign key / data type	31	18	32	50
DD19	Alter table / foreign key data	11	50	13	38
DD20	Alter table / foreign key / primary key or index key	11	13	25	63
DD21	Alter table / change column	17	25	17	58
DD22	Alter table / modify column	4	67	33	0
DD23	Alter table / drop column	14	80	20	0
DD24	Alter table / drop primary key	6	25	75	0
DD30	Alter table / rename	6	100	0	0

Table 7-2 Percentages of Student Errors in alter Table Command

Figure 7-2 shows the analytical report on percentage of error occurrence for the previous table.

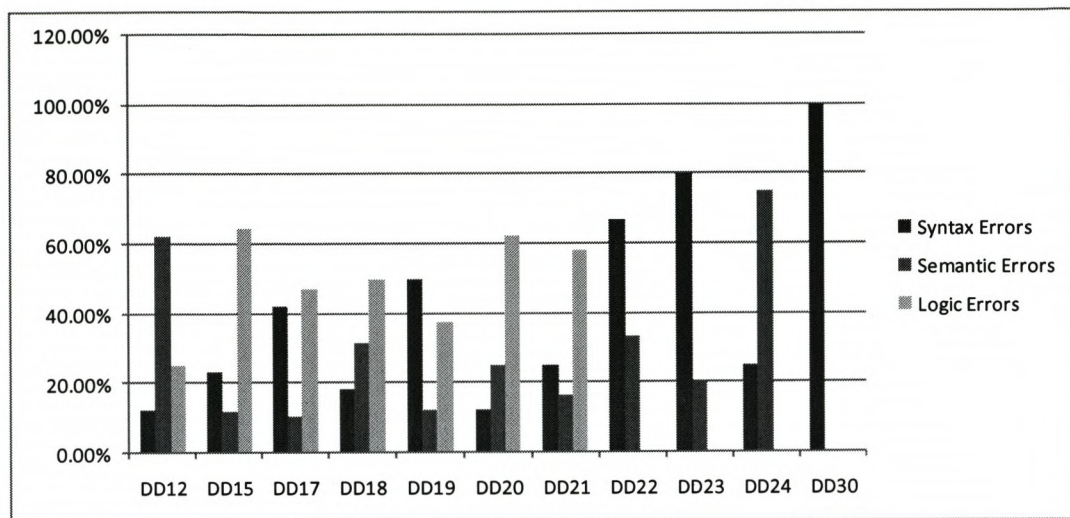


Figure 7-2 Analysis of Student Errors in alter Table Command

The last part of data definition errors are related to tables and views in databases. The percentage and results of these errors are shown in Table 7-3. The raw data for this table can be found in Table D-1, Appendix D. In this table we used 6 scripts for 7 cases. The first 3 cases are checked individually on different scripts. Two cases: DD34 and DD35 are checked on one script. The remaining 2 cases are checked on 5 individual scripts.

ErrorID	Error Name	% Scripts with Errors	Error Distribution (%)		
			Syntax	Semantic	Logic
DD31	Rename table	16	36	18	45
DD32	Describe table	11	13	38	50
DD33	Drop table	27	21	16	63
DD34	Create view	14	30	20	50
DD35	Create view / table, view	16	55	9	36
DD37	Alter view	6	75	25	0
DD38	Drop view	7	20	40	40

Table 7-3 Percentages of Student Errors in Table and View related Commands

The analytical results based on the percentage of error occurrence for previous table is shown in Figure 7-3.

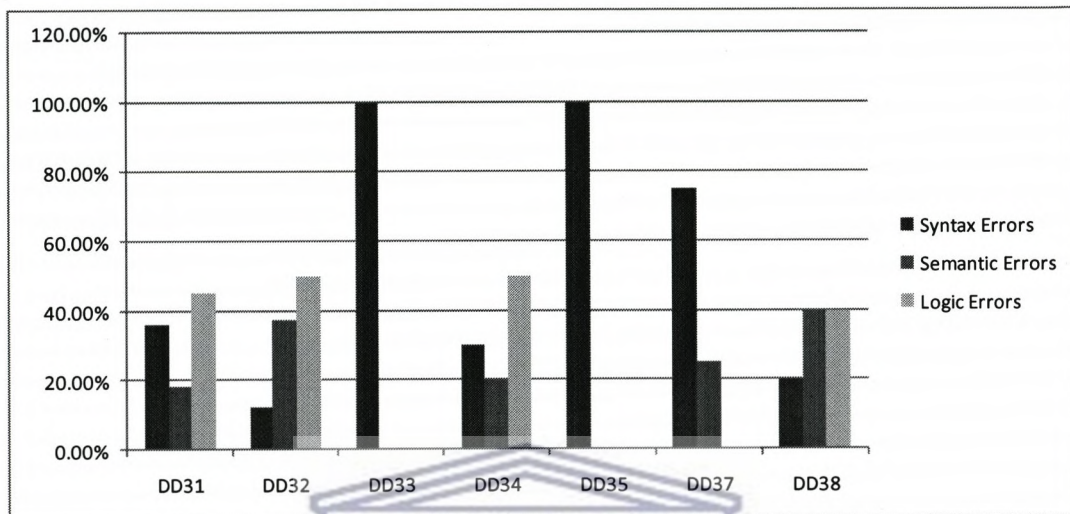


Figure 7-3 Analysis of Student Errors in Table and View related Commands

We have checked, analyzed and reported on the number and percentage of the students' MySQL errors for the data definition commands. The total errors in this section are 38 and we checked 28 out of this number in students' work. The other 10 errors occur rarely and they are ignored in this research. We have selected 20 scripts and checked the 28 cases of error occurrence in data definition commands. All these scripts are written by more than 70 students and we chose 70 samples out of them.

7.3 Student Errors—SQL-DML

The MySQL data manipulation commands form an important part of the database field. Many errors can occur in these commands. In Chapter 5, the MySQL DML errors are listed. Here, we check these errors in students' class work and match cases of this category of errors. The occurred errors are classified into three cases as syntactic, semantic and logic errors. The results for occurrences of these errors are

listed in Table D-2, Appendix D. Their percentages and the analytical report charts are explained shortly in the forthcoming paragraphs.

Table 7-4 shows the insert and replace commands and the percentage for the syntactic, semantic and logic error occurrence in these commands. The raw data for these cases is recorded in Table D-1 and Table D-2, Appendix D. In the table below error cases are checked individually on 3 different scripts.

ErrorID	Error Name	% Scripts with Errors	Error Distribution (%)		
			Syntax	Semantic	Logic
DM01	Insert	21	73	27	0
DM02	Insert / table	23	31	25	44
DM03	Insert / column	40	21	14	64
DM04	Insert / column (s)	34	42	13	46
DM05	Insert / duplicate data	14	30	20	50
DM08	Insert / select	14	30	20	50
DM09	Insert / NULL values	16	36	9	55
DM10	Replace	0	0	0	0

Table 7-4 Percentages of Student Errors in data entry Commands

Similarly, Figure 7-4 shows a histogram of error percentages of the insert and replace data entry commands in MySQL.

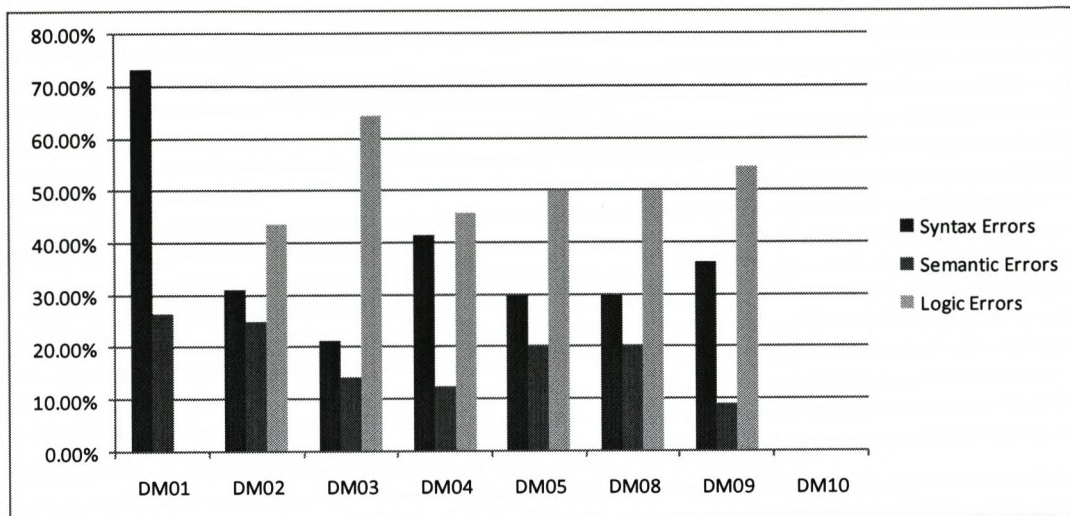


Figure 7-4 Analysis of Student Errors in data entry Commands

The following table lists the percentages of student errors in a SELECT query. The SELECT query is used to retrieve data from database tables. The raw data for these cases is recorded in Table D-1 and Table D-2 in Appendix D. In Table 7-5 we used 4 scripts for 10 cases. The first and the fourth cases in this table are checked individually in two different scripts. The second and third cases are checked in one query and the last 6 errors are checked in one script.

ErrorID	Error Name	% Scripts with Errors	Error Distribution (%)		
			Syntax	Semantic	Logic
DM11	Select	14	100	0	0
DM12	Select *	16	27	9	64
DM13	Select / column name (s) miss	11	38	0	63
DM14	Select / unknown column	10	29	14	57
DM15	Select / comma miss between column names	16	100	0	0
DM17	Select / from	26	22	11	67
DM18	Select / repeated table reference	9	17	50	33
DM19	Select / additional table reference	16	45	0	55
DM20	Select / table name miss	23	0	13	88
DM21	Select / comma miss between table names	14	30	70	0

Table 7-5 Percentages of Student Errors in data retrieve Commands—Part 1

The analytical result as percentage level of the occurred errors in data retrieval commands are shown in Figure 7-5.

