

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CSA05 Database Management Systems**

**LAB MANUAL**

**INDEX**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.NO.** | **DATE** | **Exercises** | **Page No.** | **Marks** | **Sign** |
|  |  |  |  |  |  |
| 1 |  | DDL Commands – CREATE, ALTER, DROP |  |  |  |
|  |  | DDL Commands with Constraints – PRIMARY, FOREIGN |  |  |  |
| 2 |  | KEY, UNIQUE, CHECK |  |  |  |
|  |  |  |  |  |  |
|  |  | DML Commands – INSERT, SELECT, UPDATE, DELETE |  |  |  |
| 3 |  |  |  |  |  |
|  |  | SELECT with various clause – WHERE, pattern matching |  |  |  |
| 4 |  |  |  |  |  |
|  |  | SELECT with various clause – BETWEEN, IN, Aggregate |  |  |  |
| 5 |  | Function |  |  |  |
|  |  |  |  |  |  |
|  |  | SELECT with various clause – GROUP BY, HAVING, ORDERBY |  |  |  |
| 6 |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  | SubQuery& Correlated Query |  |  |  |
| 7 |  |  |  |  |  |
|  |  | Joins – EquiJoin, InnerJoin, OuterJoin |  |  |  |
| 8 |  |  |  |  |  |
|  |  | VIEW, INDEX, SEQUENCE |  |  |  |
| 9 |  |  |  |  |  |
|  |  | Simple programming exercises using CASE, IF, ITERATE, |  |  |  |
| 10 |  | LEAVE, LOOP |  |  |  |
|  |  |  |  |  |  |
|  |  | Simple programming exercises using REPEAT, WHILE |  |  |  |
| 11 |  |  |  |  |  |
|  |  | TCL Commands – COMMIT, ROLLBACK, SAVEPOINT |  |  |  |
| 12 |  |  |  |  |  |
|  |  | DCL Commands – GRANT, REVOKE |  |  |  |
| 13 |  |  |  |  |  |
|  |  | Procedures |  |  |  |
| 14 |  |  |  |  |  |
|  |  | Function |  |  |  |
| 15 |  |  |  |  |  |
|  |  | Cursors |  |  |  |
| 16 |  |  |  |  |  |
|  |  | Triggers |  |  |  |
| 17 |  |  |  |  |  |
| 18 |  | String Functions |  |  |  |
|  |  | Database Connectivity – Using PhP& MySQL |  |  |  |
| 19 |  |  |  |  |  |

Ex.No. 1

Date:

**DDL Commands – CREATE, ALTER, DROP**

**Aim:**

To Create, Alter and Drop the table using Data Definition Language.

**Description**:

Data Definition Language (DDL) statements are used to define the database structure or schema.

DDL Commands: Create, Alter, Drop, Rename, Truncate

* CREATE - to create objects in the database
* ALTER - alters the structure of the database
* DROP - delete objects from the database
* TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
* RENAME - rename an object

**SYNTAX:**

**CREATE TABLE**

CREATE TABLE table\_name

(

column\_name1 data\_type,

column\_name2 data\_type,

column\_name3 data\_type,

*....*

);

**ALTER A TABLE**

To add a column in a table

ALTER TABLE table\_name

ADD column\_namedatatype;

To delete a column in a table

ALTER TABLE table\_name

DROP COLUMN column\_name;

**DROP TABLE**

DROP TABLE table\_name;

**TRUNCATE TABLE**

TRUNCATE TABLE table\_name;

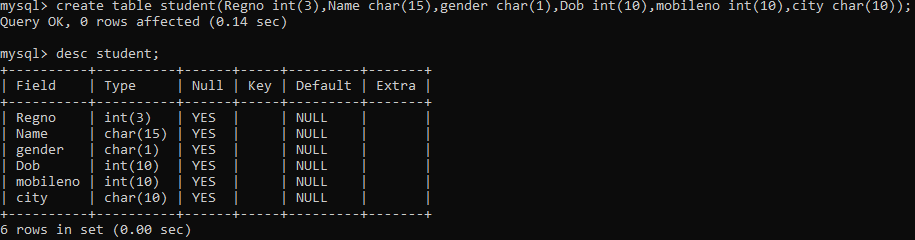
**Questions:**

1) Create a table name STUDENT with following structure.

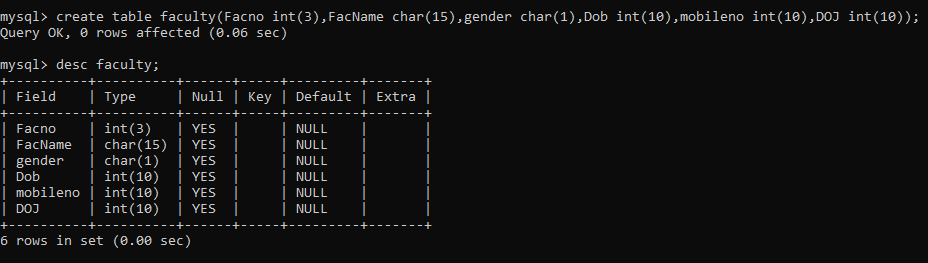
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column |  |  |  |
| # | Name | Description | Data Type |  |
|  |  |  |
|  |  | Registration |  |  |
| 1 | RegNo | Number | NUMBER(3) |  |
| 2 | Name | Student Name | VARCHAR(15) |  |
|  |  | Gender of the |  |  |
| 3 | Gender | student | CHAR(1) |  |
| 4 | DOB | Date of Birth | DATE |  |
| 5 | MobileNo | Mobile Number | NUMBER(10) |  |
| 6 | City | Location of stay | VARCHAR(15) |  |

