**MySQL Note**

**Chapter 1**

What is MySQL? Open source relational SQL database management system.

A relational database management system is a software that –

1. Enables you to implement a database with tables, columns and indexes.

2. Guarantees the referential integrity between rows of various tables.

3. Updates the indexes automatically.

4. Interprets an SQL query and combines information from various tables.

**Definitions**

1. Database – A database is a collection of tables with related data.

2. Tables – A table is a matrix with data. A table in a database looks like a simple spreadsheet.

3. Redundancy – Storing data twice, redundantly to make the system faster.

4. Primary key – A primary key is unique. A key value cannot occur twice in one table. With a key, you can only find one row.

5. Foreign key – A foreign key is the linking pin between two tables.

6. Referential integrity – Referential integrity makes sure that a foreign key value always points to an existing row.

**Setting up a MySQL user account**

For adding a new user to MySQL, I need to add a new entry to the user table in the database mysql.

Example:

*mysql> CREATE USER ‘cbuser’@’localhost’ IDENTIFIED BY ‘cbpass’;*

*mysql> GRANT ALL ON cookbook.\* TO ‘cbuser’@’localhost’;*

*mysql> FLUSH PRIVILEGES*

“FLUSH PRIVILEGES” is to tell the server to reload the grant tables.

**Run MySQL as local user**

$ mysql –h localhost –u cbuser –p

‘cbpass’

**Creating a database and set up tables within it**

Example:

mysql> CREATE DATABASE cookbook; # create a database named cookbook

mysql> USE cookbook; # use this database and it is a default database

mysql> CREATE TABLE limbs (thing VARCHAR(20), legs INT, arms INT);

mysql> INSERT INTO limbs (thing,legs,arms) VALUES('human',2,2);  
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('insect',6,0);  
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('squid',0,10);  
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('fish',0,0);  
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('centipede',100,0);

mysql> INSERT INTO limbs (thing,legs,arms) VALUES('table',4,0);  
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('armchair',4,2);  
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('phonograph',0,1);

mysql> INSERT INTO limbs (thing,legs,arms) VALUES('tripod',3,0);  
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('Peg Leg Pete',1,2);

mysql> INSERT INTO limbs (thing,legs,arms) VALUES('space alien',NULL,NULL);

And then, to display the table,

mysql> SELECT \* FROM limbs;

Let’s say you closed all program and re-initiated MySQL session. In order to get an access to this table data, you should select the database first by typing

mysql> USE cookbook

**Creating corresponding HTML or XML file**

For HTML,

% mysql -h localhost -u cbuser -p -H -e "SELECT \* FROM limbs WHERE legs=0" cookbook > limbs.html

For XML,

% mysql -h localhost -u cbuser -p -X -e "SELECT \* FROM limbs WHERE legs=0" cookbook

**Using user-defined variables in SQL statements**

For example, I would like to use MAX

Mysql> SELECT @ max\_limbs := MAX(arms+legs) FROM limbs;

Mysql> SELECT \* FROM limbs WHERE arms+legs = @max\_limbs;

User variables hold only single row value. If a statement returns multiple rows, the value from the last row is assigned. If the statement returns no rows, no assignment takes place and the variable retains its previous value.

User variable names are not case sensitive:

Mysql> SET @x = 1, @X = 2; SELECT @x, @X;

+-------+------+

| @x | @X |

+-------+------+

| 2 | 2 |

+-------+------+

**Chapter 2**

**Writing MySQL-based programs**

Every program that uses MySQL must first establish a connection to the server. Most programs also select a default database, and well-behaved MySQL programs close the connections to the server when they’re done with it. I will be learning:

*How to write library files*

*Additional techniques for obtaining connection parameters*

MySQL client API architecture

Each MySQL programming interface covered uses a two-level architecture:

* The upper level provides database-independent methods that implement database access in a portable way that’s the same whether you use MySQL, PostgreSQL, Oracle, or whatever.
* The lower level consists of a set of drivers, each of which implements the details for a single database system.

**Connecting, selecting a database, and disconnecting**

This section shows how to perform some fundamental operations common to most MySQL programs.

*import mysql.connector*

*conn\_params = {"database": "cookbook",*

*"host": "localhost",*

*"user": "cbuser",*

*"password": "cbpass",*

*}*

*try:*

*conn = mysql.connector.connect(\*\*conn\_params)*

*print("Connected")*

*except:*

*print("Cannot connect to server")*

*else:*

*conn.close()*

*print("Disconnected")*

**Writing library files**

To check the default value of sys.path on your system, run Python interactively and enter the few commands:

*% python*

*>>> import sys*

*>>> sys.path*

cookbook.py file is added to the library variable. It will connect to the SQL server for me. The usage is

*import mysql.connector*

*import cookbook*

*conn = cookbook.connect()*

DONE!

**Executing statements and retrieving results**

In here, I would like a program to send an SQL statement to the MySQL server and retrieve its results. We are going to use another dataset “profile.sql”. Go to sql folder and import the data in our database.

% mysql –h localhost –u cbuser –p cookbook < profile.sql

which will prompt the password “cbpass”. I will skip this step in the later part because this is such a basic step.

SQL statement categories

SQL statements can be grouped into two broad categories, depending on whether they return a result set:

* Statements that return no result set, such as INSERT, DELETE, or UPDATE. As a general rule, statements of this type generally change the database in some way. There are some exceptions, such as USE db\_name, which changes the default database for your session without making any changes to the database itself. The example data-modifying statement used in this section is an UPDATE:
  + UPDATE profile SET cats = cats+1 WHERE name = ‘Sybil’

We will cover how to execute this statement and dertermine the number of rows that it affects.

* Statements that return a result set, such as SELECT, SHOW, EXPLAIN, or DESCRIBE. I refer to such statement generically as SELECT statements, but you should understand that category to include any statement that returns rows. The example row-retrieval statement used in this section is a SELECT:
  + SELECT id, name, cats FROM profile

We are using “execute” attribute to perform SQL statement in python.

*cursor = conn.cursor()*

*cursor.execute(“UPDATE profile SET cats = cats + 1 WHERE name = ‘Sybil’”)*

*cursor.close() # you need to close it manually*

*conn.commit() # basically, you are saving the update*

Now, let’s print it out in Python

*import cookbook*

*conn = cookbook.connect()*

*cursor = conn.cursor()*

*cursor.execute("SELECT id, name, cats FROM profile")*

*while True:*

*row = cursor.fetchone()*

*if row is None:*

*break*

*print("id: %s, name: %s, cats: %s" % (row[0], row[1], row[2]))*

*print("Number of rows returned: %d" % cursor.rowcount)*

*cursor.close()*

The fetchone() method returns the next row as a sequence, or None when there are no more rows.

*# 2. another method - I like this method better.*

*cursor = conn.cursor()*

*cursor.execute("SELECT id, name, cats FROM profile")*

*for (id, name, cats) in cursor:*

*print("id: %s, name: %s, cats: %s" % (id, name, cats))*

*print("Number of rows returned: %d" % cursor.rowcount)*

*cursor.close()*

*# 3. and yet, there is one more way to do this*

*cursor = conn.cursor()*

*cursor.execute("SELECT id, name, cats FROM profile")*

*rows = cursor.fetchall()*

*for row in rows:*

*print("id: %s, name: %s, cats: %s" % (row[0], row[1], row[2]))*

*print("Number of rows returned: %d" % cursor.rowcount)*

*cursor.close()*

The fetchall() can be convenient when you must iterate through the rows of the result set more than once or access individual values directly.