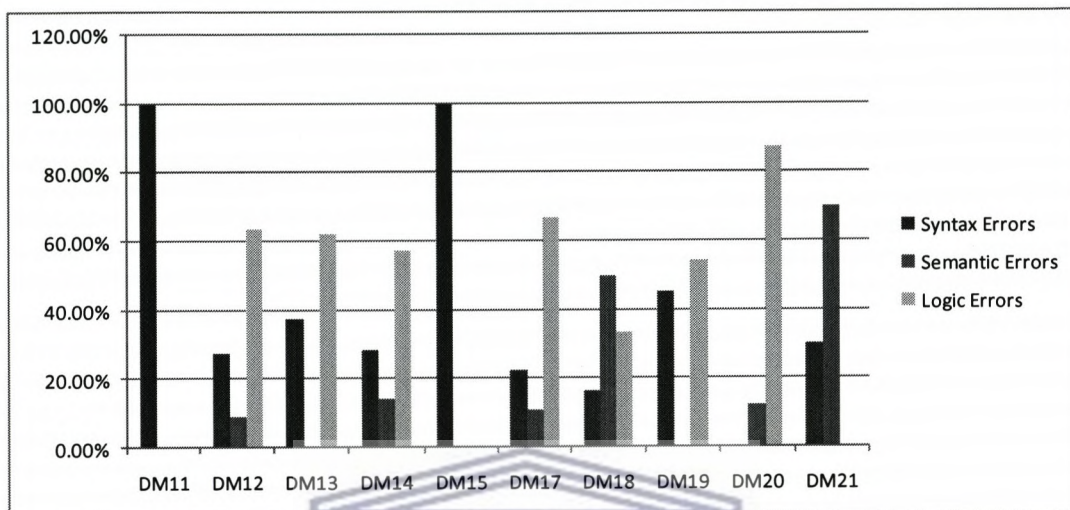


Figure 7-5 Analysis of Student Errors in data retrieve Commands—Part 1

Another part of data retrieval commands and error occurrence relate to other parts of SELECT statement. Also, subqueries and error occurrence in subqueries are explained in this part. Table 7-6 shows the error occurrence in students' work. The raw data for these cases are recorded in Table D-1 and Table D-2 in Appendix D. In the table below, we used 3 scripts for 10 cases. The first 5 errors are checked in one script. The sixth case is checked in another script. The last 4 error cases are checked on one script.

ErrorID	Error Name	% Scripts with Errors	Error Distribution (%)		
			Syntax	Semantic	Logic
DM22	Select / where / conditions	14	20	30	50
DM23	Group by / having / order by	10	14	0	86
DM24	Select / group by	17	25	17	58
DM25	Select / having	13	22	0	78
DM26	Select / order by	19	23	31	46
DM27	Select / subquery (s)	39	44	15	41
DM28	Select / subquery / table use	21	73	13	13

ErrorID	Error Name	% Scripts with Errors	Error Distribution (%)		
			Syntax	Semantic	Logic
DM30	Select / from / subquery column reference	20	57	29	14
DM31	Select / subquery / number of columns	13	33	11	56
DM32	Select / subquery / number of rows	20	29	36	36

Table 7-6 Percentages of Student Errors in data retrieve Commands—Part 2

The analyzed form of errors in previous table is shown in Figure 7-6.

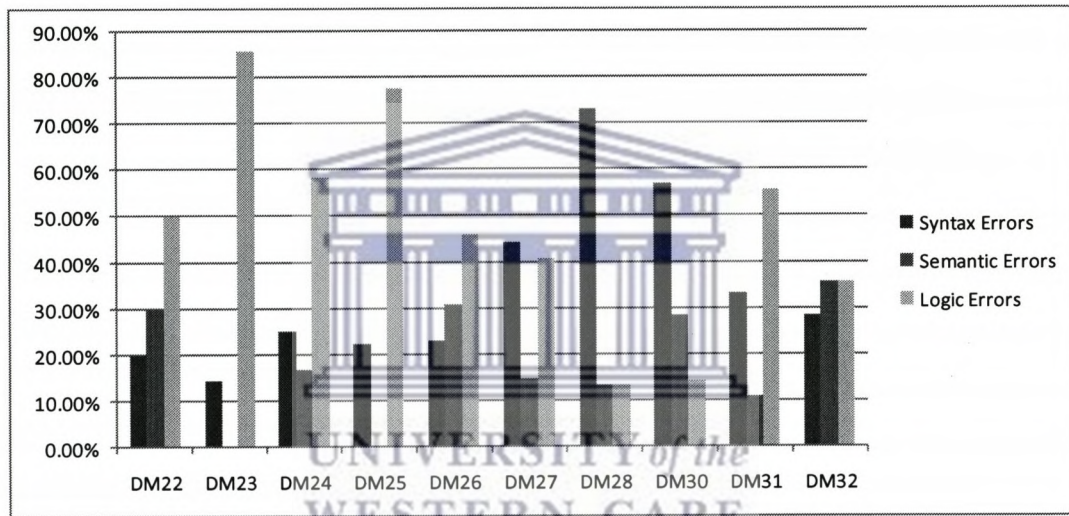


Figure 7-6 Analysis of Student Errors in data retrieve Commands—Part 2

We can combine different data sets using the **SELECT** command in MySQL. Similarly, we can implement data aggregation functions on a database's data in MySQL. Errors may occur in any parts of these commands in **SELECT** statement. Student errors are listed in Table D-2, Appendix D. Table 7-7 shows the percentage of these errors. Like most of other parts in this research, also this is checked on works of 70 students based on all 3 types of errors: syntactic, semantic and logic. In the bellow table, we used 7 scripts for 10 cases. The first 3 cases are checked

individually on 3 different scripts. The next 4 errors are checked in one script and the last 3 are checked on three different scripts.

ErrorID	Error Name	% Scripts with Errors	Error Distribution (%)		
			Syntax	Semantic	Logic
DM34	Select / union / select	14	40	0	60
DM36	Select / join / alias table (s)	11	25	13	63
DM37	Select / where / table	16	45	0	55
DM38	Select / where / aggregate function (s)	14	20	10	70
DM40	Select / avg() function	14	30	20	50
DM41	Select / min() function	10	14	0	86
DM42	Select / max() function	7	80	0	20
DM43	Select / count() function	0	0	0	0
DM52	Select / replace() function	13	22	78	0
DM55	Select / substring_index() function	10	43	57	0

Table 7-7 Percentages of Student Errors in data retrieve Commands—Part 3

Figure 7-7 shows the analytical results of Table 7-7.

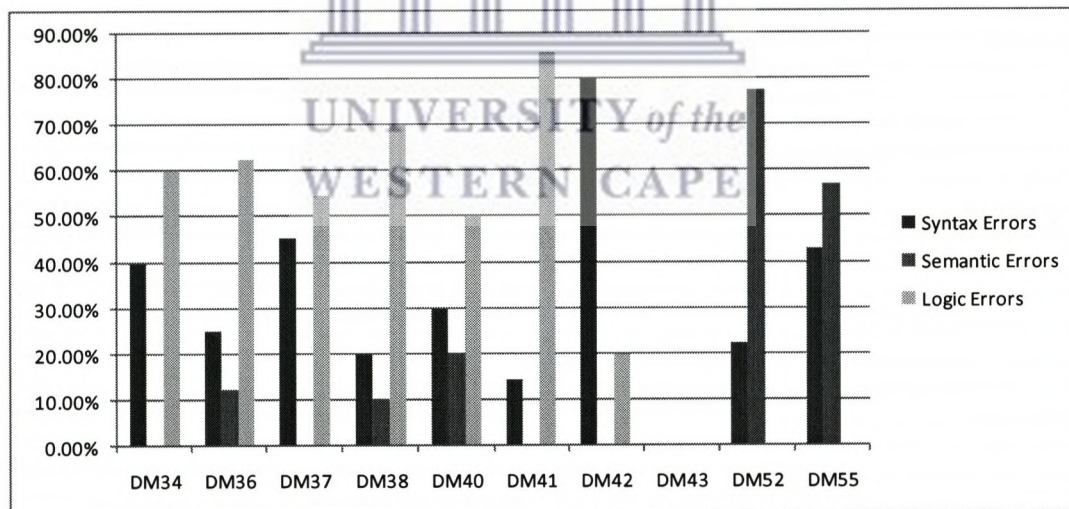


Figure 7-7 Analysis of Student Errors in data retrieve Commands—Part 3

The last part of data manipulation and the percentage error occurrences in this part are noted in Table 7-8. The raw data for these errors are shown in Table D-2,

Appendix D. In this table, we used 4 scripts for 8 cases. The first 3 errors are checked in one script. Another error, DM66, is checked on one script. Other two couples are checked on two scripts, one different script is used for each couple of errors.

ErrorID	Error Name	% Scripts with Errors	Error Distribution (%)		
			Syntax	Semantic	Logic
DM63	Select / the same column names or aliases / order by	4	0	0	100
DM64	Select / limit missed argument	0	0	0	0
DM65	Select / limit negative argument	1	100	0	0
DM66	Update table	17	25	17	58
DM67	Update / increment values	0	0	0	0
DM68	Update / decrement values	0	0	0	0
DM70	Delete data from table (s)	11	38	13	50
DM71	Truncate table / not exist	14	20	0	80

Table 7-8 Percentages of Student Errors in data retrieve, update, delete and truncate Commands

The analyzed results and the percentage chart of the occurred errors are shown in Figure 7-8.

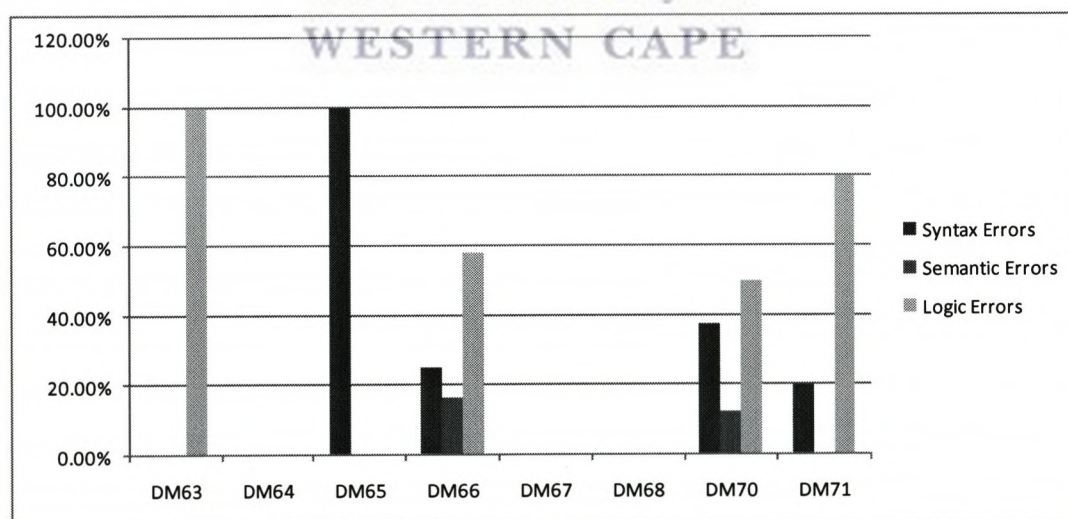


Figure 7-8 Analysis of Student Errors in data retrieve, update, delete and truncate Commands

Most of the data manipulation errors, listed in Table 5-4, are checked in students work. Each error is checked in 70 different students' class work. We have also checked these errors under three circumstances, i.e. syntax, semantic and logic. The number of occurrences for each case is listed in tables in Appendix D. Similarly, we have analyzed the data of each table and reported in charts. This is done on a percentage basis for each error.