**OUTPUTS:**

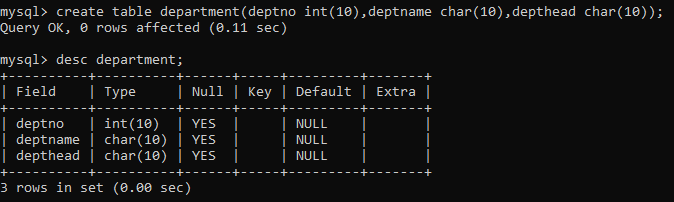
1)



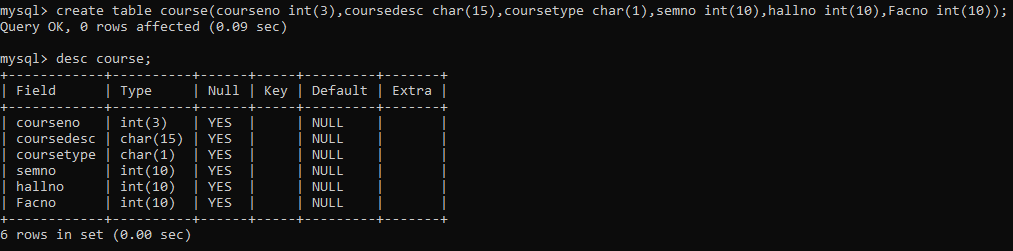
2)



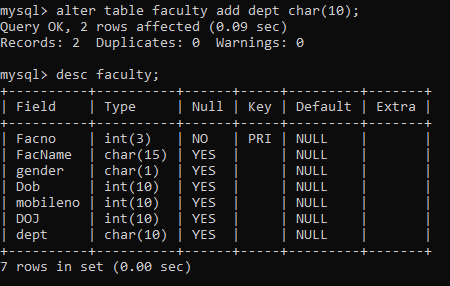
3)



4)



1. Modify the table FACULTY by adding a column name DeptNo of datatype VARCHAR(4)



**RESULT:**

Tables are created, altered and modified using DDL commands.

Ex.No. 2

Date:

**DDL Commands with Constraints – PRIMARY, FOREIGN KEY, UNIQUE, CHECK**

**AIM:**

To add the constraints like primary key, foreign key, unique key and check using DDL commands.

**Description:**

**PRIMARY KEY:**

The PRIMARY KEY constraint uniquely identifies each record in a database table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only one primary key, which may consist of single or multiple fields.

**FOREIGN KEY:**

A FOREIGN KEY is a key used to link two tables together.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

**UNIQUE Constraint:**

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

**CHECK Constraint:**

The CHECK constraint is used to limit the value range that can be placed in a column

If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

**PRIMARY:**

ALTER TABLE table\_name

ADD PRIMARY KEY(primary\_key\_column);

**FOREIGN KEY:**

ALTER TABLE table\_name

ADD CONSTRAINT constraint\_name

FOREIGN KEY foreign\_key\_name (columns)

REFERENCES parent\_table(columns)

ON DELETE action

ON UPDATE action

**UNIQUE:**

CREATE TABLE table\_1(

...

column\_name\_1 data\_type,

...

UNIQUE(column\_name\_1)

);

**CHECK**

CREATE TABLE IF NOT EXISTS parts (

part\_no VARCHAR(18) PRIMARY KEY,

description VARCHAR(40),

cost DECIMAL(10 , 2 ) NOT NULL CHECK(cost > 0), price DECIMAL (10,2) NOT NULL

);

**Questions:**

1) Alter the table STUDENT with following structure.

|  |  |  |
| --- | --- | --- |
|  | Column | Constraints |
| # | Name |  |
|  |  | PRIMARY |
| 1 | RegNo | KEY |
| 2 | MobileNo | NOT NULL |

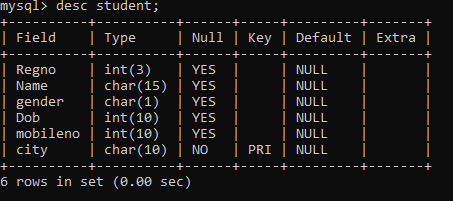
1. Alter the table name FACULTY with following structure. The DeptNo in this table refers the DeptNo in the DEPARTMENT table.

