MIT World Peace University

Database Management Systems
Second Year B. Tech, Semester 4

Designing of Entity Relationship Diagram

Assignment No. 1

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

April 30, 2023

Contents

1 Aim

Design an ER diagram for different case studies.

2 Objectives

To study creation of an ER diagram.

3 Problem Statement

Requirements of the Company The Company is organized into:

1. DEPARTMENTs. Each department has a name, number and an employee who manages the department. We keep track of the start date of the department manager.
2. Each department controls a number of PROJECTS. Each project has a name, Number, and is located at a single location.
3. We store each EMPLOYEEs social security number, address, salary, sex and birthdate. Each employee works for one department but may work on several projects. We keep track of the number of hours per week that an employee currently works on each project. We also keep track of the direct supervisor of each employee.
4. Each employee may have a number of DEPENDENTS. For each dependent we keep track of their name, sex, birthdate, and relationship to the employee.

4 Theory

4.1 Entity Relationship Diagram Definition

An ER diagram is a graphical representation of entities and their relationships to each other, typically used in computing in regard to the organization of data within databases or information systems. It is often used to depict the logical structure of databases and information systems.

4.2 Entities

An entity is a person, place or thing. It is a thing that can be distinguished from other things. It is an object or concept about which data is stored. An entity is a noun (for example, customer, employee, product, or order). The entity is the basic building block of the data model. The entity is the object that is represented in the database. An entity is a thing that exists independently of any other thing. Entities are the nouns in the problem domain.

4.3 Relationships

A relationship is a link between two or more entities. The relationship is the verb in the problem domain. The relationship is the association between two entities. Relationships are the verbs in the problem domain.

4.3.1 Binary Relationship

A binary relationship is a relationship between two entities.

4.3.2 Unary Relationship

A unary relationship is a relationship between one entity.

4.3.3 Ternary Relationship

A ternary relationship is a relationship between three entities.

4.4 Attributes

An attribute is a characteristic or quality of an entity or relationship. Attributes are the adjectives in the problem domain. Attributes are the properties of an entity or relationship. An attribute is a property of an entity that can be assigned a value. Attributes are the adjectives in the problem domain.

4.5 Cardinality

Cardinality is the number of occurrences of an entity or relationship in a relationship set. The cardinality of a relationship is the number of occurrences of an entity in a relationship. The cardinality of a relationship is the number of occurrences of an entity in a relationship.

4.6 Weak Entity

A weak entity is an entity that cannot exist on its own. It must have a relationship with another entity. A weak entity is identified by a relationship with another entity. It must use a foreign key in conjunction with its attributes to create a primary key.

4.7 Primary Key

A primary key is a unique identifier for an entity.

4.8 Foreign Key

A foreign key is a primary key of another entity.

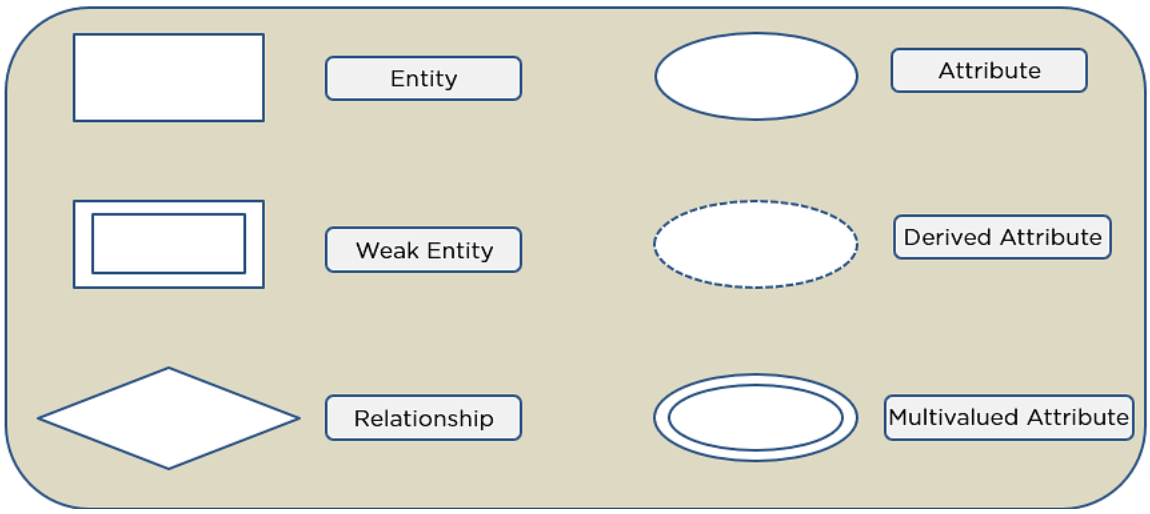

Figure 1: Symbols used in drawing ER diagrams

5 Procedure for Drawing ER Diagram

1. Ideentify the entities
2. Draw relationship table
3. Draw rough er diagram.
4. Draw each entity in depth, that have all the attributes.
5. Show the relationship with cardinality. If it is unary ternary etc.

5.1 Entities in the Problem Statement

1. Company - Department
2. Department - Name, Number, Manager, Start Date
3. Projects - Name, Number, Location
4. Employees - SSN, Address, Salary, sex, Birthdate, Department, Supervisor, no. of hours per week.
5. Dependents - Name, sex, birthdate, Relationship.

5.2 Relationships in the Problem Statement

	Company	Department	Projects	Employees	Dependents
Company		contains			
Department			controls	has	
Projects					
Employees		manages	works on		has
Dependents				work under	

Figure 2: Relationship between Entities from the Problem Statement

6 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Draw.io for Drawing the ER diagram.

7 Output

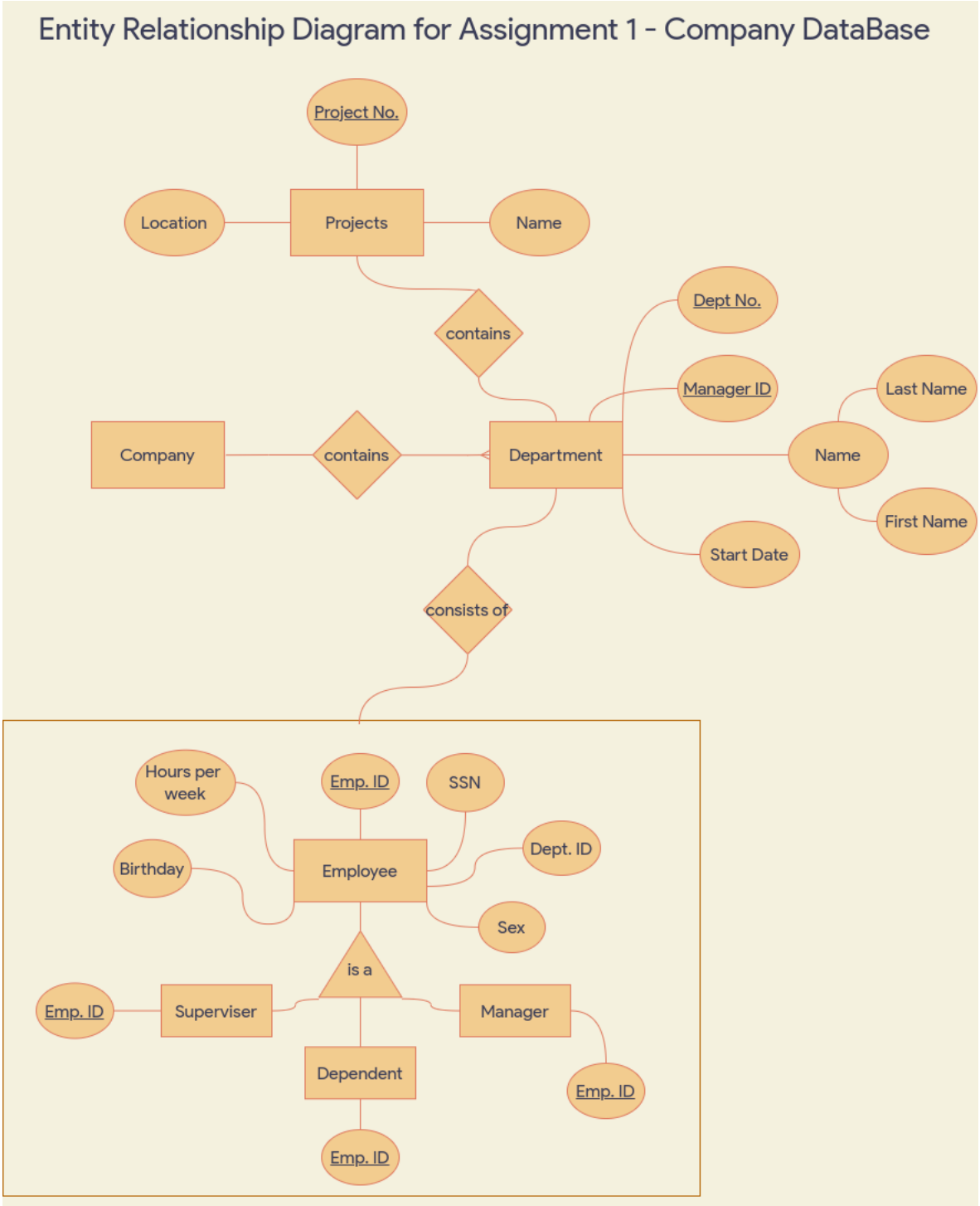

Figure 3:

8 Conclusion

Thus, we have learned to create ER diagram.

9 FAQ

1. **What are different types of attributes?**

   (a) **Simple Attributes:** The attributes which are atomic in nature are called simple attributes. For example, Name, Address, Salary, etc.

   (b) **Composite Attributes:** The attributes which are not atomic in nature are called composite attributes. For example, Name of a person is a composite attribute because it is not atomic in nature. It consists of First Name, Middle Name, and Last Name.

   (c) **Multivalued Attributes:** The attributes which can have more than one value are called multivalued attributes. For example, a person can have more than one phone number. So, Phone Number is a multivalued attribute.

   (d) **Derived Attributes:** The attributes which are not stored in the database but are derived from the other attributes are called derived attributes. For example, Age is a derived attribute because it is not stored in the database but is derived from the date of birth.

   (e) **Composite Multivalued Attributes:** The attributes which are composite and multivalued are called composite multivalued attributes. For example, a person can have more than one address. So, Address is a composite multivalued attribute.

2. **What do you mean by primary key and foreign key?**

   (a) **Primary Key:** The primary key is a unique identifier for each record in a table. It is a column or a set of columns that uniquely identifies each row in a table. The primary key must contain unique values, and cannot contain NULL values. A table can have only one primary key, which may consist of single or multiple fields.

   (b) **Foreign Key:** A foreign key is a field (or collection of fields) in one table that uniquely identifies a row of another table or the same table. In simple words, the foreign key is defined in a second table, but it refers to the primary key in the first table.

3. **What is weak entity?**
   Weak Entity: A weak entity is an entity that cannot exist on its own. It must have a relationship with another entity. A weak entity is identified by a relationship with another entity. It must use a foreign key in conjunction with its attributes to create a primary key. **Example:** A ROOM can only exist in a BUILDING. On the other hand, a TIRE might be considered as a strong entity because it also can exist without being attached to a CAR.

MIT World Peace University

Database Management Systems
Second Year B. Tech, Semester 4

Learning SQL DCL and DDL Commands

Data Definition Language and Data Control Language

Assignment No. 2

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

March 2, 2023

Contents

1 Aim

Design and Develop SQL DDL statements for different system.

2 Objectives

To study DDL, DCL commands.

3 Problem Statement

4 Theory

4.1 SQL Data Definition Language (DDL)

4.1.1 What is Data Definition Language?

Data Definition Language (DDL) is a computer language used to define the database schema. It includes commands to create, modify and drop database objects in the database. It is used to define the database structure or schema. It is also used to define the access permissions on the data, or the views that are presented to different users.

4.1.2 DDL Commands

The following are the Commands that are used in DDL:

1. CREATE - Creates a new database or a new table in a database.
2. ALTER - Modifies a database or a table.
3. DROP - Deletes a database or a table.
4. TRUNCATE - Deletes all the records from a table, including all spaces allocated for the records are removed.
5. COMMENT - Adds comments to the data dictionary.
6. RENAME - Renames an object.

4.1.3 DDL Command Syntax and Examples

1. CREATE TABLE - Creates a new database table.

```
CREATE TABLE table_name constraints
(
Column_name datatype(size) constraints default '',
Column_name datatype(size),
constraint(column_name)
);
```

2. ALTER TABLE - Changes in columns and stuff.

```
ALTER TABLE table_name
ADD column_name datatype;
```

3. DROP TABLE - Deletes a table from the database.

   ```
   DROP TABLE table_name;
   ```

4. RENAME TABLE - Renames a table.

   ```
   RENAME TABLE old_name TO new_name;
   ```

5. TRUNCATE TABLE - Deletes all the records from a table.

   ```
   TRUNCATE TABLE table_name;
   ```

6. COMMENT ON - Adds comments to the data dictionary.

   ```
   COMMENT ON TABLE table_name IS 'comment';
   ```

4.2 SQL Data Control Language (DCL)

4.2.1 What is Data Control Language?

Data Control Language (DCL) is a computer language used to define the access permissions on the data, or the views that are presented to different users. It includes commands to grant and deny privileges on database objects to users.

4.2.2 DCL Commands

The following are the Commands that are used in DCL:

1. GRANT - Gives the specified privileges to the specified user.
2. REVOKE - Takes back the specified privileges from the specified user.

4.3 DCL Command Syntax and Examples

1. GRANT - Gives the specified privileges to the specified user.

   ```
   GRANT privileges ON object_name TO user_name;
   ```

2. REVOKE - Takes back the specified privileges from the specified user.

   ```
   REVOKE privileges ON object_name FROM user_name;
   ```

5 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Draw.io for Drawing the ER diagram.

6 Input

Given Database from the Problem Statement for the Assignment for our batch. (A1 PA 20)

7 Output

```
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 10.11.2-MariaDB Arch Linux

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| class              |
| class_stuff        |
| dbms_lab           |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test               |
| test_libreoffice   |
+--------------------+
9 rows in set (0.004 sec)

MariaDB [(none)]> use dbms_lab
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [dbms_lab]> show tables;
+--------------------+
| Tables_in_dbms_lab |
+--------------------+
| books              |
| course             |
| new_book_master    |
| newauthor          |
| newbook_master     |
+--------------------+
5 rows in set (0.001 sec)

MariaDB [dbms_lab]> create table Hotel (HotelNo int Primary Key, Name varchar(50),
    City varchar(50));
Query OK, 0 rows affected (0.020 sec)

MariaDB [dbms_lab]> describe Hotel;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| HotelNo | int(11)     | NO   | PRI | NULL    |       |
```

```
| Name    | varchar(50) | YES  |     | NULL    |       |
| City    | varchar(50) | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
3 rows in set (0.002 sec)

MariaDB [dbms_lab]> create table Room (RoomNo int Primary Key, HotelNo int, Type
    varchar(50), Price int, foreign key(HotelNo) references Hotel(HotelNo));
Query OK, 0 rows affected (0.014 sec)

MariaDB [dbms_lab]> describe Room;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| RoomNo  | int(11)     | NO   | PRI | NULL    |       |
| HotelNo | int(11)     | YES  | MUL | NULL    |       |
| Type    | varchar(50) | YES  |     | NULL    |       |
| Price   | int(11)     | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
4 rows in set (0.002 sec)

MariaDB [dbms_lab]> create table Booking (HotelNo int, GuestNo int, DateFrom date,
     DateTo date, RoomNo int, foreign key(HotelNo) references Hotel(HotelNo),
    foreign key(RoomNo) references Room(RoomNo));
Query OK, 0 rows affected (0.011 sec)

MariaDB [dbms_lab]> describe Booking;
+----------+---------+------+-----+---------+-------+
| Field    | Type    | Null | Key | Default | Extra |
+----------+---------+------+-----+---------+-------+
| HotelNo  | int(11) | YES  | MUL | NULL    |       |
| GuestNo  | int(11) | YES  |     | NULL    |       |
| DateFrom | date    | YES  |     | NULL    |       |
| DateTo   | date    | YES  |     | NULL    |       |
| RoomNo   | int(11) | YES  | MUL | NULL    |       |
+----------+---------+------+-----+---------+-------+
5 rows in set (0.002 sec)

MariaDB [dbms_lab]> create table Guest(GuestNo int primary key, GuestName varchar
    (50), GuessAddress varchar(50));
Query OK, 0 rows affected (0.007 sec)

MariaDB [dbms_lab]> alter table Booking add constraint foreign key(GuestNo)
    references Guest(GuestNo);
Query OK, 0 rows affected (0.022 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [dbms_lab]> describe Booking;
+----------+---------+------+-----+---------+-------+
| Field    | Type    | Null | Key | Default | Extra |
+----------+---------+------+-----+---------+-------+
| HotelNo  | int(11) | YES  | MUL | NULL    |       |
| GuestNo  | int(11) | YES  | MUL | NULL    |       |
| DateFrom | date    | YES  |     | NULL    |       |
| DateTo   | date    | YES  |     | NULL    |       |
| RoomNo   | int(11) | YES  | MUL | NULL    |       |
+----------+---------+------+-----+---------+-------+
5 rows in set (0.004 sec)

MariaDB [dbms_lab]> describe Guest;
```

```
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| GuestNo      | int(11)     | NO   | PRI | NULL    |       |
| GuestName    | varchar(50) | YES  |     | NULL    |       |
| GuessAddress | varchar(50) | YES  |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
3 rows in set (0.001 sec)

MariaDB [dbms_lab]> describe Room;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| RoomNo  | int(11)     | NO   | PRI | NULL    |       |
| HotelNo | int(11)     | YES  | MUL | NULL    |       |
| Type    | varchar(50) | YES  |     | NULL    |       |
| Price   | int(11)     | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
4 rows in set (0.002 sec)

MariaDB [dbms_lab]> describe Hotel;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| HotelNo | int(11)     | NO   | PRI | NULL    |       |
| Name    | varchar(50) | YES  |     | NULL    |       |
| City    | varchar(50) | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
3 rows in set (0.002 sec)

ariaDB [dbms_lab]> create table emp(eno int primary key, ename varchar(50), zip
    int check(zip in (400110, 400111)), hdate date unique);
Query OK, 0 rows affected (0.009 sec)

MariaDB [dbms_lab]> describe emp;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| eno   | int(11)     | NO   | PRI | NULL    |       |
| ename | varchar(50) | YES  |     | NULL    |       |
| zip   | int(11)     | YES  |     | NULL    |       |
| hdate | date        | YES  | UNI | NULL    |       |
+-------+-------------+------+-----+---------+-------+
4 rows in set (0.002 sec)

MariaDB [dbms_lab]> create table parts(pno int primary key, pname varchar(50),
    qty_on_hand int not null, price int);
Query OK, 0 rows affected (0.007 sec)

MariaDB [dbms_lab]> describe parts;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| pno         | int(11)     | NO   | PRI | NULL    |       |
| pname       | varchar(50) | YES  |     | NULL    |       |
| qty_on_hand | int(11)     | NO   |     | NULL    |       |
| price       | int(11)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
4 rows in set (0.002 sec)
```

```
MariaDB [dbms_lab]> create table customer(cno primary key, cname varchar(50),
    street varchar(50), Zip int not null, phone int not null unique);
ERROR 4161 (HY000): Unknown data type: 'primary'
MariaDB [dbms_lab]> create table customer(cno int primary key, cname varchar(50),
    street varchar(50), Zip int not null, phone int not null unique);
Query OK, 0 rows affected (0.009 sec)

MariaDB [dbms_lab]> describe customer;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| cno    | int(11)     | NO   | PRI | NULL    |       |
| cname  | varchar(50) | YES  |     | NULL    |       |
| street | varchar(50) | YES  |     | NULL    |       |
| Zip    | int(11)     | NO   |     | NULL    |       |
| phone  | int(11)     | NO   | UNI | NULL    |       |
+--------+-------------+------+-----+---------+-------+
5 rows in set (0.002 sec)

MariaDB [dbms_lab]> create table Orders(ono int primary key, cno int, receivedDate
    date, shippedDate date, foreign key(cno) references customer(cno));
Query OK, 0 rows affected (0.010 sec)

MariaDB [dbms_lab]> describe Orders;
+--------------+---------+------+-----+---------+-------+
| Field        | Type    | Null | Key | Default | Extra |
+--------------+---------+------+-----+---------+-------+
| ono          | int(11) | NO   | PRI | NULL    |       |
| cno          | int(11) | YES  | MUL | NULL    |       |
| receivedDate | date    | YES  |     | NULL    |       |
| shippedDate  | date    | YES  |     | NULL    |       |
+--------------+---------+------+-----+---------+-------+
4 rows in set (0.002 sec)

MariaDB [dbms_lab]> create table odetails(ono int, pno int, qty int, foreign key(
    ono) references Orders(ono));
Query OK, 0 rows affected (0.009 sec)

MariaDB [dbms_lab]> describe odetails;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| ono   | int(11) | YES  | MUL | NULL    |       |
| pno   | int(11) | YES  |     | NULL    |       |
| qty   | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
3 rows in set (0.002 sec)

MariaDB [dbms_lab]> create table zipcode(zip int primary key, city varchar(50) not
    null check(city in ('Pune', 'Mumbai')));
Query OK, 0 rows affected (0.008 sec)

MariaDB [dbms_lab]> describe zipcode;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| zip   | int(11)     | NO   | PRI | NULL    |       |
| city  | varchar(50) | NO   |     | NULL    |       |
```

```
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.002 sec)
```

8 Conclusion

Thus, we have learned DDL and DCL commands thoroughly.

9 FAQ

1. *How to drop a column from a table?*

   ```
   ALTER TABLE table_name
   DROP COLUMN column_name;
   ```

2. *How to add a primary key in an already existing table?*

   ```
   ALTER TABLE table_name
   ADD PRIMARY KEY (column_name);
   ```

3. *How to create a new user in MySQL?*

   ```
   CREATE USER 'username'@'localhost' IDENTIFIED BY 'password';
   ```

MIT World Peace University

Database Management Systems
Second Year B. Tech, Semester 4

Learning SQL DML Commands
Data Manipulation Language

Assignment No. 3

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

March 9, 2023

Contents

1 Aim

Write suitable DML and select command to manipulate and retrieve requested data from tables.

2 Objectives

1. DML (Insert, Update, Delete) commands,
2. SQL Select- Logical, IN, Negation, NULL, Comparison Operators.
3. Where Clause, Between AND, Exists, ALL, LIKE

3 Problem Statement

4 Theory

4.1 SQL Data Manipulation Language (DML)

4.1.1 What is Data Manipulation Language?

Data Manipulation Language (DML) is a computer language used to access and manipulate data stored in a database. It is used to retrieve, insert, update, and delete data in a database.

4.1.2 DML Commands

The following are the Commands that are used in DML:

1. SELECT - Retrieves data from a database.
2. INSERT - Inserts data into a table.
3. UPDATE - Updates existing data within a table.
4. DELETE - Deletes existing data within a table.

4.2 DML Command Syntax and Examples

1. SELECT - Retrieves data from a database.

```
SELECT column1, column2, ...
FROM table_name;
```

2. INSERT - Inserts data into a table.

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

3. UPDATE - Updates existing data within a table.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

4. DELETE - Deletes existing data within a table.

   ```
   DELETE FROM table_name WHERE condition;
   ```

4.3 SELECT query

4.3.1 What is SELECT query?

The SELECT statement is used to select data from a database. The data returned is stored in a result table, called the result-set.

