GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

PRESIDENCY UNIVERSITY

(Established under the Presidency University Act, 2013 of the Karnataka Act 41 of 2013)

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEMS (CSE2012)

LAB MANUAL

2021-22

INSTRUCTOR- IN-CHARGE : Dr. Nagaraja S R and Mr. Sunil Kumar R M, CSE, PU

Contents:

Summary of MYSQL commands:

Different types of commands in SQL:

- **DDL commands**: - To create a database objects
- **DML commands**: - To manipulate data of a database objects
- **DQL command**: - To retrieve the data from a database.
- **DCL/DTL commands**: - To control the data of a database…

Database-Level:

```
CREATE DATABASE databaseName               -- Create a new database
CREATE DATABASE IF NOT EXISTS databaseName -- Create only if it does not exists
SHOW DATABASES                             -- Show all the databases in this server
USE databaseName                           -- Set the default (current) database
SELECT DATABASE()                          -- Show the default database
SHOW CREATE DATABASE databaseName          -- Show the CREATE DATABASE statement
DROP DATABASE databaseName                 -- Delete the database (irrecoverable!)
DROP DATABASE IF EXISTS databaseName       -- Delete if it exists
```

Table-Level:

```
DROP TABLE [IF EXISTS] tableName, ...
CREATE TABLE [IF NOT EXISTS] tableName (
  columnName columnType columnAttribute, ...
  PRIMARY KEY(columnName),
  FOREIGN KEY (columnNmae) REFERENCES tableName (columnNmae))
SHOW TABLES             -- Show all the tables in the default database
DESCRIBE|DESC tableName   -- Describe the details for a table
ALTER TABLE tableName ...  -- Modify a table, e.g., ADD COLUMN and DROP COLUMN
ALTER TABLE tableName ADD columnDefinition
ALTER TABLE tableName DROP columnName
```

ALTER TABLE *tableName* MODIFY old*columnName newcolumnname data type (to rename a column)*

ALTER TABLE *tableName* ADD FOREIGN KEY (*columnNmae*) REFERENCES *tableName* (*columnNmae*)

ALTER TABLE *tableName* DROP FOREIGN KEY *constraintName*

SHOW CREATE TABLE *tableName* -- Show the CREATE TABLE statement for this *tableName*

Row-Level:

INSERT INTO *tableName* VALUES (*column1Value*, *column2Value*,...) -- Insert on all Columns

INSERT INTO *tableName* VALUES (*column1Value*, *column2Value*,...), ... -- Insert multiple rows

INSERT INTO *tableName* (*column1Name*, ..., *columnNName*) VALUES (*column1Value*, ..., *columnNValue*) -- Insert on selected Columns

DELETE FROM *tableName* WHERE *criteria*

UPDATE *tableName* SET *columnName* = *expr*, ... WHERE *criteria*

SELECT * | *column1Name* AS *alias1*, ..., *columnNName* AS *aliasN*

FROM *tableName*

Data Types in SQL: Data Types in SQL:

i) String Data Types:

a).Fixed Length: char(n) where n is the length of the String

e.g. name char(50)

b).Variable Length: Varchar(n) – n is the maximum length of data possible for the type

All character data has to be enclosed in single quotes during specification

ii). Numeric Data Types:

Decimal : Floating point number

Float: Floating point number

Integer(size):Integer of specified length

iii). Temporal Data Types:

- DATE - format YYYY-MM-DD

- DATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
- YEAR - format YYYY or YY

Experiment No.1: To Study and Implement Data Definition Language(DDL) Commands

DDL (DATA DEFINITION LANGUAGE)

□ CREATE

□ ALTER

□ DROP

□ TRUNCATE

□ RENAME

1. To create a new database called STUDENTDB

```
mysql> CREATE DATABASE STUDENTDB;
Query OK, 1 row affected (0.12 sec)
```

2. To display the databases;

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| studentdb          |
| test               |
+--------------------+
5 rows in set (0.02 sec)
```

3. To use the created database STUDENTDB

```
mysql> USE STUDENTDB;
Database changed
```

Data Definition Language:

Syntax:

```
CREATE TABLE table_name
(Column_name        datatype[(size)],
 Column_name        datatype[(size)],
)
```

Example:

```
CREATE TABLE STUDENT
(SNUM               VARCHAR(10),
SNAME               VARCHAR(20),
MAJOR                  VARCHAR(20),
LEVEL                 VARCHAR(10),
DOB                    DATE);
```

- Creates a table with five columns

```
mysql> CREATE TABLE STUDENT(
   -> SNUM INT PRIMARY KEY,
   -> SNAME VARCHAR(20) NOT NULL,
   -> MAJOR VARCHAR(5) NOT NULL,
   -> LEVEL CHAR(5),
   -> DOB DATE);
Query OK, 0 rows affected (0.13 sec)
```

4. To display/view all the tables in the STUDENTDB database.

```
mysql> SHOW TABLES;
+---------------------+
| Tables_in_studentdb |
+---------------------+
| student             |
+---------------------+
1 row in set (0.00 sec)
```

5. To display the table description/schema/intension/structure of the database

```
mysql> DESC STUDENT;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| SNUM  | int(11)| NO   | PRI | NULL    |       |
| SNAME | varchar(20) | NO   |     | NULL    |       |
| MAJOR | varchar(5)  | NO   |     | NULL    |       |
| LEVEL | char(5)     | YES  |     | NULL    |       |
| DOB   | date        | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
5 rows in set (0.08 sec)
```

6. To add a new column called "SEM" to the existing table STUDENT and whose default value is "4"

```
mysql> ALTER TABLE STUDENT ADD SEM INT DEFAULT 4;
Query OK, 0 rows affected (0.27 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC STUDENT;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| SNUM  | int(11)     | NO   | PRI | NULL    |       |
| SNAME | varchar(20) | NO   |     | NULL    |       |
| MAJOR | varchar(5)  | NO   |     | NULL    |       |
| LEVEL | char(5)     | YES  |     | NULL    |       |
| DOB   | date        | YES  |     | NULL    |       |
| SEM   | int(11)     | YES  |     | 4       |       |
+-------+-------------+------+-----+---------+-------+
6 rows in set (0.13 sec)
```

7. To modify the definition of a column of the existing table STUDENT

```
mysql> ALTER TABLE STUDENT MODIFY MAJOR VARCHAR(20) NULL;
Query OK, 0 rows affected (0.27 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC STUDENT;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| SNUM  | int(11)     | NO   | PRI | NULL    |       |
| SNAME | varchar(20) | NO   |     | NULL    |       |
| MAJOR | varchar(20) | YES  |     | NULL    |       |
| LEVEL | char(5)     | YES  |     | NULL    |       |
| DOB   | date        | YES  |     | NULL    |       |
| SEM   | int(11)     | YES  |     | 4       |       |
+-------+-------------+------+-----+---------+-------+
6 rows in set (0.03 sec)
```

8. To make other column as primary key(unique) in an existing table STUDENT

```
mysql> ALTER TABLE STUDENT MODIFY SNAME VARCHAR(20) UNIQUE;
Query OK, 0 rows affected (0.22 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC STUDENT;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| SNUM  | int(11)     | NO   | PRI | NULL    |       |
| SNAME | varchar(20) | YES  | UNIQUE | NULL    |       |
| MAJOR | varchar(20) | YES  |     | NULL    |       |
| LEVEL | char(5)     | YES  |     | NULL    |       |
| DOB   | date        | YES  |     | NULL    |       |
| SEM   | int(11)     | YES  |     | 4       |       |
+-------+-------------+------+-----+---------+-------+
```

9. To drop a column in an already created table.

```
mysql> ALTER TABLE STUDENT DROP COLUMN  LEVEL;
Query OK, 0 rows affected (0.20 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC STUDENT;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| SNUM  | int(11)     | NO   | PRI | NULL    |       |
| SNAME | varchar(20) | YES  | UNI | NULL    |       |
| MAJOR | varchar(20) | YES  |     | NULL    |       |
| DOB   | date        | YES  |     | NULL    |       |
| SEM   | int(11)     | YES  |     | 4       |       |
+-------+-------------+------+-----+---------+-------+
5 rows in set (0.03 sec)
```

10. Truncating the tables.

Syntax:

Truncate table <tablename>;

```
mysql> TRUNCATE TABLE ENROLL1;
Query OK, 0 rows affected (0.11 sec)
```

11. To delete the definition/schema/description/structure of a table.

Syntax:

Drop table <tablename>;

Example:

```
mysql> DROP TABLE ENROLL1;
Query OK, 0 rows affected (0.08 sec)
```

Experiment No.2: To Study and Implement Data Manipulation Language commands:

Data Manipulation Language:

i)**Insert Statement**:

Allows you to add new records to the Table

Syntax:

insert into table_name[(column_list)] values (value_list)

Example:

INSERT INTO student VALUES (1, ‘Ganesh’, ‘CSE’, ‘JR’, ’2000-05-01’)

INSERT INTO Student (snum, sname, major, level, DOB) VALUES (2, ‘ramesh’, ‘CSE’, ‘SR’, ’2000-07-31’))

- Note: If the columns are not specified as in the first example the data goes in the order specified in the table

ii)**Delete Statement**:

It is used to remove records from a table of the database. The where clause in the syntax is used to restrict the rows deleted from the table otherwise all the rows from the table are deleted.

i) To remove all rows of a table

Syntax: delete from <tablename>;

```
mysql> DELETE FROM STUDENT;
```

ii) removal of a specified row/s

Syntax: DELETE FROM table_name [WHERE Condition]

```
mysql> DELETE FROM STUDENT WHERE  SNAME = ‘Ramesh’
```

- Deletes all the rows where the sname is ‘Ramesh’ keeps all the other rows.

iii) Update Statement:

It is used to make changes to existing rows of the table.

Syntax:

UPDATE table_name

SET column_name1 = value1, column_name2 = value2, …..

[WHERE Condition]

Example: UPDATE STUDENT SET SNAME = ‘Vignesh’, MAJOR = ‘IS’ WHERE snum = 1;

Updating the contents of a table.

i) updating all rows

Syntax: Update <tablename> set <col>=<exp>,<col>=<exp>;

Before updating the snapshot of Student Table

```
mysql> SELECT * FROM STUDENT;

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

| SNUM | SNAME          | MAJOR | DOB        | SEM  |

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

| 1001 | CHETHAN        | CSE   | 2000-11-03 |    4 |
| 1002 | CHETHAN CHAVAN | CSE   | 2000-11-03 |    4 |
+------+----------------+-------+------------+------+
2 rows in set (0.00 sec)
mysql> UPDATE STUDENT SET MAJOR='ISE';
Query OK, 2 rows affected (0.07 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

After updating the snapshot of Student Table

```
mysql> SELECT * FROM STUDENT;
+------+----------------+-------+------------+------+
| SNUM | SNAME          | MAJOR | DOB        | SEM  |
+------+----------------+-------+------------+------+
| 1001 | CHETHAN        | ISE   | 2000-11-03 |    4 |
| 1002 | CHETHAN CHAVAN | ISE   | 2000-11-03 |    4 |
+------+----------------+-------+------------+------+
2 rows in set (0.00 sec)
```

ii) updating seleted records.

Syntax: Update <tablename> set <col>=<exp>, <col>=<exp> where <condition>;

```
mysql> UPDATE STUDENT SET MAJOR='ECE' WHERE SNUM=1001;
```

Experiment No.3: To Study and Implement SQL Constraints

Types of Constraints in SQL

- **NOT NULL** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- **FOREIGN KEY** - Uniquely identifies a row/record in another table
- **CHECK** - Ensures that all values in a column satisfies a specific condition
- **DEFAULT** - Sets a default value for a column when no value is specified

1. To create a new table Faculty with FID auto_increment constraint

```
mysql> CREATE TABLE FACULTY
    -> ( FID INT PRIMARY KEY AUTO_INCREMENT,
    -> FNAME VARCHAR(20) NOT NULL,
    -> ADDRESS VARCHAR(20),
    -> DEPTID INT);
Query OK, 0 rows affected (0.13 sec)


mysql> DESC FACULTY;
+---------+-------------+------+-----+---------+----------------+
| Field   | Type        | Null | Key | Default | Extra          |
+---------+-------------+------+-----+---------+----------------+
| FID     | int(11)     | NO   | PRI | NULL    | auto_increment |
| FNAME   | varchar(20) | NO   |     | NULL    |                |
| ADDRESS | varchar(20) | YES  |     | NULL    |                |
| DEPTID  | int(11)     | YES  |     | NULL    |                |
+---------+-------------+------+-----+---------+----------------+
```

2. To create a relationship table called COURSE with ON DELETE CASCADE Referential Integrity Constraint.

```
mysql> CREATE TABLE COURSE (
    -> CNAME VARCHAR(10) PRIMARY KEY,
    -> MEETS_AT VARCHAR(10),
    -> ROOM VARCHAR(5),
    -> FID INTEGER,
    -> FOREIGN KEY(FID) REFERENCES FACULTY(FID) ON DELETE CASCADE);
mysql> DESC COURSE;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| CNAME    | varchar(10) | NO   | PRI | NULL    |       |
| MEETS_AT | varchar(10) | YES  |     | NULL    |       |
| ROOM     | varchar(5)  | YES  |     | NULL    |       |
| FID      | int(11)     | YES  | MUL | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.03 sec)
```

3. To CREATE a Many to Many relationship table "ENROLL" between STUDENT and COURSE relations with ON DELETE CASCADE

```
mysql> CREATE TABLE ENROLL(
   -> SNUM INTEGER,CNAME VARCHAR(10),
   -> PRIMARY KEY(SNUM,CNAME),
   -> FOREIGN KEY(SNUM) REFERENCES STUDENT(SNUM) ON DELETE CASCADE,
   -> FOREIGN KEY(CNAME) REFERENCES COURSE(CNAME) ON DELETE CASCADE);
```

```
mysql> DESC ENROLL;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| SNUM  | int(11)     | NO   | PRI | 0       |       |
| CNAME | varchar(10) | NO   | PRI |         |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.03 sec)
```

4. To CREATE a Many to Many relationship table "ENROLL" between STUDENT and COURSE relations with ON DELETE SET NULL ON UPDATE CASCADE

```
mysql> CREATE TABLE ENROLL1(
    -> SNUM INTEGER,CNAME VARCHAR(10),
    -> FOREIGN KEY (SNUM) REFERENCES STUDENT(SNUM) ON DELETE SET NULL
ON UPDATE CASCADE,
    -> FOREIGN KEY (CNAME) REFERENCES COURSE(CNAME) ON DELETE SET NULL
ON UPDATE CASCADE);
Query OK, 0 rows affected (0.14 sec)
mysql> DESC ENROLL1;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| SNUM  | int(11)     | YES  | MUL | NULL    |       |
| CNAME | varchar(10) | YES  | MUL | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

i)Entity Integrity Constraint: Primary Key Value cannot be NULL and duplicate.

```
mysql> INSERT INTO STUDENT VALUES (NULL,'CHETHAN','CSE','2000-20-03',4);
ERROR 1048 (23000): Column 'SNUM' cannot be null.
mysql> INSERT INTO STUDENT(SNUM,SNAME,MAJOR,DOB,SEM) VALUES
(1001,'CHETHAN','CSE','2000-05-03',4);
ERROR 1062 (23000): Duplicate entry '1001' for key 'PRIMARY'.
```

ii)DOMAIN Constraint: incorrect date format('YYYY-MM-DD')

```
mysql> INSERT INTO STUDENT VALUES (1002,'CHETHAN','CSE','2000-20-03',4);
ERROR 1292 (22007): Incorrect date value: '2000-20-03' for column 'DOB' at row 1
mysql> INSERT INTO COURSE VALUES('PYTHON','10:05','DGL0123',123);
ERROR 1406 (22001): Data too long for column 'ROOM' at row 1
```

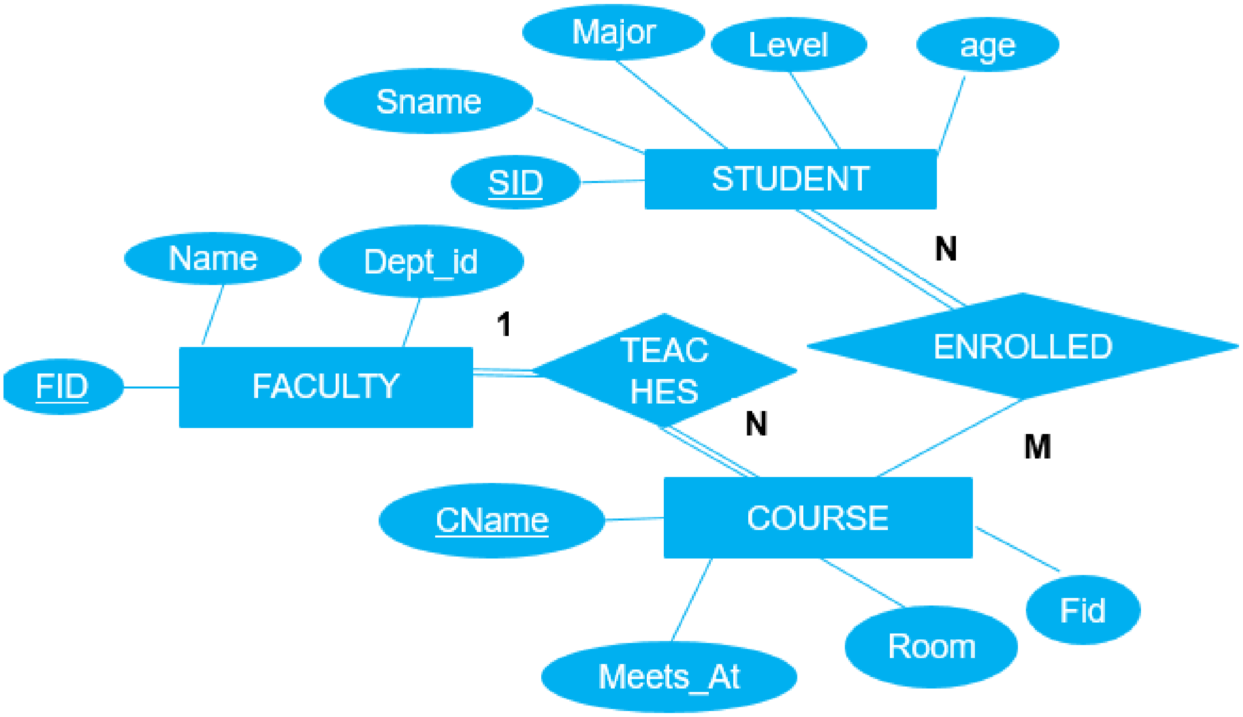
iii)Referential Integrity Constraint:

```
mysql> INSERT INTO COURSE VALUES('PYTHON','10:05','DGL01',123);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails
(`studentdb`.`course`, CONSTRAINT `course_ibfk_1` FOREIGN KEY (`FID`) REFERENCES
`faculty` (`FID`) ON DELETE CASCADE).
```

Experiment No. 4: To study and implement SQL data retrieval using SELECT, FROM and WHERE clause.

Database: Student database

ER Diagram for Student database:



Schema description for Student Database:

- **Student (*sid:* integer, *sname:* varchar(20), *major:* string, address: varchar(20) *level:* string, age *int,DOB date*)**
- **Course (*cname:* varchar(20), *meets_at:* varchar(20), *room:* varchar(20), *fid:* integer)**
- **Enrolled (*sid:* integer, *cname:* varchar(20))**
- **Faculty (fid: integer, *fname:* varchar(20), address: varchar(20), *deptid:* integer)**

Creation of Tables:

STUDENT TABLE

```
CREATE TABLE STUDENT (SID NUMBER PRIMARY KEY, SNAME VARCHAR2
(10), MAJOR VARCHAR2(10), LEVEL VARCHAR2(10), DOB DATE);
```

FACULTY TABLE

```
CREATE TABLE FACULTY (FID int PRIMARY KEY, FNAME VARCHAR(10),
DEPTID int);
```

COURSE TABLE

```
CREATE TABLE COURSE (CNAME VARCHAR(10) PRIMARY KEY, MEETS_AT
VARCHAR(10),ROOM VARCHAR(5), FID int, FOREIGN KEY (FID) REFERENCES
FACULTY (FID) ON DELETE   CASCADE);
```

ENROLLED TABLE

```
CREATE TABLE ENROLLED (SID NUMBER, CNAME VARCHAR2(10),

FOREIGN KEY(SID) REFERENCES STUDENT (SID) ON DELETE CASCADE,

FOREIGN KEY(CNAME) REFERENCES COURSE (CNAME) ON DELETE CASCADE);
```

Insert the values into the tables:

Insert into Student values (…..)

Insert into Faculty values (…)

Insert into Course values (…)

Insert into Enrolled values (…)

SELECT Command:

To retrieve information from a database we can query the databases. SQL SELECT statement is used to select rows and columns from a database/relation

Tables after insertions

```
mysql> SELECT * FROM STUDENT;
+---------+-------------------+----------+-----------+------+
| SID      | SNAME           | MAJOR | LEVEL | AGE |ADDRESS            | DOB|
+-------- +-------------------+-----------+----------+------+-----------------------+-------------+
|  101    | ABHI              | CSE      | JR        |   19   | BANGALORE    |  1980-01-23  |
|  102    | ANIL              | ISE        | JR        |   18   | DAVANAGERE |   1984-02-12  |
|  103    | BHAVYA         | ISE        | SR        |   20   | CHITRADURGA|   1998-03-22 |
|  104    | CHETHAN       | CSE       | JR        |  19  | BELAGAVI       |   1999-04-26 |
|  105    | SURESH        | MECH      | JR        |  18  |HUBLI           |   1970-12-20 |
|  106    | JAYANTH       | CSE       | SR        |  20  | DHARVAD        |    1988-11-16|
+--------+------------------+----------+---------+------+------------------+----------------+

mysql> SELECT * FROM FACULTY;
+------+------------------+----------+-----------+
| FID  | FNAME            | DEPTID |  ADDRESS  |
+------+------------------+----------+--------------
| 1001 | JAMES            |    41      | BANGALORE   |
| 1002 | SUNIL            |    42      |  BELAGAVI      |
| 1003 | SUKRUTH          |    43      | CHITRADURGA |
| 1004 | NARASIMHA |    44      | BANGALORE   |
| 1005 | AFROZ            |    41      | BANGALORE   |
| 1006 | JOHN             |    42      | BANGALORE  |
| 1007 | AHMED            |    45      | BANGALORE |
| 1008 | SUNITHA          |    46      | BANGALORE |
| 1009 | SRIVINAY         |    42      | HUBLI        |
+------+----------------+----------+--------------+

mysql> SELECT * FROM ENROLLED;

| SID | CNAME  |
+------+--------------+
|  101    DBMS           |
|  105 |   TOC             |
|  106 |    BIGDATA      |
|  106 |   AI               |
|  102 |  DATA MINING |
|  103 |  DBMS             |
|  101 | TOC               |
|  104 | TOC                |
|  105 | BIGDATA          |
|  106 | MP  |

mysql> SELECT * FROM COURSE;

| CNAME       | MEETS_AT | ROOM     | FID        |
+---------------+-----------------+-----------+------------+
| DBMS         | 9:00              | NG01 | 1001 |
| TOC            | 10:00            | KG04 | 1008 |
| BIGDATA   | 10:00             | KF04 | 1009 |
| DATA MINING | 10:00        | KF03 | 1009 |
| AI                | 10:00             | NG02 | 1001 |
| MP              | 11:15               | NG03 | 1002 |
| OOPS          | 12:10               | NG04 | 1002 |
| COA            | 01:10               | NG05 | 1003 |
| SE               | 1:05                 | NG06 | 1004 |
| OS               | 2:05                   | KG01 | 1005 |
```

Operators in SQL:

The following are the commonly used operators in SQL

Arithmetic Operators	+, -, *, /
Relational Operators	=, <, >, <=, >=, <>
Logical Operators	OR, AND, NOT

Arithmetic operators are used to perform simple arithmetic operations.

Relational Operators are used when two values are to be compared and

Logical operators are used to connect search conditions in the WHERE Clause in SQL.

SELECT Command:

SELECT count(*) AS “Total Number of Records” FROM student;

Output:

SELECT Roll_no, name, marks+20 FROM student;

Output:

SELECT name, (marks/500)*100 FROM student WHERE Roll_no > 103;

Output:

Eliminating Duplicate/Redundant data:

DISTINCT keyword is used to restrict the duplicate rows from the results of a **SELECT statement**.

e.g. **SELECT DISTINCT name FROM student;**

Conditions based on a range

SQL provides a BETWEEN operator that defines a range of values that the column value must fall for the condition to become true.

e.g. SELECT Roll_no, name FROM student WHERE Roll_no BETWEEN 100 AND 103;

The above command displays Roll_no and name of those students whose Roll_no lies in the range 100 to 103 (both 100 and 103 are included in the range).

Conditions based on Pattern

SQL provides two wild card characters that are used while comparing the strings with LIKE operator.

percent (%) Matches any string

Underscore (_) Matches any one character

e.g **SELECT Roll_no, name, city FROM student WHERE Roll_no LIKE “%3”;**

Displays those records where last digit of Roll_no is 3 and may have any number of characters in front.

Illustration:

1. Viewing data in the tables: - once data has been inserted into a table, the next most logical operation would be to view what has been inserted.

a) all rows and all columns

Syntax: Select <col> to <col n> from tablename;

```
mysql> SELECT * FROM STUDENT;
```

output:

2. Filtering table data: - while viewing data from a table, it is rare that all the data from table will be required each time. Hence, sql must give us a method of filtering out data that is not required data.

a) All columns and all rows:

Syntax: select <col1>,<col2> from <tablename>;

Output:

b) selected rows and all columns:

Syntax:

select * from <tablename> where <condition>;

output:

c) selected columns and selected rows

Syntax: select <col1>,<col2> from <tablename> where<condition>;

Output:

Basic Queries:

1.Add the columns 'Fees' & 'Email' to the STUDENT table with default value '30000' & 'someone@gmail.com'.

Query:

Result:

2. Update the fees & email of students with different values.

Query:

Result:

3. Display the Average Fees of students department-wise.

Query:

Result:

4. Find the names of students having fees between 25000 to 30000.

Query:

Result:

5. **Find the names of students having domain 'gmail.com'**

Query:

Result:

6. Display Names of students in CAPITAL Letters.

Query:

Result:

7. Increase the fees of all students by 10%;

Query:

Result:

Updating on fees column:

a. UPDATE STUDENT SET FEES=FEES*1.1;

b. UPDATE STUDENT SET FEES=FEES-FEES/11;

c. SELECT SNUM, FEES AS OLD_FEE, FEES+2000 AS NEW_FEE FROM STUDENT;

8. Display the details of the student whose email id is missing.(NULL)

Query:

SELECT * FROM STUDENT WHERE EMAILID IS NULL;

Result:

Other than NULL:

SELECT * FROM STUDENT WHERE EMAILID IS NOT NULL;

9. Display the details of the student whose student name starts with letter A.

Query:

Result:

10. Delete the first two records of a student table.

Query:

DELETE FORM STUDENT LIMIT 2;

Result:

Experiment No.5: To study and implement different SQL single row and multiple row functions.

<u>Functions available in SQL:</u>

SQL provide large collection of inbuilt functions also called library functions that can be used directly in SQL statements.

1. **Mathematical functions**
2. **String functions**
3. **Date & Time functions**

1. Mathematical functions:

Some of the commonly used mathematical functions are sum() avg(), count(), min(), max() etc.

Example: **SELECT sum(marks) FROM student;**

displays the sum of all the marks in the table student.

Example: **SELECT min(Roll_no), max(marks) FROM student;**

displays smallest Roll_no and highest marks in the table student.

2. String functions:

These functions are used to deal with the string type values like

ASCII, LOWER, UPPER, LENGTH, LEFT, RIGHT, TRIM, LTRIM, RTRIM etc.

ASCII : Returns the ASCII code value of a character (leftmost character of string).

Syntax: **ASCII(character)**

Example:

SELECT ASCII('a') returns 97

SELECT ASCII('A') returns 65

SELECT ASCII('1') returns 49

SELECT ASCII('ABC') returns 65

Note:

- For Upper character 'A' to 'Z' ASCII value 65 to 90
- For Lower character 'A' to 'Z' ASCII value 97 to 122
- For digit '0' to '9' ASCII value 48 to 57

LOWER : Convert character strings data into lowercase.

Syntax: **LOWER(string)**

SELECT LOWER('STRING FUNCTION')

returns **string function**

UPPER : Convert character strings data into Uppercase.

Syntax**: UPPER(string)**

SELECT UPPER('string function')

returns **STRING FUNCTION**

LEN : Returns the length of the character string.

Syntax: **LENGTH(string)**

SELECT LENGTH('STRING FUNCTION')

returns **15**

LEFT : Returns left part of a string with the specified number of characters counting from left.LEFT function is used to retrieve portions of the string.

Syntax: **LEFT(string,integer)**

SELECT LEFT('STRING FUNCTION', 6)

returns STRING

REVERSE : Returns reverse of a input string.

Syntax: **REVERSE(string)**

SELECT REVERSE('STRING FUNCTION')

returns **NOITCNUF GNIRTS**

SUBSTRING : Returns part of a given string.

SELECT SUBSTRING('STRING FUNCTION', 1, 6)

returns **STRING**

SELECT SUBSTRING('STRING FUNCTION', 8, 8)

returns FUNCTION

SELECT NOW();

Displays Current time and Date

```
mysql> select now();
+---------------------+
| now()               |
+---------------------+
| 2019-03-12 13:36:31 |
+---------------------+
1 row in set (0.08 sec)
```

```
mysql> select sysdate();
```

```
+---------------------+
| sysdate()           |
+---------------------+
| 2019-03-12 13:37:58 |
+---------------------+
1 row in set (0.06 sec)
```

The SQL IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

Ex: Retrieve student names who belongs to major ‘CSE’, ‘IST’, ‘ISE’

```
Select Sname
From Student
Where major in (‘CSE’,’IST’,’ISE’);
```

Not in Operator

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name NOT IN (value1, value2, ...);
```

Ex: Retrieve student names who do not belong to major ‘CSE’, ‘IST’,‘ISE’

```
Select Sname

From Student

Where sec not in (‘CSE’,’IST’,’ISE’);
```

IS NULL and IS NOT NULL

What is a NULL Value?

A field with a NULL value is a field with no value.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

Note: A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

How to Test for NULL Values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

IS NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

Ex: Retrieve student names who has not paid the fees

```
Select Sname

From Student

Where fees IS NULL;
```

IS NOT NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

Ex: Retrieve student names who has paid the fees

```
Select Sname

From Student

Where fees IS NOT NULL;
```

Experiment No.6: To study and implement aggregating Data using Group By Clause, HAVING clause and sort data using Order By clause.

GROUP BY Clause

The GROUP BY clause can be used in a SELECT statement to collect data across multiple records and group the results by one or more columns.

The syntax for the GROUP BY clause is:

```
SELECT column1,column2, ... column_n, aggregate_function (expression)
FROM tables
WHERE conditions
GROUP BY column1, column2, ... column_n;
```

Where

aggregate_function can be a function such as SUM, COUNT, MAX, MIN, AVG etc.

Example:

1. Display the Average Fees of students department-wise.

```
SELECT MAJOR, AVG(FEES)

FROM STUDENT

GROUP BY MAJOR;
```

HAVING Clause

The **HAVING** clause is used in combination with the GROUP BY clause. It can be used in a **SELECT** statement to filter the records that a **GROUP BY** returns.

The syntax for the **HAVING** clause is:

```
SELECT column1, column2, ... column_n, aggregate_function (expression)
FROM tables
WHERE predicates
GROUP BY column1, column2, ... column_n
HAVING condition1 ... condition_n;
```

1. Display sum of fees of student's department wise having count more than two for the same department.

   ```
   SELECT MAJOR, SUM(FEES)

   FROM STUDENT

   GROUP BY MAJOR

   HAVING COUNT(*)>2;
   ```

<u>ORDER BY Clause</u>

ORDER BY clause is used to display the result of a query in a specific order(sorted order).

The sorting can be done in ascending or in descending order. It should be kept in mind that the actual data in the database is not sorted but only the results of the query are displayed in sorted order.

Example: **SELECT name, city FROM student ORDER BY name;**

The above query returns name and city columns of table student sorted by name in increasing/ascending order.

Example: **SELECT * FROM student ORDER BY city DESC;**

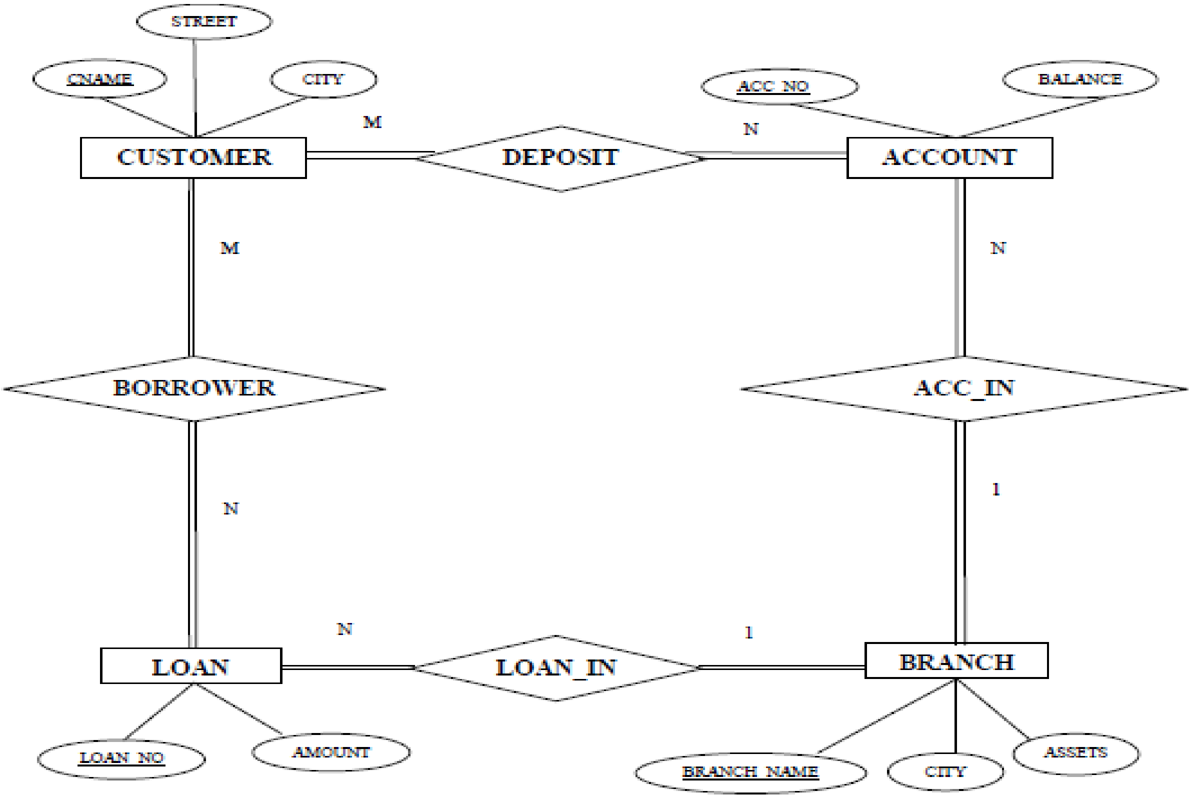
It displays all the records of table student ordered by city in descending order.