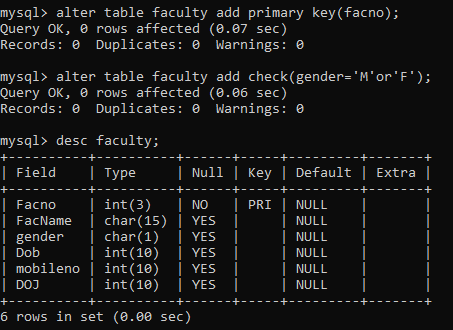
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | Column | | Constraints |  |  |  |  |
|  | # |  | Name | |  |  |  |  |  |
|  |  |  | FacNo | | PRIMARY |  |  |  |  |
|  | 1 |  | KEY |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  | Gender | | CHECK |  |  |  |  |
|  | 2 |  | ‘M’ or ‘F’ |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | 3) | After the FACULTY table is successfully created, test if you can add a constraint | | | | | | |  |
|  | FOREIGN KEY to the DeptNo of this table. | | | | | | | |  |
|  | 4) | Alter the table name DEPARTMENT with following structure. | | | | | | |  |
|  |  |  |  |  |  | |  |  |  |
|  |  |  | Column |  | Constraint | |  |  |  |
|  | # |  | Name |  |  |  |  |  |  |
|  |  |  | DeptNo |  | PRIMARY | |  |  |  |
|  | 1 |  |  | KEY | |  |  |  |
|  |  |  |  |  |  |  |
|  | 5) | Alter the table name COURSE with following structure. | | | | | | |  |
|  |  |  |  | |  | | |  |  |
|  |  |  | Column |  | Constraint | | |  |  |
|  | # |  | Name |  |  |  |  |  |  |
|  |  |  | CourseNo |  | PRIMARY | | |  |  |
|  | 1 |  |  | KEY | | |  |  |
|  |  |  |  |  |  |
|  | 2 |  | SemNo |  | 1 to 6 | | |  |  |

**OUTPUTS:**

1)

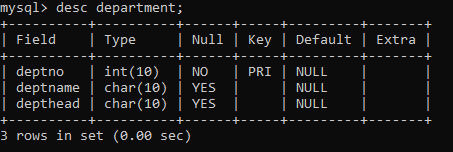




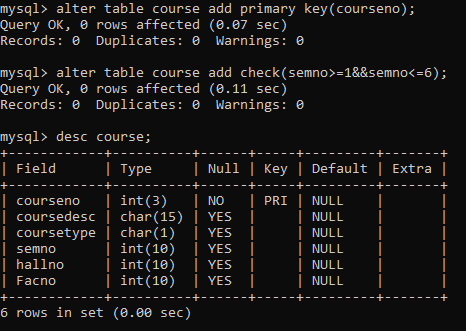


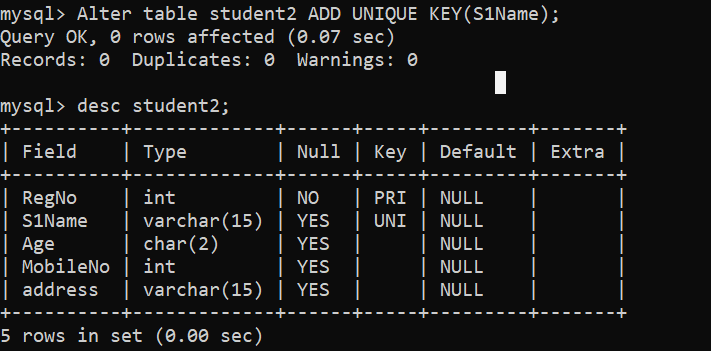
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4) | Alter the table name DEPARTMENT with following structure. | | | | | | |
|  |  |  |  |  | |  |
|  |  | Column |  | Constraint | |  |
| # |  | Name |  |  |  |  |
|  |  | DeptNo |  | PRIMARY | |  |
| 1 |  |  | KEY | |  |
|  |  |  |  |  | |  |