4.3.2 SELECT Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator value;
```

4.3.3 SELECT Operators

The following are the Operators that are used in SELECT:

1. **AND** - Returns rows where both conditions are true.
2. **OR** - Returns rows where either condition is true.
3. **NOT** - Returns rows where the condition(s) is not true.
4. **BETWEEN** - Returns rows where the value is within a range of two values.
5. **LIKE** - Returns rows where the value matches a pattern.
6. **IN** - Returns rows where the value matches any value in a list.

4.3.4 Examples of the SELECT Query

1. `SELECT * FROM CUSTOMERS;`
2. `SELECT * FROM CUSTOMERS WHERE CUST_ID = 1;`
3. `SELECT * FROM CUSTOMERS WHERE CUST_ID = 1 AND CUST_NAME = 'Krishnaraj';`
4. `SELECT * FROM CUSTOMERS WHERE CUST_ID = 1 OR CUST_NAME = 'Krishnaraj';`
5. `SELECT * FROM CUSTOMERS WHERE NOT CUST_ID = 1;`
6. `SELECT * FROM CUSTOMERS WHERE CUST_ID BETWEEN 1 AND 5;`
7. `SELECT * FROM CUSTOMERS WHERE CUST_NAME LIKE 'Krish%';`
8. `SELECT * FROM CUSTOMERS WHERE CUST_ID IN (1, 2, 3);`

4.4 SQL Operators

4.4.1 What are SQL Operators?

Operators are special symbols in SQL that allow you to perform specific operations on data.

4.4.2 SQL Operators

The following are the Operators that are used in SQL:

1. **Arithmetic Operators** - Used to perform mathematical operations on numbers.
2. **Comparison Operators** - Used to compare values.
3. **Logical Operators** - Used to combine two or more conditions.
4. **Misc Operators** - Used to perform other operations.

4.4.3 Arithmetic Operators

The following are the Arithmetic Operators that are used in SQL:

1. **+** - Addition
2. **-** - Subtraction
3. * - Multiplication
4. / - Division
5. **MOD** - Modulus

4.4.4 Comparison Operators

The following are the Comparison Operators that are used in SQL:

1. **=** - Equal
2. **<>** - Not equal. Note: In some versions of SQL this operator may be written as !=
3. **>** - Greater than
4. **<** - Less than
5. **>=** - Greater than or equal
6. **<=** - Less than or equal
7. **BETWEEN** - Between an inclusive range
8. **LIKE** - Search for a pattern
9. **IN** - To specify multiple possible values for a column

4.4.5 Logical Operators

The following are the Logical Operators that are used in SQL:

1. **AND** - Logical AND
2. **OR** - Logical OR
3. **NOT** - Logical NOT

5 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Draw.io for Drawing the ER diagram.

6 Input

Given Database from the Problem Statement for the Assignment for our batch. (A1 PA 20)

7 Output

```
MariaDB [dbms_lab]> create database Company;
Query OK, 1 row affected (0.001 sec)

MariaDB [dbms_lab]> show databases;
+--------------------+
| Database           |
+--------------------+
| Company            |
| class              |
| class_stuff        |
| dbms_lab           |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test               |
| test_libreoffice   |
+--------------------+
10 rows in set (0.001 sec)

MariaDB [dbms_lab]> use Company;
Database changed
MariaDB [Company]> create table emp(empno int primary key, empname varchar(50) not
    null, job varchar(10), mgr int not null, hiredate date, sal int not null, comm
    int, deptno int not null);
Query OK, 0 rows affected (0.008 sec)

MariaDB [Company]> describe emp;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| empno    | int(11)     | NO   | PRI | NULL    |       |
| empname  | varchar(50) | NO   |     | NULL    |       |
```

```
| job      | varchar(10) | YES  |     | NULL    |       |
| mgr      | int(11)     | NO   |     | NULL    |       |
| hiredate | date        | YES  |     | NULL    |       |
| sal      | int(11)     | NO   |     | NULL    |       |
| comm     | int(11)     | YES  |     | NULL    |       |
| deptno   | int(11)     | NO   |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
8 rows in set (0.002 sec)

MariaDB [Company]> create table dept(deptno int primary key, dname varchar(50),
    loc varchar(50) not null);
Query OK, 0 rows affected (0.008 sec)

MariaDB [Company]> describe dept;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| deptno | int(11)     | NO   | PRI | NULL    |       |
| dname  | varchar(50) | YES  |     | NULL    |       |
| loc    | varchar(50) | NO   |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.002 sec)

MariaDB [Company]> insert into emp values (7369,  "Smith",  "Clerk",  7902,  "
    1980-12-17",  800,  300,  20);
Query OK, 1 row affected (0.001 sec)

MariaDB [Company]> select * from emp;
+-------+---------+-------+------+------------+-----+------+--------+
| empno | empname | job   | mgr  | hiredate   | sal | comm | deptno |
+-------+---------+-------+------+------------+-----+------+--------+
|  7369 | Smith   | Clerk | 7902 | 1980-12-17 | 800 |  300 |     20 |
+-------+---------+-------+------+------------+-----+------+--------+
1 row in set (0.001 sec)

MariaDB [Company]> insert into emp values (7499, "Allen", "Salesman", 7698, "
    1981-02-20", 1600, 300, 30);
Query OK, 1 row affected (0.001 sec)

MariaDB [Company]> select * from emp;
+-------+---------+----------+------+------------+------+------+--------+
| empno | empname | job      | mgr  | hiredate   | sal  | comm | deptno |
+-------+---------+----------+------+------------+------+------+--------+
|  7369 | Smith   | Clerk    | 7902 | 1980-12-17 |  800 |  300 |     20 |
|  7499 | Allen   | Salesman | 7698 | 1981-02-20 | 1600 |  300 |     30 |
+-------+---------+----------+------+------------+------+------+--------+
2 rows in set (0.000 sec)

MariaDB [Company]> insert into dept values(10, "Accounting", "New York");
Query OK, 1 row affected (0.001 sec)

MariaDB [Company]> insert into dept values(20, "Research", "Dallas");
Query OK, 1 row affected (0.001 sec)

MariaDB [Company]> insert into dept values
    -> (30, "Sales", "Chicago");
Query OK, 1 row affected (0.001 sec)

MariaDB [Company]> insert into dept values(40, "Operations", "Boston");
```

```
Query OK, 1 row affected (0.001 sec)

MariaDB [Company]> select * from dept;
+--------+------------+----------+
| deptno | dname      | loc      |
+--------+------------+----------+
|     10 | Accounting | New York |
|     20 | Research   | Dallas   |
|     30 | Sales      | Chicago  |
|     40 | Operations | Boston   |
+--------+------------+----------+
4 rows in set (0.001 sec)

MariaDB [Company]> select * from emp;
+-------+---------+----------+------+------------+------+------+--------+
| empno | empname | job      | mgr  | hiredate   | sal  | comm | deptno |
+-------+---------+----------+------+------------+------+------+--------+
|  7369 | Smith   | Clerk    | 7902 | 1980-12-17 |  800 |  300 |     20 |
|  7499 | Allen   | Salesman | 7698 | 1981-02-20 | 1600 |  300 |     30 |
+-------+---------+----------+------+------------+------+------+--------+
2 rows in set (0.001 sec)

MariaDB [Company]> insert into emp values(9360, "Isaiah", "Accounting", 7940, "
    2101-9-3", 4000, 1390, 10);
Query OK, 1 row affected

MariaDB [Company]> insert into emp values(9085, "Katie", "Research", 5919, "
    1997-1-26", 8241, 1166, 20);
Query OK, 1 row affected

MariaDB [Company]> insert into emp values(5883, "Jeffery", "Research", 5817, "
    2057-8-3", 2033, 549, 20);
Query OK, 1 row affected

MariaDB [Company]> insert into emp values(5595, "Isabella", "Sales", 8245, "
    2075-9-10", 2534, 1545, 30);
Query OK, 1 row affected

MariaDB [Company]> insert into emp values(9180, "Jesse", "Accounting", 2678, "
    2101-8-22", 3238, 1796, 10);
Query OK, 1 row affected

MariaDB [Company]> insert into emp values(9487, "Amelia", "Research",7940 , "
    2123-1-17", 5368, 1998, 20);
Query OK, 1 row affected

MariaDB [Company]> insert into emp values(8467, "Mollie", "Accounting", 5919, "
    2015-2-9", 3999, 526, 10);
Query OK, 1 row affected

MariaDB [Company]> insert into emp values(9384, "Matilda", "Operations", 5817, "
    2025-5-23", 2494, 1170, 50);
Query OK, 1 row affected

MariaDB [Company]> insert into emp values(6880, "Cameron", "Sales", 8245, "
    2059-5-9", 6311, 1406, 30);
Query OK, 1 row affected

```

```
MariaDB [Company]> insert into emp values(7235, "Stephen", "Operations", 2678, "2083-0-31", 6556, 1698, 50);
Query OK, 1 row affected

MariaDB [Company]> insert into emp values(7553, "Angel", "Sales", 2678, "2099-6-3", 9352, 983, 30);
Query OK, 1 row affected
```

8 Executed Queries

8.1 Set 1

```

## Queries Set 1

1. List the number of employees and average salary for employees in department 20.

MariaDB [Company]> select avg(sal),count(*) from emp where deptno=20;
+-----------+----------+
| avg(sal)  | count(*) |
+-----------+----------+
| 5214.0000 |        3 |
+-----------+----------+
1 row in set (0.007 sec)

2. List name, salary and PF amount of all employees. (PF is calculated as 10% of basic salary)

MariaDB [Company]> select empname, sal, sal * 0.10 as PF from emp;
+----------+------+--------+
| empname  | sal  | PF     |
+----------+------+--------+
| Isabella | 2534 | 253.40 |
| Jeffery  | 2033 | 203.30 |
| Cameron  | 6311 | 631.10 |
| Stephen  | 6556 | 655.60 |
| Angel    | 9352 | 935.20 |
| Ramesh   |  500 |  50.00 |
| Krish    | 2000 | 200.00 |
| Mollie   | 3999 | 399.90 |
| Katie    | 8241 | 824.10 |
| Jesse    | 3562 | 356.20 |
| Isaiah   | 4000 | 400.00 |
| Matilda  | 2494 | 249.40 |
| Amelia   | 5368 | 536.80 |
+----------+------+--------+
13 rows in set (0.001 sec)

3. List the employee details in the ascending order of their basic salary.

MariaDB [Company]> select * from emp order by sal;
+-------+----------+------------+------+------------+------+------+--------+
| empno | empname  | job        | mgr  | hiredate   | sal  | comm | deptno |
+-------+----------+------------+------+------------+------+------+--------+
|  8343 | Ramesh   | PT         | 7698 | 2023-12-12 |  500 |  300 |     60 |
|  8344 | Krish    | PT         | 7698 | 2023-12-12 | 2000 |  300 |     60 |
|  5883 | Jeffery  | Research   | 5817 | 2057-08-03 | 2033 |  549 |     20 |
|  9384 | Matilda  | Operations | 5817 | 2025-05-23 | 2494 | 1170 |     40 |
```

```
|  5595 | Isabella | Sales      | 8245 | 2075-09-10 | 2534 | 1545 |     30 |
|  9180 | Jesse    | Accounting | 2678 | 2101-08-22 | 3562 | 1796 |     10 |
|  8467 | Mollie   | Accounting | 5919 | 2015-02-09 | 3999 |  526 |     10 |
|  9360 | Isaiah   | Accounting | 7940 | 2101-09-03 | 4000 | 1390 |     10 |
|  9487 | Amelia   | Research   | 7940 | 2123-01-17 | 5368 | 1998 |     20 |
|  6880 | Cameron  | Sales      | 8245 | 2059-05-09 | 6311 | 1406 |     30 |
|  7235 | Stephen  | Operations | 2678 | 2083-00-31 | 6556 | 1698 |     40 |
|  9085 | Katie    | Research   | 5919 | 1997-01-26 | 8241 | 1166 |     20 |
|  7553 | Angel    | Sales      | 2678 | 2099-06-03 | 9352 |  983 |     30 |
+-------+----------+------------+------+------------+------+------+--------+
13 rows in set (0.001 sec)


4. List the employee name and hire date in the descending order of the hire date.

MariaDB [Company]> select empname, hiredate from emp order by hiredate desc;
+----------+------------+
| empname  | hiredate   |
+----------+------------+
| Amelia   | 2123-01-17 |
| Isaiah   | 2101-09-03 |
| Jesse    | 2101-08-22 |
| Angel    | 2099-06-03 |
| Stephen  | 2083-00-31 |
| Isabella | 2075-09-10 |
| Cameron  | 2059-05-09 |
| Jeffery  | 2057-08-03 |
| Matilda  | 2025-05-23 |
| Ramesh   | 2023-12-12 |
| Krish    | 2023-12-12 |
| Mollie   | 2015-02-09 |
| Katie    | 1997-01-26 |
+----------+------------+
13 rows in set (0.002 sec)

5. List employee name, salary, PF, HRA, DA and gross; order the results in the
    ascending order of
gross. HRA is 50% of the salary and DA is 30% of the salary.

MariaDB [Company]> select empname, sal, sal*.10 as PF, sal*.50 as HRA, sal*.30 as
    DA, sal + sal*.90 as Gross from emp order by Gross;
+----------+------+--------+---------+---------+----------+
| empname  | sal  | PF     | HRA     | DA      | Gross    |
+----------+------+--------+---------+---------+----------+
| Ramesh   |  500 |  50.00 |  250.00 |  150.00 |   950.00 |
| Krish    | 2000 | 200.00 | 1000.00 |  600.00 |  3800.00 |
| Jeffery  | 2033 | 203.30 | 1016.50 |  609.90 |  3862.70 |
| Matilda  | 2494 | 249.40 | 1247.00 |  748.20 |  4738.60 |
| Isabella | 2534 | 253.40 | 1267.00 |  760.20 |  4814.60 |
| Jesse    | 3562 | 356.20 | 1781.00 | 1068.60 |  6767.80 |
| Mollie   | 3999 | 399.90 | 1999.50 | 1199.70 |  7598.10 |
| Isaiah   | 4000 | 400.00 | 2000.00 | 1200.00 |  7600.00 |
| Amelia   | 5368 | 536.80 | 2684.00 | 1610.40 | 10199.20 |
| Cameron  | 6311 | 631.10 | 3155.50 | 1893.30 | 11990.90 |
| Stephen  | 6556 | 655.60 | 3278.00 | 1966.80 | 12456.40 |
| Katie    | 8241 | 824.10 | 4120.50 | 2472.30 | 15657.90 |
| Angel    | 9352 | 935.20 | 4676.00 | 2805.60 | 17768.80 |
+----------+------+--------+---------+---------+----------+
13 rows in set (0.001 sec)
```

```
6. List the department numbers and number of employees in each department.

MariaDB [Company]> select deptno,count(*) from emp group by deptno;
+--------+----------+
| deptno | count(*) |
+--------+----------+
|     10 |        3 |
|     20 |        3 |
|     30 |        3 |
|     40 |        2 |
|     60 |        2 |
+--------+----------+
5 rows in set (0.002 sec)

7. Increment the Salary of salesman by 10% of basic salary.

MariaDB [Company]> update emp set sal=sal+(sal*.10) where job='sales';
Query OK, 3 rows affected (0.006 sec)
Rows matched: 3  Changed: 3  Warnings: 0

8. List the total salary, maximum and minimum salary and average salary of the
    employees, for
department 20.

select sum(sal),max(sal),min(sal),avg(sal) from emp where deptno=20;
+----------+----------+----------+-----------+
| sum(sal) | max(sal) | min(sal) | avg(sal)  |
+----------+----------+----------+-----------+
|    15642 |     8241 |     2033 | 5214.0000 |
+----------+----------+----------+-----------+
1 row in set (0.001 sec)

9. List the employees whose names contains 3 rd letter as 'I'.

MariaDB [Company]> select empname from emp where empname like '__i%';
+---------+
| empname |
+---------+
| Krish   |
+---------+
1 row in set (0.001 sec)

10. List the maximum salary paid to a salesman.

MariaDB [Company]> select *, max(sal) from emp where job='sales';
+-------+----------+-------+------+------------+------+------+--------+----------+
| empno | empname  | job   | mgr  | hiredate   | sal  | comm | deptno | max(sal) |
+-------+----------+-------+------+------------+------+------+--------+----------+
|  5595 | Isabella | Sales | 8245 | 2075-09-10 | 2787 | 1545 |     30 |    10287 |
+-------+----------+-------+------+------------+------+------+--------+----------+
1 row in set (0.001 sec)


11. Increase the salary of salesman by 10% of their current salary.

MariaDB [Company]> update emp set sal=sal+(sal*.10) where job='sales';
Query OK, 3 rows affected (0.001 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

8.2 Set 2

```
1. List the employee names and his annual salary dept wise.

MariaDB [Company]> select deptno, empname, sal*12 as Annual_Sal from emp order by
    deptno;
+--------+----------+------------+
| deptno | empname  | Annual_Sal |
+--------+----------+------------+
|     10 | Mollie   |      47988 |
|     10 | Isaiah   |      48000 |
|     10 | Jesse    |      38856 |
|     20 | Katie    |      98892 |
|     20 | Amelia   |      64416 |
|     20 | Jeffery  |      24396 |
|     30 | Angel    |     112224 |
|     30 | Cameron  |      75732 |
|     30 | Isabella |      30408 |
|     50 | Stephen  |      78672 |
|     50 | Matilda  |      29928 |
+--------+----------+------------+
11 rows in set (0.000 sec)

2. Find out least 5 earners of the company.

MariaDB [Company]> select empname from emp order by sal asc limit 5;
+----------+
| empname  |
+----------+
| Jeffery  |
| Matilda  |
| Isabella |
| Jesse    |
| Mollie   |
+----------+
5 rows in set (0.001 sec)
3. List the records from emp whose deptno is not in dept

MariaDB [Company]> select * from emp where deptno not in (select deptno from dept)
    ;
+-------+---------+------+------+------------+------+------+--------+
| empno | empname | job  | mgr  | hiredate   | sal  | comm | deptno |
+-------+---------+------+------+------------+------+------+--------+
|  8344 | Krish   | PT   | 7698 | 2023-12-12 | 2000 |  300 |     60 |
+-------+---------+------+------+------------+------+------+--------+
1 row in set (0.002 sec)

4. List those employees whose sal is odd value.

MariaDB [Company]> select * from emp where sal % 2 != 0;
+-------+---------+------------+------+------------+------+------+--------+
| empno | empname | job        | mgr  | hiredate   | sal  | comm | deptno |
+-------+---------+------------+------+------------+------+------+--------+
|  5883 | Jeffery | Research   | 5817 | 2057-08-03 | 2033 |  549 |     20 |
|  6880 | Cameron | Sales      | 8245 | 2059-05-09 | 6311 | 1406 |     30 |
```

```
|  8467 | Mollie  | Accounting | 5919 | 2015-02-09 | 3999 |  526 |     10 |
|  9085 | Katie   | Research   | 5919 | 1997-01-26 | 8241 | 1166 |     20 |
+-------+---------+------------+------+------------+------+------+--------+
4 rows in set (0.001 sec)

5. List the employees whose sal contain 3 digits.

MariaDB [Company]> select * from emp where sal/1000 < 1;
+-------+---------+------+------+------------+-----+------+--------+
| empno | empname | job  | mgr  | hiredate   | sal | comm | deptno |
+-------+---------+------+------+------------+-----+------+--------+
|  8343 | Ramesh  | PT   | 7698 | 2023-12-12 | 500 |  300 |     60 |
+-------+---------+------+------+------------+-----+------+--------+
1 row in set (0.001 sec)

6. List the employees who joined in the month of 'DEC'

MariaDB [Company]> select * from emp where hiredate like "%%%%-12-%%";
+-------+---------+------+------+------------+------+------+--------+
| empno | empname | job  | mgr  | hiredate   | sal  | comm | deptno |
+-------+---------+------+------+------------+------+------+--------+
|  8343 | Ramesh  | PT   | 7698 | 2023-12-12 |  500 |  300 |     60 |
|  8344 | Krish   | PT   | 7698 | 2023-12-12 | 2000 |  300 |     60 |
+-------+---------+------+------+------------+------+------+--------+
2 rows in set (0.001 sec)

7. List the employees whose names contains 'A'

MariaDB [Company]> select * from emp where empname like "A%";
+-------+---------+----------+------+------------+------+------+--------+
| empno | empname | job      | mgr  | hiredate   | sal  | comm | deptno |
+-------+---------+----------+------+------------+------+------+--------+
|  7553 | Angel   | Sales    | 2678 | 2099-06-03 | 9352 |  983 |     30 |
|  9487 | Amelia  | Research | 7940 | 2123-01-17 | 5368 | 1998 |     20 |
+-------+---------+----------+------+------------+------+------+--------+
2 rows in set (0.001 sec)

8. List the maximum, minimum and average salary in the company.

MariaDB [Company]> select max(sal), min(sal), avg(sal) from emp;
+----------+----------+-----------+
| max(sal) | min(sal) | avg(sal)  |
+----------+----------+-----------+
|     9352 |      500 | 4355.8462 |
+----------+----------+-----------+
1 row in set (0.001 sec)

9. Write a query to return the day of the week for any date(or HIRE_DATE) entered
   in format
'DD-MM-YY'


MariaDB [Company]> select dayname(hiredate) from emp;
+-------------------+
| dayname(hiredate) |
+-------------------+
| Tuesday           |
| Friday            |
| Friday            |
```

```
| NULL               |
| Wednesday          |
| Tuesday            |
| Tuesday            |
| Monday             |
| Sunday             |
| Monday             |
| Saturday           |
| Friday             |
| Sunday             |
+--------------------+
13 rows in set (0.002 sec)

10. Count the no of characters in employee name without considering spaces for
    each name.

MariaDB [Company]> select empname, length(replace(empname, ' ', '')) + 1 as length
    from emp;
+----------+--------+
| empname  | length |
+----------+--------+
| Isabella |      9 |
| Jeffery  |      8 |
| Cameron  |      8 |
| Stephen  |      8 |
| Angel    |      6 |
| Ramesh   |      7 |
| Krish    |      6 |
| Mollie   |      7 |
| Katie    |      6 |
| Jesse    |      6 |
| Isaiah   |      7 |
| Matilda  |      8 |
| Amelia   |      7 |
+----------+--------+
13 rows in set (0.001 sec)

11. List the employees who are drawing less than 1000. sort the output by salary.

MariaDB [Company]> select * from emp where sal < 1000 order by sal;
+-------+---------+------+------+------------+-----+------+--------+
| empno | empname | job  | mgr  | hiredate   | sal | comm | deptno |
+-------+---------+------+------+------------+-----+------+--------+
|  8343 | Ramesh  | PT   | 7698 | 2023-12-12 | 500 |  300 |     60 |
+-------+---------+------+------+------------+-----+------+--------+
1 row in set (0.002 sec)