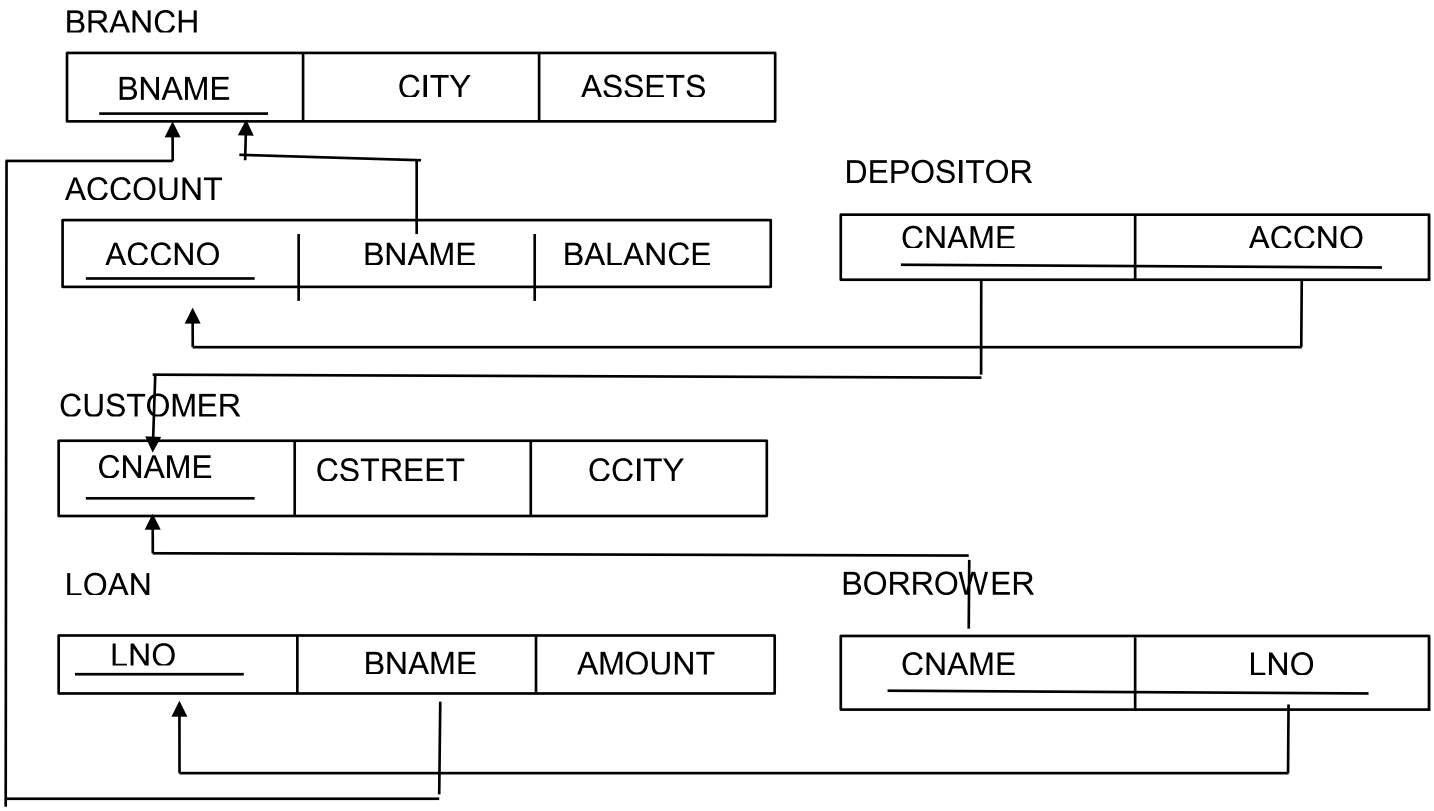
Note:- If order is not specifies that by default the sorting will be performed in ascending order.

BANK DATABASE ER DIAGRAM

SCHEMA DIAGRAM:

- BRANCH(branch-name:string, branch-city:string, assets:real)
- ACCOUNT(accno:int, branch-name:string, balance:real)
- DEPOSITOR(customer-name:string, accno:int)
- CUSTOMER(customer-name:string, customer-street:string, customer-city:string)
- LOAN(loan-number:int, branch-name:**string, amount:real)**
- **BORROWER(customer-name:string, loan-number:int)**





BRANCH

BNAME	CITY	ASSETS

ACCOUNT

ACCNO	BNAME	BALANCE

DEPOSITOR

CNAME	ACCNO

CUSTOMER

CNAME	CSTREET	CCITY

LOAN

LNO	BNAME	AMOUNT

BORROWER

CNAME	LNO

CREATE ALL THE TABLES

BRANCH TABLE

```
CREATE TABLE BRANCH (BR_NAME VARCHAR(20) PRIMARY KEY, BR_CITY
VARCHAR(20), ASSETS REAL);
```

ACCOUNT TABLE

```
CREATE TABLE ACCOUNT (ACCNO INT PRIMARY KEY, BR_NAME VARCHAR(20),
BALANCE REAL, FOREIGN KEY (BR_NAME) REFERENCES BRANCH (BR_NAME) ON
DELETE CASCADE);
```

CUSTOMER TABLE

```
CREATE TABLE CUSTOMER (CUST_NAME VARCHAR(20) PRIMARY KEY,
CUST_STREET VARCHAR (20), CUST_CITY VARCHAR (20));
```

DEPOSITOR TABLE

```
CREATE TABLE DEPOSITOR (CUST_NAME VARCHAR (20), ACCNO INT, PRIMARY
KEY (CUST_NAME, ACCNO), FOREIGN KEY (CUST_NAME) REFERENCES CUSTOMER
(CUST_NAME) ON DELETE CASCADE, FOREIGN KEY (ACCNO) REFERENCES
ACCOUNT (ACCNO) ON DELETE CASCADE);
```

LOAN TABLE

```
CREATE TABLE LOAN (LOAN_NO INT PRIMARY KEY, BR_NAME VARCHAR (20),
AMOUNT REAL, FOREIGN KEY (BR_NAME) REFERENCES BRANCH (BR_NAME) ON
DELETE CASCADE);
```

BORROWER TABLE

```
CREATE TABLE BORROWER (CUST_NAME VARCHAR (20), LOAN_NO INT, PRIMARY
KEY (CUST_NAME, LOAN_NO), FOREIGN KEY (CUST_NAME) REFERENCES
CUSTOMER (CUST_NAME) ON DELETE CASCADE, FOREIGN KEY (LOAN_NO)
REFERENCES LOAN (LOAN_NO) ON DELETE CASCADE);
```

INSERT INTO TABLES

```
INSERT INTO BRANCH VALUES
(‘KORMANGALA’ , ’BENGALURU’ , 20500.3),
(‘SADASHIVANAGAR’ , ’BENGALURU’ , 154329.5),
(‘VITTALNAGAR’ , ’HYDERABAD’ , 350000),
(‘KASTHURINAGAR’ , ’DELHI’ , 125000),
(‘MARUTINAGAR’ , ’HYDERABAD’ , 212351.6),
(‘RAJANKUNTE’ , ’MUMBAI’ , 53535.8);
```

```
INSERT INTO ACCOUNT VALUES
(123456 , ’KORMANGALA’ , 5000),
(123457 , ’SADASHIVANAGAR’ , 35000),
(123458 , ’VITTALNAGAR’ , 60000),
(123459 , ’KASTHURINAGAR’ , 255600),
(123460 , ’VITTALNAGAR’ , 37890),
(123461 , ’MARUTINAGAR’ , 20000),
(123462 , ’SADASHIVANAGAR’ , 40000);

INSERT INTO CUSTOMER VALUES
(‘KAVYA’ , ‘SADASHIVANAGAR’ , ‘BENGALURU’),
(‘ABHAY’ , ‘KAMALANAGAR’ , ‘TUMKUR’),
(‘SHEETAL’ , ‘KASTHURINAGAR’ , ‘BENGALURU’),
(‘KSHAMITHA’ , ‘MARUTILAYOUT’ , ‘TUMKUR’),
(‘LIKITH’ , ‘MADHURANAGAR’ , ‘HYDERABAD’),
(‘SACHIN’ , ‘VITTALNAGAR’ , ‘HYDERABAD’);

INSERT INTO DEPOSITOR VALUES
(‘KAVYA’ , 123457),
(‘ABHAY’ , 123456),
(‘KAVYA’ , 123456),
(‘KSHAMITHA’ , 123458),
(‘KSHAMITHA’ , 123460),
(‘LIKITH’ , 123461),
(‘KAVYA’ , 123462);

INSERT INTO LOAN VALUES
(231 , ’SADASHIVANAGAR’ , 50500.5),
(232 , ’VITTALNAGAR’ , 25000),
(233 , ’MARUTINAGAR’ , 60300.3),
(234 , ’KASTHURINAGAR’ , 45000.7),
(235 , ’KORMANGALA’ , 25534);

INSERT INTO BORROWER VALUES
(‘KAVYA’ , 231),
(‘KSHAMITHA’ , 232),
(‘ABHAY’ , 235),
(‘LIKITH’ , 234),
(‘SACHITH’ , 233);
```

Write and Execute the SQL Queries for the following statements:

1. Find bank accounts with a balance greater than 20000

   ```
   SELECT ACCNO,BALANCE
   FROM ACCOUNT
   WHERE BALANCE>20000;
   ```

2. Order results in increasing

```
SELECT ACCNO,BALANCE
 FROM ACCOUNT
WHERE BALANCE>20000
ORDER BY BALANCE DESC;
```

3. Retrieve a list of all bank branch details, ordered by branch city, with each city's branches listed in reverse order of assets.

```
SELECT BR_NAME, BR_CITY, ASSETS
FROM BRANCH
ORDER BY BR_CITY, ASSETS DESC;
```

4. Find average balance of accounts at "Sadashivanagar" branch.

```
SELECT AVG(BALANCE)
FROM ACCOUNT WHERE BR_NAME= 'SADASHIVANAGAR';
```

5. Find the sum of total account balance of any branch.

```
SELECT branch_name, SUM(balance) AS total_bal
FROM account GROUP BY branch_name;
```

Practice Queries based on Bank Database

1. Find bank accounts with a balance greater than 20000
2. Display results in increasing order of balance
3. Retrieve a list of all bank branch details, ordered by branch city, with each city's branches listed in reverse order of assets
4. Find average balance of accounts at Sadashivanagar branch
5. Find the number of branches that currently have loans
6. Find the number of branches that currently DONT have loans
7. Find branch names of Bengaluru city
8. Find number of accounts present in each branch
9. Find sum of balance of accounts at each branch
10. Find sum of balance of loan accounts at each branch
11. Find the city of a customer with account number 123456
12. Find branch names without account
13. Find the loan amount borrowed by a customer Abhay
14. Find the branch name and balance of a customer kavya with account number 123456
15. Find the loan amount taken by each customer
16. Display the loan details of a customer Kavya
17. Find the city of branch with loan number 100
18. Find the number of accounts of each customer
19. Find customers with an account but not a loan
20. Find all cities with more than two customers living in the city

21. Find all the customers who have at least two accounts at the main branch.
22. Demonstrate how you delete all account tuples at every branch located in a specific city.
23. Find all the customers who have an account at all the branches located in a specific city.
24. Find all the customers with more than one loan
25. Find branches with assets greater than all branches in Bangalore

Solutions:

```
25. SELECT branch_name FROM branch
WHERE assets > ALL (
SELECT assets FROM branch
WHERE br_city='bangalore');
```

```
24. select Cust_Name,count(Loan_no) from borrower group by Cust_Name
having count(Loan_no)>1;
```

```
23. SELECT D.CUST_NAME  FROM BRANCH B, ACCOUNT A, DEPOSITOR D
WHERE B.BR_NAME=A.BR_NAME AND A.ACCNO=D.ACCNO
AND B.BR_CITY=‘BANGALORE’
GROUP BY D.CUST_NAME
HAVING COUNT (DISTINCT A.BR_NAME) =
(SELECT COUNT (*) FROM BRANCH
WHERE BR_CITY=‘BANGALORE’);
```

```
22.DELETE FROM ACCOUNT WHERE BR_NAME IN
(SELECT BR_NAME
FROM BRANCH
WHERE BR_CITY= ‘HYDERABAD’);
```

```
21. SELECT D.CUST_NAME  FROM DEPOSITOR D, ACCOUNT A
WHERE A.ACCNO=D.ACCNO AND A.BR_NAME= 'SADASHIVANAGAR'
GROUP BY D.CUST_NAME  HAVING COUNT (D.CUST_NAME) >=2;
```

```
20. SELECT customer_city, COUNT(*) AS num_customers
FROM customer GROUP BY customer_city
HAVING COUNT(*) > 2;
```

```
19. select distinct(d.cust_name) from depositor d, borrower b
    where d.cust_name not in(select cust_name from borrower);
```

```
18.select count(accno),cust_name from depositor group by cust_name;
```

```
17.  select br_city from branch b, loan l
```

```
        where l.br_name=b.br_name and l.loan_no=100;

16. select cust_name,l.loan_no,amount from loan l, borrower b
      where cust_name='kavya' and b.loan_no=l.loan_no;

15. select cust_name,amount from loan l,borrower b
       where b.loan
```

```
SELECT DISTINCT PNUMBER
FROM PROJECT
WHERE  PNUMBER IN ( SELECT PNUMBER
                              FROM PROJECT, DEPARTMENT,EMPLOYEE
                         WHERE DNUM=DNUMBER AND MGRSSN=SSN AND
                         LNAME='SMITH')
OR
PNUMBER IN (SELECT PNO
                         FROM WORKS_ON,EMPLOYEE
                         WHERE ESSN=SSN AND LNAME='SMITH');
```

```
SELECT DISTINCT ESSN
FROM WORKS_ON
WHERE (PNO,HOURS) IN ( SELECT PNO,HOURS
                                  FROM WORKS_ON
                                 WHERE ESSN=123456789);
```

Experiment No.7: To Study and Implement different types of Set Operation in SQL.

SET OPERATIONS:

SQL has directly incorporated some set operations such as union operation (UNION), set difference (MINUS) and intersection (INTERSECT) operations. The resulting relations of these set operations are sets of tuples; duplicate tuples are eliminated from the result. The set operations apply only to union compatible relations; the two relations must have the same attributes and the attributes must appear in the same order

Illustration of all Set Operations:

1.UNION OPERATION: Display all the cities of branches and customer.

```
SELECT city FROM branch

UNION

SELECT ccity FROM customer;
```

2. INTERSECTION OPERATION: Find the number of branches that currently have loans

```
SELECT bname FROM branch

 WHERE bname  IN(SELECT bname FROM loan);
```

3.MINUS/DIFFERENCE OPEARATION: Find the number of branches that currently DONT have loans

```
SELECT bname FROM branch

 WHERE bname  NOT IN(SELECT bname FROM loan);
```

4.CARTESIAN PRODUCT:

```
select * branch cross join customer;
```

Experiment No.8: To Study and Implement different types of Joins in SQL.

SQL JOIN

SQL Join is used to fetch data from two or more tables, which is joined to appear as single set of data. It is used for combining column from two or more tables by using values common to both tables.

JOIN Keyword is used in SQL queries for joining two or more tables. Minimum required condition for joining table, is **(n-1)** where **n**, is number of tables. A table can also join to itself, which is known as, **Self Join**.

Types of JOIN:

Following are the types of JOIN that we can use in SQL:

- Inner
- Outer

- Left
- Right

Cross JOIN or Cartesian Product

This type of JOIN returns the cartesian product of rows from the tables in Join. It will return a table which consists of records which combines each row from the first table with each row of the second table.

Cross JOIN Syntax is,

SELECT column-name-list

FROM

table-name1 CROSS JOIN table-name2;

INNER Join or EQUI Join

This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the SQL query.

Inner Join Syntax is,

SELECT column-name-list FROM

table-name1 INNER JOIN table-name2

WHERE table-name1.column-name = table-name2.column-name;

Natural JOIN

Natural Join is a type of Inner join which is based on column having same name and same data type present in both the tables to be joined.

The syntax for Natural Join is,

SELECT * FROM

table-name1 NATURAL JOIN table-name2;

OUTER JOIN

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

1. Left Outer Join
2. Right Outer Join
3. Full Outer Join

LEFT Outer Join

The left outer join returns a resultset table with the **matched data** from the two tables and then the remaining rows of the **left** table and null from the **right** table's columns.

Syntax for Left Outer Join is,

SELECT column-name-list FROM

table-name1 LEFT OUTER JOIN table-name2

ON table-name1.column-name = table-name2.column-name;

RIGHT Outer Join

The right outer join returns a resultset table with the **matched data** from the two tables being joined, then the remaining rows of the **right** table and null for the remaining **left** table's columns.

Syntax for Right Outer Join is,

SELECT column-name-list FROM

table-name1 RIGHT OUTER JOIN table-name2

ON table-name1.column-name = table-name2.column-name;

Full Outer Join

The full outer join returns a resultset table with the **matched data** of two table then remaining rows of both **left** table and then the **right** table.