The total number of errors in this section is 72 and we checked 46 out of this number in students' work. The other 26 errors occur rarely and they are ignored in this research. In this section, we have used 22 different scripts and checked 46 cases of data manipulation errors.

7.4 **Student Errors—Transaction Control and Locking Tables**

The MySQL transaction control and locking commands can have errors. Chapter 5 explains occurrence of these errors. Table 5-5 lists all the errors that may counter in this part of MySQL commands. In this section, we check these errors in students' class work and find the matched cases of this category of errors. The errors occurring are classified into three classes as syntactic, semantic and logic errors. The raw data for error occurrences is noted in Table D-3, Appendix D. Results for the occurrence of these errors, their percentage and the analytical charts are explained shortly in the forthcoming paragraphs.

The following Table 7-9 shows the error occurrence in transaction control and rollback statements of MySQL. In this table, we used 8 scripts for 9 cases. The first 2 errors are checked in one script. The other cases are appear separately on different scripts.

ErrorID	Error Name	% Scripts with Errors	Error Distribution (%)		
			Syntax	Semantic	Logic
TC01	Start transaction / begin	6	50	50	0
TC02	Start transaction / begin / option	0	0	0	0
TC06	Rollback / create database	21	0	20	80
TC07	Rollback / alter database	19	0	0	100
TC08	Rollback / drop database	19	8	0	92
TC09	Rollback / create table	13	0	0	100
TC10	Rollback / alter table	11	0	13	88
TC11	Rollback / drop table	23	6	0	94
TC12	Rollback / rename table	14	0	20	80

Table 7-9 Percentages of Student errors in transaction and rollback Commands

The analytical results based the percentage of the occurred errors from the previous table are shown in Figure 7-9.

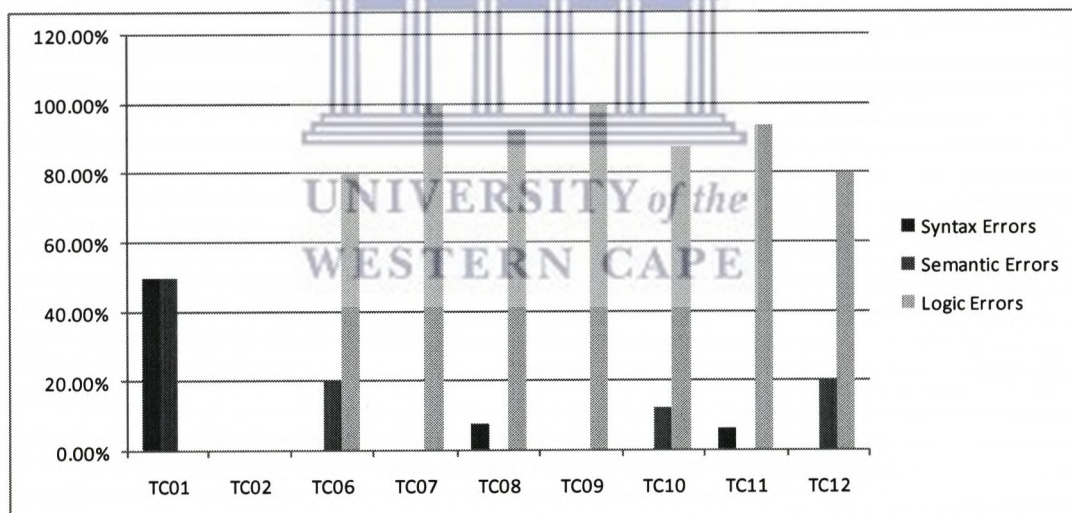


Figure 7-9 Analysis of Student Errors in transaction and rollback Commands

The second and the last part of the transaction errors in MySQL are shown in Table 7-10. The raw data for these cases is recorded in Table D-1, Table D-3, and Appendix D. In this group, for a couple of errors we used two scripts. We used 10 scripts for 8 error cases. The two errors TC17 and TC18, each one are checked in

two different scripts. In addition, the other six cases are checked on 6 different scripts.

ErrorID	Error Name	% Scripts with Errors	Error Distribution (%)		
			Syntax	Semantic	Logic
TC13	Rollback / create view	27	0	5	95
TC14	Rollback / alter view	26	6	11	83
TC15	Rollback / drop view	30	10	0	90
TC16	Rollback / truncate table	29	0	15	85
TC17	Rollback / DDL commands	23	19	6	75
TC18	Rollback / transaction control and locking statements	16	0	18	82
TC21	Savepoint / rollback to savepoint	6	100	0	0
TC22	Rollback to savepoint	14	10	10	80

Table 7-10 Percentages of Student Errors in rollback, savepoint and rollback to savepoint Commands

Figure 7-10 shows the analyzed results of Table 7-10.

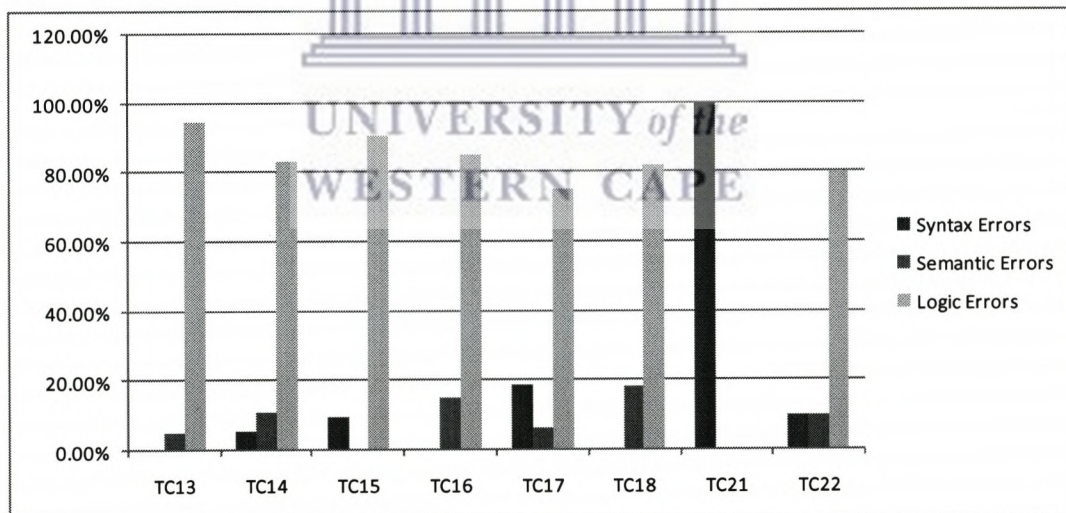


Figure 7-10 Analysis of Student Errors in rollback, savepoint and rollback to savepoint Commands

Most of the transaction control errors are checked in students work. Some errors are checked in 70 different students' class work and a couple of them are checked in 140 samples of code. Each error is checked under three cases i.e. syntactic, semantic and logic. The number of occurrences for each case is listed in tables. In addition, we have analyzed and reported on percentage of error occurrence for all tables. This is shown in charts.

The errors in this section number 22 and we have found 17 out of these occurred in students' work. The other 5 errors are rare and we have ignored them. Totally 16 scripts are used for checking cases of the transaction control sections errors.

7.5 Student Errors—Total Number of Errors in All Cases

We have checked the MySQL errors that students and novices make during class work. This process was implemented on students work from two classes in different years. The total number of students chosen for this purpose was from two classes each class containing 70 students. We have checked MySQL code portions that these students had generated. This checking was based on the three forms of semantic, syntactic and logic of each error. The previous three sections indicated three general types of MySQL commands and error occurrence in those commnds. All the results were shown clearly in data tables and appended to our document in Appendix D. The results are also mapped in charts with a percentage or occurrence. In general, we have used 58 different scripts and checked 91 cases of data definition, data manipulation and transaction control errors.

In this section we show the totals for each category of MySQL commands including data definition, data manipulation and transaction control commands. Additionally, we have included the grand total for all the commands that we checked during this research. The raw data for the totals may be found at the end of Appendix D.

Table 7-11 shows the percentages of totals and the grand total for all the checked commands.

Error Categories	Error Distribution (%)		
	Syntax	Semantic	Logic
Total - Data Definition Errors – DDL	15	36	24
Total - Data Manipulation Errors – DML	14	36	17
Total - Transaction Control Errors – TCL	18	8	9
Grand Total—All Checked Errors	15	30	17

Table 7-11 Percentages of Student Errors Totals in all Commands

Figure 7-11 shows the analyzed results of Table 7-11.

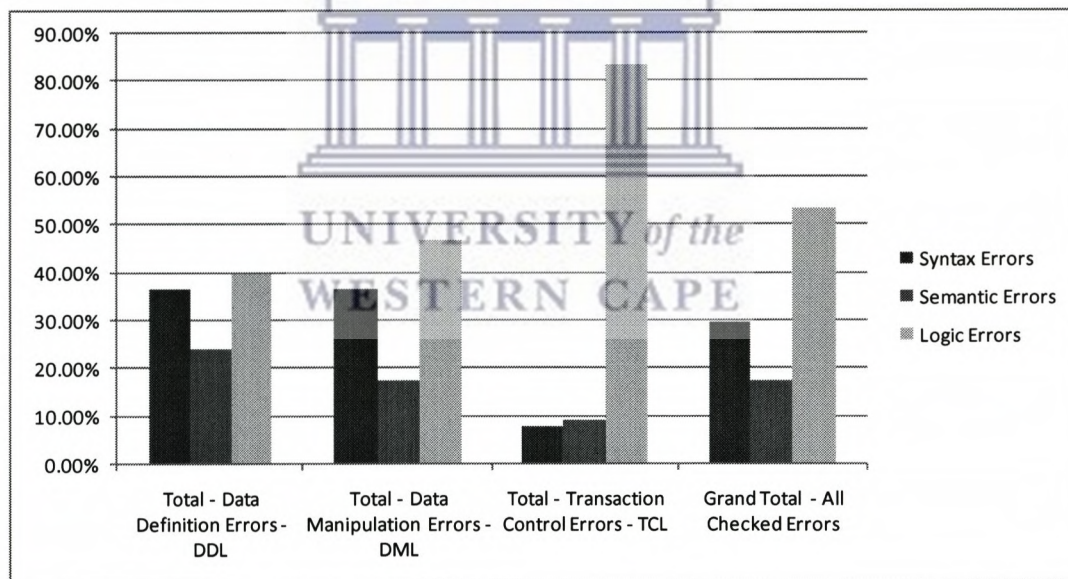


Figure 7-11 Analysis of Student Errors in all Commands

7.6 Analysis of Results

We have checked the MySQL errors that students make in a database course. The results of this checking is explained fully in preceding sections of this chapter. Ninty one cases of error occurrence is checked on 58 sample scripts written for MySQL DBMS. Each script was written by more than 70 students. We have checked each case for three different types of errors: syntactic, semantic and logic. Therefore, we have analyzed error cases in our results which are based on these three types.

