Assignment 01

(GROUP A)

Aim

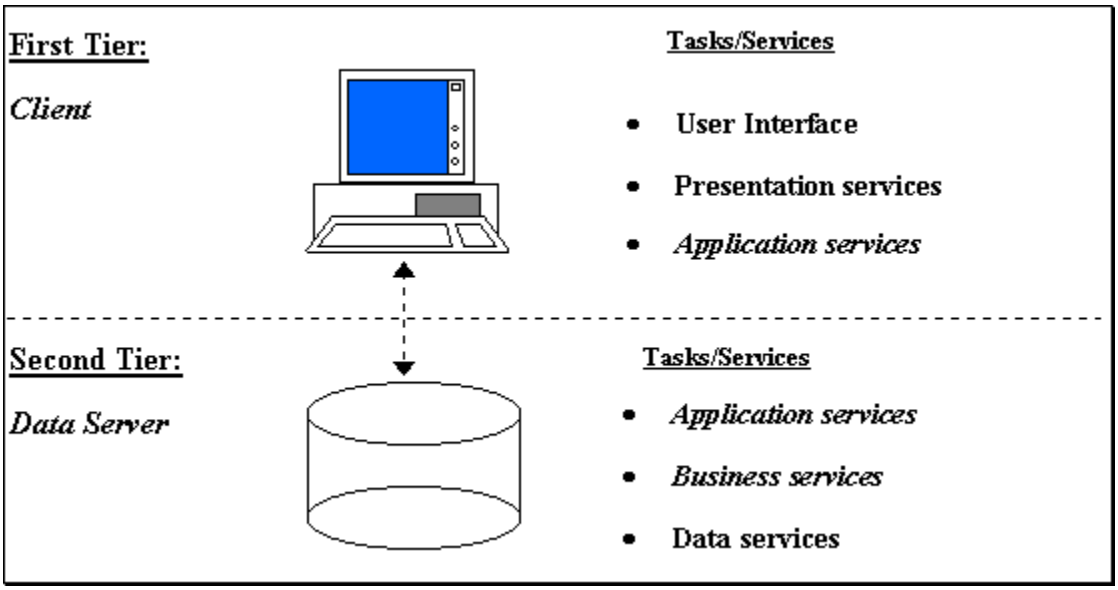
DBMS using connections(Client-Data sever, two tier) Oracle/MySQL (ODBC/JDBC), SQL prompt to create data base tables insert, update data values, delete table, use table, select queries with/without where clause,demonstrate use of stored procedure/function.

Requirements

1. Computer System with Linux/Open Source Operating System.
2. Mysql Server
3. JDK
4. JDBC Connector
5. Eclipse

Theory

Two-Tier Architecture:



This assignment provides a tutorial introduction to MySQL by showing how to use the MySQL client program to create and use a simple database. MySQL (sometimes referred to as the \ terminal monitor" or just \ monitor ") is an interactive program that allows you to connect to a MySQL server, run queries, and view the results. MySQL may also be used in batch mode: you place your queries in a *le* beforehand, then tell MySQL to execute the contents of the *le*. Both ways of using MySQL are covered here.
To see a list of options provided by MySQL, invoke it with the –help option:

shell> mysql –help
This assignment assumes that MySQL is installed on your machine and that a MySQL server is available to which you can connect. If this is not true, contact your MySQL administrator.
(If you are the administrator, you will need to consult other sections of this manual.)

Connecting to and Disconnecting from the Server

To connect to the server, you'll usually need to provide a MySQL user name when you invoke mysql and, most likely, a password. If the server runs on a machine other than the one where you log in, you'll also need to specify a hostname. Contact your administrator to find out what connection parameters you should use to connect (that is, what host, user name, and password to use). Once you know the proper parameters, you should be able to connect like this:

shell> mysql -h host -u user -p
Enter password: ********

The ******** represents your password; enter it when mysql displays the Enter password: prompt.
If that works, you should see some introductory information followed by a mysql> prompt:

shell> mysql -h host -u user -p
Enter password: ********
Welcome to the MySQL monitor. Commands end with ; or $\backslash g$.
Your MySQL connection id is 459 to server version: 3.22.20a-log
Type 'help' for help.
mysql>

The prompt tells you that mysql is ready for you to enter commands.
Some MySQL installations allow users to connect as the anonymous (unnamed) user to the server running on the local host. If this is the case on your machine, you should be able to connect to that server by invoking mysql without any options:

shell> mysql

After you have connected successfully, you can disconnect any time by typing QUIT at the mysql> prompt:

mysql> QUIT
Bye

You can also disconnect by pressing Control-D. Most examples in the following sections assume you are connected to the server. They indicate this by the mysql¿ prompt.