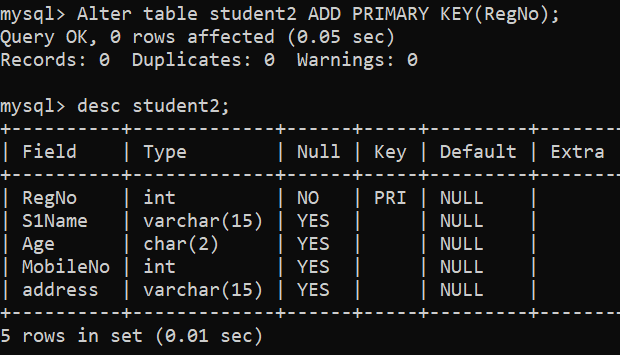


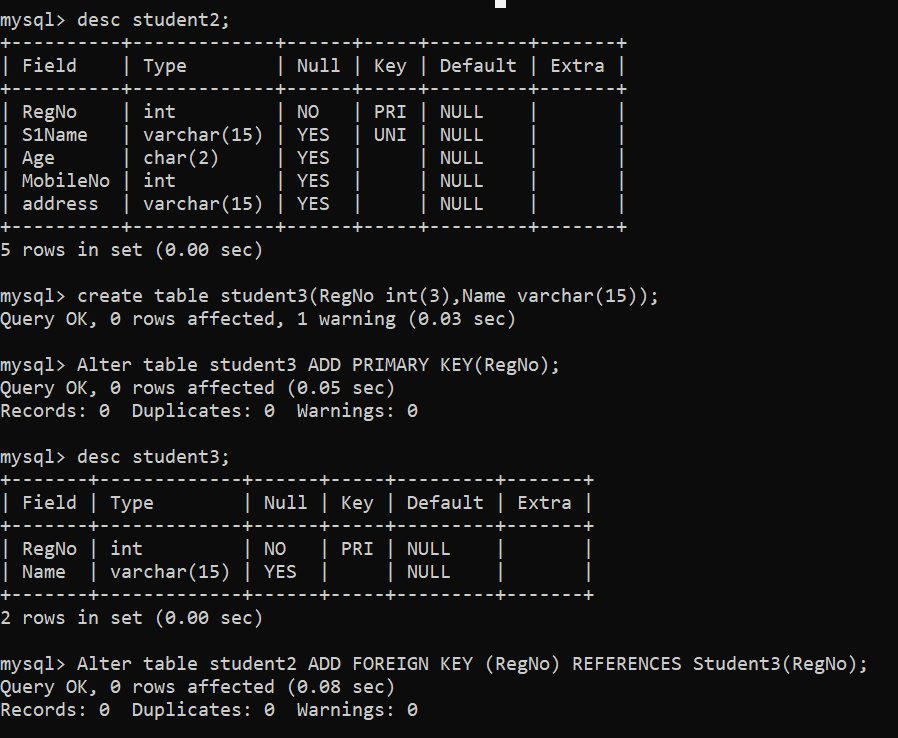


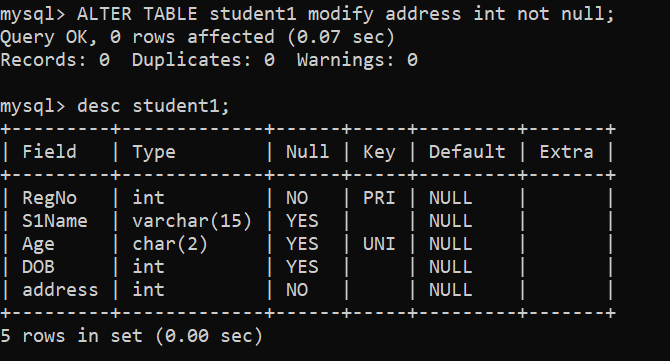
|  |  |
| --- | --- |
| 5) | After the FACULTY table is successfully created, test if you can add a constraint |
| FOREIGN KEY to the DeptNo of this table. | |









****

**Result:**

DDL Commands with Primary, Foreign, Unique, Check constraints are updated and verified.

Ex.No. : 3

Date:

**DML Commands – INSERT, SELECT, UPDATE, DELETE**

**Aim:**

To perform Data Manipulation Language (DML) Commands such as INSERT, SELECT, UPDATE, DELETE in the table.

**Description:**

Data Manipulation Language (DML) statements are used for managing data within schema objects. DML Commands: Insert , Update, Delete, Select

* + INSERT - insert data into a table
  + UPDATE - updates existing data within a table
*  DELETE - deletes all records from a table, the space for the records remain
  + SELECT - retrieve data from the a database

**INSERT**:

INSERT INTO table\_name

VALUES (value1, value2, value3,...);

( or )

INSERT INTO table\_name (column1, column2, column3,...)

VALUES (value1, value2, value3,...);

**UPDATE:**

UPDATE table\_name

SET column1=value, column2=value2,...

WHERE some\_column=some\_value;

**DELETE:**

DELETE FROM table\_name

WHERE some\_column=some\_value;

**SELECT:**

SELECT column\_name(s)

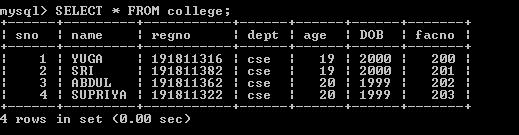
FROM table\_name;

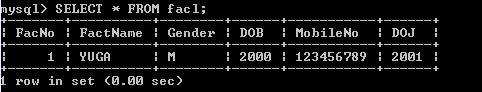
**Questions**:

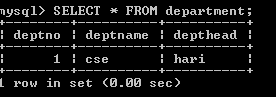
1. Populate all the five tables with your own data.
2. Update the value of student name whose register number is ‘191711342’
3. Delete the record in the table FACULTY, who resigned her job.
4. Modify the date of birth for the faculty whose name is 'RAM' with a value ‘1983-05-01’.
5. Remove all faculty who are having over 65 years
6. View all the records from the five tables.

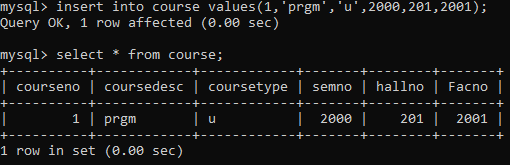
**OUTPUTS:**

1)









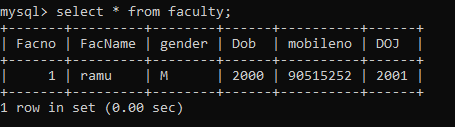
2)



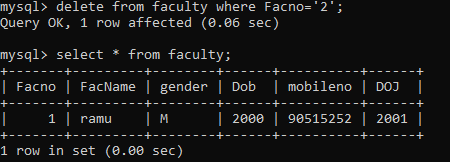
3)



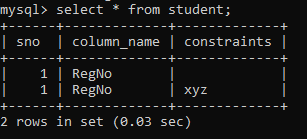
4)

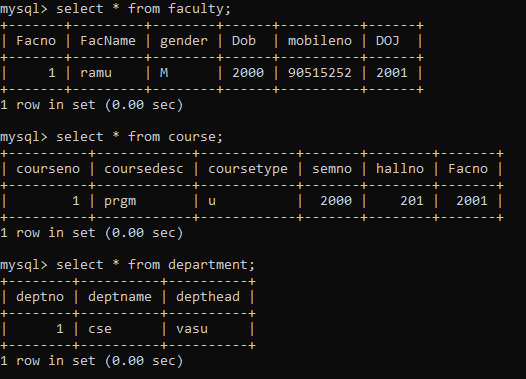


5)



6)





**RESULT**:

Data Manipulation Language (DML) Commands such as INSERT, SELECT, UPDATE, DELETE are performed in the five tables.