```

8.3 Set 3

```

1. Write a query in SQL to display the unique designations for the employees.

MariaDB [Company]> select distinct job from emp;
+------------+
| job        |
+------------+
| Sales      |
```

```
| Research   |
| Operations |
| PT         |
| Accounting |
+------------+
5 rows in set (0.001 sec)

2. Delete Employees who joined in Year 1980.

MariaDB [Company]> delete from emp where year(hiredate) = 1980;
Query OK, 0 rows affected (0.001 sec)

3. Increase the salary of Managers by 20% of their current salary.

MariaDB [Company]> update emp set sal = sal + sal*0.2 where job = 'Manager';
Query OK, 0 rows affected (0.001 sec)
Rows matched: 0  Changed: 0  Warnings: 0

4. List employees not belonging to department 30, 40, or 10.

MariaDB [Company]> select * from emp where deptno not in (30, 40, 10);
+-------+---------+----------+------+------------+------+------+--------+
| empno | empname | job      | mgr  | hiredate   | sal  | comm | deptno |
+-------+---------+----------+------+------------+------+------+--------+
|  5883 | Jeffery | Research | 5817 | 2057-08-03 | 2236 |  549 |     20 |
|  8343 | Ramesh  | PT       | 7698 | 2023-12-12 |  500 |    0 |     60 |
|  8344 | Krish   | PT       | 7698 | 2023-12-12 | 2000 |  300 |     60 |
|  9085 | Katie   | Research | 5919 | 1997-01-26 | 9065 | 1166 |     20 |
|  9487 | Amelia  | Research | 7940 | 2123-01-17 | 5905 | 1998 |     20 |
+-------+---------+----------+------+------------+------+------+--------+
5 rows in set (0.001 sec)

5. List the different designations in the company.

MariaDB [Company]> select distinct job from emp;
+------------+
| job        |
+------------+
| Sales      |
| Research   |
| Operations |
| PT         |
| Accounting |
+------------+
5 rows in set (0.001 sec)

6. List the names of employees who are not eligible for commission.

MariaDB [Company]> select * from emp where sal < 1000;
+-------+---------+------+------+------------+-----+------+--------+
| empno | empname | job  | mgr  | hiredate   | sal | comm | deptno |
+-------+---------+------+------+------------+-----+------+--------+
|  8343 | Ramesh  | PT   | 7698 | 2023-12-12 | 500 |    0 |     60 |
+-------+---------+------+------+------------+-----+------+--------+
1 row in set (0.001 sec)

7. List employees whose names either start or end with "S".

MariaDB [Company]>  select * from emp where empname like 'S%' or empname like '%S'
```

```
    ;
+-------+---------+------------+------+------------+------+------+--------+
| empno | empname | job        | mgr  | hiredate   | sal  | comm | deptno |
+-------+---------+------------+------+------------+------+------+--------+
|  7235 | Stephen | Operations | 2678 | 2083-00-31 | 6556 | 1698 |     40 |
+-------+---------+------------+------+------------+------+------+--------+
1 row in set (0.001 sec)

8. List employees whose names have letter "A" as second letter in their names.

MariaDB [Company]> select * from emp where empname like '_A%';
+-------+---------+------------+------+------------+------+------+--------+
| empno | empname | job        | mgr  | hiredate   | sal  | comm | deptno |
+-------+---------+------------+------+------------+------+------+--------+
|  6880 | Cameron | Sales      | 8245 | 2059-05-09 | 7636 | 1406 |     30 |
|  8343 | Ramesh  | PT         | 7698 | 2023-12-12 |  500 |    0 |     60 |
|  9085 | Katie   | Research   | 5919 | 1997-01-26 | 9065 | 1166 |     20 |
|  9384 | Matilda | Operations | 5817 | 2025-05-23 | 2494 | 1170 |     40 |
+-------+---------+------------+------+------------+------+------+--------+
4 rows in set (0.001 sec)

9. List the number of employees working with the company.

MariaDB [Company]> select count(*) from emp;
+----------+
| count(*) |
+----------+
|       13 |
+----------+
1 row in set (0.001 sec)

10. List the emps with hiredate in format June 4,1988.

MariaDB [Company]> select * from emp where hiredate = '1988-06-04';
Empty set (0.001 sec)

11. List the salesmen who get the commission within a range of 200 and 5000.

MariaDB [Company]> select * from emp where job = 'Sales' and comm between 200 and
    5000;
+-------+----------+-------+------+------------+-------+------+--------+
| empno | empname  | job   | mgr  | hiredate   | sal   | comm | deptno |
+-------+----------+-------+------+------------+-------+------+--------+
|  5595 | Isabella | Sales | 8245 | 2075-09-10 |  3066 | 1545 |     30 |
|  6880 | Cameron  | Sales | 8245 | 2059-05-09 |  7636 | 1406 |     30 |
|  7553 | Angel    | Sales | 2678 | 2099-06-03 | 11316 |  983 |     30 |
+-------+----------+-------+------+------------+-------+------+--------+
3 rows in set (0.001 sec)

```

8.4 Set 4

```

1. List names of employees who are more than 2 years old in the company.

MariaDB [Company]> select empname from emp where datediff(curdate(), hiredate)/365
    > 2;
+---------+
```

```
| empname |
+---------+
| Mollie  |
| Katie   |
+---------+
2 rows in set (0.001 sec)

2. List the employee details in the ascending order of their basic salary.

MariaDB [Company]> select * from emp order by sal;
+-------+----------+------------+------+------------+------+------+--------+
| empno | empname  | job        | mgr  | hiredate   | sal  | comm | deptno |
+-------+----------+------------+------+------------+------+------+--------+
|  8343 | Ramesh   | PT         | 7698 | 2023-12-12 |  500 |  300 |     60 |
|  8344 | Krish    | PT         | 7698 | 2023-12-12 | 2000 |  300 |     60 |
|  5883 | Jeffery  | Research   | 5817 | 2057-08-03 | 2033 |  549 |     20 |
|  9384 | Matilda  | Operations | 5817 | 2025-05-23 | 2494 | 1170 |     40 |
|  5595 | Isabella | Sales      | 8245 | 2075-09-10 | 2534 | 1545 |     30 |
|  9180 | Jesse    | Accounting | 2678 | 2101-08-22 | 3238 | 1796 |     10 |
|  8467 | Mollie   | Accounting | 5919 | 2015-02-09 | 3999 |  526 |     10 |
|  9360 | Isaiah   | Accounting | 7940 | 2101-09-03 | 4000 | 1390 |     10 |
|  9487 | Amelia   | Research   | 7940 | 2123-01-17 | 5368 | 1998 |     20 |
|  6880 | Cameron  | Sales      | 8245 | 2059-05-09 | 6311 | 1406 |     30 |
|  7235 | Stephen  | Operations | 2678 | 2083-00-31 | 6556 | 1698 |     40 |
|  9085 | Katie    | Research   | 5919 | 1997-01-26 | 8241 | 1166 |     20 |
|  7553 | Angel    | Sales      | 2678 | 2099-06-03 | 9352 |  983 |     30 |
+-------+----------+------------+------+------------+------+------+--------+
13 rows in set (0.002 sec)

3. Display the employees who have more salary as that of Smith

MariaDB [Company]> select * from emp where sal > (select sal from emp where
    empname = 'Mollie');
+-------+---------+------------+------+------------+------+------+--------+
| empno | empname | job        | mgr  | hiredate   | sal  | comm | deptno |
+-------+---------+------------+------+------------+------+------+--------+
|  6880 | Cameron | Sales      | 8245 | 2059-05-09 | 6311 | 1406 |     30 |
|  7235 | Stephen | Operations | 2678 | 2083-00-31 | 6556 | 1698 |     40 |
|  7553 | Angel   | Sales      | 2678 | 2099-06-03 | 9352 |  983 |     30 |
|  9085 | Katie   | Research   | 5919 | 1997-01-26 | 8241 | 1166 |     20 |
|  9360 | Isaiah  | Accounting | 7940 | 2101-09-03 | 4000 | 1390 |     10 |
|  9487 | Amelia  | Research   | 7940 | 2123-01-17 | 5368 | 1998 |     20 |
+-------+---------+------------+------+------------+------+------+--------+
6 rows in set (0.001 sec)

4. Increment the salary of Emp no. 9180 by 10% of his current salary.

MariaDB [Company]> select * from emp where empno = 9180;
+-------+---------+------------+------+------------+------+------+--------+
| empno | empname | job        | mgr  | hiredate   | sal  | comm | deptno |
+-------+---------+------------+------+------------+------+------+--------+
|  9180 | Jesse   | Accounting | 2678 | 2101-08-22 | 3562 | 1796 |     10 |
+-------+---------+------------+------+------------+------+------+--------+
1 row in set (0.001 sec)

5. List the employees whose salary is between 10000 and 25000.

MariaDB [Company]> select * from emp where sal between 10000 and 25000;
+-------+---------+-------+------+------------+-------+------+--------+
```

```
| empno | empname | job    | mgr  | hiredate   | sal   | comm | deptno |
+-------+---------+--------+------+------------+-------+------+--------+
|  7553 | Angel   | Sales  | 2678 | 2099-06-03 | 11316 |  983 |     30 |
+-------+---------+--------+------+------------+-------+------+--------+
1 row in set (0.000 sec)

6. List the names of employees who are not eligible for commission.

MariaDB [Company]> select * from emp where sal < 1000;
+-------+---------+------+------+------------+-----+------+--------+
| empno | empname | job  | mgr  | hiredate   | sal | comm | deptno |
+-------+---------+------+------+------------+-----+------+--------+
|  8343 | Ramesh  | PT   | 7698 | 2023-12-12 | 500 |    0 |     60 |
+-------+---------+------+------+------------+-----+------+--------+
1 row in set (0.001 sec)

7. Increment the Salary of Research by 10% of basic salary.

MariaDB [Company]> update emp set sal = sal + sal*0.1 where job = "Research";
Query OK, 3 rows affected (0.001 sec)
Rows matched: 3  Changed: 3  Warnings: 0

8. List the total salary, maximum and minimum salary and average salary of the
    employees
jobwise.

MariaDB [Company]> select job, sum(sal) as total, max(sal) as max, min(sal) as min
    , avg(sal) as avg from emp group by job;
+------------+-------+-------+------+-----------+
| job        | total | max   | min  | avg       |
+------------+-------+-------+------+-----------+
| Accounting | 11561 |  4000 | 3562 | 3853.6667 |
| Operations |  9050 |  6556 | 2494 | 4525.0000 |
| PT         |  2500 |  2000 |  500 | 1250.0000 |
| Research   | 17206 |  9065 | 2236 | 5735.3333 |
| Sales      | 22018 | 11316 | 3066 | 7339.3333 |
+------------+-------+-------+------+-----------+
5 rows in set (0.002 sec)

9. Delete the Employee whose name starts with P.

MariaDB [Company]> delete from emp where empname like 'P%';
Query OK, 0 rows affected (0.001 sec)

10. List the employees whose designation is "Research" and commission is > 500.

MariaDB [Company]> select * from emp where job = 'Research' and comm > 500;
+-------+---------+----------+------+------------+------+------+--------+
| empno | empname | job      | mgr  | hiredate   | sal  | comm | deptno |
+-------+---------+----------+------+------------+------+------+--------+
|  5883 | Jeffery | Research | 5817 | 2057-08-03 | 2236 |  549 |     20 |
|  9085 | Katie   | Research | 5919 | 1997-01-26 | 9065 | 1166 |     20 |
|  9487 | Amelia  | Research | 7940 | 2123-01-17 | 5905 | 1998 |     20 |
+-------+---------+----------+------+------------+------+------+--------+
3 rows in set (0.001 sec)

11. List employees belonging to department 20, 30, 40.

MariaDB [Company]> select * from emp where deptno in (20, 30, 40);
```

```
+-------+----------+------------+------+------------+-------+------+--------+
| empno | empname  | job        | mgr  | hiredate   | sal   | comm | deptno |
+-------+----------+------------+------+------------+-------+------+--------+
|  5595 | Isabella | Sales      | 8245 | 2075-09-10 |  3066 | 1545 |     30 |
|  5883 | Jeffery  | Research   | 5817 | 2057-08-03 |  2236 |  549 |     20 |
|  6880 | Cameron  | Sales      | 8245 | 2059-05-09 |  7636 | 1406 |     30 |
|  7235 | Stephen  | Operations | 2678 | 2083-00-31 |  6556 | 1698 |     40 |
|  7553 | Angel    | Sales      | 2678 | 2099-06-03 | 11316 |  983 |     30 |
|  9085 | Katie    | Research   | 5919 | 1997-01-26 |  9065 | 1166 |     20 |
|  9384 | Matilda  | Operations | 5817 | 2025-05-23 |  2494 | 1170 |     40 |
|  9487 | Amelia   | Research   | 7940 | 2123-01-17 |  5905 | 1998 |     20 |
+-------+----------+------------+------+------------+-------+------+--------+
8 rows in set (0.002 sec)

```

8.5 Set 5

```

1. List the employee names and his annual salary Job wise.

MariaDB [Company]> select job, empname, sal*12 as annual from emp;
+------------+----------+--------+
| job        | empname  | annual |
+------------+----------+--------+
| Sales      | Isabella |  36792 |
| Research   | Jeffery  |  26832 |
| Sales      | Cameron  |  91632 |
| Operations | Stephen  |  78672 |
| Sales      | Angel    | 135792 |
| PT         | Ramesh   |   6000 |
| PT         | Krish    |  24000 |
| Accounting | Mollie   |  47988 |
| Research   | Katie    | 108780 |
| Accounting | Jesse    |  42744 |
| Accounting | Isaiah   |  48000 |
| Operations | Matilda  |  29928 |
| Research   | Amelia   |  70860 |
+------------+----------+--------+
13 rows in set (0.001 sec)


2. Delete the Employee whose name starts with A & R

MariaDB [Company]> delete from emp where empname like 'A%' or empname like 'R%';
Query OK, 3 rows affected (0.002 sec)

3. Increment the salary of Emp no. 7000 by 30% of his current salary.

MariaDB [Company]> update emp set sal = sal + sal*0.3 where empno = 7000;
Query OK, 0 rows affected (0.001 sec)
Rows matched: 0  Changed: 0  Warnings: 0

4. List the total salary, maximum and minimum salary and average salary of the
    employees hire date wise.

MariaDB [Company]> select hiredate, sum(sal) as total, max(sal) as max, min(sal)
    as min, avg(sal) as avg from emp group by hiredate;
+------------+-------+------+------+-----------+
```

```
| hiredate   | total | max  | min  | avg       |
+------------+-------+------+------+-----------+
| 1997-01-26 |  9065 | 9065 | 9065 | 9065.0000 |
| 2015-02-09 |  3999 | 3999 | 3999 | 3999.0000 |
| 2023-12-12 |  2000 | 2000 | 2000 | 2000.0000 |
| 2025-05-23 |  2494 | 2494 | 2494 | 2494.0000 |
| 2057-08-03 |  2236 | 2236 | 2236 | 2236.0000 |
| 2059-05-09 |  7636 | 7636 | 7636 | 7636.0000 |
| 2075-09-10 |  3066 | 3066 | 3066 | 3066.0000 |
| 2083-00-31 |  6556 | 6556 | 6556 | 6556.0000 |
| 2101-08-22 |  3562 | 3562 | 3562 | 3562.0000 |
| 2101-09-03 |  4000 | 4000 | 4000 | 4000.0000 |
+------------+-------+------+------+-----------+
10 rows in set (0.001 sec)

5. List the employees whose names contains last letter as 'T'.

MariaDB [Company]> select * from emp where empname like '%T';
Empty set (0.001 sec)

6. Display the employees who have less salary as that of Ankush

MariaDB [Company]> select * from emp where sal < (select sal from emp where
    empname = 'Ankush');
Empty set (0.001 sec)

7. Display the employees who have salary between 10000

MariaDB [Company]> select * from emp where sal between 10000 and 20000;
Empty set (0.001 sec)

8. List employees belonging to department 30, 40, or 10.

MariaDB [Company]> select * from emp where deptno in (30, 40, 10);
+-------+----------+------------+------+------------+------+------+--------+
| empno | empname  | job        | mgr  | hiredate   | sal  | comm | deptno |
+-------+----------+------------+------+------------+------+------+--------+
|  5595 | Isabella | Sales      | 8245 | 2075-09-10 | 3066 | 1545 |     30 |
|  6880 | Cameron  | Sales      | 8245 | 2059-05-09 | 7636 | 1406 |     30 |
|  7235 | Stephen  | Operations | 2678 | 2083-00-31 | 6556 | 1698 |     40 |
|  8467 | Mollie   | Accounting | 5919 | 2015-02-09 | 3999 |  526 |     10 |
|  9180 | Jesse    | Accounting | 2678 | 2101-08-22 | 3562 | 1796 |     10 |
|  9360 | Isaiah   | Accounting | 7940 | 2101-09-03 | 4000 | 1390 |     10 |
|  9384 | Matilda  | Operations | 5817 | 2025-05-23 | 2494 | 1170 |     40 |
+-------+----------+------------+------+------------+------+------+--------+
7 rows in set (0.001 sec)

9. List the employees whose designation is 'Research' and sal is > 5000.

MariaDB [Company]> select * from emp where job = 'Research' and sal > 5000;
+-------+---------+----------+------+------------+------+------+--------+
| empno | empname | job      | mgr  | hiredate   | sal  | comm | deptno |
+-------+---------+----------+------+------------+------+------+--------+
|  9085 | Katie   | Research | 5919 | 1997-01-26 | 9065 | 1166 |     20 |
+-------+---------+----------+------+------------+------+------+--------+
1 row in set (0.001 sec)

10. List the employees details descending wise whose designation is 'Research' and
     commission is > 500.
```

```
MariaDB [Company]> select * from emp where job = 'Research' and comm > 500 order
    by comm desc;
+-------+---------+----------+------+------------+------+------+--------+
| empno | empname | job      | mgr  | hiredate   | sal  | comm | deptno |
+-------+---------+----------+------+------------+------+------+--------+
|  9085 | Katie   | Research | 5919 | 1997-01-26 | 9065 | 1166 |     20 |
|  5883 | Jeffery | Research | 5817 | 2057-08-03 | 2236 |  549 |     20 |
+-------+---------+----------+------+------------+------+------+--------+
2 rows in set (0.001 sec)
```

9 Conclusion

Thus, we have learned SQL DML commands, SELECT Command with SQL operators thoroughly.

10 FAQ

1. **What is the difference between Truncate table and Drop table command?**

   (a) *Truncate table command deletes all the records from the table and resets the identity column to 1.*

   (b) *Drop table command deletes the table and all the records from the table.*

   (c) *Truncate table command is faster than Drop table command.*

   (d) *Truncate table command cannot be rolled back.*

   (e) *Drop table command can be rolled back.*

   Example:

   (a) *Truncate table command*

   ```
   Truncate table CUSTOMERS;
   ```

   (b) *Drop table command*

   ```
   Drop table CUSTOMERS;
   ```

2. **How is the pattern matching done in the SQL?**

   (a) *The pattern matching is done using the LIKE operator.*

   (b) *The pattern matching is done using the wildcard characters.*

   (c) *The wildcard characters are:*

   - *% - Represents zero or more characters.*
   - *_ - Represents a single character.*
   - *[charlist] - Represents any single character in charlist.*

   The Syntax of the command is:

   ```
   SELECT column_name(s) FROM table_name WHERE column_name LIKE pattern;
   ```

   Example:

   ```
   SELECT * FROM CUSTOMERS WHERE CUST_NAME LIKE 'Emp%';
   SELECT * FROM STUDENTS WHERE CUST_NAME LIKE 'AssignmentNumber_';
   ```

3. **Write a DELETE command to delete all the records from CUSTOMERS table.**

   ```
   DELETE FROM CUSTOMERS;
   ```

MIT World Peace University

Database Management Systems
Second Year B. Tech, Semester 4

Group Functions, Join and Nested Queries.

Assignment No. 4

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

April 13, 2023

Contents

1 Aim

Write suitable select command to get requested data from tables

2 Objectives

1. To study Subqueries, Group, Joins and Views

3 Problem Statement

Create tables and solve given queries using , Group, Joins and Views

4 Theory

4.1 Group Functions

Group functions in SQL are functions that operate on groups of rows in a table, and return a single result for each group. They are often used in conjunction with the GROUP BY clause, which divides a table into groups based on one or more columns, and applies the group functions to each group.

The common group functions in SQL are:

1. COUNT - Counts the number of rows in a group.
2. SUM - Calculates the sum of a column in a group.
3. AVG - Calculates the average value of a column in a group.
4. MIN - Finds the minimum value of a column in a group.
5. MAX - Finds the maximum value of a column in a group.

Syntax:

```
SELECT column_name, group_function(column_name)
FROM table_name
WHERE condition
GROUP BY column_name;
```

Example:
Consider the following table "employees":

id	name	department	salary
1	Alice	HR	5000
2	Bob	IT	6000
3	Charlie	HR	4500
4	David	IT	7000
5	Emma	Sales	5500

To count the number of employees in each department, you can use the following query:

```
SELECT department, COUNT(*)
FROM employees
GROUP BY department;
```

department	**COUNT(*)**
HR	2
IT	2
Sales	1

To find the average salary of employees in each department, you can use the following query:

```
SELECT department, AVG(salary)
FROM employees
GROUP BY department;
```

department	**AVG(salary)**
HR	4750
IT	6500
Sales	5500

To find the maximum salary in the entire table, you can use the following query:

```
SELECT MAX(salary)
FROM employees;
```

MAX(salary)
7000

4.2 SQL Join Types

In SQL, join is used to combine two or more tables based on a common column between them. There are several types of joins in SQL, each with its own syntax and usage.

Consider the Following Tables

```
MariaDB [dbms_lab]> select * from booking;
+---------+---------+------------+------------+--------+
| HotelNo | GuestNo | DateFrom   | DateTo     | RoomNo |
+---------+---------+------------+------------+--------+
|       7 |      10 | 2096-04-21 | 2099-12-21 |     10 |
|       8 |       5 | 2077-09-29 | 2109-09-10 |     11 |
|      11 |       4 | 2123-01-05 | 2063-08-30 |      2 |
|      10 |       5 | 2027-02-05 | 2119-12-21 |      7 |
|       9 |       5 | 2081-07-11 | 2031-06-20 |     13 |
|       5 |       5 | 2059-11-19 | 2113-05-22 |     11 |
+---------+---------+------------+------------+--------+
6 rows in set (0.001 sec)

MariaDB [dbms_lab]> select * from Hotel;
+---------+------------------+-----------------------+
| HotelNo | Name             | City                  |
+---------+------------------+-----------------------+
|       1 | Hotel love       | Guernsey              |
|       2 | Hotel imagine    | Jordan                |
|       3 | Hotel rice       | Equatorial Guinea     |
|       4 | Hotel perhaps    | Bolivia               |
|       5 | Hotel show       | Reunion               |
|       6 | Hotel native     | Brunei                |
|       7 | Hotel pool       | Panama                |
```

```
|       8 | Hotel spin       | Guyana                 |
|       9 | Hotel toward     | St. Barthelemy         |
|      10 | Hotel expression | St. Pierre & Miquelon  |
|      11 | Hotel cheese     | Guinea-Bissau          |
|      12 | Hotel motion     | Latvia                 |
|      13 | Hotel lay        | Fiji                   |
|      14 | Hotel stiff      | Brazil                 |
|      15 | Hotel suddenly   | Lithuania              |
|      16 | Hotel stretch    | Montenegro             |
|      17 | Hotel current    | Isle of Man            |
|      18 | Hotel forest     | Haiti                  |
+---------+------------------+------------------------+
18 rows in set (0.001 sec)
```

4.3 Inner Join or Simple Join

Defintion

The inner join is used to select all matching rows or columns in both tables or as long as the defined condition is valid in SQL.