7.7 Theoretical Analysis

We have explained MySQL errors and categorized these errors in three groups: DDL, DML and TCL. Our expectations for error occurrence differ for each part. In this section, we want to theoretically analyze error occurrences in any of these parts.

The practice that we want to use is based on meaning, usage and expected results for commands in different categories. These factors may show the expectations of percentages that are resulted from the commands. In data definition commands (DDL), syntax errors and logic errors may occur in an equal percentage. The chances for semantic errors occuring is not that much. Due to the nature of these commands, we expect more syntax and logic errors and fewer semantic errors. For example, a create or alter table command may have syntax or logic errors. Semantic errors in these commands occur rarely.

DML commands may have the same number of syntax and logic errors. However, chance for logic errors is a little bit more than the syntax error. Semantic errors are also inclined to occur in these commands. After theoretical analysis, our expectation for error occurrence in data manipulation commands shows 40% logic, 35% syntax and 25% semantic errors.

Transaction control commands (TCL) commands of MySQL appear in a different form compared to the previous two categories. These commands work under different conditions and result in specific types of tables. If a user is not fully aware of these conditions, s/he may make more errors while using TCLs. Therefore, the TCL commands may be inclined to contain a higher number of logical errors in comparison to the DDL and DML commands. Our expectations are 60% logic, 25% syntax and 15% semantic errors.

7.8 Analysis of the Exact Results

Table 7-1 is chosen as an example for data analysis. In raw data, illustrated in Table D-1, Appendix D, shows that in the first line for "DD01 Create table" we have checked 70 cases. In this command, 2 syntax errors, 0 semantic errors and 10 logic errors occurred. This means that 12 of the 70 checked cases contained errors, which means 17.14% contained errors. These errors can be broken down as 16.67% syntax errors and 83.33% logical errors. Semantic errors for this case is 0.00%.

If we do this for the whole of Table 7-1, then we see that 60% of students made errors with DD11 while only 4.29% made errors with DD7 and DD08. We can also see that DD08 was the only one where no syntax errors were made. DD06 and DD08 were the only ones where most of the errors were semantic. DD04-09 had no logic errors. According to these cases the following question may arise:

Why do the students make different errors with different commands?

To answer this question, we can say one of the reasons is the type of the commands. For example, DD04 is the Use command. The user needs to type 'Use DBname;'. In this command, there is not much room for logic or semantic errors. Therefore, we expect syntactic errors and the result is also the same. 11.43% of students either misspelled 'use' or 'DBname' or left out the ';' symbol at the end of the command. If

we had an editor/IDE that did word completion and was able to predict the names of databases, then these errors could be avoided.

In Table 7-11, we have the total percentage of error occurrence for every category of MySQL commands that are explained in this document: DDL, DML and TCL. DDL has a close number of syntax (36.46%) and logic (39.58%) errors and slightly fewer semantic (23.96%). This result was expected. Theoretically, the data definition commands have less semantic errors. MySQL commands for defining database structures may have syntax errors. In our survey, we found 105 syntax errors from 1960 cases in students code. Similarly, we have encountered 114 logical errors and only 69 semantic errors from 1960 cases in 20 scripts. These results are close to our theoretical analysis explained in the previous section.

DML has 36.29% syntax, 17.06% semantic and 46.65% logic errors. The percentage of syntax errors in this category is almost the same as the syntax errors in data definition commands. It shows that students make the same number of syntax errors in both cases. More interesting point in these commands is in occurrence of logic errors. The percentage of logical errors exceeded 46.65%; in contrast, the percentage of the semantic errors decreased to 17.06%. Again, this is result expected. In our theoretical analysis, we were expecting 40% for logic, 35% for syntax and 25% for semantic errors. The resulting figures only match the syntax errors. Two other parts are different.

The data manipulation commands are mostly relate to the user's thinking. When some information is requested from a user to export from a database, the user needs to query data. For this purpose, users have different ideas and use different options and methods for their work. Most of the queries that users write, may have correct syntax and meaning. However, those queries may be logically incorrect and cause unexpected results. These are the cases where the percentage of the logic errors is higher in this category of commands compared to DDL.

TCL has 7.69% syntax, 8.97% semantic and 83.33% logic errors. This category of errors are mostly logical. The difference in percentage is high. This is result expected. Our theoretical analysis shows a moderated difference of error occurrences between the two forms: syntactic and semantic errors. The logic errors are theoretically expected to have a higher percentage: 60% logic, 25% syntax and 15% semantic.

Logic errors in transaction control commands may occur in different cases. For example, students tried to rollback changes on non-transactional tables. They tried to rollback some general commands of the data defintion section through which table structures are updated, and some other similar cases.

Overall there are 29.54% syntax, 17.16% semantic, 53.3% logic errors. If one could build a perfect syntax checker for MySQL, about 30% of student errors could be prevented.

7.9 Summary

Chapter 7 is one of the important chapters presented here. We have explained the final results of the students' work in MySQL. The MySQL code portions, which were developed by our students, have been checked for common errors. The checking was based on our classification from Chapter 5. The results on error occurrence of students' work were checked for the three cases: syntactic, semantic and logic. Further, we have analyzed the results on percentage of error occurrence. Theoretical analysis as well as the exact analysis of results are done. The percentage analyses are shown in histograms. In total we have reported the errors that have occurred in 10 tables, which are further displayed in 10 charts. In short, this chapter started with the evaluation of data definition errors and continued through data manipulation errors; and it ended by analyzing transaction control errors.

Chapter 8

Conclusion and Future Work

This research has made important findings regarding static error checking in the database field. We started with a general database study and came up through query languages to MySQL database management system. SQL and MySQL history and functions are explained. Case sensitivity of MySQL commands and functions and their proneness to errors in different platforms is explained. Then a complete literature survey was done. The survey includes SQL error checking tools and some other error checkers for different computer languages.

We have compiled a reasonably complete list of common MySQL errors that students and novices make in database courses. The errors that we found are categorized in three groups. The groups are based on the MySQL commands where they can occur. Every group covers a sufficient number of errors. The collected and grouped errors of MySQL can be used as a valuable asset for future work in database field.

Most of the listed errors are checked in students' class work. We chose two classes of Kabul University where databases are taught in two semesters for the 3rd year of Computer Science. Each error is checked in three common cases of error occurrence: syntactic, semantic and logic. The frequency of error occurrence for each case is recorded and showed in the thesis. Then, the results are analyzed and a percentage of the occurred errors are mapped in separate charts. The percentage is based on the number of code portions that we checked and the frequency of errors in each case.

In this research, we have checked different errors that may occur in SQL code. The errors are listed and grouped. Categorization of errors in groups was based on the

type of each command in SQL. Then we have recorded occurrence of errors in three forms: syntactic, semantic and logic. As a trailer of our work, this activity can be changed to re-categorize SQL errors as preventable error and non-preventable error groups. This can help developers of error checkers to recognize errors by preventable and non-preventable errors. Then they can design their tool to use only the first group of errors that can be prevented. This may differ for individual tools. Specifications of a tool can determine the preventable and non-preventable errors in SQL.

Similarly, the listed errors in this research may be used to determine whether they are predictable or detectable. Each group of predictable errors and detectable errors can enrich the design of an error checker tool.

The final results of our analyses shows that there are 29.54% syntax, 17.16% semantic, 53.3% logic errors. If one could build a perfect syntax error checker for MySQL, about 30% of student errors could be prevented. The question arises: what can be done to avoid semantic and logic errors?

Developing a tool that could determine semantic and/or logic errors is possible. However, this tool will not be that complete to be used in general cases. This kind of tool can be developed for a specific database in a limited field. This is possible only if all the rules in that specific field are included in the tool's knowledge base. For every little change, the tool may need to be updated.

We propose the following two directions for future work.

8.1 **Static Error Checker for SQL—Offline**

A static SQL error checker can be developed and installed on an individual machine to be used by a single user. The error checking tool should have facilities to find and

highlight common SQL syntax errors. It may optionally have suggestions to correct errors for its users. Its suggestions can make it more interesting. Our findings from this research can be used for designing such a tool.

The proposed tool can be used for self evaluation; it can be useful for general database users and students. It can also be used for finding similarities in students' homework that is very important for database instructors. We hope our results of this research, especially the lists of the errors, can be feasible for implementing the proposed error check for SQL.

The proposed *Static Error Checker for SQL* can have the following options and functionalities:

- Check a group of sample errors. Its developer may include the sample errors to be checked.
- It can have an additional box/menu option where it can accept new sets of errors. Checking of the code can be done on any set of errors.

8.2 **Static Error Checking of SQL - Online**

A comprehensive error checker for SQL could be programmed. This tool can have broader functionalities comparing to the previous one. The tool can check SQL syntax errors based on the results of this research. Online technologies and facilities can be used for designing such a tool. User access can be controlled by issuing a username and password for its users. The online error checker can be designed in two forms.

1. To be uploaded to the Internet
2. To be used in a small or medium network (local)

The first type can have a control board. An option to record the errors that users make by using the tool can further be analyzed. Those results will be helpful for upgrading the tool to satisfy its users. The second one can also record the errors that users make. All the recorded results will be good data for future improvements of the tool.

The proposed online tool can have the following options and functionalities:

- Like the AsseSQL explained in [35], it can provide feedback and notes to the users.
- It can have an accuracy level and give a survey option. The survey can show the frequency of any occurred error.
- Unlike the AsseQL [35], this tool can provide solutions to the users.

8.3 Concluding Notes

This research was a good experience for its researcher. During the research, valuable experiments and experiences made significant improvements in the career of the researcher. We hope the results of this research can be used as a valuable asset in academic arena, especially in database field in the world.

Appendix A

A-1 The 'HOSPITAL' database code in MySQL syntax

The code of the 'HOSPITAL' database is divided into four sections.

A-1.1. Database definition commands

The following commands drop database 'HOSPITAL' if it exists, then the database is created and activated as the defaults database.

```
DROP DATABASE HOSPITAL;  
CREATE DATABASE HOSPITAL;  
USE HOSPITAL
```



A-1.2 Table definition commands

The following commands create tables within the active database 'HOSPITAL'.

