**Chapter 5: Notes on MySQL**

MySQL is an open-source relational database management system that is owned by Oracle Corporation, which describes it as “the world’s most popular open source database”. It uses a strictly relational model, and does not support the object-relational features described in Chapter 8 of the textbook. There is a free version, called the community edition, and several others. It is available for a wide variety of platforms and includes many features besides the basic ones described here. The community edition is sufficient for our purposes. It is easily downloaded and installed from the MySQL website at <http://www.mysql.com/downloads/mysql/> MySQL Lab Exercise 5.1, which is included in a file by that name in this directory, has more details about downloading and installing the community edition.

This document covers some of the differences between the Oracle DBMS code described in Chapter 5 of the textbook and MySQL. You should read the corresponding sections in the textbook before reading each section of this document. You can also see more information about MySQL by using the help facility available from within MySQL. From command line, you can see the help topics at any time by entering the following command at the prompt, mysql>

help contents

This command displays the help categories available, and you can then ask for help on any of those categories. For additional information, you can also access the online documentation at <http://dev.mysql.com/doc/> There are also many tutorials on MySQL available on the Internet.

**Text Section 5.3: MySQL Data Definition Language**

The most useful data definition language commands for beginners are CREATE INDEX, ALTER TABLE, RENAME TABLE, DROP TABLE, and DROP INDEX. The syntax for these commands is very similar to Oracle’s, but there are some minor differences. To see all the data definition commands, you can enter, at the command prompt,

help data definition

*Text Section 5.3.1: CREATE TABLE*

To create a database in which your tables will reside, you can use the command

CREATE DATABASE [IF NOT EXISTS] *database\_name*;

You can optionally use the form CREATE SCHEMA instead. You can then connect to the database either by

USE *database\_name*;

or

CONNECT *database\_name*;

To create a table, MySQL uses a form very similar to Oracle’s

CREATE TABLE [IF NOT EXISTS] *table\_name*(*column\_list*)[*table options*][*partition options*];

See the subdirectory called MySQLUnivDB-Create&Populate in the same directory as this document. Copy and paste the contents of the MySQL\_UnivDB\_ DDL-Fig5\_2 file at the command line, one table at a time. (Note: to paste the command, click on the icon at the top left of the MySQL screen, choose Edit, then Paste.)

Read the code, and execute the commands. The column list includes the name and data type of each column, and constraints such as NULL/NOT NULL, CHECK, DEFAULT, AUTO\_INCREMENT, PRIMARY KEY, FOREIGN KEY and so on, similar to Oracle. Both column-level and table-level constraints can be used, and constraints can be given names. For foreign keys, you can optionally add ON DELETE and ON UPDATE options which include RESTRICT, CASCADE, SET NULL, and NO ACTION specifications. The AUTO\_INCREMENT specification for a column is similar to AutoNumber in Access, or sequences in Oracle. It uses a sequence of integers, starting by default at 1, which it automatically increments, assigning the next value to new records when they are entered.

To see all the variations of the CREATE TABLE command, enter at the command prompt,

help create table

To see all the tables you have created in a database, enter at the command prompt

show tables;

To see the structure of a particular table, enter

describe *table\_name*;

To see a list of all databases that exist on the server, enter

show databases;

*Data Types*

MySQL includes several numeric types, string types, date and time types, BOOLEAN, BLOB, and spatial types. The most commonly used numeric types are INTEGER(n), and DECIMAL(n,d). For strings, VARCHAR(l) and CHAR(l) are used for varying-length and fixed-length strings respectively. Note that the VARCHAR2 type, often used in Oracle, does not exist in MySQL. For the date type, MySQL uses the default format 'YYYY-MM-DD' unlike Oracle. To see all the data types, enter at the command line

help data types

*Other DDL Commands*

An index is automatically created for the primary key of a table. Other indexes can be created by using the CREATE INDEX command, which is essentially the same as in Oracle, except it does not include the CLUSTER option. The ALTER TABLE, RENAME TABLE, and DROP TABLE commands are also similar in syntax to those in Oracle. You can drop an entire database by entering

DROP DATABASE [IF EXISTS] database\_name;

**Text Section 5.4: MySQL Data Manipulation Language**

To follow this section, you should populate the Univ database you created earlier. Find the subdirectory called MySQLUnivDB-Create&Populate in the same directory as this document, open the file called MySQL\_UnivDB\_InsertStatements\_likeFig5\_3 and copy and paste the INSERT commands at the MySQL prompt.