‘

Ex. No.: 4

Date:

**SELECT with various clause – WHERE, pattern matching**

**AIM:**

To view the records from the tables using SELECT commands with WHERE Clause and Pattern matching.

**DESCRIPTION**:

The SELECT statement allows you to get the data from tables. A table consists of rows and columns like a spreadsheet. Often, you want to see a subset rows, a subset of columns, or a combination of two. The result of the SELECT statement is called a result set that is a list of rows, each consisting of the same number of columns.

**SELECT:**

SELECT

column\_1, column\_2, ...

FROM

table\_1

[INNER | LEFT |RIGHT] JOIN table\_2 ON conditions

WHERE

conditions

GROUP BY column\_1

HAVING group\_conditions

ORDER BY column\_1

LIMIT offset, length;

The SELECT statement consists of several clauses as explained in the following list:

* SELECT followed by a list of comma-separated columns or an asterisk (\*)

to indicate that you want to return all columns.

* FROM specifies the table or view where you want to query the data.
* JOIN gets related data from other tables based on specific join conditions.
* WHERE clause filters row in the result set.
* GROUP BY clause groups a set of rows into groups and applies aggregate

functions on each group.

* HAVING clause filters group based on groups defined by GROUP BY clause.
* ORDER BY clause specifies a list of columns for sorting.
* LIMIT constrains the number of returned rows.

**Questions:**

**WHERE:**

1. The student counsellor wanted to display the registration number, student name and date of birth for all the students.
2. The controller of examinations wanted to list all the female students
3. Who are the boy students registered for course with the course number “C001“
4. Display all faculty details joined before “November 2014”
5. Display all the courses not allotted to halls

**LIKE:**

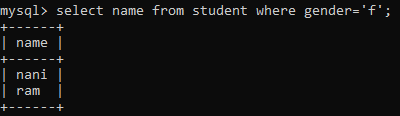
1. List the students whose name ends with the substring “ma”
2. Display all students whose name contains the substring “ma”
3. Find all the students who are located in cities having “Sal” as substring
4. Display the students whose names do not contain six letters.
5. Find all the students whose names contains “th”

**OUTPUTS:**

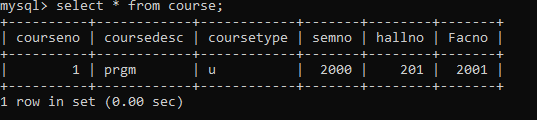
1)



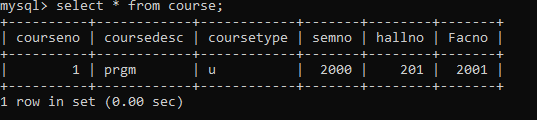
2)



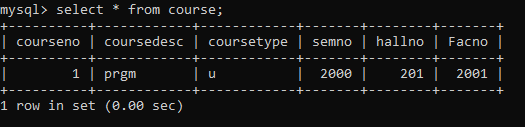
3)



4)



5)



6)



7)



**LIKE:**

The LIKE operator is commonly used to select data based on patterns. Using the LIKE operator in the right way is essential to increase the query performance.

The LIKE operator allows you to select data from a table based on a specified pattern. Therefore, the LIKE operator is often used in the WHERE clause of the SELECT statement.

MySQL provides two wildcard characters for using with the LIKE operator, the percentage % and underscore \_ .

The percentage ( % ) wildcard allows you to match any string of zero ormore characters.

The underscore ( \_ ) wildcard allows you to match any single character.

8)



9)



10)



**RESULT**:

The records from the tables are displayed using SELECT commands with WHERE Clause and Pattern matching.

Ex. No. : 5

Date:

**SELECT with various clause – BETWEEN, IN, Aggregate function**

**AIM:**

To view the records from the tables using SELECT commands with BETWEEN, IN, Aggregate functions.

**DESCRIPTION**:

The BETWEEN operator allows you to specify a range to test. We often use the BETWEEN operator in the WHERE clause of the SELECT, INSERT, UPDATE, and DELETE statements.

The IN operator allows you to determine if a specified value matches any one of a list or a sub query.

MySQL provides many aggregate functions that include AVG, COUNT, SUM, MIN, MAX, etc. An aggregate function ignores NULL values when it performs calculation except for the COUNT function.

**BETWEEN operator**:

SELECTcolumn1,column2,...

FROMtable\_name

WHERE expr [NOT] BETWEEN begin\_expr AND end\_expr;

The *expr* is the expression to test in the range that is defined by *begin\_expr* and *end\_expr*.

**IN operator:**

SELECT

column1,column2,...

FROM

table\_name

WHERE (expr|column\_1) IN ('value1','value2',...);

**Questions:**

**IN & BETWEEN**

1. List the type of the courses “Statistics” and “Programming”
2. The instructor wants to know the CourseNos whose scores are in the range 50

to 80

**AGGREGATE**

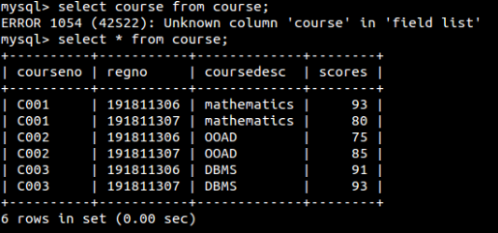
1. Find the average mark of “C002”.
2. List the maximum, minimum mark for “C021”
3. List the maximum, minimum, average mark for each subject in 5th semester
4. List the name of the courses and average mark of each courses.
5. Calculate the sum of all the scores.
6. How many students are registered for each course? Display the

course description and the number of students registered in each course.