UNIVERSITY of the
WESTERN CAPE

```
CREATE TABLE PERSON (  
Person_ID INT PRIMARY KEY,  
Name VARCHAR(40),  
DoB DATE,  
Phone CHAR(14),  
Address VARCHAR(40),  
city VARCHAR(15)  
);
```

```
CREATE TABLE EMPLOYEE (  
Employee_ID INT PRIMARY KEY,  
Date_Hired DATE,  
Salary INT,  
Care_Center_Name VARCHAR(30)  
);
```

```
CREATE TABLE NURSE (  
Qualification VARCHAR(40),
```

```
Nurse_ID INT PRIMARY KEY
);
```

```
CREATE TABLE TECHNICIAN (
Skill VARCHAR(30),
Technician_ID INT PRIMARY KEY
);
```

```
CREATE TABLE STAFF (
Job_Class VARCHAR(20),
Staff_ID INT PRIMARY KEY
);
```

```
CREATE TABLE PATIENT (
Register_Date DATE,
Patient_ID INT PRIMARY KEY
);
```

```
CREATE TABLE RESIDENT_PATIENT (
Admission_Date DATE,
Admission_Fee INT,
Res_Patient_ID INT
);
```

```
CREATE TABLE OUTPATIENT (
Out_Patient_ID INT PRIMARY KEY
);
```

```
CREATE TABLE PHYSICIAN (
Pager_Number INT,
Specialty VARCHAR(15),
Physician_ID INT PRIMARY KEY
);
```

```
CREATE TABLE VISIT (
Visit_Date DATE,
Comments VARCHAR(50),
Out_Pat_ID INT,
Phys_ID INT
);
```

```
CREATE TABLE TREATMENT (
Treatment_ID INT PRIMARY KEY,
Treatment_Name VARCHAR(30)
);
```

```
CREATE TABLE TREAT (
Treat_Date DATE,
Treat_Time TIME,
Result VARCHAR(15),
Patient_Treatment_ID INT,
```



```
Physician_Treatment_ID INT,  
Treat_Treatment_ID INT,  
Treat_Charges INT  
);
```

```
CREATE TABLE CARE_CENTER (  
Name VARCHAR(15) PRIMARY KEY,  
Building_Num INT  
);
```

```
CREATE TABLE BUILDING (  
Building_Number INT PRIMARY KEY,  
Building_Name VARCHAR(40),  
Building_Code INT  
);
```

```
CREATE TABLE VACCINE_CENTER (  
Vaccine_Center_Name VARCHAR(20) PRIMARY KEY,  
Building_Num INT  
);
```

```
CREATE TABLE LABORATORY (  
Lab_Name VARCHAR(20) PRIMARY KEY,  
Building_Num INT  
);
```

```
CREATE TABLE BED (  
Bed_Number INT PRIMARY KEY,  
Room_Number INT  
);
```

```
CREATE TABLE ITEM (  
Item_Number INT PRIMARY KEY,  
Description VARCHAR(50),  
Unit_Cost DOUBLE  
);
```

```
CREATE TABLE CONSUME (  
Quantity INT,  
Date DATE,  
Total_Cost DOUBLE,  
Item_Number INT,  
Patient_ID INT  
);
```

```
CREATE TABLE REFER (  
Refer_Physician_ID INT,  
Refer_Patient_ID INT  
);
```



A-1.4. Relationship commands

The following commands add primary keys and relationships between the 'HOSPITAL' tables.

```
ALTER TABLE EMPLOYEE
ADD FOREIGN KEY (Employee_ID) REFERENCES PERSON (Person_ID)
ON DELETE NO ACTION ON UPDATE CASCADE;
ALTER TABLE EMPLOYEE
ADD FOREIGN KEY (Care_Center_Name) REFERENCES CARE_CENTER (Name)
ON DELETE NO ACTION ON UPDATE CASCADE;
```

```
ALTER TABLE PATIENT
ADD FOREIGN KEY (Patient_ID) REFERENCES PERSON (Person_ID)
ON DELETE NO ACTION ON UPDATE CASCADE;
```

```
ALTER TABLE PHYSICIAN
ADD FOREIGN KEY (Physician_ID) REFERENCES PERSON (Person_ID)
ON DELETE NO ACTION ON UPDATE CASCADE;
```

```
ALTER TABLE TECHNICIAN
ADD FOREIGN KEY (Technician_ID) REFERENCES EMPLOYEE (Employee_ID)
ON DELETE NO ACTION ON UPDATE CASCADE;
```

```
ALTER TABLE STAFF
ADD FOREIGN KEY (Staff_ID) REFERENCES EMPLOYEE (Employee_ID)
ON DELETE NO ACTION ON UPDATE CASCADE;
```

```
ALTER TABLE NURSE
ADD FOREIGN KEY (Nurse_ID) REFERENCES EMPLOYEE (Employee_ID)
ON DELETE NO ACTION ON UPDATE CASCADE;
```

```
ALTER TABLE RESIDENT_PATIENT
ADD FOREIGN KEY (Res_Patient_ID) REFERENCES PATIENT (Patient_ID)
ON DELETE NO ACTION ON UPDATE CASCADE;
```

```
ALTER TABLE RESIDENT_PATIENT
ADD PRIMARY KEY (Admission_Date, Res_Patient_ID);
```

```
ALTER TABLE OUTPATIENT
ADD FOREIGN KEY (Out_Patient_ID) REFERENCES PATIENT (Patient_ID)
ON DELETE NO ACTION ON UPDATE CASCADE;
```

```
ALTER TABLE TREAT
ADD FOREIGN KEY (Patient_Treatment_ID) REFERENCES PATIENT (Patient_ID)
ON DELETE NO ACTION ON UPDATE CASCADE;
ALTER TABLE TREAT
ADD FOREIGN KEY (Physician_Treatment_ID) REFERENCES PHYSICIAN (Physician_ID)
```

```
ON DELETE NO ACTION ON UPDATE CASCADE;
ALTER TABLE TREAT
ADD FOREIGN KEY (Treat_Treatment_ID) REFERENCES TREATMENT (Treatment_ID)
ON DELETE NO ACTION ON UPDATE CASCADE;
```

```
ALTER TABLE VISIT
ADD FOREIGN KEY (Out_Pat_ID) REFERENCES OUTPATIENT (Out_Patient_ID)
ON DELETE NO ACTION ON UPDATE CASCADE;
ALTER TABLE VISIT
ADD FOREIGN KEY (Phys_ID) REFERENCES PHYSICIAN (Physician_ID)
ON DELETE NO ACTION ON UPDATE CASCADE;
```

```
ALTER TABLE CARE_CENTER
ADD FOREIGN KEY (Building_Num) REFERENCES BUILDING (Building_Number)
ON DELETE NO ACTION ON UPDATE CASCADE;
```

```
ALTER TABLE VACCINE_CENTER
ADD FOREIGN KEY (Building_Num) REFERENCES BUILDING (Building_Number)
ON DELETE NO ACTION ON UPDATE CASCADE;
```

```
ALTER TABLE LABORATORY
ADD FOREIGN KEY (Building_Num) REFERENCES BUILDING (Building_Number)
ON DELETE NO ACTION ON UPDATE CASCADE;
```

```
ALTER TABLE BED
ADD COLUMN (Care_Center_Name CHAR(15));
ALTER TABLE BED
ADD FOREIGN KEY (Care_Center_Name) REFERENCES CARE_CENTER (Name)
ON DELETE NO ACTION ON UPDATE CASCADE;
```

```
ALTER TABLE CONSUME
ADD FOREIGN KEY (Item_Number) REFERENCES ITEM (Item_Number)
ON DELETE NO ACTION ON UPDATE CASCADE;
ALTER TABLE CONSUME
ADD FOREIGN KEY (Patient_ID) REFERENCES PATIENT (Patient_ID)
ON DELETE NO ACTION ON UPDATE CASCADE;
```

```
ALTER TABLE NURSE
ADD COLUMN(cc_inCHARge VARCHAR(30));
ALTER TABLE NURSE
ADD FOREIGN KEY (cc_inCHARge) REFERENCES CARE_CENTER(Name)
ON DELETE NO ACTION ON UPDATE CASCADE;
```

```
ALTER TABLE REFER
ADD PRIMARY KEY (Refer_Physician_ID, Refer_Patient_ID);
ALTER TABLE REFER
ADD FOREIGN KEY(Refer_Physician_ID) REFERENCES PHYSICIAN (Physician_ID)
ON DELETE NO ACTION ON UPDATE CASCADE;
ALTER TABLE REFER
ADD FOREIGN KEY(Refer_Patient_ID) REFERENCES PATIENT (Patient_ID)
ON DELETE NO ACTION ON UPDATE CASCADE;
```