The SELECT command is similar to Oracle’s. The syntax is

SELECT [DISTINCT] *column\_list*

FROM *table\_list*

[WHERE *conditions*]

[GROUP BY *group* [HAVING *group\_conditions*]]

[ORDER BY *sort\_columns*]

[LIMIT [*beginning\_row*,] *number\_retrieved*];

The LIMIT option allows you to limit the number of rows retrieved. You can optionally specify the row to begin the query at, and the maximum number of records to be selected. MySQL DML includes all the standard operators and options, and also allows conditions of the type BETWEEN *min* AND *max* for specifying that a value is between the minimum and maximum specified, inclusive. It can be used for any ordered data type. MySQL versions of the SELECT examples explained in Section 5.4 of the text can be found in this directory in a subdirectory called MySQL Code for Examples, in a file called MySQL SELECT Examples 1\_23&LIMIT&BETWEEN. Except for the last two examples, which illustrate MySQL’s LIMIT and BETWEEN options for queries, the SQL code is identical to that for Oracle. You should copy and execute these commands in MySQL, and create and test some of your own queries.

*Text Section 5.4.4: MySQL Operators for Updating*

The UPDATE command works the same way as the corresponding Oracle command, but users can specify an ordering by using ORDER BY, and they can also specify a limit. MySQL versions of the SELECT examples explained in Section 5.4.4 of the text can be found in the MySQL Code for Examples subdirectory, in the file called MySQL Update Examples1-6&LIMIT&ORDER BY. They are identical to the Oracle SQL code, except for the last two examples which illustrate MySQL’s LIMIT and ORDER BY options for updates. Also recall that when a foreign key is defined for a table, the creator has the option of specifying the action to be taken when a parent is updated. These include RESTRICT, CASCADE, SET NULL, and NO ACTION. On an update of the parent record, this specification determines what should be done with the dependent records. Run the queries in the file and experiment with some examples of your own.

The INSERT command has the same syntax as the corresponding Oracle command, but MySQL uses a different format for DATE values, and uses AUTO\_INCREMENT instead of sequences. Examples can be found in the MySQL Code for Examples subdirectory, in the file called MySQL INSERT Examples1-6&DATE&AUTO\_INCREMENT. Note that the standard format for dates in MySQL is ’yyyy-mm-dd’, as shown in Example 5. The CURDATE() function returns the current date, as shown in Example 5(c). It is similar to Oracle’s SYSDATE. MySQL has several subtypes of the DATE type, including YEAR, which can used as the data type of a column in a CREATE TABLE command. Because this subtype exists, we can use a type-cast to convert from a full DATE type to a YEAR type, retrieving only the year portion of a data, as shown in Example 5(d). Example 5(e) shows that slashes can be used to separate the elements of a date, but the form ’yyyy/mm/dd’ must still be used. Example 6 illustrates the use of AUTO\_INCREMENT. The column must be an integer type and the auto\_increment option must be specified when the table is created (or altered). Records must be inserted without giving a value for the auto\_increment column, since the system generates the values for the column. The current value of the column can be found by using SELECT last\_insert\_ID(); as shown in Example 6(c). Because the sequence being generated does not have a name, retrieving a value from the sequence itself is limited to the last value generated during a session, so we need to work around that limitation when inserting foreign keys in a dependent table. A solution is illustrated in Example 6(d). Try out the examples in the file, and also try some of your own design.

The four DELETE examples shown in Section 5.4.4 of the textbook work without modification in MySQL, as shown in the file MySQL Delete Examples 1-4&LIMIT in the MySQL Code for Examples subdirectory. MySQL also allows records to be deleted from multiple tables, as shown in Example 5. It is possible to use ORDER BY and LIMIT when deleting records from a single table, as shown in Example 6. Note that it is important to be aware of referential integrity constraints when doing DELETE. The effect on dependent records when the parent is deleted is determined by the options chosen when the tables are created. Examples 3-6 illustrate some of these effects. You should execute the commands, study the results, and try out some of your own.