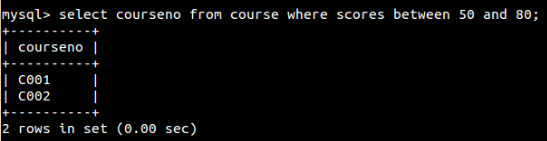
1. How many courses did each student register for? Use Assessment table.

**OUTPUTS:**

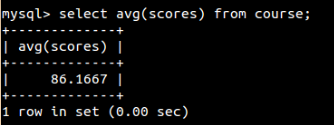
1)



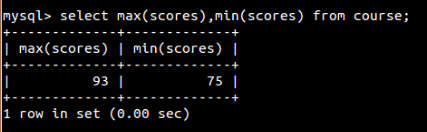
2)



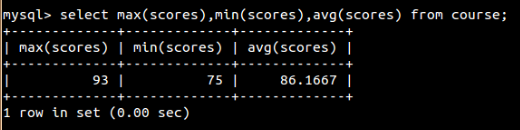
1)



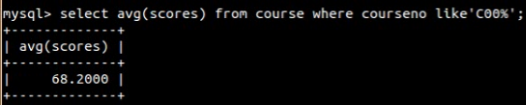
2)



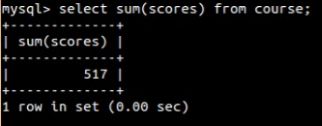
3)



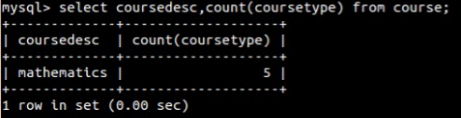
4)



5)



6)



7)



**RESULT**:

The records from the tables are displayed using SELECT commands with WHERE Clause and Pattern matching.

Ex. No.: 6

Date:

**SELECT with various clause – GROUP BY, HAVING, ORDER BY**

**AIM:**

To view the records from the tables using SELECT commands with Group By, Having, Order By

**DESCRIPTION**:

**GROUP BY – HAVING:**

The GROUP BY clause groups a set of rows into a set of summary rows by values of columns or expressions. The GROUP BY clause returns one row for each group. In other words, it reduces the number of rows in the result set.

The GROUP BY clause is used with aggregate functions such as SUM, AVG, MAX, MIN, and COUNT. The aggregate function that appears in the SELECT clause provides the information about each group.

The GROUP BY clause is an optional clause of the SELECT statement.

To filter the groups returned by GROUP BY clause, you use a HAVING clause.

**ORDER BY:**

When you use the SELECT statement to query data from a table, the result set is not sorted in any orders. To sort the result set, you use the ORDER BY clause. The ORDER BY clause allows you to:

* Sort a result set by a single column or multiple columns.
* Sort a result set by different columns in ascending or descending order.

**SYNTAX**:

**GROUP BY – HAVING:**

**SELECT**c1, c2,...,cn, aggregate\_function(ci)

**FROM**table

**WHERE**where\_conditions

**GROUP BY** c1 , c2,...,cn

**HAVING**conditionS

**Questions**:

**GROUP BY - HAVING**

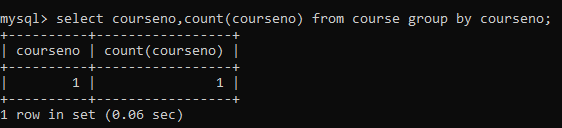
1. How many students are registered for each course? Display the course description and the number of students registered in each course.
2. How many courses did each student register for? Use Assessment table.

**ORDER BY**

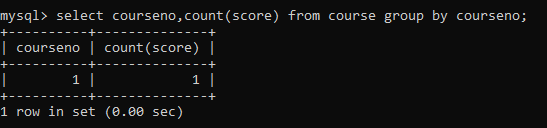
1. Retrieve Name, Gender, MobileNo of all the students in ascending order of RegNo.
2. List the faculty members in the order of older faculty first.

**OUTPUTS:**

**1)**

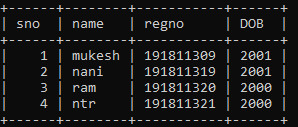


2)

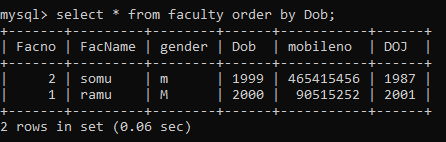


3)





4)



**ORDER BY:**

SELECT column1, column2,...

FROM tbl

ORDER BY column1 [ASC|DESC], column2 [ASC|DESC],...

ASC stands for ascending and the DESC stands for descending. By default, the ORDER BY clause sorts the result set in ascending order if you don’t specify ASC or DESC explicitly.

**RESULT**:

The records from the tables are displayed using SELECT commands with GROUP BY, HAVING and ORDER BY.

Ex. No.: 7

Date:

**SubQuery& Correlated Query**

**AIM:**

To perform subquery and correlated query on the given relation.