A-1.4. Data entry commands

The following commands enter data to the 'Hospital' tables.

```
INSERT INTO PERSON VALUES (1, "Feda", "1954-05-11", 70039, "City", "Kabul");
INSERT INTO PERSON VALUES (2, "Kabir", "1962-07-09", 70013, "Dist2", "Hirat");
INSERT INTO PERSON VALUES (3, "Mahbooba", "1945-04-01", 70018, "Dist5", "Kabul");
INSERT INTO PERSON VALUES (4, "Sabir", "1976-04-13", 70002, "Dist1", "Kabul");
INSERT INTO PERSON VALUES (5, "Raihana", "1964-03-07", 70003, "City", "Ningarhar");
INSERT INTO PERSON VALUES (6, "Karima", "1959-04-12", 70004, "City", "Kabul");
INSERT INTO PERSON VALUES (7, "Latifa", "1959-07-21", 70005, "Dist6", "Kabul");
INSERT INTO PERSON VALUES (8, "Rustam", Null, 70033, "Dist12", "Kabul");
INSERT INTO PERSON VALUES (9, "Farhad", "1953-09-19", 70034, "Center", "Maydan");
INSERT INTO PERSON VALUES (10, "Rabia", "1976-07-05", Null, "Dist1", "Kabul");
INSERT INTO PERSON VALUES (11, "Usman", "1978-09-08", 77883, "Dist1", "Kabul");
INSERT INTO PERSON VALUES (12, "Lutfullah", "1976-03-08", 79158, "Dist7", "Kabul");
INSERT INTO PERSON VALUES (13, "Fawzia", "1978-03-08", 0, "Dist12", "kabul");
INSERT INTO PERSON VALUES (14, "Javid", "1974-06-11", 79893, "Dist9", "Kabul");
INSERT INTO PERSON VALUES (15, "Rasoul", "1985-09-12", 72236, "Dist8", "kabul");
INSERT INTO PERSON VALUES (16, "Qurban", "1977-01-01", 0, "Dist5", "Kabul");
INSERT INTO PERSON VALUES (17, "Farooq", "1974-09-01", 75503, "Dist11", "kabul");
INSERT INTO PERSON VALUES (18, "Zeeba", "1974-07-17", 75023, "Jalabad", "Ningarhar");
INSERT INTO PERSON VALUES (19, "Qadria", "1959-03-19", 75360, "City", "Kabul");
INSERT INTO PERSON VALUES (20, "Arman", "1979-07-21", 70395, "Dist16", "Kabul");
INSERT INTO PERSON VALUES (21, "Roshan", Null, 75633, "Dist12", "Kabul");
INSERT INTO PERSON VALUES (22, "Karima", "1973-11-16", Null, "Center", "Jalalabad");
INSERT INTO PERSON VALUES (23, "Rabani", "1952-09-12", Null, "Dist21", "Kabul");
INSERT INTO PERSON VALUES (24, "Usman", "1962-01-10", 70027, "Dist7", "kabul");
INSERT INTO PERSON VALUES (25, "Latif", "1956-03-08", 79958, "Dist1", "kabul");
INSERT INTO PERSON VALUES (26, "Kamela", "1978-03-24", Null, "City", "kabul");
INSERT INTO PERSON VALUES (27, "Jabar", "1984-08-19", 79164, "Charikar", "Parwan");
INSERT INTO PERSON VALUES (28, "Moheb", "1985-11-12", 72223, "Dist18", "kabul");
INSERT INTO PERSON VALUES (29, "Qarib", "1987-01-15", Null, "City", "Hirat");
INSERT INTO PERSON VALUES (30, "Kamal", "1984-11-01", 50324, "Dist1", "kabul");
INSERT INTO PERSON VALUES (31, "Ruqia", Null, Null, "Dist8", "kabul");
INSERT INTO PERSON VALUES (32, "Hadia", "1985-01-01", Null, "City", "Hirat");
INSERT INTO PERSON VALUES (33, "Jamshid", "1982-11-30", 70043, "Dist1", "kabul");
INSERT INTO PERSON VALUES (34, "Gulab", "1992-01-15", Null, "City", "Hirat");
```

```
INSERT INTO PERSON VALUES (39, "Munir", "1966-03-22", 70098, "City", "kabul");
INSERT INTO PERSON VALUES (40, "Kabir", "1964-08-29", 75764, "Center", "Logar");
INSERT INTO PERSON VALUES (41, "Mirwais", "1965-11-12", 78763, "Dist2", "kabul");
INSERT INTO PERSON VALUES (42, "Laila", "1985-01-15", 79954, "City", "Hirat");
INSERT INTO PERSON VALUES (43, "Rafiq", "1974-11-01", 70734, "Dist1", "kabul");
```

```
INSERT INTO PHYSICIAN VALUES (101, "surgery", 1);
INSERT INTO PHYSICIAN VALUES (102, "General", 2);
INSERT INTO PHYSICIAN VALUES (103, "Childern", 3);
INSERT INTO PHYSICIAN VALUES (104, "General", 4);
```


INSERT INTO PHYSICIAN VALUES (105, "Emergency", 18);
INSERT INTO PHYSICIAN VALUES (106, "Emergency", 19);
INSERT INTO PHYSICIAN VALUES (107, "Surgery", 20);
INSERT INTO PHYSICIAN VALUES (108, "General", 21);

INSERT INTO BUILDING VALUES (1, "Building1", 2001);
INSERT INTO BUILDING VALUES (2, "Building2", 2002);
INSERT INTO BUILDING VALUES (3, "Building3", 2003);

INSERT INTO CARE_CENTER VALUES ("CC1", 1);
INSERT INTO CARE_CENTER VALUES ("CC2", 1);
INSERT INTO CARE_CENTER VALUES ("CC3", 1);
INSERT INTO CARE_CENTER VALUES ("CC4", 1);
INSERT INTO CARE_CENTER VALUES ("CC5", 1);
INSERT INTO CARE_CENTER VALUES ("CC6", 1);
INSERT INTO CARE_CENTER VALUES ("CC7", 2);
INSERT INTO CARE_CENTER VALUES ("CC8", 2);
INSERT INTO CARE_CENTER VALUES ("CC9", 2);
INSERT INTO CARE_CENTER VALUES ("CC10", 2);
INSERT INTO CARE_CENTER VALUES ("CC11", 3);
INSERT INTO CARE_CENTER VALUES ("CC12", 3);

INSERT INTO EMPLOYEE VALUES (5, "1997-02-16", 6000, "CC1");
INSERT INTO EMPLOYEE VALUES (6, "1997-02-16", 6500, "CC3");
INSERT INTO EMPLOYEE VALUES (8, "1986-02-01", 5500, "CC4");
INSERT INTO EMPLOYEE VALUES (9, "1989-05-01", 6000, "CC2");
INSERT INTO EMPLOYEE VALUES (7, "1990-02-01", 6000, "CC6");
INSERT INTO EMPLOYEE VALUES (22, "2002-01-01", 8000, "CC7");
INSERT INTO EMPLOYEE VALUES (23, "1987-03-16", 7200, "CC8");
INSERT INTO EMPLOYEE VALUES (24, "1986-02-16", 8000, "CC5");
INSERT INTO EMPLOYEE VALUES (25, "1988-05-01", 8000, "CC9");
INSERT INTO EMPLOYEE VALUES (26, "2001-07-01", 5500, "CC10");
INSERT INTO EMPLOYEE VALUES (39, "1999-05-16", 6000, "CC11");
INSERT INTO EMPLOYEE VALUES (40, "1989-07-01", 8000, "CC12");
INSERT INTO EMPLOYEE VALUES (41, "1998-05-01", 6000, "CC12");
INSERT INTO EMPLOYEE VALUES (42, "2003-02-01", 5000, "CC12");
INSERT INTO EMPLOYEE VALUES (43, "1987-09-01", 7000, "CC6");

INSERT INTO NURSE VALUES ("Nursing", 5, "CC1");
INSERT INTO NURSE VALUES ("Nursing", 6, "CC2");
INSERT INTO NURSE VALUES ("Nursing", 7, "CC3");
INSERT INTO NURSE VALUES ("Primary", 8, "CC4");
INSERT INTO NURSE VALUES ("Nursing", 22, "CC5");
INSERT INTO NURSE VALUES ("Private", 23, "CC6");
INSERT INTO NURSE VALUES ("Higher", 24, "CC7");
INSERT INTO NURSE VALUES ("Nursing", 25, "CC8");
INSERT INTO NURSE VALUES ("Nursing", 26, "CC9");
INSERT INTO NURSE VALUES ("Primary", 41, "CC10");
INSERT INTO NURSE VALUES ("Nursing", 39, "CC11");
INSERT INTO NURSE VALUES ("Nursing", 40, "CC12");

INSERT INTO STAFF VALUES ("Admin", 9);
INSERT INTO STAFF VALUES ("Assistant", 5);
INSERT INTO STAFF VALUES ("Admin", 26);
INSERT INTO STAFF VALUES ("Assistant", 22);
INSERT INTO STAFF VALUES ("Admin", 43);
INSERT INTO STAFF VALUES ("Assistant", 39);

INSERT INTO PATIENT VALUES ("2006-07-01", 10);
INSERT INTO PATIENT VALUES ("2006-07-02", 11);
INSERT INTO PATIENT VALUES ("2006-07-04", 12);
INSERT INTO PATIENT VALUES ("2006-07-04", 13);
INSERT INTO PATIENT VALUES ("2006-06-14", 14);
INSERT INTO PATIENT VALUES ("2006-06-19", 15);
INSERT INTO PATIENT VALUES ("2006-06-22", 16);
INSERT INTO PATIENT VALUES ("2006-06-23", 17);

INSERT INTO OUTPATIENT VALUES (10);
INSERT INTO OUTPATIENT VALUES (11);

INSERT INTO RESIDENT_PATIENT VALUES ("2006-07-04", 300, 13);
INSERT INTO RESIDENT_PATIENT VALUES ("2006-06-14", Null, 14);
INSERT INTO RESIDENT_PATIENT VALUES ("2006-06-19", Null, 15);
INSERT INTO RESIDENT_PATIENT VALUES ("2006-06-23", 400, 17);

INSERT INTO TREATMENT VALUES (1, "Flu");
INSERT INTO TREATMENT VALUES (2, "Headache");
INSERT INTO TREATMENT VALUES (3, "Bronchit");

INSERT INTO TREAT VALUES ("2006-07-02", "10:00", "Not fine", 10, 1, 3, 500);
INSERT INTO TREAT VALUES ("2006-07-03", "09:30", "Fine", 11, 2, 1, 700);
INSERT INTO TREAT VALUES ("2006-07-05", "11:00", "Fine", 10, 1, 3, 550);
INSERT INTO TREAT VALUES ("2006-07-05", "11:30", "Fine", 12, 1, 2, 1200);
INSERT INTO TREAT VALUES ("2006-07-06", "14:00", "Fine", 13, 2, 1, 800);
INSERT INTO TREAT VALUES ("2006-06-22", "14:00", "Pending", 14, 1, 3, 450);
INSERT INTO TREAT VALUES ("2006-06-22", "13:30", "Not fine", 15, 2, 2, 1150);
INSERT INTO TREAT VALUES ("2006-06-23", "13:00", "Not fine", 16, 2, 1, 700);
INSERT INTO TREAT VALUES ("2006-06-25", "11:00", "Not fine", 17, 2, 3, 500);
INSERT INTO TREAT VALUES ("2006-06-23", "13:30", "Not fine", 15, 2, 1, 720);
INSERT INTO TREAT VALUES ("2006-06-23", "08:40", "Good", 14, 2, 2, 1220);
INSERT INTO TREAT VALUES ("2006-06-24", "13:30", "Good", 15, 1, 2, 1180);

INSERT INTO BED VALUES (1, 3, "cc2");
INSERT INTO BED VALUES (2, 3, "cc2");
INSERT INTO BED VALUES (3, 3, "cc1");
INSERT INTO BED VALUES (4, 2, "cc2");

INSERT INTO REFER VALUES (3, 10);
INSERT INTO REFER VALUES (3, 11);
INSERT INTO REFER VALUES (2, 13);
INSERT INTO REFER VALUES (2, 14);
INSERT INTO REFER VALUES (2, 15);

INSERT INTO REFER VALUES (3, 16);
INSERT INTO REFER VALUES (2, 17);

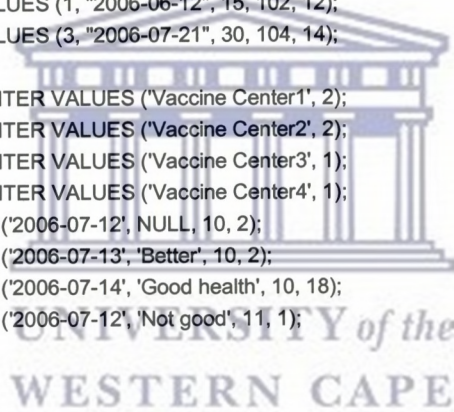
INSERT INTO ITEM VALUES (101, 'Emergency', 350);
INSERT INTO ITEM VALUES (102, 'Tube', 300);
INSERT INTO ITEM VALUES (103, 'Pense', 250);
INSERT INTO ITEM VALUES (104, 'Surgical', 150);