MySQL also has a REPLACE command that can either update an old record or insert a new one, depending on whether the record exists already. The form is similar to the INSERT form.

REPLACE INTO *tablename*(*colnames*)VALUES(*column\_values*);

The file MySQL Replace Examples shows what happens for the two cases. Execute these examples and design and execute some additional examples.

*Text Section 5.4.5: Views*

Views can be defined in MySQL using a statement similar to Oracle’s

CREATE [OR REPLACE] VIEW viewname[(col-list)]AS *SELECT-statement*;

The file called MySQL Create View Examples 1\_6 shows that the same SQL statements shown in Section 5.4.5 for Oracle run on MySQL. We added the OR REPLACE option to each one, but otherwise they are unchanged. Views can be used the same way as they are in Oracle. Try out the commands in the file and add some of your own.

**Text Section 5.5: MySQL Constraints and Triggers**

Triggers can be written for MySQL databases using syntax similar to Oracle’s, and they function the same way as they do in Oracle. The OR REPLACE is not used as an option in the CREATE TRIGGER statement. To signal the end of the trigger, the delimiter must be changed from the usual semicolon to some other special character, so that MySQL will not treat semicolons that are part of the trigger code as the delimiter for the entire trigger. This is done by a simple statement such as delimiter $$ at the beginning of the code. The delimiter should be set back to the semicolon by writing delimiter ; at the end of the trigger code. Another difference is that MySQL uses the prefix NEW. instead of :NEW. to refer to a new record , and OLD. rather than :OLD for the old record. The MySQLCode for Examples subdirectory contains another directory called MySQL Triggers with files for the MySQL code for Figure 5.4, including creating the tables and the trigger code for the examples. Note that the declarations for local variables go after the BEGIN statement, as shown in the code for trigger EnrollRequest.

Constraints, including primary key and foreign key constraints, are checked immediately, with no DEFERRED option. Since this can cause a problem in a transaction that involves foreign key constraints that need to be temporarily violated, you can suspend foreign key checking using the following code

SET FOREIGN\_KEY\_CHECKS=0;

... do the transaction...

SET FOREIGN\_KEY\_CHECKS=1;

An example is shown in the file called My\_SQL ConstraintChecking, which also shows how to use the show create table *tablename* command to show the constraints that exist for a table. Note that the DESCRIBE TABLE command does not list the table constraints.

**Text Section 5.7.1: MySQL Stored Routines - Procedures and Functions**

MySQL allows users to create stored procedures and functions that are similar to SQL PSMs in Oracle. A simple example of a procedure is shown in the file MySQLFig5\_6-ProcedureFindName. It shows that the form for creating a stored procedure is

CREATE PROCEDURE *name* (*parameter list*)

BEGIN

*procedure body*

END;

The Oracle option OR REPLACE is not used. As with triggers, it is necessary to reset the delimiter to a character other than the semicolon before the procedure code, and to reset it afterwards. We have used $$ as the delimiter. The parameter list can specify IN, OUT or INOUT for each parameter. This word is followed by the name and type of the parameter. If there are no parameters, an empty list ( ) must follow the procedure name. The BEGIN and END are not needed if the body consists of a single statement. Declarations of local variables, if any, follow the BEGIN. Most valid SQL statements can be placed in the body. The SELECT *database attributes* INTO *program variables* WHERE *condition* form of the SELECT statement can be used to retrieve values from the database and store them in local variables, as shown in the example, where we simply display their values using a simple SELECT. Once the procedure has been compiled, it is executed using a CALL statement and the actual parameters, as in

CALL PROCEDURE FINDNAME(’S1001’);

Unlike SQL\*Plus, MySQL will not complete compilation if the procedure contains an error. To drop a procedure, use

DROP PROCEDURE *procedurename*;

Functions are created using

CREATE FUNCTION *function\_name* (*parameter\_list*) RETURNS *type*

BEGIN

*procedure body, which must include a RETURN statement*

END;

Since only IN parameters can be used for functions, the parameter list includes only the name and type of each parameter. Functions can be written for user-defined functions as well as for stored routines.