**DESCRIPTION**:

**SUBQUERY**

A MySQL subquery is a query nested within another query such as SELECT, INSERT, UPDATE or DELETE. In addition, a MySQL subquery can be nested inside another subquery.

A MySQL subquery is called an inner query while the query that contains the subquery is called an outer query. A subquery can be used anywhere that expression is used and must be closed in parentheses.

**CORRELATED QUERY:**

A correlated subquery is a subquery that uses the data from the outer query. In other words, a correlated subquery depends on the outer query. A correlated subquery is evaluated once for each row in the outer query.

**SYNTAX**:

**SUBQUERY:**

**SELECT**c1, c2,...,cn

**FROM**table

**WHERE**c1 IN (SELECTc1, c2,...,cn

FROMtable

WHEREwhere\_conditions);

**CORRELATED QUERY:**

SELECT\*

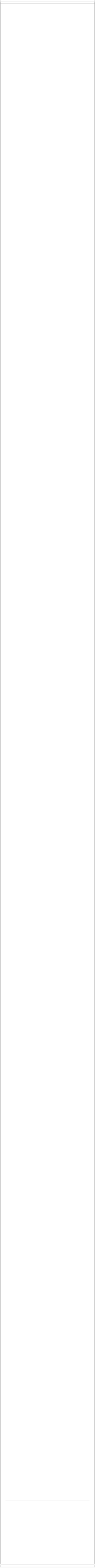
FROMtable\_name

WHEREEXISTS(subquery );

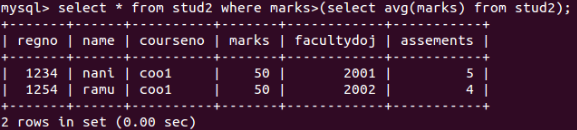
**Questions**:

**Sub-Query and Correlated Sub-Query:**

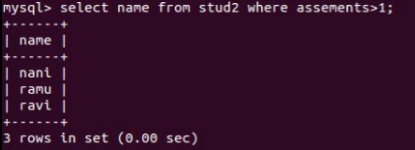
1. Which of the student’s score is greater than the average score?
2. Which of the students’ have written more than one assessment test?
3. Which faculty has joined recently and when?
4. List the course and score of assessments that have the value more than the average score each Course

**OUTPUTS:**

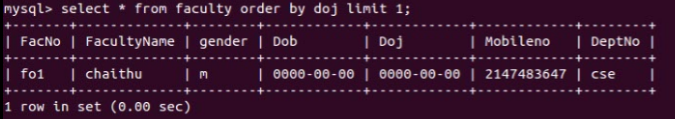
1) Which of the student’s score is greater than the avg score?



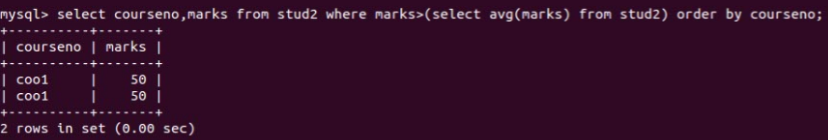
2) Which of the students’ have written more than one assessment test?



3) Which faculty has joined recently and when?



4) List the course and score of assessments that have the value more than the average score each Course



**RESULT**:

The records from the tables are displayed using Sub-Query and Correlated Sub-Query.

Ex. No.: 8

Date:

**Joins – EquiJoin, InnerJoin, OuterJoin**

**AIM:**To perform JOIN using EquiJoin, InnerJoin, OuterJoin on the given relation.

**DESCRIPTION**:

**JOIN**

A MySQL join is a method of linking data from one or more table based on values of the common column between tables.

MySQL supports the following types of joins:

1. [Cross join](http://www.mysqltutorial.org/www.mysqltutorial.org/mysql-cross-join/)
2. [Inner join](http://www.mysqltutorial.org/www.mysqltutorial.org/mysql-inner-join.aspx)
3. [Left join](http://www.mysqltutorial.org/www.mysqltutorial.org/mysql-left-join.aspx)
4. [Right join](http://www.mysqltutorial.org/www.mysqltutorial.org/mysql-right-join/)

**CROSS JOIN**

The CROSS JOIN makes a Cartesian product of rows from multiple tables. Suppose, you join t1 and t2 tables using the CROSS JOIN, the result set will include the combinations of rows from the t1 table with the rows in the t2 table.

**INNER JOIN**

To join two tables, the INNER JOIN compares each row in the first table with each row in the second table to find pairs of rows that satisfy the join-predicate. Whenever the join-predicate is satisfied by matching non-NULL values, column values for each matched pair of rows of the two tables are included in the result set.

**LEFT JOIN**

Unlike an INNER JOIN, a LEFT JOIN returns all rows in the left table including rows that satisfy join-predicate and rows do not. For the rows that do not match the join-predicate, NULLs appear in the columns of the right table in the result set.

**RIGHT JOIN**

A RIGHT JOIN is similar to the LEFT JOIN except that the treatment of tables is reversed. With a RIGHT JOIN, every row from the right table ( t2) will appear in the result set. For the rows in the right table that do not have the matching rows in the left table ( t1), NULLs appear for columns in the left table ( t1).