Creating and Using a Database

Suppose you have several pets in your home (your menagerie) and you'd like to keep track of various types of information about them. You can do so by creating tables to hold your data and loading them with the desired information. Then you can answer different sorts of questions about your animals by retrieving data from the tables. This section shows you how to:

- Create a database
- Create a table
- Load data into the table

- Retrieve data from the table in various ways
- Use multiple tables

Use the SHOW statement to find out what databases currently exist on the server:

```
mysql> SHOW databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| te                 |
| test               |
+--------------------+
```

If the test database exists, try to access it:
mysql> USE test
Database changed
mysql>

Note that USE, like QUIT, does not require a semicolon. (You can terminate such statements with a semicolon if you like; it does no harm.) The USE statement is special in another way, too: it must be given on a single line.
You can use the test database (if you have access to it) for the examples that follow, but anything you create in that database can be removed by anyone else with access to it. For this reason, you should probably ask your MySQL administrator for permission to use a database of your own. Suppose you want to call yours enagerie. The administrator needs to execute a command like this:

mysql> GRANT ALL ON menagerie ∗ TO your_mysql_name;

where your_mysql_name is the MySQL user name assigned to you.

If the administrator creates your database for you when setting up your permissions, you can begin using it. Otherwise, you need to create it yourself:
mysql> CREATE DATABASE menagerie;

Under Unix, database names are case-sensitive (unlike SQL keywords), so you must always refer to your database as menagerie, not as Menagerie, MENAGERIE, or some other variant. This is also true for table names. (Under Windows, this restriction does not apply, although you must refer to databases and tables using the same lettercase throughout a given query.)

Creating a database does not select it for use; you must do that explicitly. To make menagerie the current database, use this command:
mysql> USE menagerie
Database changed

Your database needs to be created only once, but you must select it for use each time you begin a mysql session. You can do this by issuing a USE statement as shown above. Alternatively, you can select the database on the command-line when you invoke mysql. Just specify its name after any connection parameters that you might need to provide. For example:

shell> mysql -h host -u user -p menagerie
Enter password: ********

Steps Required using JDBC

- **Import the packages** Requires that you include the packages containing the JDBC classes needed for database programming . Most often, using $importjava.sql.*$ will suffice.
- **Register the JDBC driver** Requires that you initialize a driver so you can open a communication channel with the database.
- **Open a connection** Requires using the $DriverManager.getConnection()$ method to create a Connection object, which represents a physical connection with database server.
  T o create a new database, you need not to g ive any database name while preparing database URL as mentioned in the below example.
- **Execute a query** Requires using an object of type Statement for building and submitting an SQL statement to the database.
- **Clean up the environment** Requires explicitly closing all database resources versus relying on the JVM's garbag e collection.

CREATE TABLE

Creating the database is the easy part, but at this point it's empty, as SHOW TABLES will tell you:
mysql> SHOW TABLES;
Empty set (0.00 sec)
Use a CREATE TABLE statement to specify the layout of your table:

Syntax:

create table < *tablename* >
{
fieldname-1 datatype constraints if any,
fieldname-2 datatype constraints if any,
.
fieldname-n datatype constraints if any,
};
create table < *tablename* > as
(
select(att-list) from < *existing_tablename* >
);

mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);

VARCHAR is a good choice for the name, owner, and species columns because the column values will vary in length. The lengths of those columns need not all be the same, and need not be 20. You can pick any length from 1 to 255, whatever seems most reasonable to you. (If you make a poor choice and it turns out later that you need a longer field, MySQL provides an ALTER TABLE statement.)

Now that you have created a table, SHOW TABLES should produce some output:
mysql> SHOW TABLES;

```
+----------------+
| Tables_in_test |
+----------------+
| e1             |
| pet            |
+----------------+
```