The flow of control statements are similar to Oracle’s and include IF…THEN…ELSE…END IF, LOOP…END LOOP,REPEAT…UNTIL…END REPEAT, and WHILE…END WHILE.

Cursors are used in almost the same way as in Oracle, as shown in the file MySQLFig5\_7-ProcedureCursorSample. A cursor is declared using

DECLARE *cursor-variable* CURSOR FOR *query*;

The cursor must be opened to execute the query, using

*OPEN cursor\_variable*;

The FETCH statement is used to retrieve a row of the results, as in

FETCH *cursor-variable* INTO *variable, variable*,,,

In our example, we simply display the values stored in those variables by using a SELECT statement. At the end, the cursor is closed using

CLOSE *cursor-variable*

*MySQL Error Handling*

Like Oracle, MySQL has options for the user to specify what actions should be taken when an error occurs in a procedure or program. Some MySQL error handling is illustrated in the previous procedure in the file MySQLFig5\_7-ProcedureCursorSample. As described in Section 5.7.2 for embedded SQL in Oracle, a status variable called SQLSTATE is used for communicating error conditions. A value of ‘02000’ means no data has been found. In our example, this variable is used to control the loop. We have declared a CONTINUE HANDLER that specifies that when the value of SQLSTATE is ‘02000’, the BOOLEAN variable done, which we declared earlier, should be set to 1. When this occurs, we end the repeat. Besides the CONTINUE handler, which allows the procedure to continue to execute, we can also use EXIT, which ends the procedure. The general form for declaring a handler is

DECLARE {CONTINUE/EXIT} HANDLER FOR *condition\_value* [,*condition\_value*….] statement;

The *condition\_value* can be a value of SQLSTATE (as illustrated in our example for SQLSTATE of ‘02000’), a named condition that we have declared previously, SQLWARNING, NOT FOUND, SQLEXCEPTION, or a MySQL\_error\_code. The same handler can be used for several conditions. If the condition occurs the statement specified in the declaration is executed. It can be either a simple statement such as the one illustrated in our example, SET done=1, or a compound statement delimited by BEGIN…END. If the handler is a CONTINUE, the procedure continues after that statement is executed. If it is an EXIT handler, the procedure ends. The following command is used to create a named condition

DECLARE condition\_name CONDITION FOR condition\_value;

The *condition\_value* may be a value of SQLSTATE or a MySQL error code. A complete list of these values and codes can be found in the MySQL documentation. For example, we can declare the conditions foreign\_key\_error and duplicate key values

DECLARE foreign\_key\_error CONDITION FOR 1216;

DECLARE duplicate\_key CONDITION FOR SQLSTATE '23000';

Then we can write a handler for these conditions using

DECLARE EXIT HANDLER FOR foreign\_key\_error, duplicate\_key;

*Text Sections 5.7.2 and 5.7.3:Connectors and APIs*

There are many connectors and APIs available for MySQL, allowing users to connect to a MySQL database from a variety of languages and environments. There are connectors for ODBC, JDBC and .NET, among others, and APIs for C, C++, Eiffel, Java, PHP, Perl, Python, Ruby and other languages.

**Text Section 5.8: Schema Information**

The MySQL version of a system catalog is called the INFORMATION\_SCHEMA. It contains information about all the databases on the MySQL server. Users can query the schema using SELECT statements of the form

SELECT *columns*

FROM INFORMATION\_SCHEMA.*schema\_table\_name*

WHERE *condition*;

There are INFORMATION\_SCHEMA tables called TABLES, COLUMNS, TABLE CONSTRAINTS, VIEWS, TRIGGERS, ROUTINES, and many others. Each of these tables contains information about the corresponding objects in the database. For example, the TABLES table has columns called TABLE\_NAME, TABLE\_ROWS, and TABLE\_SCHEMA, among others. The COLUMNS table has TABLE\_NAME, SCHEMA\_NAME, COLUMN\_NAME, DATATYPE, ORDINAL\_POSITION, and others. For VIEWS, the columns include TABLE\_NAME, VIEW\_DEFINITION, DEFINER, and others. Typical queries are shown in the file called MySQL INFORMATION\_SCHEMA queries. You should execute these queries and examine the results. Details about all the options for these and other tables in INFORMATION\_SCHEMA can be found in the MySQL documentation.