Figure

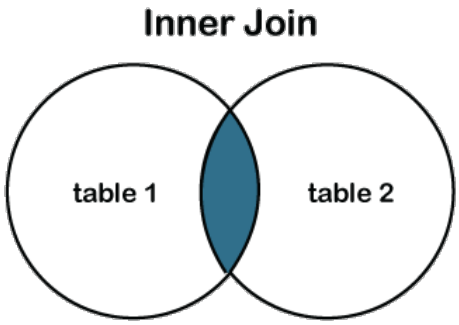

Figure 1: Inner Join

Syntax

```
Select column_1, column_2, column_3 FROM table_1 INNER JOIN table_2 ON table_1.
    column = table_2.column;
```

Example

```
MariaDB [dbms_lab]> select booking.HotelNo, Hotel.name from booking inner join
    Hotel on booking.HotelNo = Hotel.HotelNo;
+---------+------------------+
| HotelNo | name             |
+---------+------------------+
|       5 | Hotel show       |
|       7 | Hotel pool       |
|       8 | Hotel spin       |
```

```
|        9 | Hotel toward      |
|       10 | Hotel expression  |
|       11 | Hotel cheese      |
+----------+-------------------+
6 rows in set (0.001 sec)
```

4.4 Left Join

Defintion

The LEFT JOIN is used to retrieve all records from the left table (table1) and the matched rows or columns from the right table (table2). If both tables do not contain any matched rows or columns, it returns the NULL.

Figure

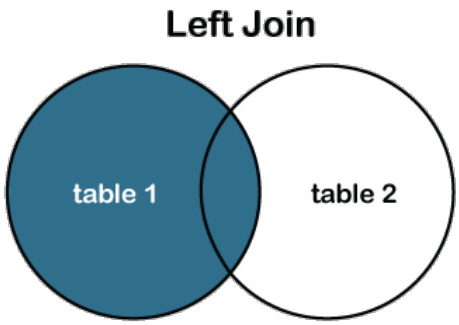

Figure 2: Left Join

Syntax

```
Select column_1, column_2, column(s) FROM table_1 LEFT JOIN table_2 ON table_1.
    column_name = table_2.column_name;
```

Example

```
MariaDB [dbms_lab]> select * from booking left join Hotel on booking.HotelNo =
    Hotel.HotelNo;
+--------+---------+-------------------+------------------------+
| RoomNo | HotelNo | Name              | City                   |
+--------+---------+-------------------+------------------------+
|     10 |       7 | Hotel pool        | Panama                 |
|     11 |       8 | Hotel spin        | Guyana                 |
|      2 |      11 | Hotel cheese      | Guinea-Bissau          |
|      7 |      10 | Hotel expression  | St. Pierre & Miquelon  |
|     13 |       9 | Hotel toward      | St. Barthelemy         |
|     11 |       5 | Hotel show        | Reunion                |
+--------+---------+-------------------+------------------------+
6 rows in set (0.001 sec)
```

4.5 Right Join

Defintion

The RIGHT JOIN is used to retrieve all records from the right table (table2) and the matched rows or columns from the left table (table1). If both tables do not contain any matched rows or columns, it returns the NULL.

Figure

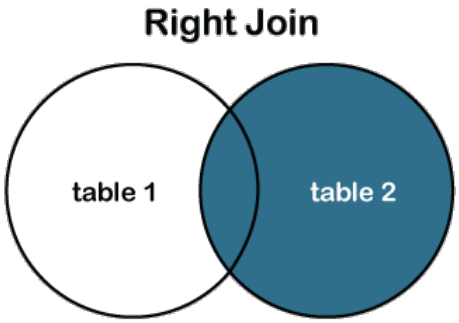

Figure 3: Right Join

Syntax

```
Select column_1, column_2, column_3 FROM table_1 RIGHT JOIN table_2 ON table_1.
    column = table_2.column;
```

Example

```

MariaDB [dbms_lab]> select * from booking right join Hotel on booking.HotelNo =
    Hotel.HotelNo;
+--------+---------+-------------------+-----------------------+
| RoomNo | HotelNo | Name              | City                  |
+--------+---------+-------------------+-----------------------+
|     10 |       7 | Hotel pool        | Panama                |
|     11 |       8 | Hotel spin        | Guyana                |
|      2 |      11 | Hotel cheese      | Guinea-Bissau         |
|      7 |      10 | Hotel expression  | St. Pierre & Miquelon |
|     13 |       9 | Hotel toward      | St. Barthelemy        |
|     11 |       5 | Hotel show        | Reunion               |
|   NULL |       1 | Hotel love        | Guernsey              |
|   NULL |       2 | Hotel imagine     | Jordan                |
|   NULL |       3 | Hotel rice        | Equatorial Guinea     |
|   NULL |       4 | Hotel perhaps     | Bolivia               |
|   NULL |       6 | Hotel native      | Brunei                |
|   NULL |      12 | Hotel motion      | Latvia                |
|   NULL |      13 | Hotel lay         | Fiji                  |
|   NULL |      14 | Hotel stiff       | Brazil                |
|   NULL |      15 | Hotel suddenly    | Lithuania             |
|   NULL |      16 | Hotel stretch     | Montenegro            |
```

```
|   NULL |       17 | Hotel current    | Isle of Man           |
|   NULL |       18 | Hotel forest     | Haiti                 |
+--------+----------+------------------+-----------------------+
18 rows in set (0.002 sec)
```

4.6 Natural Join

Defintion

It is a type of inner type that joins two or more tables based on the same column name and has the same data type present on both tables.

Syntax

```
Select * from tablename1 Natural JOIN tablename_2;
```

Example

```
MariaDB [dbms_lab]> select * from booking natural join Hotel
-> ;
+--------+------------------+-----------------------+
| RoomNo | Name             | City                  |
+--------+------------------+-----------------------+
|     10 | Hotel pool       | Panama                |
|     11 | Hotel spin       | Guyana                |
|      2 | Hotel cheese     | Guinea-Bissau         |
|      7 | Hotel expression | St. Pierre & Miquelon |
|     13 | Hotel toward     | St. Barthelemy        |
|     11 | Hotel show       | Reunion               |
+--------+------------------+-----------------------+
6 rows in set (0.001 sec)
```

4.7 Full Outer Join

Defintion

It is a combination result set of both LEFT JOIN and RIGHT JOIN. The joined tables return all records from both the tables and if no matches are found in the table, it places NULL. It is also called a FULL OUTER JOIN.

Figure

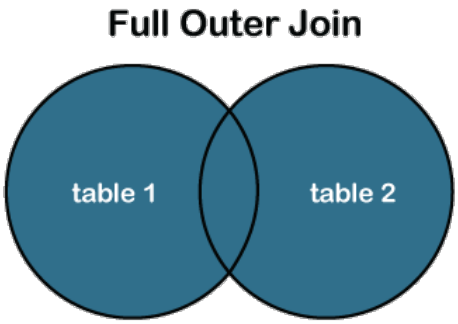

Figure 4: Right Join

Syntax

```
  Select column_1, column_2, column(s) FROM table_1 FULL JOIN table_2 ON table_1.
    column_name = table_2.column_name;
```

Example

```

MariaDB [dbms_lab]> select * from booking left join Hotel on booking.HotelNo =
    Hotel.HotelNo
-> union
-> select * from booking right join Hotel on booking.HotelNo = Hotel.HotelNo;
+--------+---------+-------------------+-----------------------+
| RoomNo | HotelNo | Name              | City                  |
+--------+---------+-------------------+-----------------------+
|     10 |       7 | Hotel pool        | Panama                |
|     11 |       8 | Hotel spin        | Guyana                |
|      2 |      11 | Hotel cheese      | Guinea-Bissau         |
|      7 |      10 | Hotel expression  | St. Pierre & Miquelon |
|     13 |       9 | Hotel toward      | St. Barthelemy        |
|     11 |       5 | Hotel show        | Reunion               |
|   NULL |       1 | Hotel love        | Guernsey              |
|   NULL |       2 | Hotel imagine     | Jordan                |
|   NULL |       3 | Hotel rice        | Equatorial Guinea     |
|   NULL |       4 | Hotel perhaps     | Bolivia               |
|   NULL |       6 | Hotel native      | Brunei                |
|   NULL |      12 | Hotel motion      | Latvia                |
|   NULL |      13 | Hotel lay         | Fiji                  |
|   NULL |      14 | Hotel stiff       | Brazil                |
|   NULL |      15 | Hotel suddenly    | Lithuania             |
|   NULL |      16 | Hotel stretch     | Montenegro            |
|   NULL |      17 | Hotel current     | Isle of Man           |
|   NULL |      18 | Hotel forest      | Haiti                 |
+--------+---------+-------------------+-----------------------+
18 rows in set (0.008 sec)
```

4.8 Cross Join

Defintion

It is also known as CARTESIAN JOIN, which returns the Cartesian product of two or more joined tables. The CROSS JOIN produces a table that merges each row from the first table with each second table row. It is not required to include any condition in CROSS JOIN.

Syntax

```
Select * from table_1 cross join table_2;
```

Example

```

MariaDB [dbms_lab]>
MariaDB [dbms_lab]> select booking.HotelNo, booking.RoomNo, Hotel.name, Hotel.City
    from booking cross join Hotel;
+---------+--------+------------------+-----------------------+
| HotelNo | RoomNo | name             | City                  |
+---------+--------+------------------+-----------------------+
|       7 |     10 | Hotel love       | Guernsey              |
|       8 |     11 | Hotel love       | Guernsey              |
|      11 |      2 | Hotel love       | Guernsey              |
|      10 |      7 | Hotel love       | Guernsey              |
|       9 |     13 | Hotel love       | Guernsey              |
|       5 |     11 | Hotel love       | Guernsey              |
|       7 |     10 | Hotel imagine    | Jordan                |
|       8 |     11 | Hotel imagine    | Jordan                |
|      11 |      2 | Hotel imagine    | Jordan                |
|      10 |      7 | Hotel imagine    | Jordan                |
|       9 |     13 | Hotel imagine    | Jordan                |
|       5 |     11 | Hotel imagine    | Jordan                |
|       7 |     10 | Hotel rice       | Equatorial Guinea     |
|       8 |     11 | Hotel rice       | Equatorial Guinea     |
|      11 |      2 | Hotel rice       | Equatorial Guinea     |
|      10 |      7 | Hotel rice       | Equatorial Guinea     |
|       9 |     13 | Hotel rice       | Equatorial Guinea     |
|       5 |     11 | Hotel rice       | Equatorial Guinea     |
|       7 |     10 | Hotel perhaps    | Bolivia               |
|       8 |     11 | Hotel perhaps    | Bolivia               |
|      11 |      2 | Hotel perhaps    | Bolivia               |
|      10 |      7 | Hotel perhaps    | Bolivia               |
|       9 |     13 | Hotel perhaps    | Bolivia               |
|       5 |     11 | Hotel perhaps    | Bolivia               |
|       7 |     10 | Hotel show       | Reunion               |
|       8 |     11 | Hotel show       | Reunion               |
|      11 |      2 | Hotel show       | Reunion               |
|      10 |      7 | Hotel show       | Reunion               |
|       9 |     13 | Hotel show       | Reunion               |
|       5 |     11 | Hotel show       | Reunion               |
|       7 |     10 | Hotel native     | Brunei                |
|       8 |     11 | Hotel native     | Brunei                |
|      11 |      2 | Hotel native     | Brunei                |
|      10 |      7 | Hotel native     | Brunei                |
|       9 |     13 | Hotel native     | Brunei                |
```

```
|        5 |     11 | Hotel native     | Brunei                |
|        7 |     10 | Hotel pool       | Panama                |
|        8 |     11 | Hotel pool       | Panama                |
|       11 |      2 | Hotel pool       | Panama                |
|       10 |      7 | Hotel pool       | Panama                |
|        9 |     13 | Hotel pool       | Panama                |
|        5 |     11 | Hotel pool       | Panama                |
|        7 |     10 | Hotel spin       | Guyana                |
|        8 |     11 | Hotel spin       | Guyana                |
|       11 |      2 | Hotel spin       | Guyana                |
|       10 |      7 | Hotel spin       | Guyana                |
|        9 |     13 | Hotel spin       | Guyana                |
|        5 |     11 | Hotel spin       | Guyana                |
|        7 |     10 | Hotel toward     | St. Barthelemy        |
|        8 |     11 | Hotel toward     | St. Barthelemy        |
|       11 |      2 | Hotel toward     | St. Barthelemy        |
|       10 |      7 | Hotel toward     | St. Barthelemy        |
|        9 |     13 | Hotel toward     | St. Barthelemy        |
|        5 |     11 | Hotel toward     | St. Barthelemy        |
|        7 |     10 | Hotel expression | St. Pierre & Miquelon |
|        8 |     11 | Hotel expression | St. Pierre & Miquelon |
|       11 |      2 | Hotel expression | St. Pierre & Miquelon |
|       10 |      7 | Hotel expression | St. Pierre & Miquelon |
|        9 |     13 | Hotel expression | St. Pierre & Miquelon |
|        5 |     11 | Hotel expression | St. Pierre & Miquelon |
|        7 |     10 | Hotel cheese     | Guinea-Bissau         |
|        8 |     11 | Hotel cheese     | Guinea-Bissau         |
|       11 |      2 | Hotel cheese     | Guinea-Bissau         |
|       10 |      7 | Hotel cheese     | Guinea-Bissau         |
|        9 |     13 | Hotel cheese     | Guinea-Bissau         |
|        5 |     11 | Hotel cheese     | Guinea-Bissau         |
|        7 |     10 | Hotel motion     | Latvia                |
|        8 |     11 | Hotel motion     | Latvia                |
|       11 |      2 | Hotel motion     | Latvia                |
|       10 |      7 | Hotel motion     | Latvia                |
|        9 |     13 | Hotel motion     | Latvia                |
|        5 |     11 | Hotel motion     | Latvia                |
|        7 |     10 | Hotel lay        | Fiji                  |
|        8 |     11 | Hotel lay        | Fiji                  |
|       11 |      2 | Hotel lay        | Fiji                  |
|       10 |      7 | Hotel lay        | Fiji                  |
|        9 |     13 | Hotel lay        | Fiji                  |
|        5 |     11 | Hotel lay        | Fiji                  |
|        7 |     10 | Hotel stiff      | Brazil                |
|        8 |     11 | Hotel stiff      | Brazil                |
|       11 |      2 | Hotel stiff      | Brazil                |
|       10 |      7 | Hotel stiff      | Brazil                |
|        9 |     13 | Hotel stiff      | Brazil                |
|        5 |     11 | Hotel stiff      | Brazil                |
|        7 |     10 | Hotel suddenly   | Lithuania             |
|        8 |     11 | Hotel suddenly   | Lithuania             |
|       11 |      2 | Hotel suddenly   | Lithuania             |
|       10 |      7 | Hotel suddenly   | Lithuania             |
|        9 |     13 | Hotel suddenly   | Lithuania             |
|        5 |     11 | Hotel suddenly   | Lithuania             |
|        7 |     10 | Hotel stretch    | Montenegro            |
|        8 |     11 | Hotel stretch    | Montenegro            |
|       11 |      2 | Hotel stretch    | Montenegro            |
|       10 |      7 | Hotel stretch    | Montenegro            |
```

```
|        9 |      13 | Hotel stretch    | Montenegro             |
|        5 |      11 | Hotel stretch    | Montenegro             |
|        7 |      10 | Hotel current    | Isle of Man            |
|        8 |      11 | Hotel current    | Isle of Man            |
|       11 |       2 | Hotel current    | Isle of Man            |
|       10 |       7 | Hotel current    | Isle of Man            |
|        9 |      13 | Hotel current    | Isle of Man            |
|        5 |      11 | Hotel current    | Isle of Man            |
|        7 |      10 | Hotel forest     | Haiti                  |
|        8 |      11 | Hotel forest     | Haiti                  |
|       11 |       2 | Hotel forest     | Haiti                  |
|       10 |       7 | Hotel forest     | Haiti                  |
|        9 |      13 | Hotel forest     | Haiti                  |
|        5 |      11 | Hotel forest     | Haiti                  |
+----------+---------+------------------+------------------------+
108 rows in set (0.000 sec)
```

4.9 Self Join

Defintion

It is a SELF JOIN used to create a table by joining itself as there were two tables. It makes temporary naming of at least one table in an SQL statement.

Syntax

```
  Select column1, column2, column(s) FROM table_1 Tbl1, table_1 Tbl2 WHERE
    condition;
```

5 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Draw.io for Drawing the ER diagram.

6 Input

Given Database from the Problem Statement for the Assignment for our batch. (A1 PA 20)

7 Executed Queries

7.1 Questions SetA

```
MariaDB [dbms_lab]> select * from Room;
+--------+---------+-------+-------+
| RoomNo | HotelNo | Type  | Price |
+--------+---------+-------+-------+
|      1 |       1 | Suite |  1646 |
|      2 |       2 | Suite |  1264 |
|      3 |       1 | 2 Bed |   773 |
|      4 |       4 | 2 Bed |  1949 |
|      5 |       1 | 3 Bed |  1959 |
|      6 |       3 | 3 Bed |   674 |
```

```
|      7 |       1 | 1 Bed |  1018 |
|      8 |       3 | 1 Bed |  1314 |
|      9 |       1 | Suite |  1308 |
|     10 |       9 | 3 Bed |  1366 |
|     11 |      10 | 1 Bed |   666 |
|     12 |       7 | 2 Bed |  1498 |
|     13 |       7 | Suite |   984 |
+--------+---------+-------+-------+
13 rows in set (0.001 sec)

MariaDB [dbms_lab]> select * from Hotel;
+---------+------------------+-----------------------+
| HotelNo | Name             | City                  |
+---------+------------------+-----------------------+
|       1 | Hotel love       | Guernsey              |
|       2 | Hotel imagine    | Jordan                |
|       3 | Hotel rice       | Equatorial Guinea     |
|       4 | Hotel perhaps    | Bolivia               |
|       5 | Hotel show       | Reunion               |
|       6 | Hotel native     | Brunei                |
|       7 | Hotel pool       | Panama                |
|       8 | Hotel spin       | Guyana                |
|       9 | Hotel toward     | St. Barthelemy        |
|      10 | Hotel expression | St. Pierre & Miquelon |
|      11 | Hotel cheese     | Guinea-Bissau         |
|      12 | Hotel motion     | Latvia                |
|      13 | Hotel lay        | Fiji                  |
|      14 | Hotel stiff      | Brazil                |
|      15 | Hotel suddenly   | Lithuania             |
|      16 | Hotel stretch    | Montenegro            |
|      17 | Hotel current    | Isle of Man           |
|      18 | Hotel forest     | Haiti                 |
+---------+------------------+-----------------------+
18 rows in set (0.001 sec)

MariaDB [dbms_lab]> select * from booking;
+---------+---------+------------+------------+--------+
| HotelNo | GuestNo | DateFrom   | DateTo     | RoomNo |
+---------+---------+------------+------------+--------+
|       7 |      10 | 2096-04-21 | 2099-12-21 |     10 |
|       8 |       5 | 2077-09-29 | 2109-09-10 |     11 |
|      11 |       4 | 2123-01-05 | 2063-08-30 |      2 |
|      10 |       5 | 2027-02-05 | 2119-12-21 |      7 |
|       9 |       5 | 2081-07-11 | 2031-06-20 |     13 |
|       5 |       5 | 2059-11-19 | 2113-05-22 |     11 |
+---------+---------+------------+------------+--------+
6 rows in set (0.000 sec)

MariaDB [dbms_lab]> select * from Guest;
+---------+----------------+-----------------+
| GuestNo | GuestName      | GuessAddress    |
+---------+----------------+-----------------+
|       2 | Patrick Taylor | Lebanon         |
|       4 | Mattie Vargas  | St. Barthelemy  |
|       5 | Travis Frazier | Gambia          |
|      10 | Sarah Ramsey   | Jamaica         |
|      11 | Rachel Keller  | Kenya           |
|      15 | Nathan Higgins | Puerto Rico     |
|      16 | Maude Gonzales | St. Lucia       |
```

```
+---------+----------------+-----------------+
7 rows in set (0.000 sec)

MariaDB [dbms_lab]>
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> -- 1. many hotels are there?
MariaDB [dbms_lab]> select count(*) from Hotel;
+----------+
| count(*) |
+----------+
|       18 |
+----------+
1 row in set (0.000 sec)

MariaDB [dbms_lab]>
MariaDB [dbms_lab]> -- 2. the price and type of all rooms at the Grosvenor Hotel.
MariaDB [dbms_lab]> select price, type, Name from Room, Hotel where Room.HotelNo =
    Hotel.HotelNo and Name = 'Hotel love';
+-------+-------+------------+
| price | type  | Name       |
+-------+-------+------------+
|  1646 | Suite | Hotel love |
|   773 | 2 Bed | Hotel love |
|  1959 | 3 Bed | Hotel love |
|  1018 | 1 Bed | Hotel love |
|  1308 | Suite | Hotel love |
+-------+-------+------------+
5 rows in set (0.001 sec)

MariaDB [dbms_lab]>
MariaDB [dbms_lab]> -- 3. the number of rooms in each hotel.
MariaDB [dbms_lab]> select Room.HotelNo, Hotel.NAME,  count(*) from Room, Hotel
   where Room.HotelNo = Hotel.HotelNo group by HotelNo;
+---------+------------------+----------+
| HotelNo | NAME             | count(*) |
+---------+------------------+----------+
|       1 | Hotel love       |        5 |
|       2 | Hotel imagine    |        1 |
|       3 | Hotel rice       |        2 |
|       4 | Hotel perhaps    |        1 |
|       7 | Hotel pool       |        2 |
|       9 | Hotel toward     |        1 |
|      10 | Hotel expression |        1 |
+---------+------------------+----------+
7 rows in set (0.000 sec)

MariaDB [dbms_lab]>
MariaDB [dbms_lab]> -- 4. Update the price of all rooms by 5%.
MariaDB [dbms_lab]> select r.Price, r.Price + r.Price * 0.05 as Updated_price from
    Room r;
+-------+---------------+
| Price | Updated_price |
+-------+---------------+
|  1646 |       1728.30 |
|  1264 |       1327.20 |
|   773 |        811.65 |
|  1949 |       2046.45 |
|  1959 |       2056.95 |
|   674 |        707.70 |
```

```
|   1018 |        1068.90 |
|   1314 |        1379.70 |
|   1308 |        1373.40 |
|   1366 |        1434.30 |
|    666 |         699.30 |
|   1498 |        1572.90 |
|    984 |        1033.20 |
+-------+----------------+
13 rows in set (0.000 sec)

MariaDB [dbms_lab]>
MariaDB [dbms_lab]> -- 5. full details of all hotels in London.
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> select * from Hotel where City = 'Jordan';
+---------+---------------+--------+
| HotelNo | Name          | City   |
+---------+---------------+--------+
|       2 | Hotel imagine | Jordan |
+---------+---------------+--------+
1 row in set (0.000 sec)

MariaDB [dbms_lab]>
MariaDB [dbms_lab]> -- 6. What is the average price of a room?
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> select avg(Price) from Room;
+------------+
| avg(Price) |
+------------+
|  1263.0000 |
+------------+
1 row in set (0.000 sec)

MariaDB [dbms_lab]>
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> -- 7. all guests currently staying at the Grosvenor Hotel.
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> select Guest.* from Guest, booking, Hotel where Guest.GuestNo
    = booking.GuestNo and booking.HotelNo = Hotel.HotelNo and Hotel.Name = 'Hotel
    pool';
+---------+--------------+--------------+
| GuestNo | GuestName    | GuessAddress |
+---------+--------------+--------------+
|      10 | Sarah Ramsey | Jamaica      |
+---------+--------------+--------------+
1 row in set (0.001 sec)

MariaDB [dbms_lab]>
MariaDB [dbms_lab]> -- 8. the number of rooms in each hotel in London.
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> select count(*) from Room, Hotel where Room.HotelNo = Hotel.
    HotelNo and Hotel.City = 'Jordan';
+----------+
| count(*) |
+----------+
|        1 |
+----------+
1 row in set (0.000 sec)

MariaDB [dbms_lab]>
```