INSERT INTO LABORATORY VALUES ('Lab1', 2);
INSERT INTO LABORATORY VALUES ('Lab2', 2);
INSERT INTO LABORATORY VALUES ('Lab3', 1);
INSERT INTO LABORATORY VALUES ('Lab4', 3);

INSERT INTO TECHNICIAN VALUES ('Lab', 5);
INSERT INTO TECHNICIAN VALUES ('Lab', 7);
INSERT INTO TECHNICIAN VALUES ('First Aid', 8);
INSERT INTO TECHNICIAN VALUES ('Operation Aid', 6);

INSERT INTO CONSUME VALUES (2, "2006-07-14", 20, 102, 10);
INSERT INTO CONSUME VALUES (1, "2006-06-12", 15, 102, 12);
INSERT INTO CONSUME VALUES (3, "2006-07-21", 30, 104, 14);

INSERT INTO VACCINE_CENTER VALUES ('Vaccine Center1', 2);
INSERT INTO VACCINE_CENTER VALUES ('Vaccine Center2', 2);
INSERT INTO VACCINE_CENTER VALUES ('Vaccine Center3', 1);
INSERT INTO VACCINE_CENTER VALUES ('Vaccine Center4', 1);
INSERT INTO VISIT VALUES ('2006-07-12', NULL, 10, 2);
INSERT INTO VISIT VALUES ('2006-07-13', 'Better', 10, 2);
INSERT INTO VISIT VALUES ('2006-07-14', 'Good health', 10, 18);
INSERT INTO VISIT VALUES ('2006-07-12', 'Not good', 11, 1);



Appendix B

B-1. Sample MySQL Queries

The following sample queries / commands are requested from our students. The students asked to run them on the 'HOSPITAL' database. These queries are just taken as sample and they are a sub part of all the queries.

- Recreate the 'HOSPITAL' database
 - The 'HOSPITAL' database already exists in the system.
- Remove the 'dbOne' database from the system.
 - The 'dbOne' database may or may not exist in the system.
- List person ID, name, phone and city.
- List physician Name and Date of Birth whose specialty is surgery.
 - Field names should change to one in the question.
- List name, phone and address for all the patients who do not have phone numbers. List name, phone and address for all the patients.
- List resident patient names, date of admission and phone numbers for the patients who did not pay admission fees.
- List person IDs and city names whose names are started with 'R'.
- List the Care Center Names and number of beds each care center have.
- List the Care Center Names and number of beds each care center have.
- Use 'IN' to list the names of persons who are located in Kabul city.
- Use 'BETWEEN' to list the IDs of patients whose treat charges are between 480 and 820.
- What is the average fee in resident patients table?
- List the largest charges taken by each physician and the physician's pager number.

- List the largest charges taken by each physician and the physician's pager number, but only where the largest charge is not more than 1160.
- List the name of the employees and their salary in order of their salary. Within each salary, have the names in alphabetical order.

For the following two Queries – subqueries have to be used. Correlated queries are not needed.

- S List the ID and treatment charges of the patient who paid the largest charges for treatment.
- List the data from the treat table for treatments attributed to physicians living in Kabul.
- Create a new table like the person table that has only people who live in Kabul. Name the new table 'PERSON_KBL'.
 - You have to use one single MySQL command to do this.
- Convert 12 from decimal to binary. Use the CONV() function.
- Convert 111011011 from binary to octal system. Use the CONV() function.
- Convert AAA from hexadecimal to binary system. Use the CONV() function.
- Convert C12A from hexadecimal to decimal system. Use the CONV() function.
- Find the square root of 16. Use the SQRT() function.
- Find the square root of -16. Use the SQRT() function.

Appendix C

C-1. MySQL Versions

The following table describes MySQL features in different series.

MySQL Series	Features
MySQL 3.x	<ul style="list-style-type: none">- Replication- Full-text search in MyISAM tables- Transactions with InnoDB tables- Referential integrity for InnoDB tables
MySQL 4.0	<ul style="list-style-type: none">- Delete and update across several tables- ACID transactions- Cascading UPDATE and DELETE in foreign key values- Row level locking of userdata- Speed enhancement- Unions- Full-text search- Truncate table- Identity as a synonym for auto-increment keys- Support for new character set 'latin1_de'- Multiple table DELETE and UPDATE- SQL_CALC_FOUND_ROWS and FOUND_ROWS() functions- InnoDB database engine support
MySQL 4.1	<ul style="list-style-type: none">- Sub-queries- Unicode support (UTF8 and UCS2 = UTF16)- GIS support (GEOMETRY data type, R-tree index)- Binary trees (B-Trees) for heap tables- Prepared Statements (SQL commands with parameters)- Derived tables- Create table tblTwo LIKE tblOne- INSERT ... ON DUPLICATE KEY UPDATE ...

	<ul style="list-style-type: none"> - Show warnings - Unicode - Through the MyISAM storage engine, the OpenGIS spatial data types for storing geographical data - File sort behavior - Large table support - A number of storage engines were added: <ul style="list-style-type: none"> - EXAMPLE - NDBCLUSTER - ARCHIVE - CSV - BLACKHOLE - The GROUP_CONCAT() aggregate function
MySQL 5.0	<ul style="list-style-type: none"> - BIT data type - VARCHAR data type with more than 255 characters - Data dictionary to access system tables - Instance manager – users can stop or start MySQL services from operating system - Index merge - Stored procedures - Views - Optimizer tuning - Server side cursors - XA Transactions - Triggers - FEDERATED storage engine
MySQL 5.1	<ul style="list-style-type: none"> - Event scheduler - Partitioning, - Pluggable storage engine API - Plug in API - Row based applications - Server log tables

Table C-1 MySQL versions and features for different series

Appendix D

D-1 Raw Data

The following tables show the number of error occurrences for different cases. This operation is based on the results of our work in Chapter 5. Only the common errors are checked and the rare ones are ignored. The first three tables have 6 columns each. Column 1 shows the error id, which is already explained in Chapter 5, column 2 is the error name. Column 3 shows the total number of scripts checked for errors based on the scripts that students had written in Kabul University. The last 3 columns show the type of each occurred error within the total checked samples. The last table shows the totals for each section and the grand total for the errors occurring in all cases.

ErrorID	Error Name	Total Checked	Occurred Errors		
			Syntax	Semantic	Logic
DD01	Create database	70	2	0	10
DD03	Drop database	70	4	0	2
DD04	Use database	70	8	0	0
DD05	Create table	70	6	3	0
DD06	Create table / column name	70	1	5	0
DD07	Create table / data type	70	2	1	0
DD08	Create table / constraint	70	0	3	0
DD09	Create table / reference table	70	2	2	0
DD10	Create table / reference table / primary key or index key	70	1	3	8
DD11	Create table ... select	70	17	11	14
DD12	Alter table / add column	70	1	5	2
DD15	Alter table / add primary key	70	4	2	11
DD17	Alter table / add foreign key	70	8	2	9
DD18	Alter table / foreign key / data type	70	4	7	11
DD19	Alter table / foreign key data	70	4	1	3
DD20	Alter table / foreign key / primary key or index key	70	1	2	5

ErrorID	Error Name	Total Checked	Occurred Errors		
			Syntax	Semantic	Logic
DD21	Alter table / change column	70	3	2	7
DD22	Alter table / modify column	70	2	1	0
DD23	Alter table / drop column	70	8	2	0
DD24	Alter table / drop primary key	70	1	3	0
DD30	Alter table / rename	70	4	0	0
DD31	Rename table	70	4	2	5
DD32	Describe table	70	1	3	4
DD33	Drop table	70	4	3	12
DD34	Create view	70	3	2	5
DD35	Create view / table, view	70	6	1	4
DD37	Alter view	70	3	1	0
DD38	Drop view	70	1	2	2
Totals		1960	105	69	114

Table D-1 Student Errors in Data Definition Commands (SQL-DDL)

ErrorID	Error Name	Total Checked	Occurred Errors		
			Syntax	Semantic	Logic
DM01	Insert	70	11	4	0
DM02	Insert / table	70	5	4	7
DM03	Insert / column	70	6	4	18
DM04	Insert / column (s)	70	10	3	11
DM05	Insert / duplicate data	70	3	2	5
DM08	Insert / select	70	3	2	5
DM09	Insert / NULL values	70	4	1	6
DM10	Replace	70	0	0	0
DM11	Select	70	10	0	0
DM12	Select *	70	3	1	7
DM13	Select / column name (s) miss	70	3	0	5
DM14	Select / unknown column	70	2	1	4
DM15	Select / comma miss between column names	70	11	0	0
DM17	Select / from	70	4	2	12
DM18	Select / repeated table reference	70	1	3	2
DM19	Select / additional table reference	70	5	0	6
DM20	Select / table name miss	70	0	2	14
DM21	Select / comma miss between table names	70	3	7	0
DM22	Select / where	70	2	3	5
DM23	Group by / having / order by	70	1	0	6
DM24	Select / group by	70	3	2	7
DM25	Select / having	70	2	0	7
DM26	Select / order by	70	3	4	6
DM27	Select / subquery (s)	70	12	4	11
DM28	Select / subquery / table use	70	11	2	2

ErrorID	Error Name	Total Checked	Occurred Errors		
			Syntax	Semantic	Logic
DM30	Select / from / subquery column reference	70	8	4	2
DM31	Select / subquery / number of columns	70	3	1	5
DM32	Select / subquery / number of rows	70	4	5	5
DM34	Select / union / select	70	4	0	6
DM36	Select / join / alias table (s)	70	2	1	5
DM37	Select / where	70	5	0	6
DM38	Select / where / aggregate function (s)	70	2	1	7
DM40	Select / avg() function	70	3	2	5
DM41	Select / min() function	70	1	0	6
DM42	Select / max() function	70	4	0	1
DM43	Select / count() function	70	0	0	0
DM52	Select / replace() function	70	2	7	0
DM55	Select / substring_index() function	70	3	4	0
DM63	Select / the same column names or aliases / order by	70	0	0	3
DM64	Select / limit missed argument	70	0	0	0
DM65	Select / limit negative argument	70	1	0	0
DM66	Update table	70	3	2	7
DM67	Update / increment values	70	0	0	0
DM68	Update / decrement values	70	0	0	0
DM70	Delete data from table (s)	70	3	1	4
DM71	Truncate table / not exist	70	2	0	8
Totals		3220	168	79	216