Syntax of Full Outer Join is,

SELECT column-name-list FROM

table-name1 FULL OUTER JOIN table-name2

ON table-name1.column-name = table-name2.column-name;

Illustration:

CREATE TABLE DEPARTMENT(DEPT_ID INT PRIMARY KEY, DEPT_NAME VARCHAR(20));

CREATE TABLE EMPLOYEE(EMP_ID INT PRIMARY KEY, EMP_NAME VARCHAR(20), DEPT_NUM INT, FOREIGN KEY(DEPT_NUM) REFERENCES DEPARTMENT(DEPT_ID) ON DELETE CASCADE);

INSERT INTO DEPARTMENT VALUES(1,'Accounting');

INSERT INTO DEPARTMENT VALUES(2,'Sales');

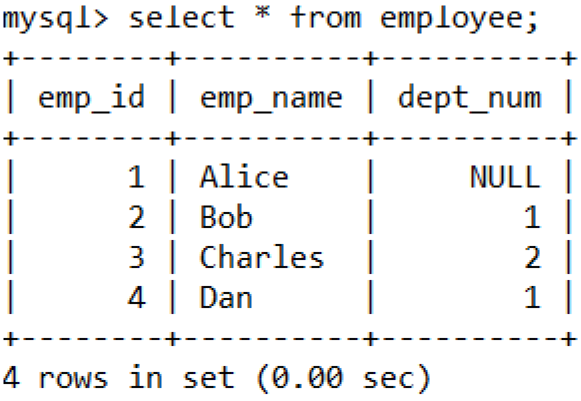
INSERT INTO DEPARTMENT VALUES(3,'Marketing');

INSERT INTO EMPLOYEE VALUES(1,'Alice',NULL);

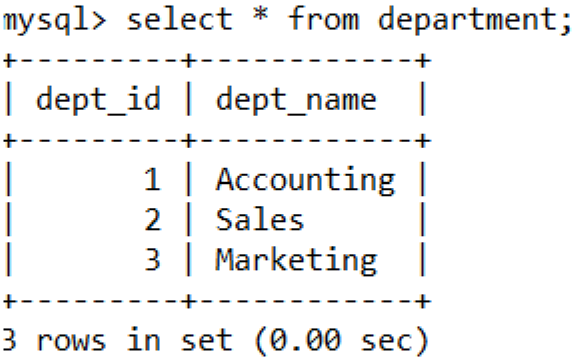
INSERT INTO EMPLOYEE VALUES(2,'Bob',1);

INSERT INTO EMPLOYEE VALUES(3,'Charles',2);

INSERT INTO EMPLOYEE VALUES(4,'Dan',1);



```
mysql> select * from employee;
+--------+----------+----------+
| emp_id | emp_name | dept_num |
+--------+----------+----------+
|      1 | Alice    |     NULL |
|      2 | Bob      |        1 |
|      3 | Charles  |        2 |
|      4 | Dan      |        1 |
+--------+----------+----------+
4 rows in set (0.00 sec)
```

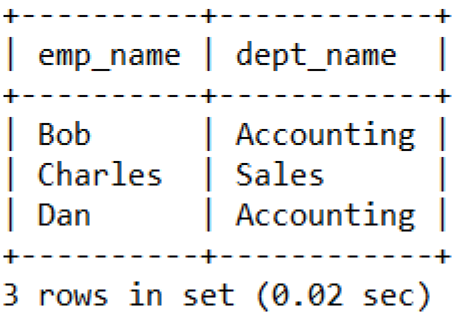


```
mysql> select * from department;
+---------+------------+
| dept_id | dept_name  |
+---------+------------+
|       1 | Accounting |
|       2 | Sales      |
|       3 | Marketing  |
+---------+------------+
3 rows in set (0.00 sec)
```

1. **EQUI JOIN OPERATION: Display employee and their respective branch where employee department number is same as department's department id.**

SELECT EMP_NAME, DEPT_NAME

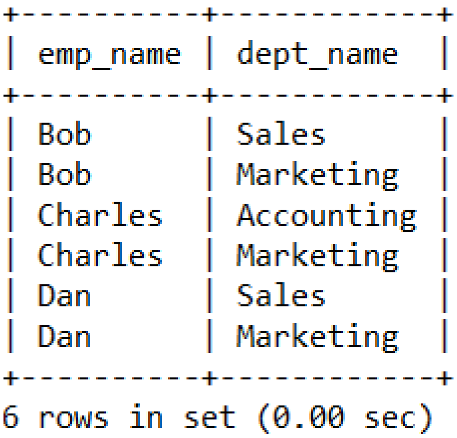
FROM EMPLOYEE E JOIN DEPARTMENT D ON E.DEPT_NUM=D.DEPT_ID;



```
+----------+------------+
| emp_name | dept_name  |
+----------+------------+
| Bob      | Accounting |
| Charles  | Sales      |
| Dan      | Accounting |
+----------+------------+
3 rows in set (0.02 sec)
```

2. **JOIN WITH NOT EQUALITY OPERATOR(NON EQUI JOIN): Display employee and their respective Department where employee department number is not same as department's department id.**

SELECT EMP_NAME, DEPT_NAME FROM EMPLOYEE E JOIN DEPARTMENT D ON E.DEPT_NUM<>D.DEPT_ID;



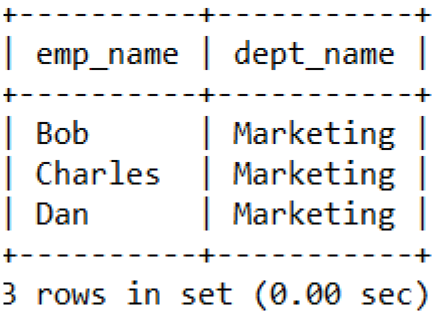
```
+----------+------------+
| emp_name | dept_name  |
+----------+------------+
| Bob      | Sales      |
| Bob      | Marketing  |
| Charles  | Accounting |
| Charles  | Marketing  |
| Dan      | Sales      |
| Dan      | Marketing  |
+----------+------------+
6 rows in set (0.00 sec)
```

3. EQUI JOIN WITH SPECIFIED CONDITION: Display employee and their respective Department where employee department number is not same as department's department id and Department name is MARKETING

SELECT EMP_NAME, DEPT_NAME

FROM EMPLOYEE E JOIN DEPARTMENT D ON E.DEPT_NUM<>D.DEPT_ID WHERE

D.DEPT_NAME='MARKETING';



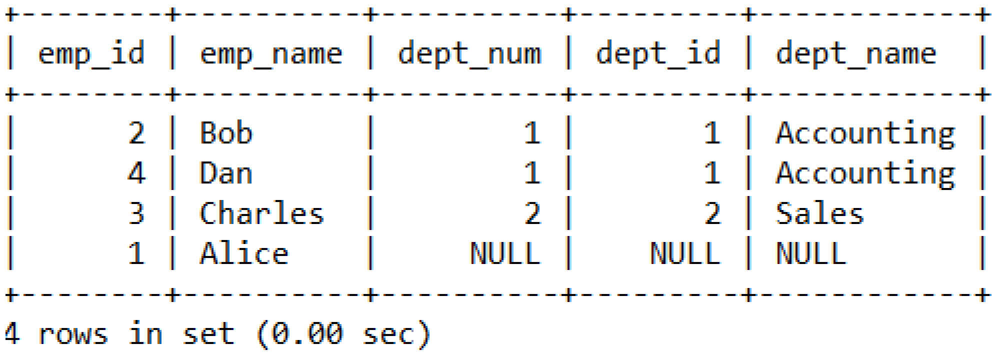
```
+----------+-----------+
| emp_name | dept_name |
+----------+-----------+
| Bob      | Marketing |
| Charles  | Marketing |
| Dan      | Marketing |
+----------+-----------+
3 rows in set (0.00 sec)
```

OUTER JOIN:

1. LEFT OUTER JOIN OR LEFT JOIN: Join Employee and department tables with reference to employee table

SELECT *

FROM EMPLOYEE E LEFT JOIN DEPARTMENT D ON E.DEPT_NUM=D.DEPT_ID;

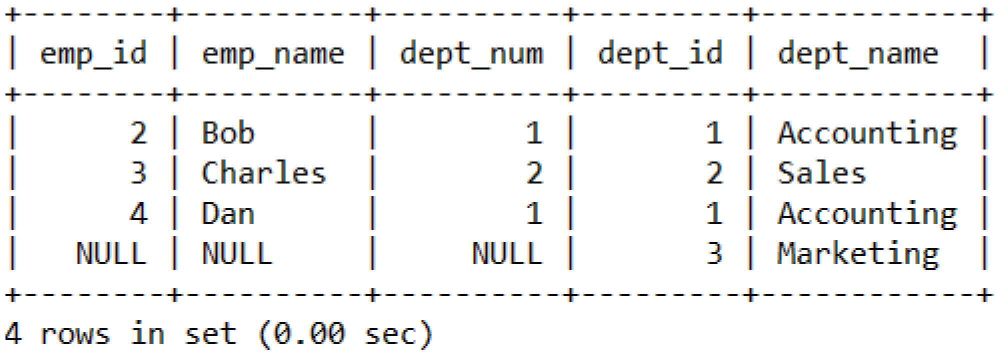


```
+--------+----------+----------+---------+------------+
| emp_id | emp_name | dept_num | dept_id | dept_name  |
+--------+----------+----------+---------+------------+
|      2 | Bob      |        1 |       1 | Accounting |
|      4 | Dan      |        1 |       1 | Accounting |
|      3 | Charles  |        2 |       2 | Sales      |
|      1 | Alice    |     NULL |    NULL | NULL       |
+--------+----------+----------+---------+------------+
4 rows in set (0.00 sec)
```

2. RIGHT OUTER JOIN OR RIGHT JOIN: Join Employee and department tables with reference to department table

SELECT * FROM

EMPLOYEE E RIGHT JOIN DEPARTMENT D ON E.DEPT_NUM=D.DEPT_ID;



```
+--------+----------+----------+---------+------------+
| emp_id | emp_name | dept_num | dept_id | dept_name  |
+--------+----------+----------+---------+------------+
|      2 | Bob      |        1 |       1 | Accounting |
|      3 | Charles  |        2 |       2 | Sales      |
|      4 | Dan      |        1 |       1 | Accounting |
|   NULL | NULL     |     NULL |       3 | Marketing  |
+--------+----------+----------+---------+------------+
4 rows in set (0.00 sec)
```

3. COMBINATION OF SET AND JOIN OPERATIONS

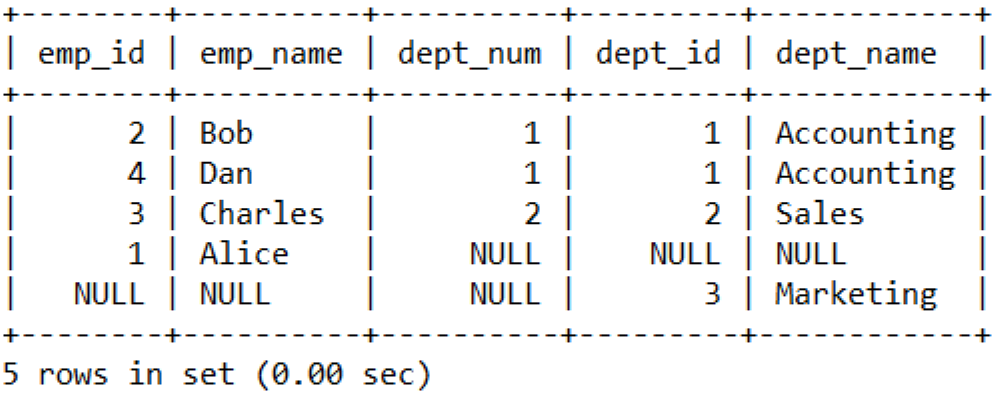
SELECT *

FROM EMPLOYEE E LEFT JOIN DEPARTMENT D ON E.DEPT_NUM=D.DEPT_ID

UNION

SELECT *

FROM EMPLOYEE E RIGHT JOIN DEPARTMENT D ON E.DEPT_NUM=D.DEPT_ID;

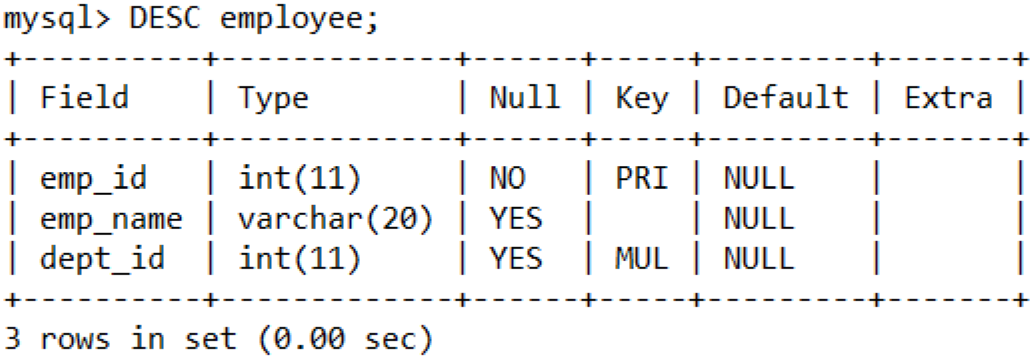


```
+--------+----------+----------+---------+------------+
| emp_id | emp_name | dept_num | dept_id | dept_name  |
+--------+----------+----------+---------+------------+
|      2 | Bob      |        1 |       1 | Accounting |
|      4 | Dan      |        1 |       1 | Accounting |
|      3 | Charles  |        2 |       2 | Sales      |
|      1 | Alice    |     NULL |    NULL | NULL       |
|   NULL | NULL     |     NULL |       3 | Marketing  |
+--------+----------+----------+---------+------------+
5 rows in set (0.00 sec)
```

4.**NATURAL JOIN Operation:**

NOTE: First rename the column

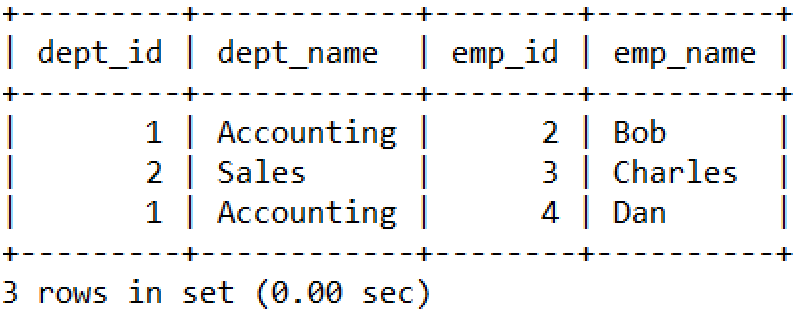
ALTER TABLE employee Change dept_num dept_id int;



```
mysql> DESC employee;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| emp_id   | int(11)     | NO   | PRI | NULL    |       |
| emp_name | varchar(20) | YES  |     | NULL    |       |
| dept_id  | int(11)     | YES  | MUL | NULL    |       |
+----------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

SELECT *

FROM DEPARTMENT NATURAL JOIN EMPLOYEE



```
+---------+------------+--------+----------+
| dept_id | dept_name  | emp_id | emp_name |
+---------+------------+--------+----------+
|       1 | Accounting |      2 | Bob      |
|       2 | Sales      |      3 | Charles  |
|       1 | Accounting |      4 | Dan      |
+---------+------------+--------+----------+
3 rows in set (0.00 sec)
```

Experiment No.9: To study and implement Sub queries/Nested queries, Correlated nested queries in SQL.

NESTING OF QUERIES

A complete SELECT query, called a nested query, can be specified within the WHERE-clause of another query, called the outer query.

Syntax:

```
SELECT      select_list
FROM        table
WHERE       expr operator
                    (SELECT     select_list
                     FROM       table);
```

Subqueries:

A subquery is a SELECT statement that is embedded in a clause of another SELECT statement. They can be very useful when you need to select rows from a table with a condition that depends on the data in the table itself.

You can place the subquery in a number of SQL clauses:

WHERE clause

HAVING clause

FROM clause

The comparison operator IN compares a value v with a set (or multi-set) of values V, and evaluates to TRUE if v is one of the elements in V. The subquery (inner query) executes once before the main query. The result of the subquery is used by the main query (outer query).

Single-Row Subqueries:

- Return only one row
- Use single-row comparison operators(=, >,<= ,>= ,< >)

Multiple-Row Subqueries:

- Return more than one row
- Use multiple-row comparison operators(IN, ANY, ALL)

CORRELATED NESTED QUERIES:

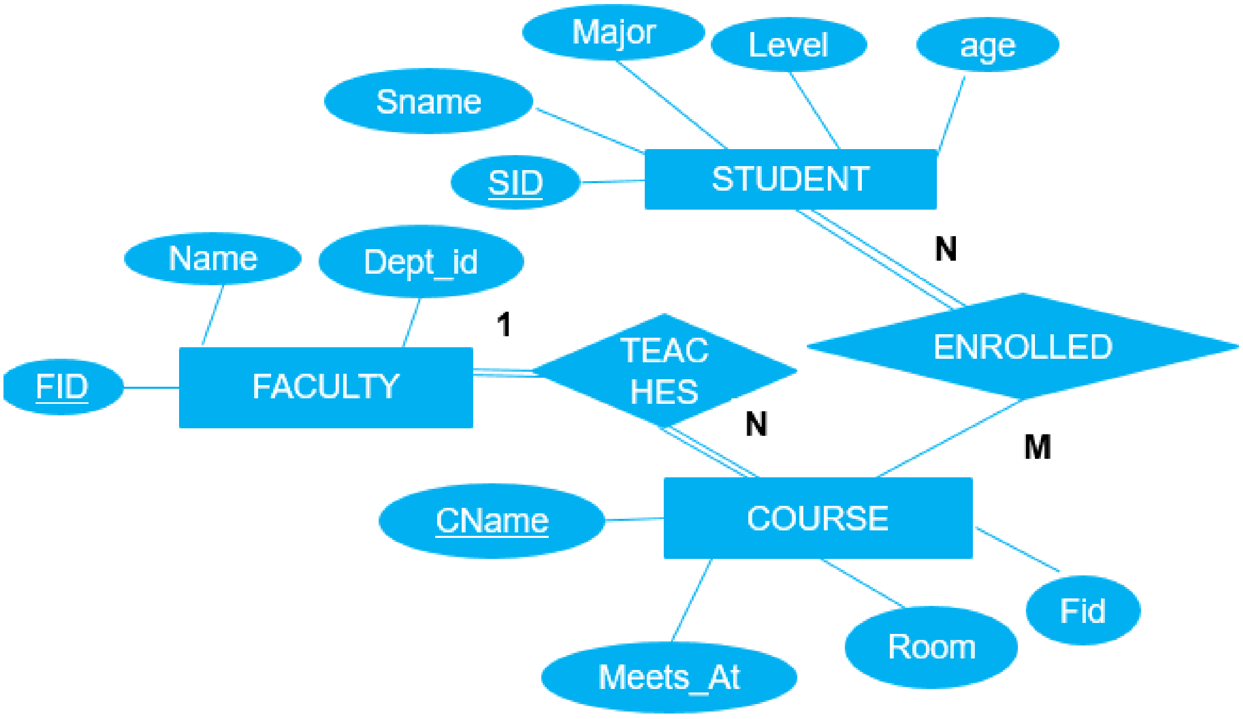
If a condition in the WHERE-clause of a nested query references an attribute of a relation declared in the outer query, the two queries are said to be correlated. The result of a correlated nested query is different for each tuple (or combination of tuples) of the relation(s) the outer query.

Use Student database:

Note:

- Already created for experiment-4.
- But sharing the ER diagram, schema description and creations for students, reference.

ER-Diagram



Schema description for Student Database:

- **Student (*sid:* integer, *sname:* varchar(20), *major:* string, address: varchar(20) *level:* string, age *int, DOB date*)**
- **Course (*cname:* varchar(20), *meets_at:* varchar(20), *room:* varchar(20), *fid:* integer)**
- **Enrolled (*sid:* integer, *cname:* varchar(20))**
- **Faculty (fid: integer, *fname:* varchar(20), address: varchar(20), *deptid:* integer)**

Creation of Tables:

STUDENT TABLE

```
CREATE TABLE STUDENT (SID NUMBER PRIMARY KEY, SNAME VARCHAR2 (10),
MAJOR VARCHAR2(10), LEVEL VARCHAR2(10), DOB DATE);
```

FACULTY TABLE

```
CREATE TABLE FACULTY (FID int PRIMARY KEY, FNAME VARCHAR(10), DEPTID int);
```

COURSE TABLE

```
CREATE TABLE COURSE (CNAME VARCHAR(10) PRIMARY KEY, MEETS_AT
VARCHAR(10),ROOM VARCHAR(5), FID int, FOREIGN KEY (FID) REFERENCES
FACULTY (FID) ON DELETE   CASCADE);
```

ENROLLED TABLE