7.2 Questions Set B

```
MariaDB [dbms_lab]> CREATE TABLE zipcode (
    ->   zip VARCHAR(10) PRIMARY KEY,
    ->   city VARCHAR(50) NOT NULL
    -> );
Query OK, 0 rows affected (0.50 sec)

MariaDB [dbms_lab]> CREATE TABLE customers (
    ->   cno INT PRIMARY KEY,
    ->   cname VARCHAR(50) NOT NULL,
    ->   street VARCHAR(50),
    ->   zip VARCHAR(10),
    ->   phone VARCHAR(20),
    ->   CONSTRAINT zip_fk FOREIGN KEY (zip) REFERENCES zipcode (zip)
    -> );
Query OK, 0 rows affected (0.39 sec)


MariaDB [dbms_lab]> CREATE TABLE emp (
    ->   eno INT PRIMARY KEY,
    ->   ename VARCHAR(50) NOT NULL,
    ->   zip VARCHAR(10),
    ->   hdate DATE,
    ->   FOREIGN KEY (zip) REFERENCES zipcode (zip)
    -> );
Query OK, 0 rows affected (0.44 sec)

MariaDB [dbms_lab]> CREATE TABLE parts (
    ->   pno INT PRIMARY KEY,
    ->   pname VARCHAR(50) NOT NULL,
    ->   qty_on_hand INT CHECK (qty_on_hand >= 0),
    ->   price DECIMAL(10, 2) CHECK (price >= 0)
    -> );
Query OK, 0 rows affected (0.40 sec)

MariaDB [dbms_lab]> ^C
MariaDB [dbms_lab]> CREATE TABLE orders (
    ->   ono INT PRIMARY KEY,
    ->   cno INT,
    ->   receivedate DATE,
    ->   shippeddate DATE,
    ->   CONSTRAINT cno_fk FOREIGN KEY (cno) REFERENCES customers (cno)
    -> );
Query OK, 0 rows affected (0.22 sec)

MariaDB [dbms_lab]> CREATE TABLE odetails (
    ->   ono INT,
    ->   pno INT,
    ->   qty INT CHECK (qty >= 0),
    ->   PRIMARY KEY (ono, pno),
    ->   CONSTRAINT ono_fk FOREIGN KEY (ono) REFERENCES orders (ono),
    ->   CONSTRAINT pno_fk FOREIGN KEY (pno) REFERENCES parts (pno)
    -> );
Query OK, 0 rows affected (0.24 sec)

MariaDB [dbms_lab]> INSERT INTO zipcode (zip, city) VALUES
    ->   ('10001', 'New York'),
    ->   ('10002', 'Los Angeles'),
```

```
    ->   ('10003', 'Chicago');
Query OK, 3 rows affected (0.44 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [dbms_lab]> INSERT INTO emp (eno, ename, zip, hdate) VALUES
    ->   (1, 'John Smith', '10001', '2022-01-01'),
    ->   (2, 'Jane Doe', '10002', '2022-02-01'),
    ->   (3, 'Bob Johnson', '10003', '2022-03-01');
Query OK, 3 rows affected (0.39 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [dbms_lab]> INSERT INTO parts (pno, pname, qty_on_hand, price) VALUES
    ->   (1, 'Widget', 10, 19.99),
    ->   (2, 'Gizmo', 5, 29.99),
    ->   (3, 'Doodad', 20, 9.99);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [dbms_lab]> INSERT INTO customers (cno, cname, street, zip, phone) VALUES
    ->   (1, 'Acme Inc.', '123 Main St.', '10001', '555-1234'),
    ->   (2, 'Globex Corp.', '456 Elm St.', '10002', '555-5678'),
    ->   (3, 'Initech Ltd.', '789 Oak St.', '10003', '555-9101');
Query OK, 3 rows affected (0.38 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [dbms_lab]> INSERT INTO orders (ono, cno, receivedate, shippeddate) VALUES
    ->   (1, 1, '2022-01-01', '2022-01-02'),
    ->   (2, 2, '2022-02-01', NULL),
    ->   (3, 3, '2022-03-01', '2022-03-02');
Query OK, 3 rows affected (0.08 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [dbms_lab]> INSERT INTO odetails (ono, pno, qty) VALUES
    ->   (1, 1, 5),
    ->   (1, 2, 2),
    ->   (2, 3, 10),
    ->   (3, 1, 3),
    ->   (3, 2, 1),
    ->   (3, 3, 5);
Query OK, 6 rows affected (0.13 sec)
Records: 6  Duplicates: 0  Warnings: 0


-- q1
SELECT pno, pname
FROM parts
WHERE price < 20.00 at line 1
MariaDB [dbms_lab]> SELECT pno, pname
    -> FROM parts
    -> WHERE price < 20.00;
+-----+--------+
| pno | pname  |
+-----+--------+
|   1 | Widget |
|   3 | Doodad |
+-----+--------+
2 rows in set (0.00 sec)

MariaDB [dbms_lab]> -- q2
```

```
MariaDB [dbms_lab]> SELECT orders.ono, customers.cname
    -> FROM orders
    -> JOIN customers ON orders.cno = customers.cno
    -> WHERE orders.shippeddate IS NULL;
+-----+--------------+
| ono | cname        |
+-----+--------------+
|   2 | Globex Corp. |
+-----+--------------+
1 row in set (0.00 sec)

MariaDB [dbms_lab]> -- q4
MariaDB [dbms_lab]> SELECT DISTINCT zipcode.city
    -> FROM emp
    -> JOIN zipcode ON emp.zip = zipcode.zip
    -> UNION
    -> SELECT DISTINCT zipcode.city
    -> FROM customers
    -> JOIN zipcode ON customers.zip = zipcode.zip;

+------------+
| city       |
+------------+
| New York   |
| Los Angeles|
| Chicago    |
+------------+

MariaDB [dbms_lab]> -- q5
MariaDB [dbms_lab]> SELECT SUM(qty)
    -> FROM odetails
    -> WHERE pno = '1060';
+----------+
| SUM(qty) |
+----------+
|     NULL |
+----------+
1 row in set (0.00 sec)

MariaDB [dbms_lab]> -- q6
MariaDB [dbms_lab]> SELECT COUNT(*) as total_customers
    -> FROM customers;
+-----------------+
| total_customers |
+-----------------+
|               3 |
+-----------------+
1 row in set (0.37 sec)

MariaDB [dbms_lab]> -- q7
MariaDB [dbms_lab]> CREATE VIEW customer_orders AS
    -> SELECT orders.ono, orders.receivedate, orders.shippeddate, customers.cname
    -> FROM orders
    -> JOIN customers ON orders.cno = customers.cno;
Query OK, 0 rows affected (0.44 sec)


MariaDB [dbms_lab]> select * from customer_orders;
+-----+-------------+-------------+--------------+
```

```
| ono | receivedate | shippeddate | cname        |
+-----+-------------+-------------+--------------+
|   1 | 2022-01-01  | 2022-01-02  | Acme Inc.    |
|   2 | 2022-02-01  | NULL        | Globex Corp. |
|   3 | 2022-03-01  | 2022-03-02  | Initech Ltd. |
+-----+-------------+-------------+--------------+
3 rows in set (0.35 sec)
```

8 Conclusion

Thus, we have learned to Select Group By, Joins and Subqueries commands thoroughly.

9 FAQ

1. **When to use self join? How does it differ from other joins?**

   A self join is used when you need to join a table with itself, typically to find relationships between rows in the same table. It differs from other joins in that you are joining a table with itself rather than joining two separate tables. A self join can be performed using an alias to distinguish between the two copies of the table being joined.

```sql
SELECT t1.employee_name, t2.employee_name
FROM employees t1
JOIN employees t2 ON t1.manager_id = t2.employee_id;
```

2. **Compare Cross Join with Natural Join. Share your comments.**

   A cross join produces the Cartesian product of two tables, resulting in a combination of all rows from one table with all rows from another table. A natural join matches two tables based on their common column names. It automatically eliminates duplicate columns from the result set, and the result set only contains the columns with the same name from both tables.

```sql
  SELECT *
  FROM table1
  CROSS JOIN table2;
```

3. **What is the importance of SQL joins in database management? Explain its types.**

   SQL joins are important in database management because they allow you to combine data from two or more tables into a single result set. This allows you to extract meaningful information from your data by revealing relationships between tables. There are four main types of SQL joins: inner join, left join, right join, and full outer join.

4. **What are the different types of Joins in SQL?**

   The different types of SQL joins are:

   - Inner join: returns only the matching rows from both tables based on the specified join condition.
   - Left join: returns all the rows from the left table and the matching rows from the right table based on the specified join condition.
   - Right join: returns all the rows from the right table and the matching rows from the left table based on the specified join condition.
   - Full outer join: returns all the rows from both tables, matching rows where possible and filling in NULL values for non-matching rows.

```sql
SELECT *
FROM table1
INNER JOIN table2
ON table1.column = table2.column;

```

```sql
SELECT *
FROM table1
LEFT JOIN table2
ON table1.column = table2.column;
```

5. **State the difference between inner join and left join.**

   The main difference between an inner join and a left join is that an inner join only returns matching rows from both tables based on the specified join condition, while a left join returns all the rows from the left table and the matching rows from the right table based on the specified join condition.

```sql
SELECT *
FROM table1
LEFT JOIN table2
ON table1.column = table2.column;
```

```sql
SELECT *
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

6. **State difference between left join and right join.**

   The main difference between a left join and a right join is that a left join returns all the rows from the left table and the matching rows from the right table based on the specified join condition, while a right join returns all the rows from the right table and the matching rows from the left table based on the specified join condition.

```sql
SELECT *
FROM table1
LEFT JOIN table2
ON table1.column = table2.column;
```

```sql
SELECT *
FROM table1
RIGHT JOIN table2
ON table1.column = table2.column;

```

MIT World Peace University

Database Management Systems
Second Year B. Tech, Semester 4

SQL Queries on Functions, Data Sorting, Subquery, Group by, Having, Set Operations and View

Assignment No. 5

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

April 13, 2023

Contents

1 Aim

Write suitable select commands to execute queries on the given data set.

2 Objectives

1. To get basic understanding of Aggregate Functions, Order By clause
2. To get basic understanding of Subquery or Inner query or Nested query and Select using subquery.
3. To understand the basic concept of Correlated Subquery.
4. To get familiar with the basic ALL, ANY, EXISTS, SOME functionality.
5. To understand basic TCL commands

3 Problem Statement

Create tables and solve given queries using Subqueries

4 Theory

4.1 Aggregate Functions

Aggregate functions are used to perform calculations on a set of values and return a single value. The following are the aggregate functions:

- COUNT() - Returns the number of rows that matches a specified criteria
- SUM() - Returns the sum of all the values in a column
- AVG() - Returns the average value of a numeric column
- MIN() - Returns the smallest value of the selected column
- MAX() - Returns the largest value of the selected column

4.2 Order By Clause

The ORDER BY clause is used to sort the result-set in ascending or descending order. The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

Syntax

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC;
```

4.3 Group By Clause

The GROUP BY clause is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;
```

4.4 Subqueries

A subquery (sub-select) is a query within a query. The subquery is executed first, and the main query uses the subquery as a source of data.

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator
(SELECT STATEMENT);
```

4.5 Views

A view is a virtual table based on the result-set of an SQL statement.

Syntax

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition;
```

4.6 TCL Commands

TCL stands for Transaction Control Language. It is a set of SQL commands that are used to control the transaction. The following are the TCL commands:

- COMMIT - permanently saves all changes made by the transaction.
- ROLLBACK - cancels all changes made by the transaction
- SAVEPOINT - sets a savepoint within a transaction
- SET TRANSACTION - sets the transaction characteristics for the current transaction

5 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Draw.io for Drawing the ER diagram.

6 Input

Given Database from the Problem Statement for the Assignment for our batch. (A1 PA 20)

7 Creation and Insertion of Values in the Tables

```
-- Active: 1678946907415@@127.0.0.1@3306@dbms_lab
use dbms_lab;

show tables;

create table airline(name varchar(50) primary key);

create table airplane(reg_no int primary key, model_no int, capacity int, name
    varchar(50), foreign key(name) references airline(name));
create table flights(flight_no int primary key, place_from varchar(50), place_to
    varchar(50), departure_date date, departure_time time, arrival_date date,
    arrival_time time, reg_no int, foreign key(reg_no) references airplane(reg_no))
    ;

create table passenger(email varchar(50) primary key, first_name varchar(50),
    surname varchar(50));
create table flight_booking(email varchar(50), flight_no int, no_seats int,
    foreign key(email) references passenger(email), foreign key(flight_no)
    references flights(flight_no), primary key(email, flight_no));

describe airplane;

insert into airline values("Qatar Airways");
insert into airline values("Emirates");
insert into airline values("Air India");

insert into airplane values(111,007,180,"Qatar Airways");
insert into airplane values(112,007,169,"Qatar Airways");
insert into airplane values(113,008,200,"Qatar Airways");
insert into airplane values(221,017,150,"Emirates");
insert into airplane values(222,017,140,"Emirates");
insert into airplane values(223,018,175,"Emirates");
insert into airplane values(333,027,200,"Air India");
insert into airplane values(334,027,150,"Air India");
insert into airplane values(335,028,175,"Air India");

select * from airplane;

describe flights;
insert into flights values(12345,"Mumbai","London","2021-07-27","12:12:12","
    2021-07-28","23:59:56",111);
insert into flights values(67890,"Pune","Bangalore","2021-07-27","12:12:12","
    2021-07-27","16:59:56",221);
```

```
insert into flights values(23456,"London","Pune","2021-07-27","12:12:12","
    2021-07-28","22:59:56",333);

select * from flights;

describe passenger;

insert into passenger values("love@gmail.com","Love","Quinn");
insert into passenger values("joe@gmail.com","Joe","Goldberg");
insert into passenger values("beck@gmail.com","Gwen","Beck");

describe flight_booking;

insert into flight_booking values("love@gmail.com",12345,6);
insert into flight_booking values("joe@gmail.com",23456,2);
insert into flight_booking values("beck@gmail.com",67890,6);

select * from flight_booking;
select * from flights;

-- QUERIES

-- 1. Display the Passenger email ,Flight_no,Source and Destination Airport Names
    for all flights
-- booked

select b.email, b.flight_no, f.place_from, f.place_to from flight_booking as b
    inner join flights as f where b.flight_no = f.flight_no;

-- 2.
--  Display the flight and passenger details for the flights booked having
    Departure Date between
-- 23-08-2021 and 25-08-2021

select * from flights as f, passenger as p, flight_booking as b where b.email = p.
    email and b.flight_no = f.flight_no and departure_date between "2021-07-27" and
     "2021-07-28";

-- 3.
--  Display the top 5 airplanes that participated in Flights from Mumbai to London
     based on the
-- airplane capacity

select * from airplane as a, flights as f where a.reg_no = f.reg_no and f.
    place_from = "Mumbai" and f.place_to = "London" order by a.capacity desc limit
    5;

-- 4.Display the passenger first names who have booked the no_of seats smaller
    than the average
-- number of seats booked by all passengers for the arrival airport:New Delhi

select * from passenger as p, flight_booking as b, flights as f where p.email = b.
    email and f.flight_no = b.flight_no and f.place_to = "New Delhi" and b.no_seats
     < all(select avg(no_seats) from flight_booking);


/*5.Display the surnames of passengers who have not booked a flight from Pune to
    Bangalore*/
select surname
```

```
from passenger
where email not in(
        select email
        from flight_booking
        where flight_no in (
                select flight_no
                from flights
                where place_from = 'Pune'
                    and place_to = 'Bangalore'
            )
    );

/*6. Display the Passenger details only if they have booked flights on 21st July
    2021. (Use Exists)*/
select *
from passenger
where exists (
        select email
        from flight_booking
        where flight_no in(
                select flight_no
                from flights
                where departure_date = '2021-07-27'
            )
    );
/*--7.Display the Flight-wise total time duration of flights if the duration is
    more than 8 hours (Hint : Date function,Aggregation,Grouping)*/

select flight_no, timediff(f.arrival_time, f.departure_time ) from flights as f
    where timediff(f.arrival_time, f.departure_time ) > "8:00:00" group by
    flight_no;

/*8.Display the Airplane-wise average seating capacity for any airline*/
select name,
    avg(capacity)
from airplane
group by name;

/*9.Display the total number of flights which are booked and travelling to London
    airport.*/
select count(b.flight_no) as total
from flight_booking b,
    flights f
where f.place_to = 'London';

/*10. Create a view having information about flight_no,airplane_no,capacity.*/
create view flightinfo as
select f.flight_no,
    a.reg_no,
    a.capacity
from flights f,
    airplane a
where a.reg_no = f.reg_no;

select * from flightinfo;
```

8 Tables

```
MariaDB [dbms_lab]> select * from passenger;
+-----------------+------------+----------+
| email           | first_name | surname  |
+-----------------+------------+----------+
| beck@gmail.com | Gwen       | Beck     |
| joe@gmail.com  | Joe        | Goldberg |
| love@gmail.com | Love       | Quinn    |
+-----------------+------------+----------+
3 rows in set (0.001 sec)

MariaDB [dbms_lab]> select * from airplane;
+--------+----------+----------+---------------+
| reg_no | model_no | capacity | name          |
+--------+----------+----------+---------------+
|    111 |        7 |      180 | Qatar Airways |
|    112 |        7 |      169 | Qatar Airways |
|    113 |        8 |      200 | Qatar Airways |
|    221 |       17 |      150 | Emirates      |
|    222 |       17 |      140 | Emirates      |
|    223 |       18 |      175 | Emirates      |
|    333 |       27 |      200 | Air India     |
|    334 |       27 |      150 | Air India     |
|    335 |       28 |      175 | Air India     |
+--------+----------+----------+---------------+
9 rows in set (0.001 sec)

MariaDB [dbms_lab]> select * from airline;
+---------------+
| name          |
+---------------+
| Air India     |
| Emirates      |
| Qatar Airways |
+---------------+
3 rows in set (0.001 sec)

MariaDB [dbms_lab]> select * from flights;
+--
    ---------+------------+-----------+-----------------+----------------+--------------+--------

| flight_no | place_from | place_to  | departure_date | departure_time |
    arrival_date | arrival_time | reg_no |
+--
    ---------+------------+-----------+-----------------+----------------+--------------+--------

|     12345 | Mumbai     | London    | 2021-07-27     | 12:12:12       |
    2021-07-28   | 23:59:56     |    111 |
|     23456 | London     | Pune      | 2021-07-27     | 12:12:12       |
    2021-07-28   | 22:59:56     |    333 |
|     67890 | Pune       | Bangalore | 2021-07-27     | 12:12:12       |
    2021-07-27   | 16:59:56     |    221 |
+--
    ---------+------------+-----------+-----------------+----------------+--------------+--------

3 rows in set (0.001 sec)

MariaDB [dbms_lab]> select * from flight_booking;
+-----------------+-----------+----------+
| email           | flight_no | no_seats |
```

```
+----------------+-----------+----------+
| beck@gmail.com |     67890 |        6 |
| joe@gmail.com  |     23456 |        2 |
| love@gmail.com |     12345 |        6 |
+----------------+-----------+----------+
3 rows in set (0.001 sec)
```

9 Queries

```
MariaDB [dbms_lab]> -- QUERIES
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> -- 1. Display the Passenger email ,Flight_no,Source and
    Destination Airport Names for all flights
MariaDB [dbms_lab]> -- booked
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> select b.email, b.flight_no, f.place_from, f.place_to from
    flight_booking as b inner join flights as f where b.flight_no = f.flight_no;
+----------------+-----------+------------+-----------+
| email          | flight_no | place_from | place_to  |
+----------------+-----------+------------+-----------+
| love@gmail.com |     12345 | Mumbai     | London    |
| joe@gmail.com  |     23456 | London     | Pune      |
| beck@gmail.com |     67890 | Pune       | Bangalore |
+----------------+-----------+------------+-----------+
3 rows in set (0.001 sec)

MariaDB [dbms_lab]>
MariaDB [dbms_lab]> -- 2.
MariaDB [dbms_lab]> --  Display the flight and passenger details for the flights
    booked having Departure Date between
MariaDB [dbms_lab]> -- 23-08-2021 and 25-08-2021
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> select * from flights as f, passenger as p, flight_booking as
    b where b.email = p.email and b.flight_no = f.flight_no and departure_date
    between "2021-07-27" and "2021-07-28";
+--
    ---------+------------+-----------+----------------+----------------+--------------+-------

| flight_no | place_from | place_to  | departure_date | departure_time |
    arrival_date | arrival_time | reg_no | email          | first_name | surname  |
     email          | flight_no | no_seats |
+--
    ---------+------------+-----------+----------------+----------------+--------------+-------

|     67890 | Pune       | Bangalore | 2021-07-27     | 12:12:12       |
    2021-07-27   | 16:59:56     |    221 | beck@gmail.com | Gwen       | Beck     |
     beck@gmail.com |     67890 |        6 |
|     23456 | London     | Pune      | 2021-07-27     | 12:12:12       |
    2021-07-28   | 22:59:56     |    333 | joe@gmail.com  | Joe        | Goldberg |
     joe@gmail.com  |     23456 |        2 |
|     12345 | Mumbai     | London    | 2021-07-27     | 12:12:12       |
    2021-07-28   | 23:59:56     |    111 | love@gmail.com | Love       | Quinn    |
     love@gmail.com |     12345 |        6 |
+--
    ---------+------------+-----------+----------------+----------------+--------------+-------

3 rows in set (0.004 sec)
```

```
30
31 MariaDB [dbms_lab]>
32 MariaDB [dbms_lab]> -- 3.
33 MariaDB [dbms_lab]> --  Display the top 5 airplanes that participated in Flights
       from Mumbai to London based on the
34 MariaDB [dbms_lab]> -- airplane capacity
35 MariaDB [dbms_lab]>
36 MariaDB [dbms_lab]> select * from airplane as a, flights as f where a.reg_no = f.
       reg_no and f.place_from = "Mumbai" and f.place_to = "London" order by a.
       capacity desc limit 5;
37 +--
       ------+----------+----------+---------------+-----------+------------+----------+-----------
38 | reg_no | model_no | capacity | name          | flight_no | place_from | place_to
        | departure_date | departure_time | arrival_date | arrival_time | reg_no |
39 +--
       ------+----------+----------+---------------+-----------+------------+----------+-----------
40 |    111 |        7 |      180 | Qatar Airways |     12345 | Mumbai     | London
        | 2021-07-27     | 12:12:12       | 2021-07-28   | 23:59:56     |    111 |
41 +--
       ------+----------+----------+---------------+-----------+------------+----------+-----------
42 1 row in set (0.002 sec)
43
44 MariaDB [dbms_lab]>
45 MariaDB [dbms_lab]> -- 4.Display the passenger first names who have booked the
       no_of seats smaller than the average
46 MariaDB [dbms_lab]> -- number of seats booked by all passengers for the arrival
       airport:New Delhi
47 MariaDB [dbms_lab]>
48 MariaDB [dbms_lab]> select * from passenger as p, flight_booking as b, flights as
       f where p.email = b.email and f.flight_no = b.flight_no and f.place_to = "New
       Delhi" and b.no_seats < all(select avg(no_seats) from flight_booking);
49 Empty set (0.002 sec)
50
51 MariaDB [dbms_lab]>
52 MariaDB [dbms_lab]>
53 MariaDB [dbms_lab]> /*5.Display the surnames of passengers who have not booked a
       flight from Pune to Bangalore*/
54 MariaDB [dbms_lab]> select surname
55     -> from passenger
56     -> where email not in(
57     ->         select email
58     ->         from flight_booking
59     ->         where flight_no in (
60     ->                 select flight_no
61     ->                 from flights
62     ->                 where place_from = 'Pune'
63     ->                     and place_to = 'Bangalore'
64     ->             )
65     ->     );
66 +----------+
67 | surname  |
68 +----------+
69 | Goldberg |
70 | Quinn    |
71 +----------+
72 2 rows in set (0.003 sec)
```