To verify that your table was created the way you expected, use a DESCRIBE statement:
mysq> DESCRIBE pet;

```
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

You can use DESCRIBE any time, for example, if you forget the names of the columns in your table or what types they are.

INSERT

This is used to add one or more rows to a table. The values are separated by commas and the data types char and date are enclosed in apostrophes. The values must br entered in the same order as they are defined.

- **Inserting a single row into a table**
  insert into < *tablename* > values(fieldvalue-1,fieldvalue-2,,fieldvalue-n);
- **Inserting more than one record using a single insert command**
  insert into < *tablename* > values(&fieldname-1,&fieldname-2,&fieldname-n);
- **Skipping the fields while inserting**
  insert into <tablename(coln names to which datas to b inserted)> values (list of values);
  Other way is to give null while passing the values.
  insert into < *tablename* >(select(att_list) from <existing table name>);

SELECT

It is used to retrieve information from the table.it is generally referred to as querying the table. We can either display all columns in a table or only specify column from the table.

SELECT(att_list) FROM < *tablename* > [WHERE <condition/expression>];

- **Retrieval of all columns from a table**
  Select ∗ from tablename; // This query selects all rows from the table.
- **Retrieval of specific columns from a table**
  It retrieves the specified columns from the table.
  Syntax: Select column_name1, ..,column_namen from table name;

- **Elimination of duplicates from the select clause**
  It prevents retrieving the duplicated values .Distinct keyword is to be used.
  Syntax: Select DISTINCT col1, col2 from table name;

- **Select command with where clause**
  To select specific rows from a table we include where clause in the select command. It can appear only after the from clause.
  Syntax: Select column_name1, ..,column_namen from table name where condition;

- **Select command with order by clause**
  Syntax: Select column_name1, ..,column_namen from table name where condition order by colmnname;

- **Select command to create a table**
  Syntax: create table tablename as select $*$ from existing_tablename;

- **Select command to insert records**
  Syntax: insert into tablename (select columns from existing_tablename);

UPDATE

It is used to alter the column values in a table. A single column may be updated or more than one column could be updated.
Syntax: update $<$ *tablename* $>$ set(fieldname-1 = value, fieldname-2 = value,,fieldname-n = value)[WHERE ¡condition/expression>];

DELETE

After inserting row in a table we can also delete them if required. The delete command consists of a from clause followed by an optional where clause.
Syntax:delete from $<$ *tablename* $>$ [where ¡condition/expression>];

STORED PROCEDURE

Stored procedures are similar to user-defined functions (UDFs). The major difference is that UDFs can be used like any other expression within SQL statements, whereas stored procedures must be invoked using the CALL statement.

Syntax:

DELIMITER //
CREATE PROCEDURE *procedure_name*()
BEGIN
SELECT * FROM products;
END //
DELIMITER ;

CALL *procedure_name*() //Calling a procedure;

Mathematical Modelling

Terms used

Consider S as the system, then

$$S = \{q_0, q_f, Q, X, Y, D, ND, S, F, Fn\}$$

where,

q_0**:** Initial State

Q: Set of Intermediate States

X: Set of Expected Input and Possible Inputs

Y: Set of Expected Output and Possible Outputs

D: Set of Deterministic Data

ND: Set of Non-Deterministic Data

S: Set of Success Cases

F: Set of Failure Cases

Fn: Set of Functions used in the System

q_f**:** Final state

System Description

let S be the System for the given problem

$$S = \{q_0, q_f, Q, X, Y, D, ND, S, F, Fn\}$$

where,

q_0 : Program is compiled and executed.

Q $=\{q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}$ where,
- q_1 : Data Source Net created
- q_2 : Connection Established
- q_3 : Display menu
- q_4 : Table Creation
- q_5 : Data Insertion
- q_6 : Data Updation
- q_7 : Data Deletion
- q_8 : Display Table Contents
- q_9 : Procedure is called

X $=\{x_1\}$ where,
- x_1 : Data values for column

Y $=\{y_1, y_2\}$ where,
- y_1 : operation completed
- y_2 : operation failed

D $=\{d_1\}$ where,
- d_1 : database and table name

ND $=\{n_1\}$ where,
- n_1 : Output, since output depends upon input. $n_1 \epsilon Y$

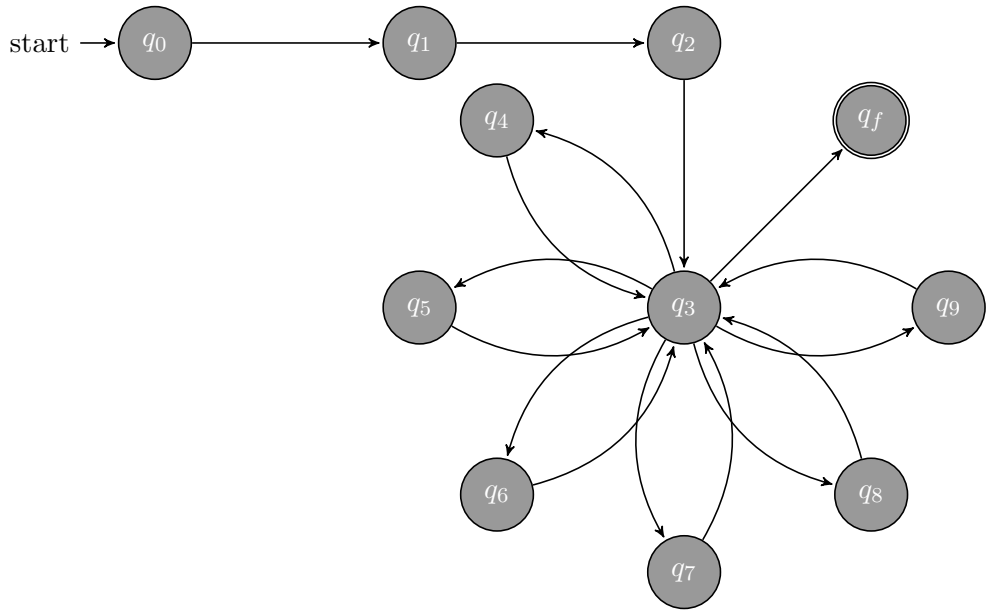
S $=\{y_1\}$

F $=\{y_2\}$

Fn $=\{fn_1, fn_2, fn_3, fn_4, fn_5, fn_6, \}$where,
fn_1 : create table function.
fn_2 : insert function.
fn_3 : update function.
fn_4 : delete function.
fn_5 : select function.
fn_6 : call procedure function.

q_f : final state when program finished the execution after displaying the data.

State Diagram



Result

Thus, we implemented DBMS queries using connections (Client-Data sever, two tier) Oracle/MySQL (ODBC/JDBC), SQLprompt to create data base tables insert, update data values, delete table, use table, select queries with/without where clause,demonstrate use of stored procedure/function.