```
    CREATE TABLE ENROLLED (SID NUMBER, CNAME VARCHAR2(10),
    FOREIGN KEY(SID) REFERENCES STUDENT (SID) ON DELETE CASCADE,
    FOREIGN KEY(CNAME) REFERENCES COURSE (CNAME) ON DELETE CASCADE);
```

Insert the values into the tables:

Insert into Student values (…..)

Insert into Faculty values (…)

Insert into Course values (…)

Insert into Enrolled values (…)

SELECT Command:

To retrieve information from a database we can query the databases. SQL SELECT statement is used to select rows and columns from a database/relation

Tables after insertions

```
mysql> SELECT * FROM STUDENT;
+---------+-------------------+----------+-----------+------+
| SID      | SNAME           | MAJOR | LEVEL | AGE |ADDRESS             | DOB|
+-------- +-------------------+-----------+----------+------+-----------------------+-------------+
|  101     | ABHI                | CSE       | JR         |   19   | BANGALORE    |  1980-01-23  |
|  102     | ANIL                 | ISE         | JR         |  18   | DAVANAGERE |   1984-02-12  |
|  103     | BHAVYA         | ISE         | SR         |  20   | CHITRADURGA|   1998-03-22 |
|  104     | CHETHAN       | CSE        | JR         |  19   | BELAGAVI       |    1999-04-26 |
|  105     | SURESH           | MECH    | JR         |  18   |HUBLI                 |    1970-12-20 |
|  106     | JAYANTH        | CSE        | SR        |  20   | DHARVAD       |      1988-11-16|
+--------+--------------------+-----------+---------+------+--------------------+------------------+
```

```
mysql> SELECT * FROM FACULTY;
+------+-------------------+-----------+------------+
| FID  | FNAME           | DEPTID |  ADDRESS  |
+------+-------------------+-----------+---------------
| 1001 | JAMES            |     41      | BANGALORE    |
| 1002 | SUNIL             |     42      |  BELAGAVI        |
| 1003 | SUKRUTH       |     43      | CHITRADURGA |
| 1004 | NARASIMHA |     44       | BANGALORE    |
| 1005 | AFROZ           |     41       | BANGALORE    |
| 1006 | JOHN              |     42       | BANGALORE   |
| 1007 | AHMED         |     45       | BANGALORE  |
| 1008 | SUNITHA       |     46       | BANGALORE  |
| 1009 | SRIVINAY     |     42       | HUBLI         |
+------+------------------+-----------+----------------+
```

```
mysql> SELECT * FROM ENROLLED;

| SID | CNAME  |
+------+---------------+
|  101    DBMS             |
|  105 |   TOC               |
|  106 |    BIGDATA       |
|  106 |   AI                   |
|  102 |  DATA MINING |
|  103 |  DBMS               |
|  101 | TOC                   |
|  104 | TOC                    |
|  105 | BIGDATA          |
|  106 | MP  |
mysql> SELECT * FROM COURSE;

| CNAME       | MEETS_AT | ROOM     | FID        |
+---------------+-----------------+-----------+------------+
| DBMS         | 9:00              | NG01 | 1001 |
| TOC            | 10:00            | KG04 | 1008 |
| BIGDATA   | 10:00            | KF04 | 1009 |
| DATA MINING | 10:00        | KF03 | 1009 |
| AI               | 10:00            | NG02 | 1001 |
| MP              | 11:15              | NG03 | 1002 |
| OOPS          | 12:10              | NG04 | 1002 |
| COA            | 01:10              | NG05 | 1003 |
| SE              | 1:05                   | NG06 | 1004 |

| OS              | 2:05                   | KG01 | 1005 |
```

1. Find the names of all Juniors (level = JR) who are enrolled in a COURSE taught by Prof.Narasimha.

```
mysql> SELECT DISTINCT S.SNAME

          FROM STUDENT S,CLASS C,ENROLLED E,FACULTY F

          WHERE S.SNUM=E.SNUM AND E.CNAME=C.CNAME AND F.FID=C.FID

          AND F.FNAME='NARASIMHA' AND S.LEVEL='JR';
```

2. Find the names of all COURSEs that either meet in room KG04 or have five or more Students enrolled.

```
mysql> SELECT C.CNAME

          FROM CLASS C

          WHERE C.ROOM='KG04' OR C.CNAME IN (SELECT E.CNAME

                                                                FROM ENROLLED E

                                                                GROUP BY E.CNAME

                                                                 HAVING COUNT (*)>=5);
```

3. Find the names of all students who are enrolled in two classes that meet at the same time

```
mysql> SELECT DISTINCT S.*
```

```
          FROM STUDENT S
          WHERE S.SNUM IN
  (SELECT E1.SNUM
 FROM ENROLLED E1, ENROLLED E2, CLASS C1, CLASS C2
 WHERE E1.SNUM = E2.SNUM AND E1.CNAME <>    E2.CNAME AND
 E1.CNAME = C1.CNAME AND E2.CNAME = C2.CNAME AND                 C1.MEETS_AT
= C2.MEETS_AT);
```

4. .Find the names of students enrolled in the maximum number of courses.

```
mysql> SELECT DISTINCT S.sname
          FROM Student S
          WHERE S.snum IN (SELECT E.snum
                              FROM Enrolled E
                              GROUP BY E.snum);
```

5. Find the names of faculty members for whom the combined enrolment of the courses that they teach is less than five.

```
mysql> SELECT DISTINCT F.FNAME
          FROM FACULTY F
          WHERE F.FID IN (SELECT C.FID
                            FROM CLASS C, ENROLLED E
                            WHERE C.CNAME=E.CNAME
                            GROUP BY C.FID
                             HAVING COUNT(*)<5);
```

Experiment 10: To Study and Implement Views in SQL

(Note: use Student Database)

Views in SQL:

- Views are relations, except that they are not physically stored.
- Views are created for presenting different information to different users
- A view is a “virtual table” or a “stored query” which takes the output of a query and treats it as a table. The table upon which a view is created is called as base table.

- A view is a logical table based on a table or another view. A view contains no data of its own but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called base tables.
- The view is stored as a SELECT statement in the data dictionary

OBJECTIVE

- Views Helps to encapsulate complex query and make it reusable.
- Provides user security on each view - it depends on your data policy security.

SQL COMMANDS

Creating views:

Syntax:

Create view <view name>;

Description:

This command is used to create view by combining two tables.

Viewing single row of table:

Syntax:

Create view<view name> as select from <table name>;

Description:

This command is used to view a single row from a particular table.

Viewing all columns from a single table:

Syntax:

Create view<view name> as select * from <table name>;

Description:

This is used to create view which displays all columns from a single table.

View specified column from a single table:

Syntax:

Create view<view table name> as select column1, column2 from <tablename>;

Description:

This command is used to create view which displays on a specified column from a single table.

View specified column from a multiple table:

Syntax:

Create view<view table name> as select column1, column2,….columnn where ‘condition’;

Description:

This is used to create view to display specified columns from multiple tables.

View all column from a multiple table:

Syntax:

Create view<view table name> as select * from <table name> where ‘condition’;

Description:

This is used to create view which displays all the columns of a table.

Inserting into views:

Syntax:

Insert into <view name> values <’data1’,’data2’,……>;

Description:

This is used to do inserting of information or data into values.

Updating in view: **is done by using query materialization and query modification.**

Deleting a view:

Syntax:

Drop view <view name>;

Illustration:

```
mysql> use studentdb;

Database changed
```

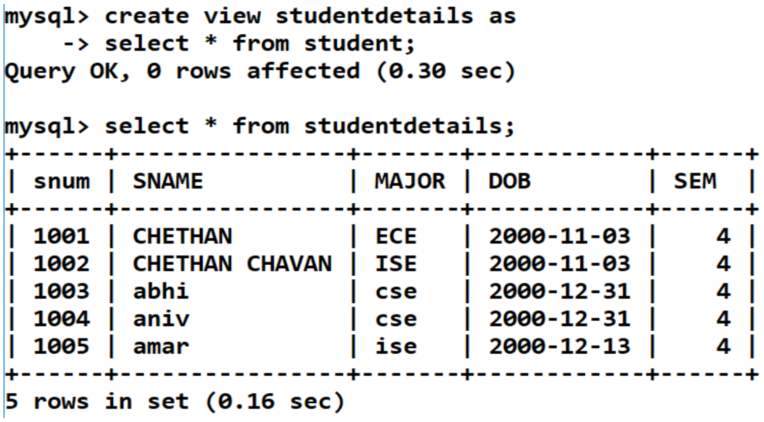
1. **Create a simple view named as “Studentdetails”**

Solution:

```
mysql> create view studentdetails as

     select * from student;

Query OK, 0 rows affected (0.30 sec)
```

2. **To retrieve all the data in the studentdetails view**

Solution:



```
mysql> create view studentdetails as
    -> select * from student;
Query OK, 0 rows affected (0.30 sec)

mysql> select * from studentdetails;
+------+----------------+-------+------------+------+
| snum | SNAME          | MAJOR | DOB        | SEM  |
+------+----------------+-------+------------+------+
| 1001 | CHETHAN        | ECE   | 2000-11-03 |    4 |
| 1002 | CHETHAN CHAVAN | ISE   | 2000-11-03 |    4 |
| 1003 | abhi           | cse   | 2000-12-31 |    4 |
| 1004 | aniv           | cse   | 2000-12-31 |    4 |
| 1005 | amar           | ise   | 2000-12-13 |    4 |
+------+----------------+-------+------------+------+
5 rows in set (0.16 sec)
```

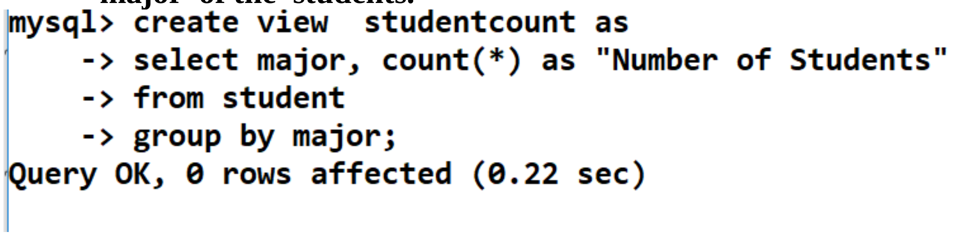
Query on a view.

3. Retieve the name and id student of ‘cse’ department (major).

```
mysql> select snum, sname  from studentdetails where major='cse';

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

| snum | SNAME |

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

| 1003 | abhi  |

| 1004 | aniv  |

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

2 rows in set (0.11 sec)
```

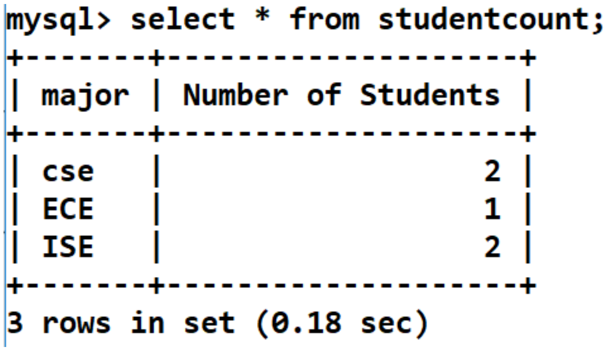
Complex Views:

4. **Write a Query view for each department retrieve the details like number of students, major of the students.**



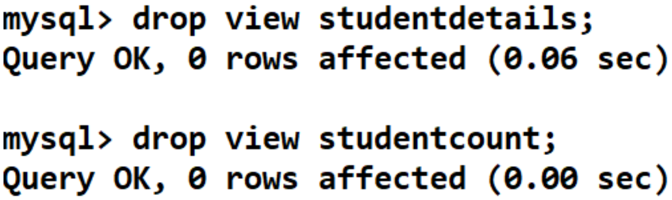
```
mysql> create view  studentcount as
    -> select major, count(*) as "Number of Students"
    -> from student
    -> group by major;
Query OK, 0 rows affected (0.22 sec)
```

Result of the query



```
mysql> select * from studentcount;
+-------+--------------------+
| major | Number of Students |
+-------+--------------------+
| cse   |                  2 |
| ECE   |                  1 |
| ISE   |                  2 |
+-------+--------------------+
3 rows in set (0.18 sec)
```

5. Remove the view called studentdetails and studentcount



```
mysql> drop view studentdetails;
Query OK, 0 rows affected (0.06 sec)

mysql> drop view studentcount;
Query OK, 0 rows affected (0.00 sec)
```

Experiment No.11: To study and implement Functions and Procedures.

A stored procedure contains a sequence of SQL commands stored in the database catalog so that it can be invoked later by a program

- Stored procedures are declared using the following syntax:

Create Procedure <proc-name>

(param_spec$_1$, param_spec$_2$, …, param_spec$_n$)

begin

-- execution code

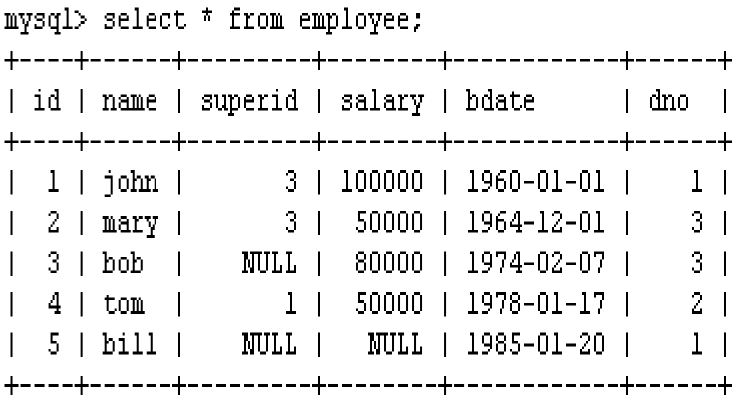
end;

where each param_spec is of the form:

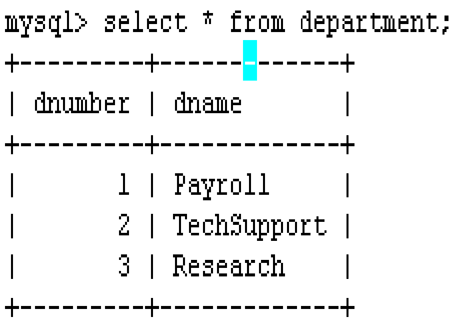
[in | out | inout] <param_name> <param_type>

- in mode: allows you to pass values into the procedure,
- out mode: allows you to pass value back from procedure to the calling program

An example consider the Employee and Department tables which are shown below.

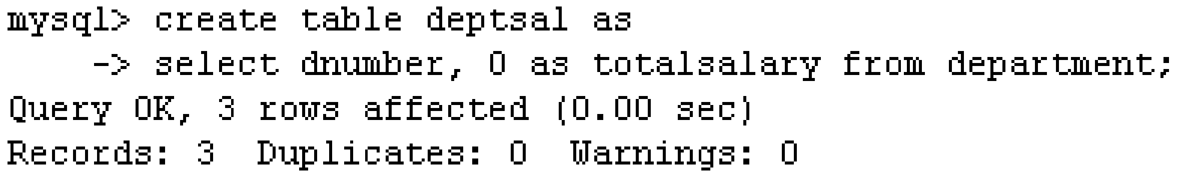


```
mysql> select * from employee;
+----+------+---------+--------+------------+------+
| id | name | superid | salary | bdate      | dno  |
+----+------+---------+--------+------------+------+
|  1 | john |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob  |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom  |       1 |  50000 | 1978-01-17 |    2 |
|  5 | bill |    NULL |   NULL | 1985-01-20 |    1 |
+----+------+---------+--------+------------+------+
```

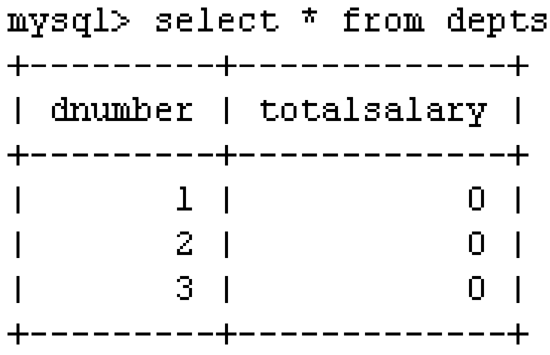


```
mysql> select * from department;
+---------+-------------+
| dnumber | dname       |
+---------+-------------+
|       1 | Payroll     |
|       2 | TechSupport |
|       3 | Research    |
+---------+-------------+
```

Suppose we want to keep track of the total salaries of employees working for each department

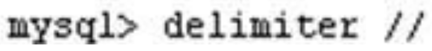


```
mysql> create table deptsal as
    -> select dnumber, 0 as totalsalary from department;
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
```



```
mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |           0 |
|       2 |           0 |
|       3 |           0 |
+---------+-------------+
```

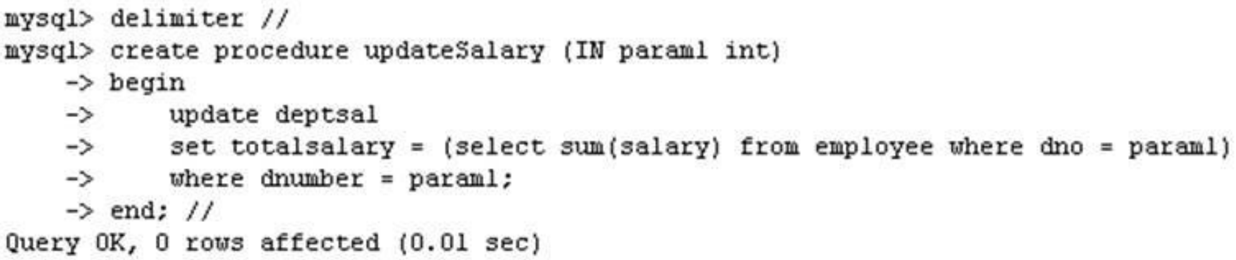
Illustration Example:



```
mysql> delimiter //
```

Step 1: Change the delimiter (i.e., terminating character) of SQL statement from semicolon (;) to something else (e.g., //) So that you can distinguish between the semicolon of the SQL statements in the procedure and the terminating character of the procedure definition.

Example:

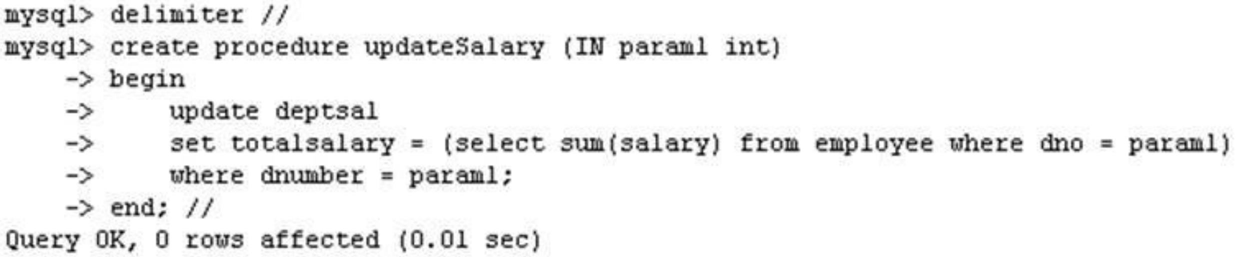