```
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> /*6. Display the Passenger details only if they have booked
    flights on 21st July 2021. (Use Exists)*/
MariaDB [dbms_lab]> select *
    -> from passenger
    -> where exists (
    ->          select email
    ->          from flight_booking
    ->          where flight_no in(
    ->                  select flight_no
    ->                  from flights
    ->                  where departure_date = '2021-07-27'
    ->              )
    ->     );
+----------------+------------+----------+
| email          | first_name | surname  |
+----------------+------------+----------+
| beck@gmail.com | Gwen       | Beck     |
| joe@gmail.com  | Joe        | Goldberg |
| love@gmail.com | Love       | Quinn    |
+----------------+------------+----------+
3 rows in set (0.001 sec)

MariaDB [dbms_lab]> /*--7.Display the Flight-wise total time duration of flights
    if the duration is more than 8 hours (Hint : Date function,Aggregation,Grouping
    )*/
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> select flight_no, timediff(f.arrival_time, f.departure_time )
    from flights as f where timediff(f.arrival_time, f.departure_time ) > "8:00:00"
     group by flight_no;
+-----------+-----------------------------------------------+
| flight_no | timediff(f.arrival_time, f.departure_time ) |
+-----------+-----------------------------------------------+
|     12345 | 11:47:44                                      |
|     23456 | 10:47:44                                      |
+-----------+-----------------------------------------------+
2 rows in set (0.001 sec)

MariaDB [dbms_lab]>
MariaDB [dbms_lab]> /*8.Display the Airplane-wise average seating capacity for any
     airline*/
MariaDB [dbms_lab]> select name,
    ->     avg(capacity)
    -> from airplane
    -> group by name;
+---------------+---------------+
| name          | avg(capacity) |
+---------------+---------------+
| Air India     |      175.0000 |
| Emirates      |      155.0000 |
| Qatar Airways |      183.0000 |
+---------------+---------------+
3 rows in set (0.001 sec)

MariaDB [dbms_lab]>
MariaDB [dbms_lab]> /*9.Display the total number of flights which are booked and
    travelling to London airport.*/
MariaDB [dbms_lab]> select count(b.flight_no) as total
```

```
    -> from flight_booking b,
    ->      flights f
    -> where f.place_to = 'London';
+-------+
| total |
+-------+
|     3 |
+-------+
1 row in set (0.000 sec)

MariaDB [dbms_lab]>
MariaDB [dbms_lab]> /*10. Create a view having information about flight_no,
    airplane_no,capacity.*/
MariaDB [dbms_lab]> create view flightinfo as
    -> select f.flight_no,
    ->      a.reg_no,
    ->      a.capacity
    -> from flights f,
    ->      airplane a
    -> where a.reg_no = f.reg_no;
ERROR 1050 (42S01): Table 'flightinfo' already exists
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> select * from flightinfo;
+-----------+--------+----------+
| flight_no | reg_no | capacity |
+-----------+--------+----------+
|     12345 |    111 |      180 |
|     67890 |    221 |      150 |
|     23456 |    333 |      200 |
+-----------+--------+----------+
3 rows in set (0.001 sec)
```

10 Conclusion

Thus, we have learned Subqueries commands thoroughly.

11 FAQ

1. *Explain following types of subqueries*

   - *Single-row subquery*
   - *Multiple-row subquery*
   - *Multiple-column subquery*

   The Given Subqueries can be explained as such:

   - *Single-row subquery* : A subquery that returns a single row is called a single-row subquery. A special case is the scalar subquery, which returns a single row with one column. Scalar subqueries are acceptable (and often very useful) in virtually any situation where you could use a literal value, a constant, or an expression.
   - *Multiple-row subquery* : A subquery that returns multiple rows is called a multiple-row subquery. These queries are commonly used to generate result sets that will be passed to a DML or SELECT statement for further processing. Both single-row and multiple-row subqueries will be evaluated once, before the parent query is run. Single- and multiple-row subqueries can be used in the WHERE and HAVING clauses of the parent query, but there are restrictions on the legal comparison operators
   - *Multiple-column subquery* : A subquery that returns multiple columns is called a multiple-column subquery. It is also called a correlated subquery. A correlated subquery has a more complex method of execution than single- and multiple-row subqueries and is potentially much more powerful. If a subquery references columns in the parent query, then its result will be dependent on the parent query. This makes it impossible to evaluate the subquery before evaluating the parent query.

2. *When subquery is used?*

   Subqueries are queires within queries.

   They are used for the following purposes:

   - *To find the value that is to be used in the outer query*
   - *To find the rows that are to be used in the outer query*
   - *To find the columns that are to be used in the outer query*

3. *Explain SQL SubQueries with ALL, ANY, EXISTS, SOME, With UPDATE*

   Sql subqueries can be used with the following operators, ALL, ANY, EXISTS, SOME, IN, NOT IN, =, <>, >, <, >=, <=, !=, IS NULL, IS NOT NULL, BETWEEN, NOT BETWEEN, LIKE, NOT LIKE, etc.

   They can be explained as follows:

   - *ALL* - Returns true if the subquery returns a value that is less than or equal to all the values in the subquery.

- *ANY* - Returns true if the subquery returns a value that is less than or equal to any of the values in the subquery.
- *EXISTS* - Returns true if the subquery returns any rows.
- *SOME* - Returns true if the subquery returns a value that is less than or equal to any of the values in the subquery.
- *IN* - Returns true if the subquery returns a value that is equal to any of the values in the subquery.
- *NOT IN* - Returns true if the subquery returns a value that is not equal to any of the values in the subquery

4. *How to get groupwise data from a table. What is use of Having Clause*

   Groupwise data can be obtained using the GROUP BY clause. The HAVING clause is used to filter the groups.

   An Example query would be

```
SELECT column_name, aggregate_function(column_name) from table_name having
    aggregate_function(column_name) operator value group by column_name;
```

5. *What is 'having' clause and when to use it?*

   The Having clause is used to filter the groups. It is used with the GROUP BY clause.

6. *How to display data from View. Are the views updatable? Explain*

   Data from a view can be displayed using the SELECT statement.

   Views are not updatable. They are read-only. This is because the view is not a physical table. It is a virtual table.

MIT World Peace University

Database Management Systems
Second Year B. Tech, Semester 4

Stored Procedures and Functions in PL/SQL

Assignment No. 6

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

April 30, 2023

Contents

1 Aim

Write PLSQL Procedures and Function for given problem statements

2 Objectives

1. To study PLSQL procedures and functions

3 Problem Statement

Create tables and solve given queries using Subqueries

4 Theory

4.1 PL/SQL

PL/SQL is Oracle's procedural extension to industry-standard SQL. PL/SQL naturally, efficiently, and safely extends SQL for developers. Its primary strength is in providing a server-side, stored procedural language that is easy-to-use, seamless with SQL, robust, portable, and secure.

4.2 Stored Procedures

A stored procedure is a subroutine available to applications that access a relational database system. Stored procedures (sometimes called a proc, sproc, StoPro, or SP) are actually stored in the database data dictionary.

4.3 Functions

A function is a subroutine available to applications that access a relational database management system (RDBMS). Such applications can include multiple programming languages, APIs, and communication protocols. Functions are also called procedures, modules, or subroutines. Functions are stored in and callable from the database.

4.4 Difference between Stored Procedures and Functions

The following are the key differences between a stored procedure and a function.

1. A function must return a value but in Stored Procedure it is optional.
2. A function can have only input parameters for it whereas a stored procedure can have input/output parameters .
3. Functions can be called from Procedure whereas Procedures cannot be called from a Function.
4. Exception can be handled by try-catch block in a Procedure whereas try-catch block cannot be used in a Function.
5. We can go for Transaction Management in Procedure whereas we can't go in Function.
6. We can use a procedure in a select statement but we can't use Function in a select statement.

7. We can't use a function in DML (insert, update, delete) statement. But we can use a procedure in DML statement.

5 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Draw.io for Drawing the ER diagram.

6 Input

Given Database from the Problem Statement for the Assignment for our batch. (A1 PA 20)

7 Creation and Insertion of Values in the Tables

8 Queries

```
-- Procedures and Functions

-- BATCH 2 EXERCISE 1 - PROCEDURES

-- product(prod_id, prod_name, qty_on_hand)
-- Order(cust_id, prod_id, order_date, qty_ordered)
-- Customer(cust_id, cust_name, phone, address)

-- Write a stored procedure to take the cust_id, prod_id and qty_ordered as
-- input. Procedure should check if the order for a particular customer can be
-- fulfilled and if yes then insert the new order and update the product
-- quantity on hand. Display appropriate message if the order cannot be
-- fulfilled. Output parameter must have updated value of the qty_on_hand

-- 1. Create database and tables

create database if not exists lab_procedures;
use lab_procedures;

CREATE TABLE `product` (
  `product_id` INT NOT NULL AUTO_INCREMENT,
  `prod_name` varchar(255) NOT NULL,
  `qty_on_hand` INT(255) NOT NULL,
  PRIMARY KEY (`product_id`)
);

CREATE TABLE `customer` (
  `cust_id` INT NOT NULL AUTO_INCREMENT,
  `cust_name` varchar(255) NOT NULL,
  `phone` VARCHAR(255) NOT NULL,
  `address` VARCHAR(255) NOT NULL,
  PRIMARY KEY (`cust_id`)
);

CREATE TABLE `order_details` (
  `cust_id` INT NOT NULL,
```

```
  `product_id` INT NOT NULL,
  `order_date` DATE NOT NULL,
  `qty_order` INT NOT NULL
);

INSERT INTO product (prod_name, qty_on_hand) VALUES
    ('Product A', 50),
    ('Product B', 100),
    ('Product C', 75);

INSERT INTO customer (cust_name, phone, address) VALUES
    ('John Smith', '123-456-7890', '123 Main St'),
    ('Jane Doe', '555-555-1212', '456 Oak Ave'),
    ('Bob Johnson', '555-123-4567', '789 Elm St');

INSERT INTO order_details values
    (1, 1, '2023-04-25', 10),
    (1, 2, '2023-04-26', 5),
    (2, 3, '2023-04-27', 8),
    (3, 1, '2023-04-26', 15),
    (3, 3, '2023-04-27', 3);


ALTER TABLE `order_details` ADD CONSTRAINT `order_fk0` FOREIGN KEY (`cust_id`)
    REFERENCES `customer`(`cust_id`);

ALTER TABLE `order_details` ADD CONSTRAINT `order_fk1` FOREIGN KEY (`product_id`)
    REFERENCES `product`(`product_id`);


-- Creating Procedure

DELIMITER $$
CREATE PROCEDURE Fulfill_Order_proc3 (
    IN p_cust_id INT,
    IN p_prod_id INT,
    IN p_qty_ordered INT,
    OUT p_qty_on_hand INT
)


BEGIN
    DECLARE v_qty_on_hand INT;

    -- Get the current quantity on hand for the product
    SELECT qty_on_hand INTO v_qty_on_hand
    FROM product
    WHERE product_id = p_prod_id;

    -- Check if the order can be fulfilled
    IF v_qty_on_hand >= p_qty_ordered THEN
        -- Insert the new order
        INSERT INTO order_details (cust_id, product_id, order_date, qty_order)
        VALUES (p_cust_id, p_prod_id, CURDATE(), qty_order);

        -- Update the quantity on hand for the product
        UPDATE product
        SET qty_on_hand = qty_on_hand - p_qty_ordered
        WHERE product_id = p_prod_id;
```

```
94
95          -- Set the output parameter to the updated quantity on hand
96          SELECT qty_on_hand INTO p_qty_on_hand
97          FROM product
98          WHERE product_id = p_prod_id;
99
100         -- Display a success message
101         SELECT CONCAT('Order fulfilled. New quantity on hand for product ',
        p_prod_id, ' is ', p_qty_on_hand) AS message;
102     ELSE
103         -- Display an error message
104         SELECT CONCAT('Order cannot be fulfilled. Only ', v_qty_on_hand, ' units
        of product ', p_prod_id, ' are available.') AS message;
105     END IF;
106 END$$
107 DELIMITER ;
108
109
110 -- Calling Procedure
111
112 -- Get the current quantity on hand for product 1
113 SELECT qty_on_hand FROM product WHERE product_id = 1;
114
115 -- Attempt to place an order for 20 units of product 1 for customer 1
116 CALL Fulfill_Order_proc3(1, 1, 20, @qty_on_hand);
117
118 -- Get the error message returned by the stored procedure
119 SELECT message FROM (SELECT @p_qty_on_hand AS message) AS result;
120
121 -- BATCH 2 EXERCISE 2 - FUNCTIONS
122
123 -- Write a function to find total quantity ordered by taking
124 -- cust_id and prod_id as input parameter
125 -- Also write a code to call the function
126
127 -- Creating Function
128
129 DELIMITER $$
130 CREATE FUNCTION Total_Qty_Ordered2(cust_id INT, prod_id INT)
131 RETURNS INT deterministic
132 BEGIN
133     DECLARE total_qty INT;
134     SELECT SUM(qty_order) INTO total_qty
135     FROM order_details
136     WHERE cust_id = cust_id AND prod_id = product_id;
137     RETURN total_qty;
138 END $$
139
140 DELIMITER ;
141
142 -- Calling Function
143
144 SELECT Total_Qty_Ordered2(1, 1) AS total_qty;
```

9 Outputs

```
1 MariaDB [(none)]> -- Procedures and Functions
2 MariaDB [(none)]>
```

```
MariaDB [(none)]> -- BATCH 2 EXERCISE 1 - PROCEDURES
MariaDB [(none)]>
MariaDB [(none)]> -- product(prod_id, prod_name, qty_on_hand)
MariaDB [(none)]> -- Order(cust_id, prod_id, order_date, qty_ordered)
MariaDB [(none)]> -- Customer(cust_id, cust_name, phone, address)
MariaDB [(none)]>
MariaDB [(none)]> -- Write a stored procedure to take the cust_id, prod_id and
    qty_ordered as
MariaDB [(none)]> -- input. Procedure should check if the order for a particular
    customer can be
MariaDB [(none)]> -- fulfilled and if yes then insert the new order and update the
     product
MariaDB [(none)]> -- quantity on hand. Display appropriate message if the order
    cannot be
MariaDB [(none)]> -- fulfilled. Output parameter must have updated value of the
    qty_on_hand
MariaDB [(none)]>
MariaDB [(none)]> -- 1. Create database and tables
MariaDB [(none)]>
MariaDB [(none)]> create database if not exists lab_procedures;
Query OK, 1 row affected (0.000 sec)

MariaDB [(none)]> use lab_procedures;
Database changed
MariaDB [lab_procedures]>
MariaDB [lab_procedures]> CREATE TABLE `product` (
    -> `product_id` INT NOT NULL AUTO_INCREMENT,
    -> `prod_name` varchar(255) NOT NULL,
    -> `qty_on_hand` INT(255) NOT NULL,
    -> PRIMARY KEY (`product_id`)
    -> );
Query OK, 0 rows affected (0.004 sec)

MariaDB [lab_procedures]>
MariaDB [lab_procedures]> CREATE TABLE `customer` (
    -> `cust_id` INT NOT NULL AUTO_INCREMENT,
    -> `cust_name` varchar(255) NOT NULL,
    -> `phone` VARCHAR(255) NOT NULL,
    -> `address` VARCHAR(255) NOT NULL,
    -> PRIMARY KEY (`cust_id`)
    -> );
Query OK, 0 rows affected (0.004 sec)

MariaDB [lab_procedures]>
MariaDB [lab_procedures]> CREATE TABLE `order_details` (
    -> `cust_id` INT NOT NULL,
    -> `product_id` INT NOT NULL,
    -> `order_date` DATE NOT NULL,
    -> `qty_order` INT NOT NULL
    -> );
Query OK, 0 rows affected (0.003 sec)

MariaDB [lab_procedures]>
MariaDB [lab_procedures]> INSERT INTO product (prod_name, qty_on_hand) VALUES
    ->     ('Product A', 50),
    ->     ('Product B', 100),
    ->     ('Product C', 75);
Query OK, 3 rows affected (0.001 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```
57
58 MariaDB [lab_procedures]>
59 MariaDB [lab_procedures]> INSERT INTO customer (cust_name, phone, address) VALUES
60     ->     ('John Smith', '123-456-7890', '123 Main St'),
61     ->     ('Jane Doe', '555-555-1212', '456 Oak Ave'),
62     ->     ('Bob Johnson', '555-123-4567', '789 Elm St');
63 Query OK, 3 rows affected (0.001 sec)
64 Records: 3  Duplicates: 0  Warnings: 0
65
66 MariaDB [lab_procedures]>
67 MariaDB [lab_procedures]> INSERT INTO order_details values
68     ->     (1, 1, '2023-04-25', 10),
69     ->     (1, 2, '2023-04-26', 5),
70     ->     (2, 3, '2023-04-27', 8),
71     ->     (3, 1, '2023-04-26', 15),
72     ->     (3, 3, '2023-04-27', 3);
73 Query OK, 5 rows affected (0.001 sec)
74 Records: 5  Duplicates: 0  Warnings: 0
75
76 MariaDB [lab_procedures]>
77 MariaDB [lab_procedures]>
78 MariaDB [lab_procedures]> ALTER TABLE `order_details` ADD CONSTRAINT `order_fk0`
    FOREIGN KEY (`cust_id`) REFERENCES `customer`(`cust_id`);
79 Query OK, 5 rows affected (0.011 sec)
80 Records: 5  Duplicates: 0  Warnings: 0
81
82 MariaDB [lab_procedures]>
83 MariaDB [lab_procedures]> ALTER TABLE `order_details` ADD CONSTRAINT `order_fk1`
    FOREIGN KEY (`product_id`) REFERENCES `product`(`product_id`);
84 Query OK, 5 rows affected (0.011 sec)
85 Records: 5  Duplicates: 0  Warnings: 0
86
87 MariaDB [lab_procedures]>
88 MariaDB [lab_procedures]>
89 MariaDB [lab_procedures]> -- Creating Procedure
90 MariaDB [lab_procedures]>
91 MariaDB [lab_procedures]> DELIMITER $$
92 MariaDB [lab_procedures]> CREATE PROCEDURE Fulfill_Order_proc3 (
93     ->     IN p_cust_id INT,
94     ->     IN p_prod_id INT,
95     ->     IN p_qty_ordered INT,
96     ->     OUT p_qty_on_hand INT
97     -> )
98     ->
99     ->
100    -> BEGIN
101    ->     DECLARE v_qty_on_hand INT;
102    ->
103    ->     -- Get the current quantity on hand for the product
104    ->     SELECT qty_on_hand INTO v_qty_on_hand
105    ->     FROM product
106    ->     WHERE product_id = p_prod_id;
107    ->
108    ->     -- Check if the order can be fulfilled
109    ->     IF v_qty_on_hand >= p_qty_ordered THEN
110    ->         -- Insert the new order
111    ->         INSERT INTO order_details (cust_id, product_id, order_date,
    qty_order)
112    ->         VALUES (p_cust_id, p_prod_id, CURDATE(), qty_order);
```

```
    ->
    ->         -- Update the quantity on hand for the product
    ->         UPDATE product
    ->         SET qty_on_hand = qty_on_hand - p_qty_ordered
    ->         WHERE product_id = p_prod_id;
    ->
    ->         -- Set the output parameter to the updated quantity on hand
    ->         SELECT qty_on_hand INTO p_qty_on_hand
    ->         FROM product
    ->         WHERE product_id = p_prod_id;
    ->
    ->         -- Display a success message
    ->         SELECT CONCAT('Order fulfilled. New quantity on hand for product ',
     p_prod_id, ' is ', p_qty_on_hand) AS message;
    ->     ELSE
    ->         -- Display an error message
    ->         SELECT CONCAT('Order cannot be fulfilled. Only ', v_qty_on_hand, '
    units of product ', p_prod_id, ' are available.') AS message;
    ->     END IF;
    -> END$$
Query OK, 0 rows affected (0.001 sec)

MariaDB [lab_procedures]> DELIMITER ;
MariaDB [lab_procedures]>
MariaDB [lab_procedures]>
MariaDB [lab_procedures]> -- Calling Procedure
MariaDB [lab_procedures]>
MariaDB [lab_procedures]> -- Get the current quantity on hand for product 1
MariaDB [lab_procedures]> SELECT qty_on_hand FROM product WHERE product_id = 1;
+-------------+
| qty_on_hand |
+-------------+
|          50 |
+-------------+
1 row in set (0.000 sec)

MariaDB [lab_procedures]>
MariaDB [lab_procedures]> -- Attempt to place an order for 20 units of product 1
    for customer 1
MariaDB [lab_procedures]> CALL Fulfill_Order_proc3(1, 1, 20, @qty_on_hand);
+-------------------------------------------------------------+
| message                                                     |
+-------------------------------------------------------------+
| Order fulfilled. New quantity on hand for product 1 is 30 |
+-------------------------------------------------------------+
1 row in set (0.002 sec)

Query OK, 4 rows affected (0.002 sec)

MariaDB [lab_procedures]>
MariaDB [lab_procedures]> -- Get the error message returned by the stored
    procedure
MariaDB [lab_procedures]> SELECT message FROM (SELECT @p_qty_on_hand AS message)
    AS result;
+---------+
| message |
+---------+
| NULL    |
+---------+
```

```
1 row in set (0.000 sec)

MariaDB [lab_procedures]>
MariaDB [lab_procedures]> -- BATCH 2 EXERCISE 2 - FUNCTIONS
MariaDB [lab_procedures]>
MariaDB [lab_procedures]> -- Write a function to find total quantity ordered by
    taking
MariaDB [lab_procedures]> -- cust_id and prod_id as input parameter
MariaDB [lab_procedures]> -- Also write a code to call the function
MariaDB [lab_procedures]>
MariaDB [lab_procedures]> -- Creating Function
MariaDB [lab_procedures]>
MariaDB [lab_procedures]> DELIMITER $$
MariaDB [lab_procedures]> CREATE FUNCTION Total_Qty_Ordered2(cust_id INT, prod_id
    INT)
    -> RETURNS INT deterministic
    -> BEGIN
    ->     DECLARE total_qty INT;
    ->     SELECT SUM(qty_order) INTO total_qty
    ->     FROM order_details
    ->     WHERE cust_id = cust_id AND prod_id = product_id;
    ->     RETURN total_qty;
    -> END $$
Query OK, 0 rows affected (0.003 sec)

MariaDB [lab_procedures]>
MariaDB [lab_procedures]> DELIMITER ;
MariaDB [lab_procedures]>
MariaDB [lab_procedures]> -- Calling Function
MariaDB [lab_procedures]>
MariaDB [lab_procedures]> SELECT Total_Qty_Ordered2(1, 1) AS total_qty;
+-----------+
| total_qty |
+-----------+
|        25 |
+-----------+
1 row in set (0.001 sec)

MariaDB [lab_procedures]> select * from product;
+------------+-----------+-------------+
| product_id | prod_name | qty_on_hand |
+------------+-----------+-------------+
|          1 | Product A |          30 |
|          2 | Product B |         100 |
|          3 | Product C |          75 |
+------------+-----------+-------------+
3 rows in set (0.001 sec)

MariaDB [lab_procedures]> select * from customer;
+---------+-------------+--------------+-------------+
| cust_id | cust_name   | phone        | address     |
+---------+-------------+--------------+-------------+
|       1 | John Smith  | 123-456-7890 | 123 Main St |
|       2 | Jane Doe    | 555-555-1212 | 456 Oak Ave |
|       3 | Bob Johnson | 555-123-4567 | 789 Elm St  |
+---------+-------------+--------------+-------------+
3 rows in set (0.000 sec)