Table D-2 Student Errors in Data Manipulation Commands (SQL-DML)

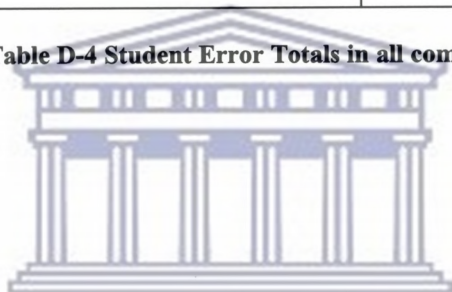
ErrorID	Error Name	Total Checked	Occurred Errors		
			Syntax	Semantic	Logic
TC01	Start transaction / begin	70	2	2	0
TC02	Start transaction / begin / option	70	0	0	0
TC06	Rollback / create database	70	0	3	12
TC07	Rollback / alter database	70	0	0	13
TC08	Rollback / drop database	70	1	0	12
TC09	Rollback / create table	70	0	0	9
TC10	Rollback / alter table	70	0	1	7
TC11	Rollback / drop table	70	1	0	15
TC12	Rollback / rename table	70	0	2	8
TC13	Rollback / create view	70	0	1	18
TC14	Rollback / alter view	70	1	2	15
TC15	Rollback / drop view	70	2	0	19
TC16	Rollback / truncate table	70	0	3	17
TC17	Rollback / DDL commands	140	6	2	24
TC18	Rollback / transaction control and locking	140	0	4	18

ErrorID	Error Name	Total Checked	Occurred Errors		
			Syntax	Semantic	Logic
	statements				
TC21	Savepoint / rollback to savepoint	70	4	0	0
TC22	Rollback to savepoint	70	1	1	8
Totals		1330	18	21	195

Table D-3 Student Errors in Transaction Control Commands (SQL-TCL)

Error Categories	Total Checked	Occurred Errors		
		Syntax	Semantic	Logic
Total - Data Definition Errors – DDL	1960	105	69	114
Total - Data Manipulation Errors – DML	3220	168	79	216
Total - Transaction Control Errors – TCL	1330	18	21	195
Grand Total - All Checked Errors	6510	291	169	525

Table D-4 Student Error Totals in all commands



UNIVERSITY of the
WESTERN CAPE

References

- [1] ISO/IEC 9075-10:2000, Information Technology–Database Languages–SQL–Part10: Object Language Bindings(SQL/OLB).
- [2] M. A. Bain. How to create stored procedures and functions in MySQL database. [Online] Available at http://database-programming.suite101.com/article.cfm/mysql_stored_procedures_and_functions, (Accessed June 2010).
- [3] C. Barker. Static error checking of C applications ported from UNIX to WIN32 systems using LCLint. Bachelor's thesis, University of Virginia, USA, 2001.
- [4] M. Barnett, K. Rustan, M. Leino and W. R. Bush. The Spec# programming system: An overview. Microsoft Research, Redmond, WA, USA, 2004.
- [5] S. Bhagat, L. Bhagat, J. Kavalan, and M. Sasikumar. Acharya: An intelligent tutoring environment for learning SQL. 2002.
- [6] R. S. Boyer, B. Elspas and K. N. Levitt. SELECT—A formal system for testing and debugging programs by symbolic execution. In *Proceedings of International Conference on Reliable Software*. ACM, 1975.
- [7] P. Brusilovsky, S. Sosnovsky, D. H. Lee, M. V. Yudelso, V. Zadorozny, and X. Zhou. An open integrated exploratorium for database courses. In

Proceedings of the 13th annual Conference on Innovation and technology in computer science education. Madrid, Spain, 2008.

- [8] J. R. Burch, E. M. Clarke and K. L. McMillan. Symbolic model checking: 1020 states and beyond. 1992.
- [9] W. R. Bush, J. D. Pincus and D. J. Sielaff. A static analyzer for finding dynamic programming errors. ACM, 2000.
- [10] B. Chen, D. Engler and S. Hallem. How to write system-specific, static checkers in Metal. In *Proceedings of the 2002 ACM SIGPLAN-SIGSOFT workshop for software tools and engineering*. South Carolina, USA, 2002.
- [11] M. G. Chinwala. Algebraic languages for XML databases. Georgia, 2001.
- [12] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. ACM, 1977.
- [13] Darklizener. MySQL user defined functions. [Online] Available at http://www.codeproject.com/KB/database/MySQL_UDFs.aspx, (Accessed: April 2010).
- [14] C. J. Date. *An introduction to database systems*. International Edition, 2004.
- [15] S. W. Dietrich, E. Eckert, and K. Piscator. WinRDBI: A Windows-based relational database educational tool. In *Proceedings of the 28th ACM SIGCSE Technical Symposium on Computer Science Education*. California, USA, 1997.

- [16] D. Engler, D. Y. Chen, S. Hallem, A. Chou and B. Chelf. Bugs as deviant behavior: A general approach to inferring errors in systems code. ACM, 2001.
- [17] D. Evans, J. V. Guttag, J. J. Horning and Y. M. Tan. LCLint: A tool for using specifications to check code. 1994.
- [18] D. Evans. LCLint User's Guide. Department of Computer Science, University of Virginia, 2000.
- [19] C. Flanagan and J. B. Saxe. Avoiding exponential explosion: generating compact verification conditions. In *Proceedings of the 28th ACM SIGCSE Technical Symposium on the Principals of Programming Languages*. California, USA, 1997.
- [20] C. Flanagan and K. R. M. Leino. Houdini, an annotation assistant for ESC/Java. In *Proceedings of the International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity*. London, United Kingdom, 2001.
- [21] C. Flanagan, K. R. M. Leino, M. Lillibridge, G. Nelson, J. B. Saxe and R. Stata. Extended Static Checking for Java. ACM, 2002.
- [22] I. Gilfillan. 2009. MySQL Stored Functions. [Online] Available at <http://www.databasejournal.com/features/mysql/article.php/3569846/MySQL-Stored-Functions.htm> (Accessed May 2010)
- [23] D. Jackson. Aspect: detecting bugs with abstract dependencies. 1995.
- [24] J. L. Jensen, M. E. Jorgensen, N. Klarlund and M. I. Schwartzbach. Automatic verification of pointer programs using monadic second-order logic.

In *Proceedings of the ACM SIGPLAN 1997 Conference on Programming language design and implementation*. Las Vegas, Nevada, USA, 1997.

- [25] R. Kearns, S. Shead, and A. Fekete. A teaching system for SQL. In *Proceedings of the 2nd Australasian conference on Computer science education*. The University of Melbourne, Australia, 1997.
- [26] C. Kenny and C. Pahl. Automated tutoring for a database skills training environment. In *Proceedings of the 36th SIGCSE technical symposium on Computer science education*. St. Luis, Missouri, USA, 2005.
- [27] D. M. Kroenke. *Database Concepts*. Prentice Hall Professional Technical Reference, 2002.
- [28] F. Kunst. Lint, a C Program Checker. Vrije Universiteit, Amsterdam, 1988.
- [29] B. M. Leavenworth and J. E. Sammet. An overview of nonprocedural languages. Before 1985.
- [30] H. Lu, H. C. Chan and K. K. Wei. A survey on usage of SQL. New York, USA, 1993.
- [31] R. A. McClure and I. H. Cruger. SQL DOM: Compile time checking of dynamic SQL statements. In *Proceedings of the 27th international conference on Software engineering*. St. Louis, MO, USA, 2005.
- [32] M. J. Minock. Knowledge representation using schema tuple queries. 2003.
- [33] A. Mitrovic. Learning SQL with a Computerized Tutor. In *Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*. Atlanta, Georgia, USA, 1998.

- [34] A. Mitrovic. A knowledge-based teaching system for SQL. AACE, 1998.
- [35] J. C. Prior and R. Lister. The Backwash Effect on SQL Skills Grading. In *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*. Leeds, United Kingdom, 2004.
- [36] R. Ragu. *Database Management Systems*. 1997.
- [37] A. Ranganathan, and Z. Liu. Information retrieval from relational databases using semantic queries. In *Proceedings of the 15th ACM international conference on Information and knowledge management*. Arlington, Virginia, USA, 2006.
- [38] G. Reese, R. Jay, Yarger, and T. King. *Managing & Using MySQL*. 2nd Edition. O'Reilly, 2002.
- [39] P. Rob and C. Coronel. *DATABASE SYSTEMS: Design, Implementation, and Management*. International Thomson Publishing, 1997.
- [40] K. Rustan, M. Leino and G. Nelson. TOOL DEMONSTRATION An extended static checker for modula-3. Springer Berlin / Heidelberg, 1998.
- [41] S. Sadiq, M. Orłowska, W. Sadiq, and J. Lin. SQLator – An online sql learning workbench. In *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*. Leeds, United Kingdom, 2004.
- [42] S. Sanders and T. Danciu. 2005. Creating a DBA dashboard for MySQL. [Online] Available at <http://dev.mysql.com/tech-resources/articles/dba-dashboard.html> (Accessed May 2010)

- [43] R. Schumacher and A. Lentz. 2005. Dispelling the Myths. [Online] Available at <http://dev.mysql.com/tech-resources/articles/dispelling-the-myths.html>, (Accessed: June 2010)
- [44] M. Seltzer. Beyond relational databases. ACM, 2005.
- [45] A. Silberschatz, H. F. Korth, and S. Sudarshan. *DATABASE SYSTEM CONCEPTS*. 4th Edition. McGraw-Hill Higher Education, 2001.
- [46] M. Stonebraker and U. Cetinental. One size fits all: An idea whose time has come and gone. In *Proceedings of the 21st International Conference on Data Engineering*. IEEE Computer Society, 2005.
- [47] R. K. Stephens and R. R. Plew. *SAMS Teach Yourself SQL*. 3rd Edition. Sams, 1999.
- [48] Tool Command Language. Tool command official website. [Online] Available at: <http://www.tcl.tk/>, (Accessed: December 2009).
- [49] C. Turker and M. Gertz. Semantic integrity support in SQL:1999 and commercial (object-)relational database management systems. Springer-Verlag New York Inc, 2001.
- [50] A. Weiss. The MySQL Model. [Online] Available at http://www.wdvl.com/Authoring/Languages/Perl/PerlfortheWeb/mysql_mode1.html, (Accessed: March 2010)
- [51] D. N. Xu. Extended Static Checking for Haskell. In *Proceedings of the 2006 ACM SIGPLAN workshop on Haskell*. Portland, Oregon, USA, 2006.

[52] K. Usman. 2004. Introducing MySQL: A powerful RDBMS. [Online]
Available at <http://www.webmasterstop.com/34.html>, (Accessed: June 2010)



UNIVERSITY *of the*
WESTERN CAPE