```
mysql> delimiter //
mysql> create procedure updateSalary (IN param1 int)
    -> begin
    ->     update deptsal
    ->     set totalsalary = (select sum(salary) from employee where dno = param1)
    ->     where dnumber = param1;
    -> end; //
Query OK, 0 rows affected (0.01 sec)
```

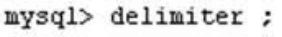
Step 2:

1. Define a procedure called updateSalary which takes as input a department number.
2. The body of the procedure is an SQL command to update the totalsalary column of the deptsal table.
3. Terminate the procedure definition using the delimiter you had defined in step 1 (//)

Example:



```
mysql> delimiter //
mysql> create procedure updateSalary (IN param1 int)
    -> begin
    ->     update deptsal
    ->     set totalsalary = (select sum(salary) from employee where dno = param1)
    ->     where dnumber = param1;
    -> end; //
Query OK, 0 rows affected (0.01 sec)
```



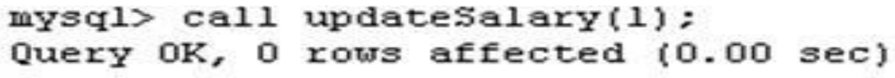
```
mysql> delimiter ;
```

Step 3: Change the delimiter back to semicolon (;)

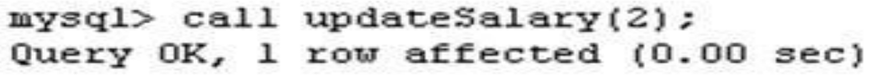
mysql> delimeter ;

Step 4: Call the procedure to update the totalsalary for each department

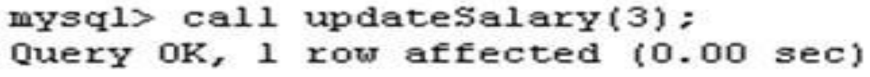
Example:



```
mysql> call updateSalary(1);
Query OK, 0 rows affected (0.00 sec)
```

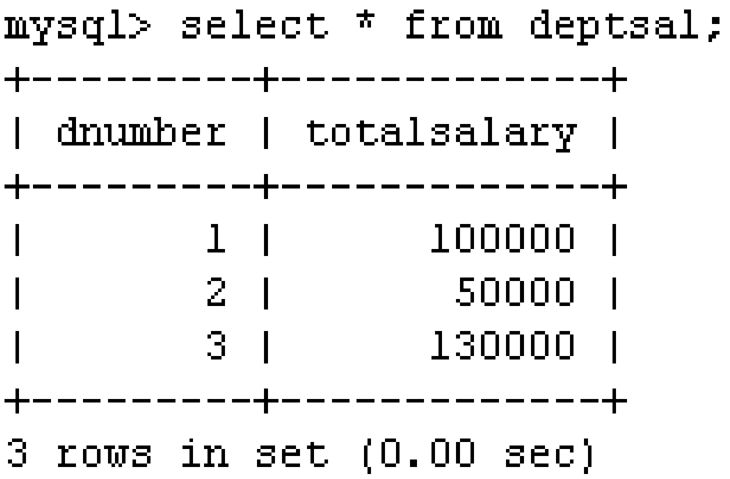


```
mysql> call updateSalary(2);
Query OK, 1 row affected (0.00 sec)
```



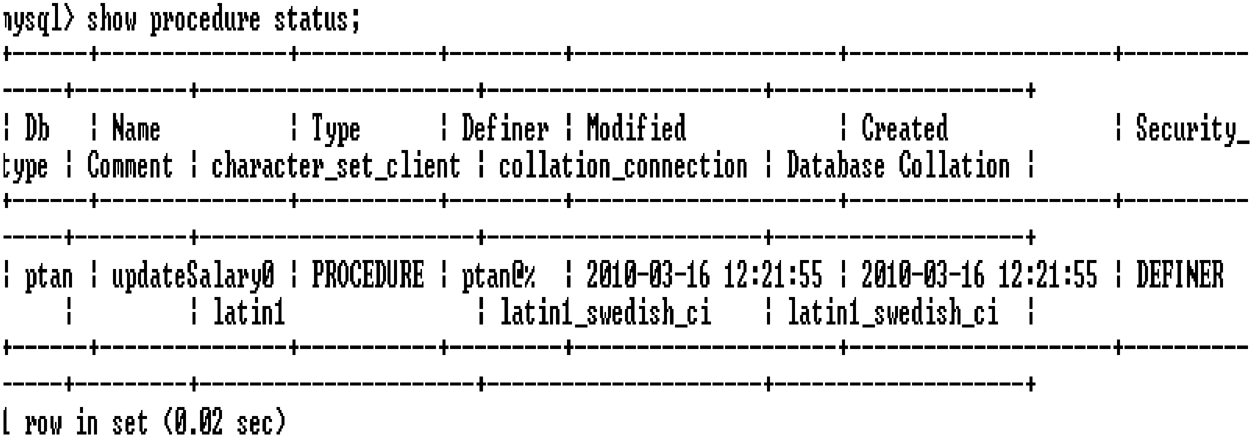
```
mysql> call updateSalary(3);
Query OK, 1 row affected (0.00 sec)
```

Step 5: Show the updated total salary in the deptsal table

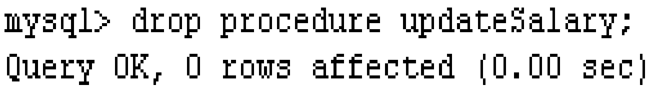


```
mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

- **Use show procedure status to display the list of stored procedures you have created**



```
mysql> show procedure status;
+------+---------------+-----------+---------+---------------------+---------------------+----------
-----+---------+----------------------+----------------------+--------------------+
| Db   | Name          | Type      | Definer | Modified            | Created             | Security_
type | Comment | character_set_client | collation_connection | Database Collation |
+------+---------------+-----------+---------+---------------------+---------------------+----------
-----+---------+----------------------+----------------------+--------------------+
| ptan | updateSalary0 | PROCEDURE | ptan@%  | 2010-03-16 12:21:55 | 2010-03-16 12:21:55 | DEFINER
     |         | latin1               | latin1_swedish_ci    | latin1_swedish_ci  |
+------+---------------+-----------+---------+---------------------+---------------------+----------
-----+---------+----------------------+----------------------+--------------------+
1 row in set (0.02 sec)
```



```
mysql> drop procedure updateSalary;
Query OK, 0 rows affected (0.00 sec)
```

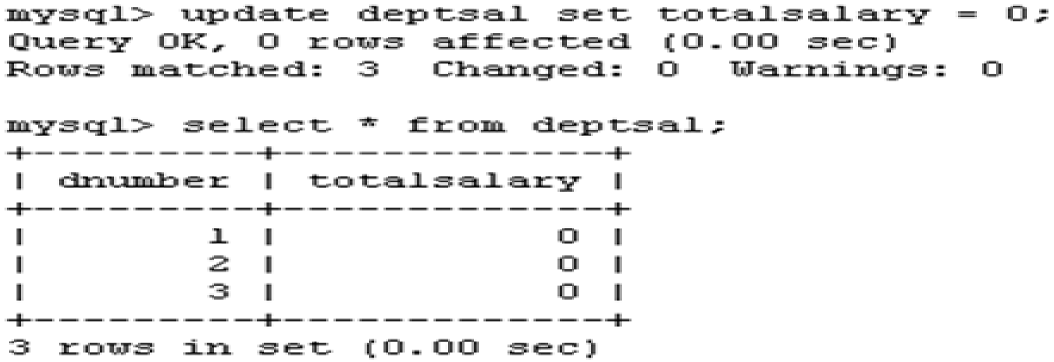
Stored Procedures in MySQL:

- You can declare variables in stored procedures
    - You can use flow control statements (conditional IF-THEN-ELSE or loops such as WHILE and REPEAT)
- MySQL also supports cursors in stored procedures.
    - A cursor is used to iterate through a set of rows returned by a query so that we can process each individual row.

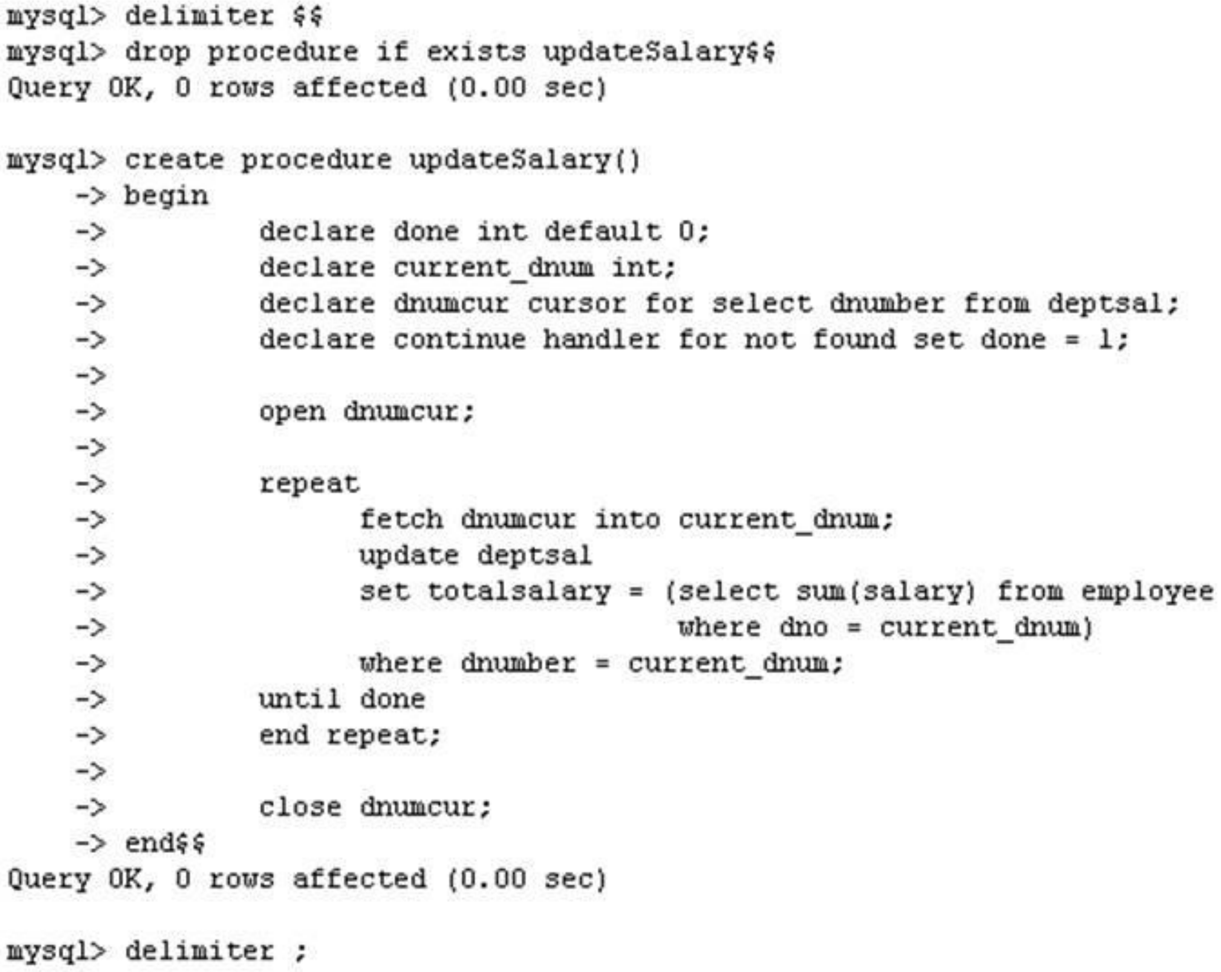
Example using Cursors:

- The previous procedure updates one row in deptsal table based on input parameter
- Suppose we want to update all the rows in deptsal simultaneously

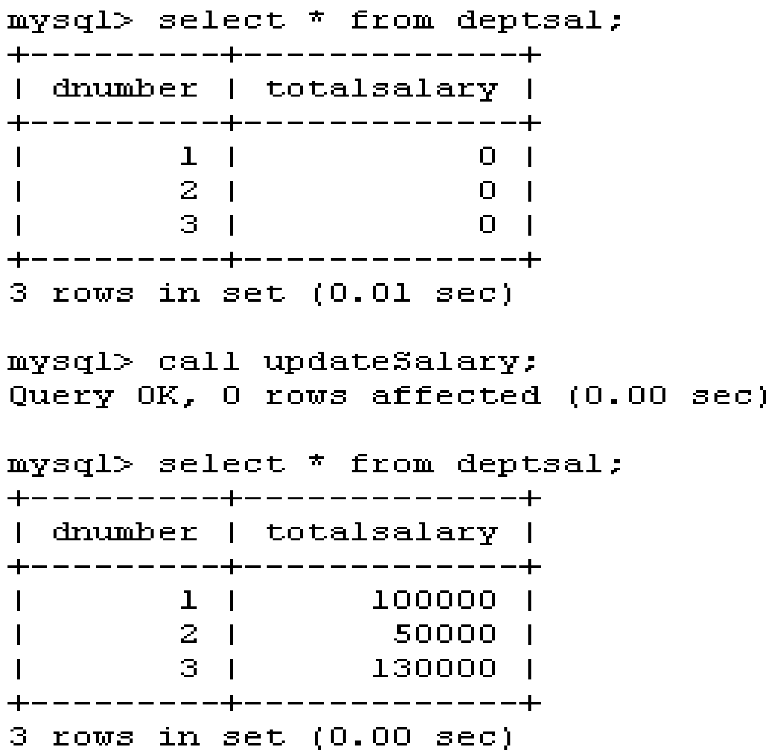
- First, let’s reset the totalsalary in deptsal to zero



```
mysql> update deptsal set totalsalary = 0;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 3  Changed: 0  Warnings: 0

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |           0 |
|       2 |           0 |
|       3 |           0 |
+---------+-------------+
3 rows in set (0.00 sec)
```



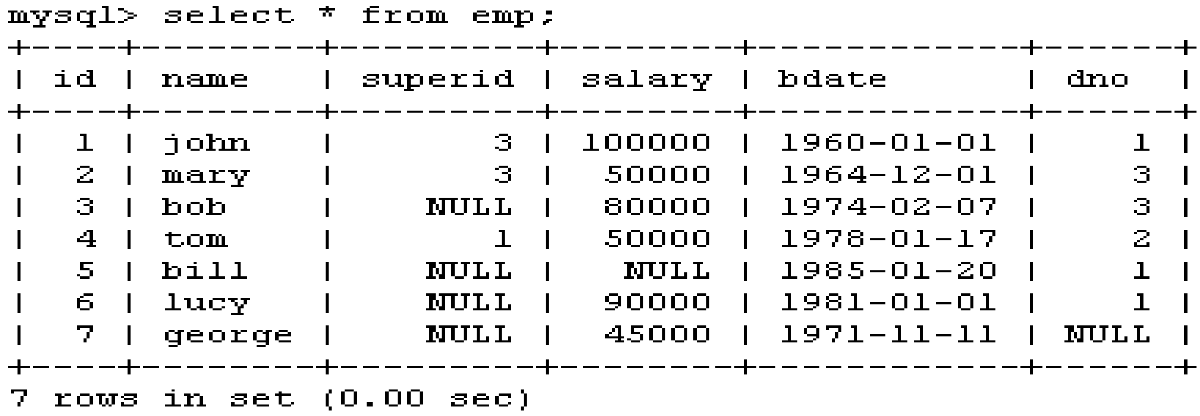
```
mysql> delimiter $$
mysql> drop procedure if exists updateSalary$$
Query OK, 0 rows affected (0.00 sec)

mysql> create procedure updateSalary()
    -> begin
    ->          declare done int default 0;
    ->          declare current_dnum int;
    ->          declare dnumcur cursor for select dnumber from deptsal;
    ->          declare continue handler for not found set done = 1;
    ->
    ->          open dnumcur;
    ->
    ->          repeat
    ->                fetch dnumcur into current_dnum;
    ->                update deptsal
    ->                set totalsalary = (select sum(salary) from employee
    ->                                   where dno = current_dnum)
    ->                where dnumber = current_dnum;
    ->          until done
    ->          end repeat;
    ->
    ->          close dnumcur;
    -> end$$
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
```