MariaDB [lab_procedures]> select * from order_details;
```

```
+---------+------------+------------+-----------+
| cust_id | product_id | order_date | qty_order |
+---------+------------+------------+-----------+
|       1 |          1 | 2023-04-25 |        10 |
|       1 |          2 | 2023-04-26 |         5 |
|       2 |          3 | 2023-04-27 |         8 |
|       3 |          1 | 2023-04-26 |        15 |
|       3 |          3 | 2023-04-27 |         3 |
|       1 |          1 | 2023-04-27 |         0 |
+---------+------------+------------+-----------+
6 rows in set (0.001 sec)

MariaDB [lab_procedures]>
```

10 Conclusion

Thus, we have learned PLSQL Database Programming.

11 FAQ

1. **What is PLSQL? What are Applications of PLSQL?**

   PL/SQL (Procedural Language/Structured Query Language) is a procedural extension of SQL that is used to write and execute program units such as stored procedures, functions, and triggers in Oracle Database. It offers a wide range of features such as exception handling, variable declaration, loops, conditional statements, and more, which make it a powerful tool for developing complex database applications. The applications of PL/SQL include building database applications, automating database administration tasks, creating reports, and more.

2. **What is deterministic in stored funcions mean?**

   In PL/SQL, a stored function is deterministic if it always returns the same result for the same set of input parameters. This means that if the input parameters for a deterministic function remain the same, the function will always return the same output. This property is important because it enables developers to write functions that can be used in a wider range of contexts and can be optimized by the Oracle database engine.

3. **Explain Various Input Parameter in PLSQL**

   Various input parameters in PL/SQL include:

   (a) IN: This parameter is used to pass values into a stored procedure or function. The values of the IN parameter are read-only within the program unit and cannot be modified.

   (b) OUT: This parameter is used to return values from a stored procedure or function. The values of the OUT parameter are write-only within the program unit and must be assigned a value before the program unit completes.

   (c) IN OUT: This parameter is used to pass values into a stored procedure or function and return values back to the calling program. The values of the IN OUT parameter can be read and modified within the program unit.

   (d) DEFAULT: This parameter is used to provide a default value for a parameter. If a value is not specified for the parameter when the program unit is called, the default value will be used instead.

MIT World Peace University

Database Management Systems
Second Year B. Tech, Semester 4

Create Triggers using PL/SQL

Assignment No. 7

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

April 30, 2023

Contents

1 Aim

Write PL/SQL Triggers for the creation of insert trigger, delete trigger and update trigger on the given problem statements.

2 Objectives

1. To study and use Triggers using MySQL PL/SQL block

3 Problem Statement

Create tables and solve given queries

4 Theory

4.1 Triggers in PL/SQL

A trigger is a named PL/SQL unit that is stored in the database and can be invoked repeatedly. A trigger automatically executes whenever an event associated with a table occurs. There are two types of triggers based on the which level it is triggered. They are:

1. Row Level Trigger
2. Statement Level Trigger
3. Database Level Trigger
4. Instead of Trigger
5. DDL Trigger
6. System Trigger
7. Compound Trigger

4.2 Advantages of Triggers

1. Triggers can be used to enforce complex business rules.
2. Triggers can be used to enforce complex integrity constraints.
3. Triggers can be used to propagate data to other tables.
4. Triggers can be used to log all changes to a table.
5. Triggers can be used to prevent invalid transactions.
6. Triggers can be used to notify external applications of database changes.

4.3 Disadvantages of Triggers

1. Triggers are hidden within the database and can be difficult to find.
2. Triggers are not visible in the SQL source code.
3. Triggers can be difficult to debug.
4. Triggers can cause performance issues.
5. Triggers can cause infinite loops.
6. Triggers can cause locking issues.
7. Triggers can cause cascading failures.
8. Triggers can cause unpredictable results.
9. Triggers can cause security issues.

4.4 Usage of Triggers

Usages of Triggers

1. Auditing
2. Data Integrity
3. Data Validation
4. Notification
5. Replication
6. Security
7. Synchronization

4.5 Difference between Stored Procedure and Trigger

1. Triggers are invoked automatically in response to the associated DML statement.
2. Stored procedures are invoked explicitly by users or applications.
3. Triggers are attached to tables and are implicitly invoked.
4. Stored procedures are stand-alone and are explicitly invoked.
5. Triggers execute implicitly in response to DML statements on the table with which they are associated.
6. Stored procedures execute explicitly when they are invoked by a user or application.
7. Triggers execute under the security privileges of the owner of the trigger.
8. Stored procedures execute under the security privileges of the user who invokes them.

9. Triggers are attached to tables and are implicitly invoked.
10. Stored procedures are stand-alone and are explicitly invoked.
11. Triggers execute implicitly in response to DML statements on the table with which they are associated.
12. Stored procedures execute explicitly when they are invoked by a user or application.
13. Triggers execute under the security privileges of the owner of the trigger.
14. Stored procedures execute under the security privileges of the user who invokes them.

Syntax of Triggers:

```
CREATE TRIGGER Trigger_Name
[ BEFORE | AFTER ]  [ Insert | Update | Delete]
ON [Table_Name]
[ FOR EACH ROW | FOR EACH COLUMN ]
AS
Set of SQL Statement
```

4.6 Types of Triggers

1. Row Level Trigger
   - Row level trigger is triggered when a row is inserted, updated or deleted from a table.
   - Row level trigger is declared at row level.
   - Row level trigger is used to perform an action when a DML event (INSERT, UPDATE, DELETE) occurs.
   - Row level trigger can be used to enforce complex business rules or integrity constraints.
2. Statement Level Trigger
   - Statement level trigger is triggered when a DML event (INSERT, UPDATE, DELETE) occurs.
   - Statement level trigger is declared at statement level.
   - Statement level trigger is used to perform an action when a DML event (INSERT, UPDATE, DELETE) occurs.
   - Statement level trigger can be used to enforce complex business rules or integrity constraints.
3. Database Level Trigger
   - Database level trigger is triggered when a DDL event (CREATE, ALTER, DROP, RENAME, TRUNCATE) occurs.
   - Database level trigger is declared at database level.
   - Database level trigger is used to perform an action when a DDL event (CREATE, ALTER, DROP, RENAME, TRUNCATE) occurs.
   - Database level trigger can be used to enforce complex business rules or integrity constraints.

4. Instead of Trigger
    - Instead of trigger is triggered when a DML event (INSERT, UPDATE, DELETE) occurs.
    - Instead of trigger is declared at view level.
    - Instead of trigger is used to perform an action when a DML event (INSERT, UPDATE, DELETE) occurs.
    - Instead of trigger can be used to enforce complex business rules or integrity constraints.
5. DDL Trigger
    - DDL trigger is triggered when a DDL event (CREATE, ALTER, DROP, RENAME, TRUNCATE) occurs.
    - DDL trigger is declared at database level.
    - DDL trigger is used to perform an action when a DDL event (CREATE, ALTER,

4.7 Trigger Level

Trigger Levels are used to specify when the trigger should be fired. There are two types of trigger levels. They are:

1. Before Trigger
    - Before trigger is triggered before the triggering statement is executed.
    - Before trigger is declared using BEFORE keyword.
    - Before trigger is used to perform an action before the triggering statement is executed.
    - Before trigger can be used to enforce complex business rules or integrity constraints.
2. After Trigger
    - After trigger is triggered after the triggering statement is executed.
    - After trigger is declared using AFTER keyword.
    - After trigger is used to perform an action after the triggering statement is executed.
    - After trigger can be used to enforce complex business rules or integrity constraints.

4.8 Trigger Events

Trigger Events are used to specify when the trigger should be fired. There are three types of trigger events. They are:

1. Insert Trigger
    - Insert trigger is triggered when an insert event occurs.
    - Insert trigger is declared using INSERT keyword.
    - Insert trigger is used to perform an action when an insert event occurs.
    - Insert trigger can be used to enforce complex business rules or integrity constraints.
2. Update Trigger

- Update trigger is triggered when an update event occurs.
- Update trigger is declared using UPDATE keyword.
- Update trigger is used to perform an action when an update event occurs.
- Update trigger can be used to enforce complex business rules or integrity constraints.

3. Delete Trigger

- Delete trigger is triggered when a delete event occurs.
- Delete trigger is declared using DELETE keyword.
- Delete trigger is used to perform an action when a delete event occurs.
- Delete trigger can be used to enforce complex business rules or integrity constraints.

An SQL example for trigger events is given below:

```
CREATE TRIGGER trigger_name
    BEFORE INSERT ON table_name
    FOR EACH ROW
    BEGIN
        -- trigger body
    END;
```

4.9 NEW and OLD Clause /Trigger Variables

Clauses are used to specify the columns of the table. There are two types of clauses. They are:

1. NEW Clause

- NEW clause is used to specify the columns of the table that are affected by the triggering statement.
- NEW clause is used in INSERT and UPDATE triggers.
- NEW clause is used to specify the columns of the table that are affected by the triggering statement.
- NEW clause is used in INSERT and UPDATE triggers.

2. OLD Clause

- OLD clause is used to specify the columns of the table that are affected by the triggering statement.
- OLD clause is used in UPDATE and DELETE triggers.
- OLD clause is used to specify the columns of the table that are affected by the triggering statement.
- OLD clause is used in UPDATE and DELETE triggers.

An SQL Example for NEW and OLD clause would be:

```
    CREATE TRIGGER trigger_name
    BEFORE INSERT ON table_name
    FOR EACH ROW
    BEGIN
        -- trigger body
        IF NEW.column_name = 'value' THEN
            -- trigger body
        END IF;
    END;
```

4.10 Dropping Triggers

Dropping Triggers is used to drop the trigger from the database. An SQL Example for this would be:

```
    DROP TRIGGER trigger_name;
```

5 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Draw.io for Drawing the ER diagram.

6 Input

Given Database from the Problem Statement for the Assignment for our batch. (A1 PA 20)

7 Queries

```
-- Assignment 7 Triggers

-- Consider the following relational schema: BOOK (Isbn, Title,
-- SoldCopies) WRITING (Isbn, Name) AUTHOR (Name, SoldCopies)

-- Define a set of triggers for keeping SoldCopies in AUTHOR updated
-- with respect to: updates on SoldCopies in BOOK insertion of new tuples
-- in the WRITING relation

-- Create Database and Tables

CREATE database if not exists store;
use store;
CREATE TABLE BOOK (
    Isbn VARCHAR(10) PRIMARY KEY,
    Title VARCHAR(100),
    SoldCopies INT
);

CREATE TABLE WRITING (
    Isbn VARCHAR(10),
    Name VARCHAR(50),
    PRIMARY KEY (Isbn, Name),
    FOREIGN KEY (Isbn) REFERENCES BOOK(Isbn)
);

CREATE TABLE AUTHOR (
    Name VARCHAR(50) PRIMARY KEY,
    SoldCopies INT
);

CREATE TABLE Customer (
    cust_id INT PRIMARY KEY,
    Principal_amount DOUBLE,
    Rate_of_interest DOUBLE,
    Years INT
```

```
);


INSERT INTO BOOK (Isbn, Title, SoldCopies)
VALUES ('9783161', 'Crime and Punishment', 500);

INSERT INTO WRITING (Isbn, Name)
VALUES ('9783161', 'Fyodor Dostoevsky');

INSERT INTO AUTHOR (Name, SoldCopies)
VALUES ('Fyodor Dostoevsky', 500);

INSERT INTO Customer (cust_id, Principal_amount, Rate_of_interest, Years)
VALUES (1, 1000, 0.05, 5);

-- Create Triggers

DELIMITER //
CREATE TRIGGER update_author_soldcopies
AFTER UPDATE ON BOOK
FOR EACH ROW
BEGIN
    IF NEW.SoldCopies != OLD.SoldCopies THEN
        UPDATE AUTHOR
        SET SoldCopies = SoldCopies + (NEW.SoldCopies - OLD.SoldCopies) WHERE Name
     IN (SELECT Name FROM WRITING WHERE Isbn = NEW.Isbn);
    END IF;
END//
DELIMITER ;

delimiter $$
CREATE TRIGGER insert_author_soldcopies
AFTER INSERT ON WRITING
FOR EACH ROW
BEGIN
    UPDATE AUTHOR
    SET SoldCopies = SoldCopies + (SELECT SoldCopies FROM BOOK WHERE Isbn = NEW.
    Isbn)
    WHERE Name = NEW.Name;
END $$
delimiter ;
```

8 Outputs

```


MariaDB [(none)]>
MariaDB [(none)]> CREATE database if not exists store;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> use store;
Database changed
MariaDB [store]> CREATE TABLE BOOK (
    ->     Isbn VARCHAR(10) PRIMARY KEY,
    ->     Title VARCHAR(100),
    ->     SoldCopies INT
    -> );
Query OK, 0 rows affected (0.006 sec)
```

```
MariaDB [store]>
MariaDB [store]> CREATE TABLE WRITING (
    ->     Isbn VARCHAR(10),
    ->     Name VARCHAR(50),
    ->     PRIMARY KEY (Isbn, Name),
    ->     FOREIGN KEY (Isbn) REFERENCES BOOK(Isbn)
    -> );
Query OK, 0 rows affected (0.006 sec)

MariaDB [store]>
MariaDB [store]> CREATE TABLE AUTHOR (
    ->     Name VARCHAR(50) PRIMARY KEY,
    ->     SoldCopies INT
    -> );
Query OK, 0 rows affected (0.005 sec)

MariaDB [store]>
MariaDB [store]> CREATE TABLE Customer (
    ->     cust_id INT PRIMARY KEY,
    ->     Principal_amount DOUBLE,
    ->     Rate_of_interest DOUBLE,
    ->     Years INT
    -> );
Query OK, 0 rows affected (0.005 sec)

MariaDB [store]>
MariaDB [store]>
MariaDB [store]> INSERT INTO BOOK (Isbn, Title, SoldCopies)
    -> VALUES ('9783161', 'Crime and Punishment', 500);
Query OK, 1 row affected (0.001 sec)

MariaDB [store]>
MariaDB [store]> INSERT INTO WRITING (Isbn, Name)
    -> VALUES ('9783161', 'Fyodor Dostoevsky');
Query OK, 1 row affected (0.001 sec)

MariaDB [store]>
MariaDB [store]> INSERT INTO AUTHOR (Name, SoldCopies)
    -> VALUES ('Fyodor Dostoevsky', 500);
Query OK, 1 row affected (0.001 sec)

MariaDB [store]>
MariaDB [store]> INSERT INTO Customer (cust_id, Principal_amount, Rate_of_interest
    , Years)
    -> VALUES (1, 1000, 0.05, 5);
Query OK, 1 row affected (0.001 sec)

MariaDB [store]>
MariaDB [store]> -- Create Triggers
MariaDB [store]>
MariaDB [store]> DELIMITER //
MariaDB [store]> CREATE TRIGGER update_author_soldcopies
    -> AFTER UPDATE ON BOOK
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.SoldCopies != OLD.SoldCopies THEN
    ->         UPDATE AUTHOR
    ->         SET SoldCopies = SoldCopies + (NEW.SoldCopies - OLD.SoldCopies)
```

```
    WHERE Name IN (SELECT Name FROM WRITING WHERE Isbn = NEW.Isbn);
    ->     END IF;
    -> END//
Query OK, 0 rows affected (0.003 sec)

MariaDB [store]> DELIMITER ;
MariaDB [store]>
MariaDB [store]> delimiter $$
MariaDB [store]> CREATE TRIGGER insert_author_soldcopies
    -> AFTER INSERT ON WRITING
    -> FOR EACH ROW
    -> BEGIN
    ->     UPDATE AUTHOR
    ->     SET SoldCopies = SoldCopies + (SELECT SoldCopies FROM BOOK WHERE Isbn =
     NEW.Isbn)
    ->     WHERE Name = NEW.Name;
    -> END $$
Query OK, 0 rows affected (0.003 sec)

MariaDB [store]> delimiter ;
MariaDB [store]>
MariaDB [store]> select * from BOOK;
+---------+----------------------+------------+
| Isbn    | Title                | SoldCopies |
+---------+----------------------+------------+
| 9783161 | Crime and Punishment |        500 |
+---------+----------------------+------------+
1 row in set (0.002 sec)

MariaDB [store]> select * from AUTHOR;
+--------------------+------------+
| Name               | SoldCopies |
+--------------------+------------+
| Fyodor Dostoevsky |        500 |
+--------------------+------------+
1 row in set (0.001 sec)

MariaDB [store]> select * from WRITING;
+---------+--------------------+
| Isbn    | Name               |
+---------+--------------------+
| 9783161 | Fyodor Dostoevsky |
+---------+--------------------+
1 row in set (0.001 sec)

MariaDB [store]> select * from Customer;
+---------+------------------+------------------+-------+
| cust_id | Principal_amount | Rate_of_interest | Years |
+---------+------------------+------------------+-------+
|       1 |             1000 |             0.05 |     5 |
+---------+------------------+------------------+-------+
1 row in set (0.001 sec)

-- Test Triggers


MariaDB [store]> select * from AUTHOR;
+--------------------+------------+
| Name               | SoldCopies |
```

```
+--------------------+------------+
| Fyodor Dostoevsky |        500 |
+--------------------+------------+
1 row in set (0.001 sec)

MariaDB [store]> UPDATE BOOK SET SoldCopies = 600 WHERE Isbn = '9783161';
Query OK, 1 row affected (0.004 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [store]> select * from AUTHOR;
+--------------------+------------+
| Name               | SoldCopies |
+--------------------+------------+
| Fyodor Dostoevsky |        600 |
+--------------------+------------+
1 row in set (0.000 sec)
```

9 Conclusion

Thus, we have learned creating and using triggers in SQL. We have also learned the advantages and disadvantages of using triggers in SQL.

10 FAQ

1. **Enlist Advantages of Triggers ?**

   Advantages of Triggers:

   - Data Integrity: Triggers can help to ensure that data in a database remains consistent and accurate, by automatically enforcing data validation rules.
   - Automation: Triggers can automate repetitive or complex database operations, such as updating related records or sending notifications based on specific events.
   - Security: Triggers can be used to implement security policies, such as preventing unauthorized access or monitoring user activity.
   - Performance: Triggers can improve database performance by reducing the number of queries needed to perform certain operations.
   - Audit trail: Triggers can be used to create an audit trail of changes made to data, which can be useful for compliance or troubleshooting purposes.

2. **Enlist Disadvantages of Triggers ?**

   Disadvantages of Triggers:

   - Complexity: Triggers can add complexity to database design and maintenance, making it harder to understand and modify the database schema.
   - Debugging: Debugging triggers can be difficult, as they are executed automatically and can be triggered by a wide range of events.
   - Performance: Triggers can also have a negative impact on database performance, particularly if they are poorly designed or executed frequently.
   - Scalability: Triggers can make it more difficult to scale a database system, as they can increase the number of transactions and the amount of data being processed.

3. **What are the Applications of Triggers.**

   Applications of Triggers:

   - Data validation: Triggers can be used to enforce data validation rules, such as checking that certain fields are not left blank or that certain values fall within a specified range.
   - Data synchronization: Triggers can be used to synchronize data between different tables or databases, ensuring that related records are always up-to-date.
   - Notifications: Triggers can be used to send notifications or alerts based on specific events, such as when a new record is added or an existing record is updated.
   - Security: Triggers can be used to implement security policies, such as preventing unauthorized access or monitoring user activity.
   - Audit trail: Triggers can be used to create an audit trail of changes made to data, which can be useful for compliance or troubleshooting purposes.

MIT World Peace University

Database Management Systems
Second Year B. Tech, Semester 4

Cursors using PL/SQL

Assignment No. 8

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

April 30, 2023

Contents

1 Aim

Write PL/SQL Cursor for the given problem statements

2 Objectives

1. To study and use Cursor using MySQL PL/SQL block

3 Problem Statement

Create tables and solve given queries.

4 Theory

Explain following points

4.1 What is Cursor?

A cursor is a database object that allows a program to retrieve data from a database one row at a time. It acts like a pointer that points to a specific row in a result set returned by a SQL query. Cursors are useful in situations where we need to manipulate data row by row, such as in a loop. They allow us to navigate through the rows of a result set and manipulate them as needed.

Example:

```
DECLARE
    CURSOR c1 IS
        SELECT * FROM employees;
BEGIN
    FOR employee IN c1 LOOP
        DBMS_OUTPUT.PUT_LINE(employee.first_name || ' ' || employee.last_name)
;
    END LOOP;
END;
```

4.2 Why do we need the Cursors?

Cursors are necessary in situations where we need to retrieve data from a database one row at a time and process it. For example, if we want to perform a complex calculation on a large dataset, it may be more efficient to retrieve the data row by row rather than all at once. Cursors enable us to retrieve data one row at a time and process it as needed, which can be particularly useful in situations where we need to perform complex processing on data.

1. Cursors are used to retrieve data from a database one row at a time.
2. Cursors are used to process data row by row.
3. Cursors are used to perform complex calculations on large datasets.

4.3 Different types of Cursors

There are two types of cursors in PL/SQL:

1. Implicit Cursor: This is a cursor that is automatically created by the Oracle database when a SQL statement is executed. It is used for simple, one-time queries that return a small number of rows.
2. Explicit Cursor: This is a cursor that is explicitly declared and defined in a PL/SQL program. It is used for more complex queries that require multiple rows to be returned and processed.

4.4 Drawbacks of Implicit Cursors

Implicit cursors have a few drawbacks, including:

1. They do not provide much control over the result set returned by the query.
2. They do not allow us to manipulate data row by row.
3. They can cause performance issues if used in a loop or if the query returns a large number of rows.

4.5 PL/SQL variables in a Cursor

PL/SQL variables can be used in a cursor to pass values into the query. For example, we can declare a variable in the cursor declaration and use it in the WHERE clause of the query. This can be particularly useful when we need to retrieve a subset of data from a larger dataset.

4.6 Opening a Cursor

Before we can use a cursor, we need to open it using the OPEN statement. This statement prepares the cursor for fetching data from the result set. Once the cursor is open, we can begin fetching data from it.

```
    DECLARE
        CURSOR c1 IS
            SELECT * FROM employees;
    BEGIN
        OPEN c1;
        ...
    END;
```

4.7 Fetching from a Cursor

To retrieve data from a cursor, we use the FETCH statement. This statement retrieves the next row from the result set and makes it available for processing. We can fetch data from the cursor one row at a time, or we can retrieve all of the rows at once.

```
    DECLARE
        CURSOR c1 IS
            SELECT * FROM employees;
        emp_row employees%ROWTYPE;
    BEGIN
        OPEN c1;
```

```
        FETCH c1 INTO emp_row;
        ...
    END;
```

4.8 Closing a Cursor

Once we are done processing the result set, we need to close the cursor using the CLOSE statement. This statement releases the resources used by the cursor and frees up memory. It is important to close cursors when we are done using them to avoid wasting system resources.

```
    DECLARE
        CURSOR c1 IS
            SELECT * FROM employees;
        emp_row employees%ROWTYPE;
    BEGIN
        OPEN c1;
        FETCH c1 INTO emp_row;
        CLOSE c1;
    END;
```

4.9 Cursor attributes

Cursor attributes are properties of a cursor that provide information about its status. Some examples of cursor attributes include %FOUND, which indicates whether the last fetch retrieved a row, and %NOTFOUND, which indicates whether the last fetch did not retrieve a row.