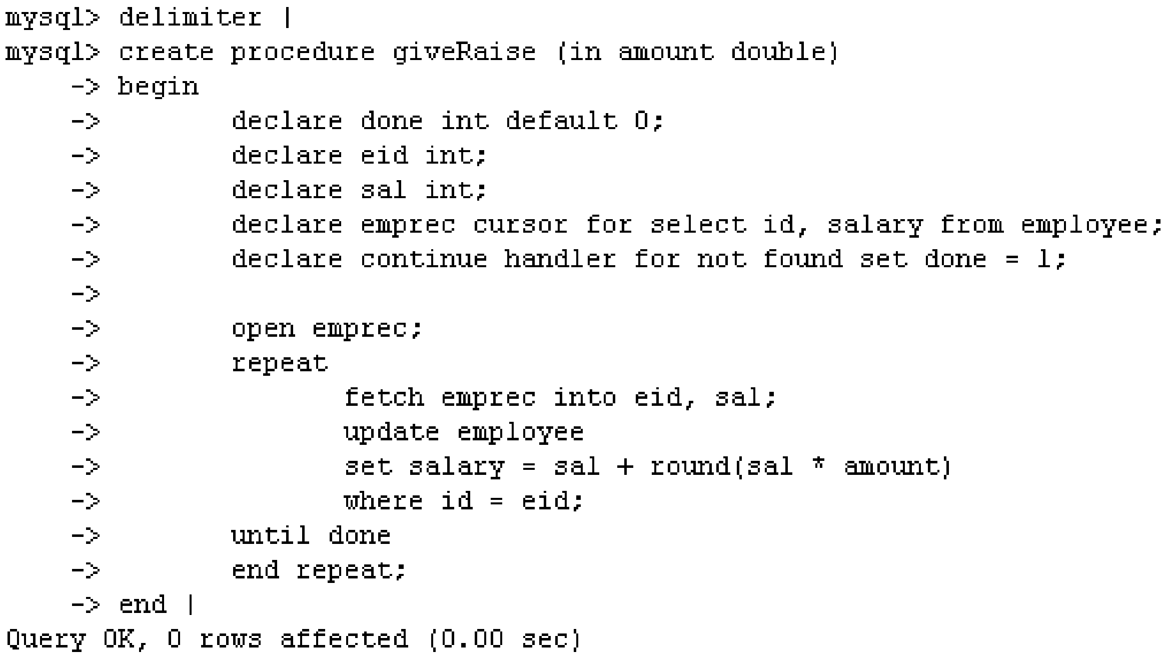
```
mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |           0 |
|       2 |           0 |
|       3 |           0 |
+---------+-------------+
3 rows in set (0.01 sec)

mysql> call updateSalary;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

An example illustration:

- **Create a stored procedure to give a raise to all employees**

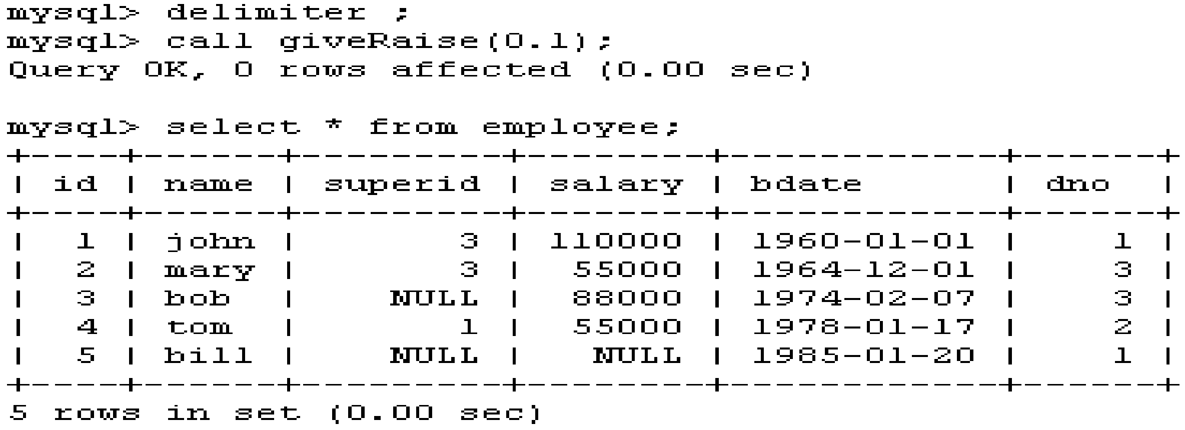


```
mysql> select * from emp;
+----+--------+---------+--------+------------+------+
| id | name   | superid | salary | bdate      | dno  |
+----+--------+---------+--------+------------+------+
|  1 | john   |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary   |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob    |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom    |       1 |  50000 | 1978-01-17 |    2 |
|  5 | bill   |    NULL |   NULL | 1985-01-20 |    1 |
|  6 | lucy   |    NULL |  90000 | 1981-01-01 |    1 |
|  7 | george |    NULL |  45000 | 1971-11-11 | NULL |
+----+--------+---------+--------+------------+------+
7 rows in set (0.00 sec)
```



```
mysql> delimiter |
mysql> create procedure giveRaise (in amount double)
    -> begin
    ->        declare done int default 0;
    ->        declare eid int;
    ->        declare sal int;
    ->        declare emprec cursor for select id, salary from employee;
    ->        declare continue handler for not found set done = 1;
    ->
    ->        open emprec;
    ->        repeat
    ->              fetch emprec into eid, sal;
    ->              update employee
    ->              set salary = sal + round(sal * amount)
    ->              where id = eid;
    ->        until done
    ->        end repeat;
    -> end |
Query OK, 0 rows affected (0.00 sec)
```

An Example:



```
mysql> delimiter ;
mysql> call giveRaise(0.1);
Query OK, 0 rows affected (0.00 sec)

mysql> select * from employee;
+----+------+---------+--------+------------+------+
| id | name | superid | salary | bdate      | dno  |
+----+------+---------+--------+------------+------+
|  1 | john |       3 | 110000 | 1960-01-01 |    1 |
|  2 | mary |       3 |  55000 | 1964-12-01 |    3 |
|  3 | bob  |    NULL |  88000 | 1974-02-07 |    3 |
|  4 | tom  |       1 |  55000 | 1978-01-17 |    2 |
|  5 | bill |    NULL |   NULL | 1985-01-20 |    1 |
+----+------+---------+--------+------------+------+
5 rows in set (0.00 sec)
```

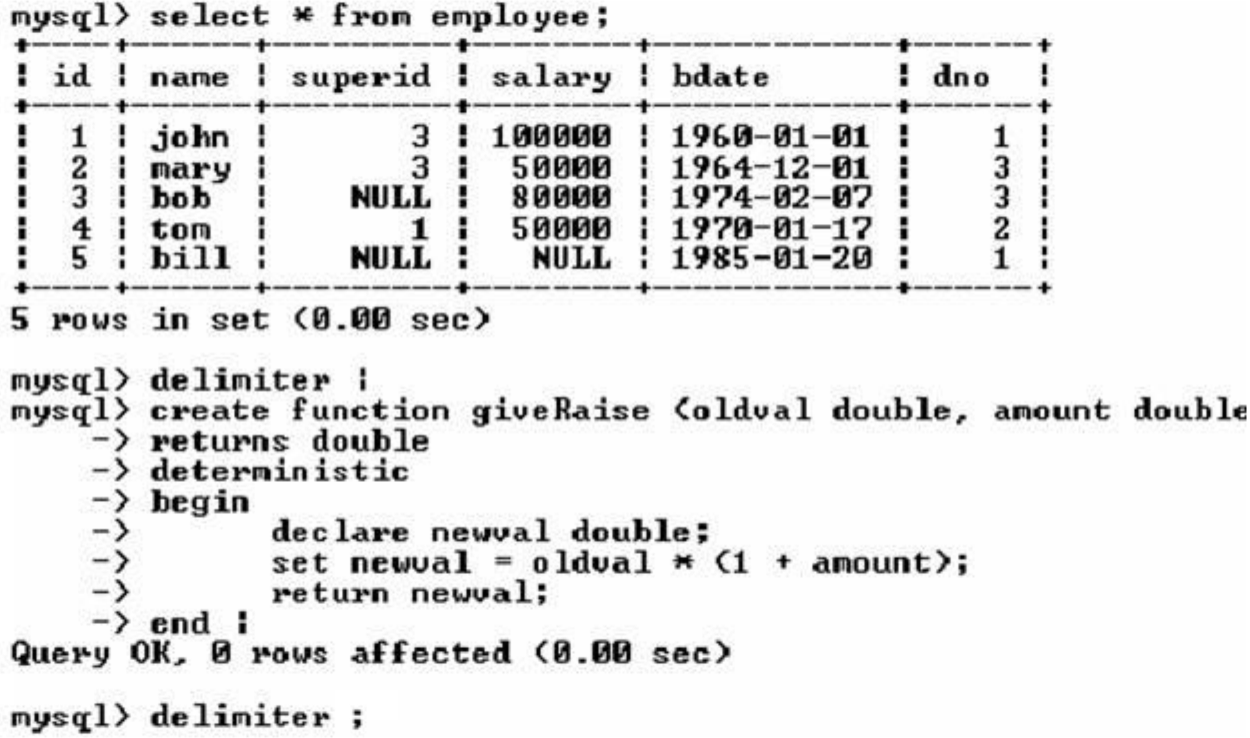
Functions:

- **Functions are declared using the following syntax:**

Create function <function-name> (param_spec$_1$, …, param_spec$_k$)
returns <return_type>
[not] deterministic
Begin
-- execution code
end;

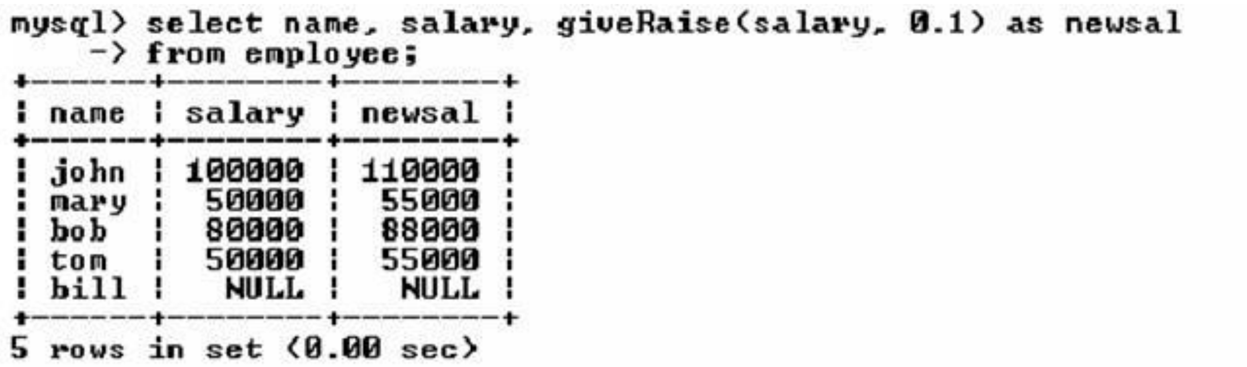
where param_spec is:
[in | out | in out] <param_name> <param_type>

Example of Functions:



```
mysql> select * from employee;
+----+------+---------+--------+------------+------+
| id | name | superid | salary | bdate      | dno  |
+----+------+---------+--------+------------+------+
|  1 | john |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob  |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom  |       1 |  50000 | 1970-01-17 |    2 |
|  5 | bill |    NULL |   NULL | 1985-01-20 |    1 |
+----+------+---------+--------+------------+------+
5 rows in set (0.00 sec)

mysql> delimiter |
mysql> create function giveRaise (oldval double, amount double
    -> returns double
    -> deterministic
    -> begin
    ->       declare newval double;
    ->       set newval = oldval * (1 + amount);
    ->       return newval;
    -> end |
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
```

Example of calling function in the SELECT clause:



```
mysql> select name, salary, giveRaise(salary, 0.1) as newsal
    -> from employee;
+------+--------+--------+
| name | salary | newsal |
+------+--------+--------+
| john | 100000 | 110000 |
| mary |  50000 |  55000 |
| bob  |  80000 |  88000 |
| tom  |  50000 |  55000 |
| bill |   NULL |   NULL |
+------+--------+--------+
5 rows in set (0.00 sec)
```

Experiment No.12: To study and implement SQL Triggers

To monitor a database and take a corrective action when a condition occurs

Triggers are nothing but the procedures/functions that involve actions and fired/executed automatically whenever an event occurs such as an insert, delete, or update operation or pressing a button or when mouse button is clicked.

Examples:

- Charge $10 overdraft fee if the balance of an account after a withdrawal transaction is less than $500

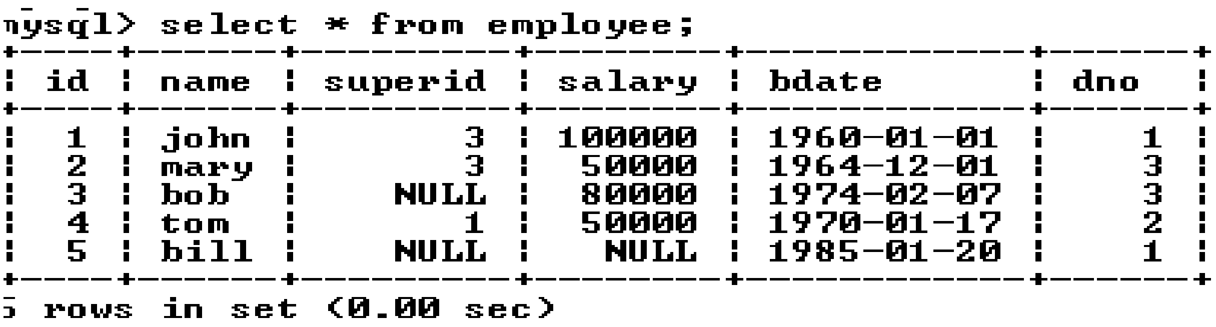
- Limit the salary increase of an employee to no more than 5% raise

Syntax:

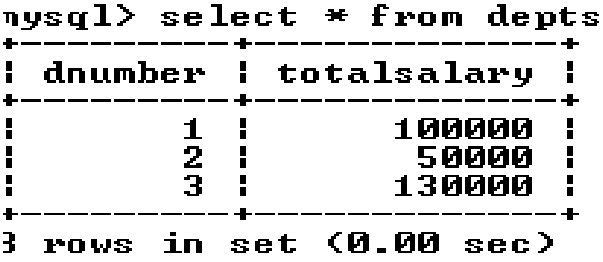
CREATE TRIGGER trigger-name
trigger-time trigger-event
ON table-name
FOR EACH ROW
trigger-action;

- trigger-time ∈ {BEFORE, AFTER}
- trigger-event ∈ {INSERT,DELETE,UPDATE}

SQL Triggers: An Example



```
mysql> select * from employee;
+----+------+---------+--------+------------+-----+
| id | name | superid | salary | bdate      | dno |
+----+------+---------+--------+------------+-----+
|  1 | john |       3 | 100000 | 1960-01-01 |   1 |
|  2 | mary |       3 |  50000 | 1964-12-01 |   3 |
|  3 | bob  |    NULL |  80000 | 1974-02-07 |   3 |
|  4 | tom  |       1 |  50000 | 1970-01-17 |   2 |
|  5 | bill |    NULL |   NULL | 1985-01-20 |   1 |
+----+------+---------+--------+------------+-----+
5 rows in set (0.00 sec)
```

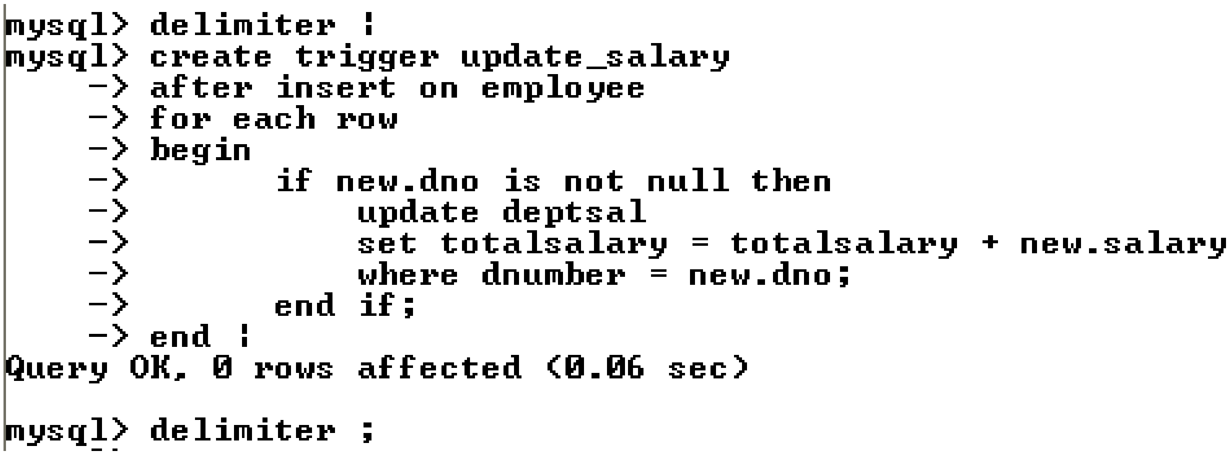


```
mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

An example: We want to create a trigger to update the total salary of a department when a new employee is hired

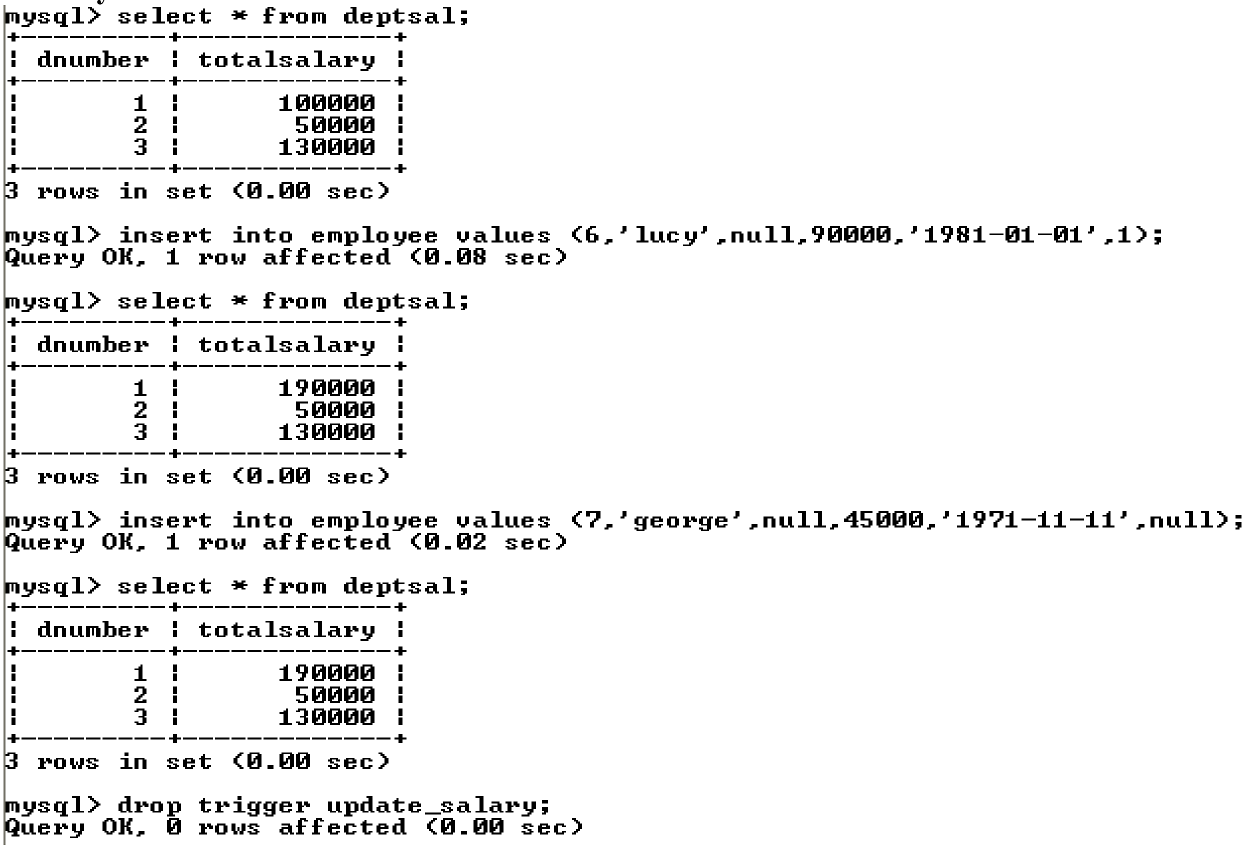
Problem Statemenet:

1 **Create a trigger to update the total salary of a department when a new employee is hired:**
Create definer='root'@'localhost' trigger update_salary



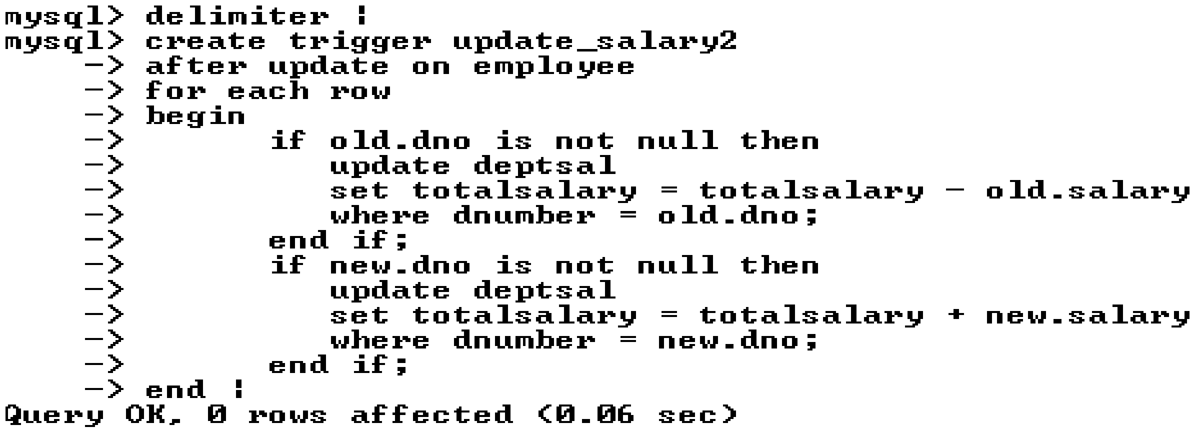
```
mysql> delimiter |
mysql> create trigger update_salary
    -> after insert on employee
    -> for each row
    -> begin
    ->        if new.dno is not null then
    ->            update deptsal
    ->            set totalsalary = totalsalary + new.salary
    ->            where dnumber = new.dno;
    ->        end if;
    -> end |
Query OK, 0 rows affected (0.06 sec)

mysql> delimiter ;
```

The keyword "new" refers to the new row inserted

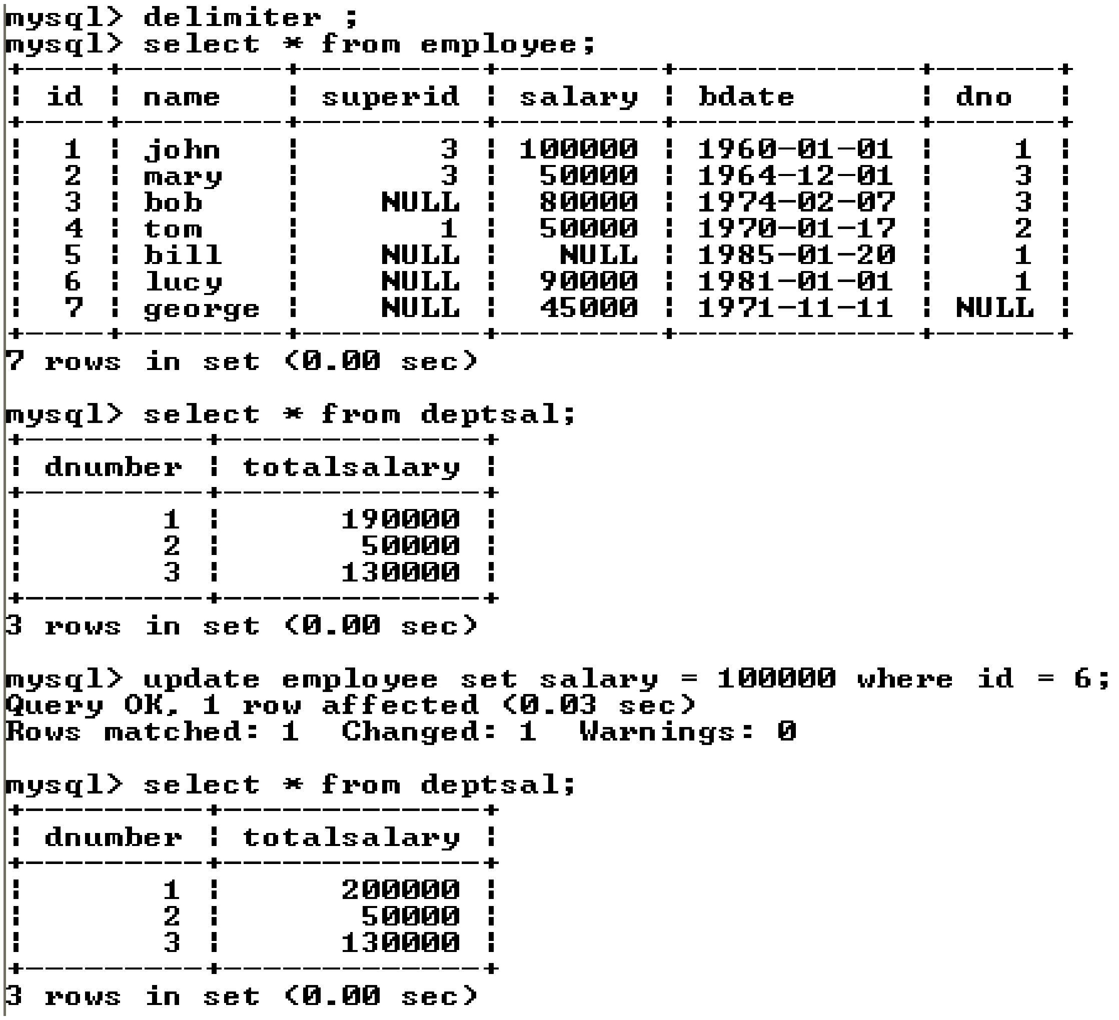


```
mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)

mysql> insert into employee values (6,'lucy',null,90000,'1981-01-01',1);
Query OK, 1 row affected (0.08 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      190000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)

mysql> insert into employee values (7,'george',null,45000,'1971-11-11',null);
Query OK, 1 row affected (0.02 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      190000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)

mysql> drop trigger update_salary;
Query OK, 0 rows affected (0.00 sec)
```

2. Create a trigger to update the total salary of a department when an employee tuple is modified:

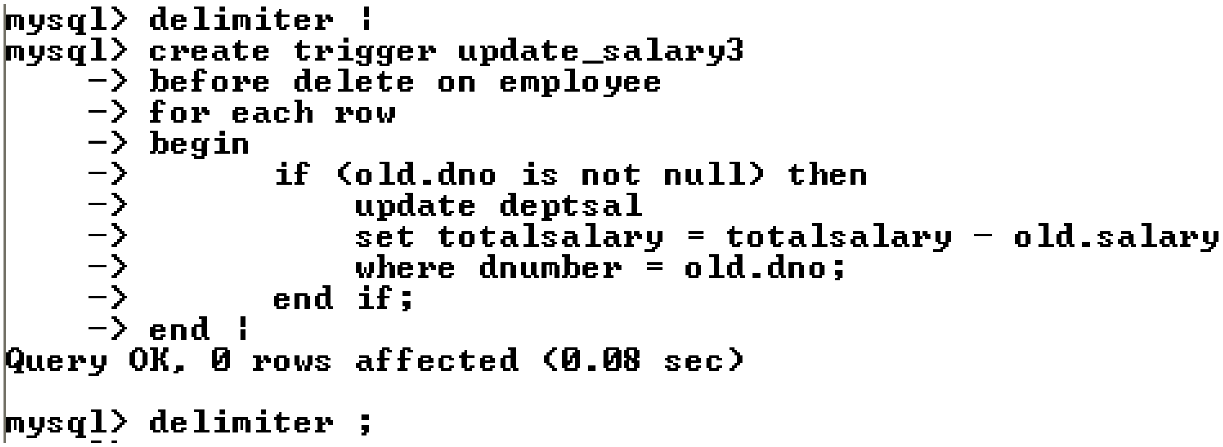


```
mysql> delimiter !
mysql> create trigger update_salary2
    -> after update on employee
    -> for each row
    -> begin
    ->       if old.dno is not null then
    ->          update deptsal
    ->          set totalsalary = totalsalary - old.salary
    ->          where dnumber = old.dno;
    ->       end if;
    ->       if new.dno is not null then
    ->          update deptsal
    ->          set totalsalary = totalsalary + new.salary
    ->          where dnumber = new.dno;
    ->       end if;
    -> end !
Query OK, 0 rows affected (0.06 sec)
```



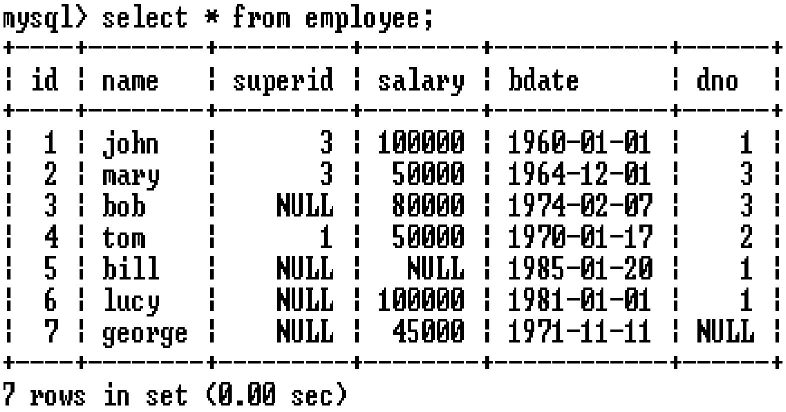
```
mysql> delimiter ;
mysql> select * from employee;
+----+--------+---------+--------+------------+------+
| id | name   | superid | salary | bdate      | dno  |
+----+--------+---------+--------+------------+------+
|  1 | john   |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary   |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob    |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom    |       1 |  50000 | 1970-01-17 |    2 |
|  5 | bill   |    NULL |   NULL | 1985-01-20 |    1 |
|  6 | lucy   |    NULL |  90000 | 1981-01-01 |    1 |
|  7 | george |    NULL |  45000 | 1971-11-11 | NULL |
+----+--------+---------+--------+------------+------+
7 rows in set (0.00 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      190000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)

mysql> update employee set salary = 100000 where id = 6;
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      200000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

3. Create a trigger to update the total salary of a department when an employee tuple is deleted:

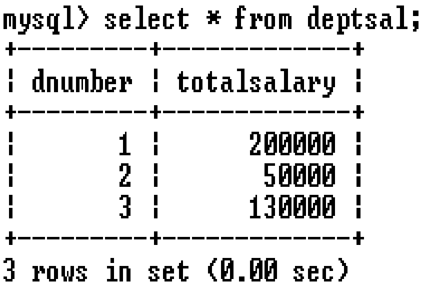


```
mysql> delimiter |
mysql> create trigger update_salary3
    -> before delete on employee
    -> for each row
    -> begin
    ->        if (old.dno is not null) then
    ->            update deptsal
    ->            set totalsalary = totalsalary - old.salary
    ->            where dnumber = old.dno;
    ->        end if;
    -> end |
Query OK, 0 rows affected (0.08 sec)

mysql> delimiter ;
```

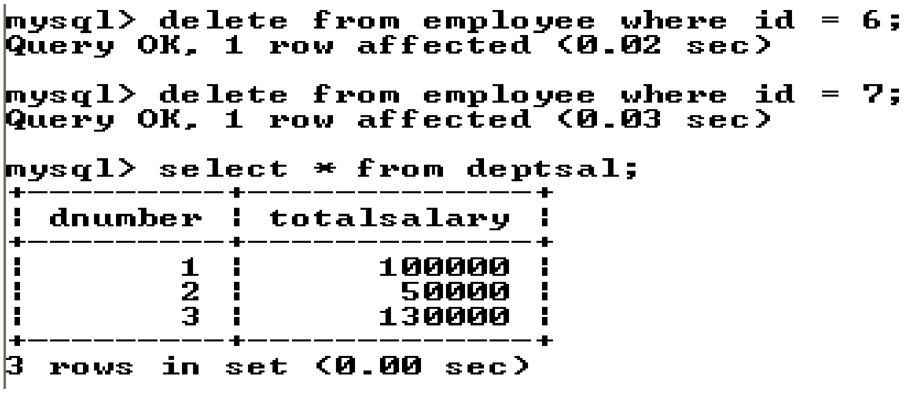
SQL Triggers: An Example



```
mysql> select * from employee;
+----+--------+---------+--------+------------+------+
| id | name   | superid | salary | bdate      | dno  |
+----+--------+---------+--------+------------+------+
|  1 | john   |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary   |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob    |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom    |       1 |  50000 | 1970-01-17 |    2 |
|  5 | bill   |    NULL |   NULL | 1985-01-20 |    1 |
|  6 | lucy   |    NULL | 100000 | 1981-01-01 |    1 |
|  7 | george |    NULL |  45000 | 1971-11-11 | NULL |
+----+--------+---------+--------+------------+------+
7 rows in set (0.00 sec)
```



```
mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      200000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```



```
mysql> delete from employee where id = 6;
Query OK, 1 row affected (0.02 sec)

mysql> delete from employee where id = 7;
Query OK, 1 row affected (0.03 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

To list all the triggers you have created:

mysql> show triggers;

To drop a trigger or a trigger is no longer required.

mysql> drop trigger trigger_name

Lab Exercises:

1. **The following relations keep track of airline flight information:**

Flights (flno: integer, from: string, to: string, distance: integer, departs: time, arrives: time, price: integer)

Aircraft (aid: integer, aname: string, cruisingrange: integer)

Certified (eid: integer, aid: integer)

Employees (eid: integer, ename: string, salary: integer)

Note that the Employees relation describes pilots and other kinds of employees as well; every pilot is certified for some aircraft, and only pilots are certified to fly.

For the above schema, perform the following.

a) Create the above tables by specifying primary keys and foreign keys.

b) Insert around 10 records in each of the tables.

c) Find the names of aircraft such that all pilots certified to operate them earn more than 80,000.

d) For each pilot who is certified for more than three aircraft, find the eid and the maximum cruising range of the aircraft that he (or she) is certified for.

e). Find the names of pilots whose salary is less than the price of the cheapest route from Los Angeles to Honolulu.

f) Find the second highest salary of an employee.

g) Create a stored procedure that remove all employees.

2. Create a relational database schema for a Minor-Project, described by the following relations.

STUDENT (Rollno: integer, Name: String, Sem: integer, Degree: String, Contact no: integer, Guide_No: integer)

GUIDE (Guide_name: String, Guide_No: integer, Guide_reserach_domain: String, Contat_No: integer, Email_Id: String)

PROJECT (Project_No: Integer, Project_title: String, Project_Area: String, Start_dt, date, Guide_No:integer)

GROUP (Group_Code:integer, Roll_No:integer)

PROJECT_GROUP (Group_Code:integer, Project_No: integer, no_of_students:integer)

For the above schema, perform the following.

a) Create the tables with the appropriate integrity constraints

b) Insert around 10 records in each of the tables

c) Find the list of guide, who are guiding more than two student groups.

d) Find the list of project no, project name & name of guide, in domain of DataBase.

e) Update guide details of a roll no „110011‟, new guide is “Ram Mohan‟ & id “112200‟.

f) Remove the Guide details, guide no is „112211‟ and assign guide no “133113” to all respective students project group.

g) Create a view as student_project details that lists student name, project name and guide name

3. Consider a relational database schema for a Sailors database below

Sailors(sid: integer, sname: string, rating: integer, age: real);

Boats(bid: integer, bname: string, color: string);

Reserves(sid: integer, bid: integer, day: date).

For the above schema, perform the following.

a) Create the above tables by specifying primary keys and foreign keys.

b) Insert around 10 records in each of the tables.

c) Find the names of sailors who have reserved a red boat, and list in the order of age.

d) Find the names of sailors who have reserved boat 103

e) Find the name and the age of the youngest sailor.

f) Find the average age of sailors for each rating level that has at least two sailors.

g) Create a stored procedure that gives details of sailors for a specified color of a boat.

4. Consider a relational database schema for a Company database below.

Employee (F_name: string, L_name: string , SSN:integer, Bdate: date, Address:string, Gender:string, Salary: integer, Super_Emp_id: integer, D_no: integer)

Department (D_name:string, D_no:integer, D_Mgr_id:integer, Mgr_start_date: date)

Dept_Location(D_no: integer, D_location :string)

Project (P_name:string, P_number:integer, P_location:string, D_no:integer)

Works_on (ESSN:integer, P_no:integer, Hours: int)

Dependent(SSN:integer,Dependent_name:string,Gender:string,Bdate:date,Relationship:String)

For the above schema, perform the following

a) Create the above tables by specifying primary keys and foreign keys.

b) Insert around 10 records in each of the tables.

c) Company decided to give a raise on salaries of every employee, working on the „ProductX‟ project by 10 percent.

d) Find the names of employees who have no dependents

e) List the name and address of all employees who work for the "Research" department.

f) Retrieve a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, then first name.

g) Create a view Dept_info that gives details of department name, Number of employees and total salary of each employee.

5. Database Schema for a Student Library scenario

Student(Stud_no : integer,Stud_name: string)

Membership(Mem_no: integer,Stud_no: integer)

Book(book_no: integer, book_name:string, author: string)

Iss_rec(iss_no:integer, iss_date: date, Mem_no: integer, book_no: integer)

For the above schema, perform the following—

a) Create the tables with the appropriate integrity constraints

b) Insert around 10 records in each of the tables

c) List all the student names with their membership numbers

d) List all the issues for the current date with student and Book names

e) Give a count of how many books have been bought by each student

f) Give a list of books taken by student with stud_no as 5

g) i)Create a view which lists out the iss_no, iss _date, stud_name, book name

ii)Create a procedure that gives the details of student.