```
  DECLARE
    CURSOR employee_cursor IS
      SELECT * FROM employees;
    emp_row employees%ROWTYPE;
  BEGIN
    OPEN employee_cursor;
    LOOP
      FETCH employee_cursor INTO emp_row;
      EXIT WHEN employee_cursor%NOTFOUND;
      -- process data here
    END LOOP;
    CLOSE employee_cursor;
  END;
```

5 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Draw.io for Drawing the ER diagram.

6 Input

Given Database from the Problem Statement for the Assignment for our batch. (A1 PA 20)

7 Queries

```

-- Use a cursor to calculate compound interest for each customer
-- and insert customer id and simple interest in another table
-- named TEMPLIST.
-- Customer(cust_id, Prinicipal_amount, Rate_of_interest, No. of
-- Years)

-- Create Database and Tables

CREATE database if not exists store;
use store;
CREATE TABLE BOOK (
    Isbn VARCHAR(10) PRIMARY KEY,
    Title VARCHAR(100),
    SoldCopies INT
);

CREATE TABLE WRITING (
    Isbn VARCHAR(10),
    Name VARCHAR(50),
    PRIMARY KEY (Isbn, Name),
    FOREIGN KEY (Isbn) REFERENCES BOOK(Isbn)
);

CREATE TABLE AUTHOR (
    Name VARCHAR(50) PRIMARY KEY,
    SoldCopies INT
);

CREATE TABLE Customer (
    cust_id INT PRIMARY KEY,
    Principal_amount DOUBLE,
    Rate_of_interest DOUBLE,
    Years INT
);


INSERT INTO BOOK (Isbn, Title, SoldCopies)
VALUES ('9783161', 'Crime and Punishment', 500);

INSERT INTO WRITING (Isbn, Name)
VALUES ('9783161', 'Fyodor Dostoevsky');

INSERT INTO AUTHOR (Name, SoldCopies)
VALUES ('Fyodor Dostoevsky', 500);

INSERT INTO Customer (cust_id, Principal_amount, Rate_of_interest, Years)
VALUES (1, 1000, 0.05, 5);


-- Create Cursors
USE store;

DROP TABLE IF EXISTS TEMPLIST;
CREATE TABLE TEMPLIST (
  cust_id INT PRIMARY KEY,
```

```
  Compound_interest DOUBLE
);

DROP PROCEDURE IF EXISTS calculate_interest;
DELIMITER //
CREATE PROCEDURE calculate_interest()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE custid INT;
  DECLARE princ_amt, rate, num_years, interest_amt, temp_amt DOUBLE;
  DECLARE cur CURSOR FOR SELECT cust_id, Principal_amount, Rate_of_interest, Years
     FROM Customer;

  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  OPEN cur;
  loop1: LOOP
    FETCH cur INTO custid, princ_amt, rate, num_years;
    IF done THEN
      LEAVE loop1;
    END IF;

    SET temp_amt = 1 + rate / 12;
    SET interest_amt = princ_amt * POWER(temp_amt, num_years * 12) - princ_amt;

    INSERT INTO TEMPLIST (cust_id, Compound_interest) VALUES (custid, interest_amt
    );
  END LOOP;
  CLOSE cur;
END//
DELIMITER ;

CALL calculate_interest();
SELECT * FROM TEMPLIST;
```

8 Outputs in Execution

```


MariaDB [(none)]>
MariaDB [(none)]> CREATE database if not exists store;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> use store;
Database changed
MariaDB [store]> CREATE TABLE BOOK (
    ->     Isbn VARCHAR(10) PRIMARY KEY,
    ->     Title VARCHAR(100),
    ->     SoldCopies INT
    -> );
Query OK, 0 rows affected (0.006 sec)

MariaDB [store]>
MariaDB [store]> CREATE TABLE WRITING (
    ->     Isbn VARCHAR(10),
    ->     Name VARCHAR(50),
    ->     PRIMARY KEY (Isbn, Name),
    ->     FOREIGN KEY (Isbn) REFERENCES BOOK(Isbn)
```

```
    -> );
Query OK, 0 rows affected (0.006 sec)

MariaDB [store]>
MariaDB [store]> CREATE TABLE AUTHOR (
    ->     Name VARCHAR(50) PRIMARY KEY,
    ->     SoldCopies INT
    -> );
Query OK, 0 rows affected (0.005 sec)

MariaDB [store]>
MariaDB [store]> CREATE TABLE Customer (
    ->     cust_id INT PRIMARY KEY,
    ->     Principal_amount DOUBLE,
    ->     Rate_of_interest DOUBLE,
    ->     Years INT
    -> );
Query OK, 0 rows affected (0.005 sec)

MariaDB [store]>
MariaDB [store]>
MariaDB [store]> INSERT INTO BOOK (Isbn, Title, SoldCopies)
    -> VALUES ('9783161', 'Crime and Punishment', 500);
Query OK, 1 row affected (0.001 sec)

MariaDB [store]>
MariaDB [store]> INSERT INTO WRITING (Isbn, Name)
    -> VALUES ('9783161', 'Fyodor Dostoevsky');
Query OK, 1 row affected (0.001 sec)

MariaDB [store]>
MariaDB [store]> INSERT INTO AUTHOR (Name, SoldCopies)
    -> VALUES ('Fyodor Dostoevsky', 500);
Query OK, 1 row affected (0.001 sec)

MariaDB [store]>
MariaDB [store]> INSERT INTO Customer (cust_id, Principal_amount, Rate_of_interest
    , Years)
    -> VALUES (1, 1000, 0.05, 5);
Query OK, 1 row affected (0.001 sec)

MariaDB [store]>
MariaDB [store]> -- Create Triggers
MariaDB [store]>
MariaDB [store]> DELIMITER //
MariaDB [store]> CREATE TRIGGER update_author_soldcopies
    -> AFTER UPDATE ON BOOK
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.SoldCopies != OLD.SoldCopies THEN
    ->         UPDATE AUTHOR
    ->         SET SoldCopies = SoldCopies + (NEW.SoldCopies - OLD.SoldCopies)
    WHERE Name IN (SELECT Name FROM WRITING WHERE Isbn = NEW.Isbn);
    ->     END IF;
    -> END//
Query OK, 0 rows affected (0.003 sec)

MariaDB [store]> DELIMITER ;
MariaDB [store]>
```

```
MariaDB [store]> delimiter $$
MariaDB [store]> CREATE TRIGGER insert_author_soldcopies
    -> AFTER INSERT ON WRITING
    -> FOR EACH ROW
    -> BEGIN
    ->     UPDATE AUTHOR
    ->     SET SoldCopies = SoldCopies + (SELECT SoldCopies FROM BOOK WHERE Isbn =
     NEW.Isbn)
    ->     WHERE Name = NEW.Name;
    -> END $$
Query OK, 0 rows affected (0.003 sec)

MariaDB [store]> delimiter ;
MariaDB [store]>
MariaDB [store]> select * from BOOK;
+---------+----------------------+------------+
| Isbn    | Title                | SoldCopies |
+---------+----------------------+------------+
| 9783161 | Crime and Punishment |        500 |
+---------+----------------------+------------+
1 row in set (0.002 sec)

MariaDB [store]> select * from AUTHOR;
+-------------------+------------+
| Name              | SoldCopies |
+-------------------+------------+
| Fyodor Dostoevsky |        500 |
+-------------------+------------+
1 row in set (0.001 sec)

MariaDB [store]> select * from WRITING;
+---------+-------------------+
| Isbn    | Name              |
+---------+-------------------+
| 9783161 | Fyodor Dostoevsky |
+---------+-------------------+
1 row in set (0.001 sec)

MariaDB [store]> select * from Customer;
+---------+------------------+------------------+-------+
| cust_id | Principal_amount | Rate_of_interest | Years |
+---------+------------------+------------------+-------+
|       1 |             1000 |             0.05 |     5 |
+---------+------------------+------------------+-------+
1 row in set (0.001 sec)

MariaDB [store]> USE store;
Database changed
MariaDB [store]>
MariaDB [store]> DROP TABLE IF EXISTS TEMPLIST;
Query OK, 0 rows affected (0.007 sec)

MariaDB [store]> CREATE TABLE TEMPLIST (
    ->   cust_id INT PRIMARY KEY,
    ->   Compound_interest DOUBLE
    -> );
Query OK, 0 rows affected (0.007 sec)

MariaDB [store]>
```

```
MariaDB [store]> DROP PROCEDURE IF EXISTS calculate_interest;
Query OK, 0 rows affected (0.003 sec)

MariaDB [store]> DELIMITER //
MariaDB [store]> CREATE PROCEDURE calculate_interest()
    -> BEGIN
    ->   DECLARE done INT DEFAULT FALSE;
    ->   DECLARE custid INT;
    ->   DECLARE princ_amt, rate, num_years, interest_amt, temp_amt DOUBLE;
    ->   DECLARE cur CURSOR FOR SELECT cust_id, Principal_amount, Rate_of_interest
   , Years FROM Customer;
    ->
    ->   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    ->
    ->   OPEN cur;
    ->   loop1: LOOP
    ->     FETCH cur INTO custid, princ_amt, rate, num_years;
    ->     IF done THEN
    ->       LEAVE loop1;
    ->     END IF;
    ->
    ->     SET temp_amt = 1 + rate / 12;
    ->     SET interest_amt = princ_amt * POWER(temp_amt, num_years * 12) -
   princ_amt;
    ->
    ->     INSERT INTO TEMPLIST (cust_id, Compound_interest) VALUES (custid,
   interest_amt);
    ->   END LOOP;
    ->   CLOSE cur;
    -> END//
Query OK, 0 rows affected (0.002 sec)

MariaDB [store]> DELIMITER ;
MariaDB [store]>
MariaDB [store]> CALL calculate_interest();
Query OK, 1 row affected (0.001 sec)

MariaDB [store]> SELECT * FROM TEMPLIST;
+---------+--------------------+
| cust_id | Compound_interest |
+---------+--------------------+
|       1 | 283.3586785035118 |
+---------+--------------------+
1 row in set (0.000 sec)

MariaDB [store]>
```

9 Conclusion

Thus, we have learned creating and using Cursor in SQL. We have also learned about the different types of Cursors and their advantages and disadvantages.

10 FAQ

1. **Enlist Advantages of Cursors ?**
   Advantages of Cursors:
   (a) Flexibility: Cursors offer a flexible way to process database records one at a time, allowing for complex processing logic to be applied to each record individually.
   (b) Control: Cursors give developers more control over the data being processed, allowing for precise manipulation of records and fields.
   (c) Sequential processing: Cursors allow records to be processed in a sequential order, which can be important in cases where records need to be processed in a specific order.
   (d) Record-level operations: Cursors allow for record-level operations such as inserting, updating, and deleting individual records in a result set.
2. **Enlist Disadvantages of Cursors?**

   Disadvantages of Cursors:
   (a) Overhead: Cursors can add overhead to the database server, as they require resources to be allocated to hold the result set and manage the cursor.
   (b) Performance: Cursors can have a negative impact on database performance, particularly if they are used to process large result sets.
   (c) Complexity: Cursors can make code more complex and difficult to understand, particularly if they are nested or used in conjunction with other database operations.
3. **What are the Applications of Cursors.**

   Applications of Cursors:
   (a) Report generation: Cursors can be used to generate reports based on complex queries that require record-level processing.
   (b) Data validation: Cursors can be used to validate data against complex business rules that cannot be easily expressed in a single query.
   (c) Data migration: Cursors can be used to migrate data between databases, particularly when data needs to be transformed or manipulated during the migration process.
   (d) Batch processing: Cursors can be used for batch processing operations, such as updating a large number of records based on a specific criteria.
4. **Why do we need the Cursors?**

   Cursors are needed in situations where processing of individual records in a result set is required. They provide a way to traverse and manipulate data one record at a time, allowing for complex processing logic to be applied to each record individually. Cursors are particularly useful in scenarios where complex business rules need to be applied to individual records, and in cases where record-level operations such as inserting, updating, and deleting are required. However, cursors can also have a negative impact on database performance and should be used judiciously.

MIT World Peace University

Database Management Systems
Second Year B. Tech, Semester 4

Design of XML schema and XQUERY

Assignment No. 9

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

April 30, 2023

Contents

1 Aim

Study XML Query usage and write XQUERY to display the data XQuery FLOWR expression

2 Objectives

1. To study and use XML Query using XQuery FLOWR expression

3 Problem Statement

Create tables and solve given queries.

4 Theory

4.1 What is XML, XML document

XML stands for eXtensible Markup Language. It is a markup language that is used to store and transport data in a structured format. An XML document is a text file that contains data in a structured format using a set of predefined tags, also known as elements. These tags define the structure of the data and allow it to be easily processed and manipulated by software programs.

4.2 Components of XML

The main components of an XML document include elements, attributes, and entities.

1. Elements: They are the building blocks of an XML document and are defined by tags. Elements can contain other elements, text, or both.
2. Attributes: They provide additional information about an element and are defined within the opening tag of an element.
3. Entities: They are used to represent special characters or symbols within an XML document, such as &, <, >, and ".

4.3 XML Databases

XML Databases are databases that store data in XML format. They are designed to provide efficient access and retrieval of data in XML format, as well as support for querying and updating the data. XML databases are often used in conjunction with XML processors to provide a powerful and flexible way to access and manipulate XML data.

XML databases are used to store and manage XML data. They are designed to provide efficient access and retrieval of data in XML format, as well as support for querying and updating the data.

4.4 XML Database Applications

Some examples of XML database applications include:

1. E-commerce systems that store product data in XML format.
2. Financial systems that store transaction data in XML format.
3. Publishing systems that store content in XML format.

4.5 XQUERY

XQuery is a query language that is used to retrieve and manipulate data stored in XML format. It provides a way to search, filter, and transform XML data, and supports a wide range of data types, including text, numbers, and dates. XQuery is often used in conjunction with XML databases to provide a powerful and flexible way to access and manipulate XML data.

4.6 FLOWR Syntax and Example

FLOWR (pronounced "flower") is a query language used in XQuery to construct complex queries. The FLOWR syntax consists of four clauses: for, let, where, and return.

For: This clause is used to specify the source of the data to be queried. Let: This clause is used to define variables that can be used in the query. Where: This clause is used to filter the data based on a set of conditions. Return: This clause is used to specify the data to be returned by the query. Here is an example of a FLOWR query:

```
for $book in //book
let $price := $book/price
where $price > 10
return <result>
<BookTitle>{data($book/title)}</BookTitle>
<BookPrice>{data($price)}</BookPrice>
</result>
```

5 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Draw.io for Drawing the ER diagram.

6 Input

Given Database from the Problem Statement for the Assignment for our batch. (A1 PA 20)

7 Queries

8 Outputs

9 Conclusion

Thus, we have learned creating and using XML Document and XQuery.

10 FAQ

1. **Enlist Advantages of XML over HTML ?**

   - Customizable tags: XML allows for the creation of custom tags, which can be tailored to specific data structures and applications.
   - Data validation: XML provides the ability to define data types and validation rules for data, which helps to ensure data accuracy and consistency.
   - Data interchange: XML is widely used for data interchange between systems, as it provides a standard way to represent data that is platform-independent.
   - Extensibility: XML is extensible, which means that new tags can be added to support new types of data or functionality.
   - Separation of content and presentation: XML separates content from presentation, making it easier to create different views of the same data.

2. **How XML is used to handle data?**

   XML is used to handle data by providing a standard format for representing data structures and their relationships. XML documents consist of elements, attributes, and text, which can be used to represent complex data structures such as hierarchical data or nested records. XML also provides the ability to define data types and validation rules for data, which helps to ensure data accuracy and consistency. XML can be processed uing a wide range of programming languages and technologies, making it a popular choice for data exchange and integration.

3. **Enlist applications of XQuery.**

   (a) Data integration: XQuery can be used to extract, transform, and load data from different sources, making it useful for data integration and migration projects.

   (b) Content management: XQuery can be used to manage and manipulate XML documents in content management systems, enabling more powerful and flexible search capabilities.

   (c) Web services: XQuery can be used to create web services that expose XML data over the internet, allowing for platform-independent data exchange between different systems.

   (d) Business intelligence: XQuery can be used to extract and analyze data from large XML documents, making it useful for business intelligence and reporting applications.

   (e) Data mining: XQuery can be used to extract patterns and trends from large XML datasets, making it useful for data mining and predictive analytics applications.

MIT World Peace University

Database Management Systems
Second Year B. Tech, Semester 4

Basics of JSON

Assignment No. 9

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

May 4, 2023

Contents

1 Aim

Create a json document and write the json query to displayt the data.

2 Objectives

1. To understand the key structure elements of a json file: object names and values.
2. To study the core data types that json files can store including: boolean, numeric and string; hierarchical json structures including: objects, arrays and data elements.
3. To use MySQL JSON data type to store JSON documents in the database and perform CRUD operations.

3 Problem Statement

Create tables and solve given queries.

4 Theory

4.1 Introduction to JSON

JSON, short for JavaScript Object Notation, is a lightweight data interchange format. It is a text format that is easy to read and write and is often used for transmitting data between a server and a web application. JSON has become popular because of its simplicity and flexibility in handling data structures, making it a popular choice for data exchange between systems.

JSON is based on two basic structures: key-value pairs and arrays. Key-value pairs consist of a key and a value, separated by a colon, and are enclosed in curly braces. Arrays are lists of values and are enclosed in square brackets. JSON is human-readable, which means that it can be easily understood by people, and it is also machine-readable, which means that computers can easily parse and generate JSON data.

4.2 Reading and Writing Files in Python

Python provides built-in support for reading and writing files. To open a file in Python, you can use the open() function, which takes two arguments: the file name and the mode in which you want to open the file. The mode can be "r" for reading, "w" for writing, "a" for appending, and "x" for exclusive creation. Once you have opened a file, you can read or write data to it using the file object's methods.

To read data from a file, you can use the read() method, which reads the entire contents of the file as a string. You can also use the readline() method to read a single line at a time or the readlines() method to read all the lines into a list. To write data to a file, you can use the write() method, which writes a string to the file, or the writelines() method, which writes a list of strings to the file.

Syntax for reading a file in Python:

```
with open("file.txt", "r") as f:
    data = f.read()
```

Syntax for writing to a file in Python:

```
with open("file.txt", "w") as f:
    f.write("Hello, world!")
```

4.3 Writing to a JSON File

Python provides a built-in module called json for working with JSON data. To write JSON data to a file, you can use the json.dump() function, which takes two arguments: the data you want to write and the file object you want to write it to. The json.dump() function automatically converts the data to JSON format and writes it to the file.

Syntax for writing JSON data to a file in Python:

```
import json

data = {"name": "John", "age": 30, "city": "New York"}

with open("data.json", "w") as f:
json.dump(data, f)
```

4.4 Introduction to MySQL and JSON data type

MySQL is a popular open-source relational database management system that provides a way to store and manage structured data. MySQL supports a JSON data type, which allows you to store JSON data in a column of a table. The JSON data type is a flexible and efficient way to store and manipulate data that has a variable structure.

When you store JSON data in a MySQL database, you can use SQL to query and manipulate the data just like you would with any other data type. MySQL provides a set of functions and operators for working with JSON data, such as JSONEXTRACT() for extracting data from a JSON object, JSONARRAY() for creating a JSON array, and JSONOBJECT() for creating a JSON object.

Syntax for creating a table with a JSON column in MySQL:

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    json_column JSON,
    ...
);
```

5 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Interpreters Used: Python 3.11

6 Input

1. Read CSV file and convert it into json file with python
2. Read text file and convert it into json file with python
3. Database with JSON Data field

7 Outputs

```python
## read a csv file and convert it into json file
import csv
import json

# Open the CSV file and read its contents
with open('data.csv', newline='') as csv_file:
    csv_reader = csv.DictReader(csv_file)

    # Create an empty list to hold the JSON objects
    json_objects = []

    # Loop through each row in the CSV file
    for row in csv_reader:
        # Convert the row to a JSON object and append it to the list
        json_objects.append(row)

# Write the JSON objects to a file with indentation
with open('data.json', 'w') as json_file:
    json.dump(json_objects, json_file, indent=4)
```

Listing 1: Python Code

```
ID, NAME, AGE, CITY
1, John, 20, New York
2, Mary, 25, Boston
3, Peter, 30, Chicago
4, John, 35, New York
5, Mary, 40, Boston
6, Peter, 45, Chicago
7, John, 50, New York
8, Mary, 55, Boston
9, Peter, 60, Chicago
10, Randy, 65, Los Angeles
```

Listing 2: CSV Input

```
[
    {
        "ID": "1",
        " NAME": " John",
        " AGE": " 20",
        " CITY": " New York"
    },
    {
        "ID": "2",
        " NAME": " Mary",
        " AGE": " 25",
        " CITY": " Boston"
    },
    {
        "ID": "3",
        " NAME": " Peter",
        " AGE": " 30",
        " CITY": " Chicago"
    },
    {
        "ID": "4",
```

```
        " NAME": " John",
        " AGE": " 35",
        " CITY": " New York"
    },
    {
        "ID": "5",
        " NAME": " Mary",
        " AGE": " 40",
        " CITY": " Boston"
    },
    {
        "ID": "6",
        " NAME": " Peter",
        " AGE": " 45",
        " CITY": " Chicago"
    },
    {
        "ID": "7",
        " NAME": " John",
        " AGE": " 50",
        " CITY": " New York"
    },
    {
        "ID": "8",
        " NAME": " Mary",
        " AGE": " 55",
        " CITY": " Boston"
    },
    {
        "ID": "9",
        " NAME": " Peter",
        " AGE": " 60",
        " CITY": " Chicago"
    },
    {
        "ID": "10",
        " NAME": " Randy",
        " AGE": " 65",
        " CITY": " Los Angeles"
    }
]
```


Listing 3: JSON Output

8 Conclusion

Thus, we have learned about the data types, libraries and methods to create JSON documents for storing and retrieving the data.

9 FAQ

1. What is JSON?

   JSON stands for JavaScript Object Notation, which is a lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate. It is often used for transmitting data between a server and a web application.

2. Mention what is the rule for JSON syntax rules? Give an example of JSON object?

   JSON syntax follows a set of rules that determine how data is represented in a JSON object. Some of the key rules for JSON syntax are:

   (a) JSON data is represented as key-value pairs, enclosed in curly braces .

   (b) Each key in a JSON object must be a string enclosed in double quotes, followed by a colon, and then its value.

   (c) Multiple key-value pairs in a JSON object are separated by commas.

   Here's an example of a JSON object that represents information about a person:

```
{
"name": "John Doe",
"age": 30,
"address": {
"street": "123 Main St",
"city": "Anytown",
"state": "CA",
"zip": "12345"
},
"phone": [
{
  "type": "home",
  "number": "555-1234"
},
{
"type": "work",
"number": "555-5678"
}
]
}
```

   In this example, the JSON object represents a person named John Doe, including their name, age, address, and phone numbers.

3. Mention what are the data types supported by JSON?
   JSON supports a limited set of data types, including:

   (a) string: a sequence of characters enclosed in double quotes

   (b) number: an integer or floating-point value

   (c) boolean: either true or false

   (d) null: a special value representing "no value"

   (e) array: an ordered list of values, enclosed in square brackets [ ]

(f) object: an unordered collection of key-value pairs, enclosed in curly braces

4. List out the uses of JSON?

   JSON is a widely used format for data interchange between systems, and it has many applications, including:

   (a) Web APIs: JSON is often used as the data format for APIs that provide access to web services and data.

   (b) Configuration files: JSON can be used to store configuration data for applications and systems.

   (c) Data storage: JSON can be used to store data in databases or file systems.

   (d) Data exchange: JSON can be used to exchange data between different programming languages and platforms.

   (e) Front-end development: JSON can be used to represent data in web applications and JavaScript programs.