**SYNTAX**:

**CROSS JOIN:**

**SELECT**t1.id, t2.id

FROMt1**CROSS JOIN**t2;

**INNER JOIN:**

**SELECT**t1.id, t2.id

**FROM**t1**INNER JOIN**t2 **ON** t1.pattern **=** t2.pattern;

**LEFT JOIN:**

**SELECT**t1.id, t2.id

**FROM**t1**LEFT JOIN**t2 **ON** t1.pattern = t2.pattern

**ORDER BY t1.id;**

**RIGHT JOIN**:

**SELECT**t1.id, t2.id

**FROM**t1**RIGHT JOIN**t2 **ON** t1.pattern = t2.pattern

**ORDER BYt2.id;**

**Questions**:

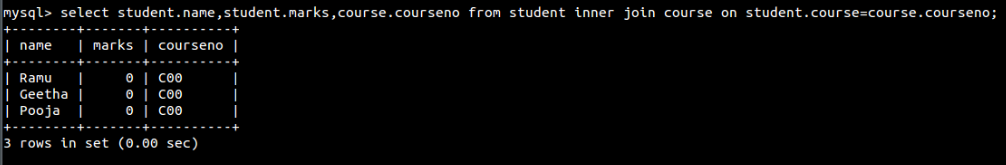
1. List the departments where the faculty members are working.
2. Find the student who has no score in any of the courses. List student name and course number.
3. The office clerk needs the names of the courses taken by the faculty belonging to ‘ECE department’ whose name is ‘Kamal’

**OUTPUTS:**

1. List the departments where the faculty members are working.



1. Find the student who has no score in any of the courses. List student name and course number.



3) The office clerk needs the names of the courses taken by the faculty belonging to ‘Sales’ whose name is ‘Ratnam’



**RESULT**: The records from the tables are displayed using JOIN using EquiJoinInnerJoin, OuterJoin.

Ex. No.: 9

Date:

**VIEW, INDEX, SEQUENCE**

**AIM:**

To create view, index and sequence on the given relation.

**DESCRIPTION:**

**VIEW**

MySQL has supported database views since version 5+. In MySQL, almost features of views conform to the SQL: 2003 standard. MySQL processes query against the views in two ways:

1.In a first way, MySQL creates a temporary table based on the view definition

2.statement and executes the incoming query on this temporary table.

3.In a second way, MySQL combines the incoming query with the query defined the view into one query and executes the combined query.

**INDEX:**

A database index, or just index, helps **speed up the retrieval of data from tables**. When you query data from a table, first MySQL **checks if the indexes exist**, then MySQL uses the indexes to select exact physical corresponding rows of the table instead of scanning the whole table..

**SEQUENCE:**

In MySQL, a sequence is a **list of integers generated in the ascending order i.e., 1,2,3…** Many applications need sequences to generate unique numbers mainly for identification e.g., customer ID in CRM, employee numbers in HR, equipment numbers in services management system, etc.

To create a sequence in MySQL automatically, you set the AUTO\_INCREMENT attribute to a column, which typically is a primary key column.

**SYNTAX:**

**VIEW:**

**CREATE**[ALGORITHM = {MERGE | TEMPTABLE | UNDEFINED}]

**VIEW**[**database\_name**].[view\_name]

**AS**[SELECT statement]

**INDEX:**

**CREATE** [UNIQUE|FULLTEXT|SPATIAL] **INDEX**index\_name

**USING** [BTREE | HASH | RTREE]

**ON** table\_name (column\_name [(length)] [ASC | DESC],...)

**SEQUENCE:**

**CREATE TABLE**table\_name(

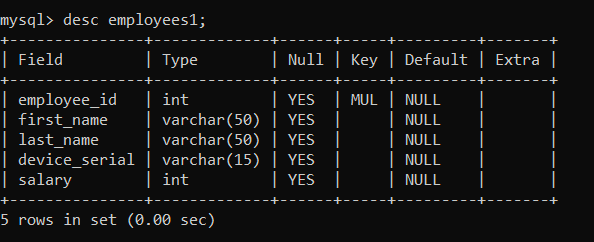
col\_name1 **AUTO\_INCREMENT PRIMARY**

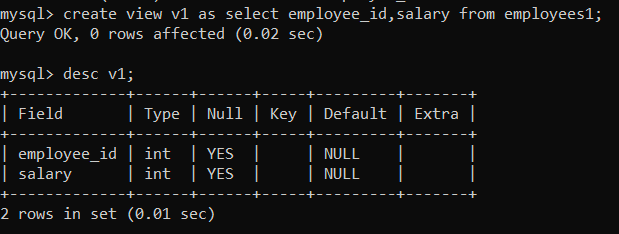
**KEY,** col\_name2,col\_name3, ….);

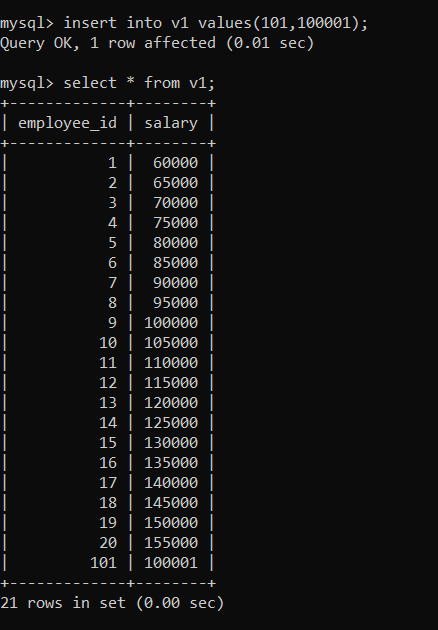
**Questions:**

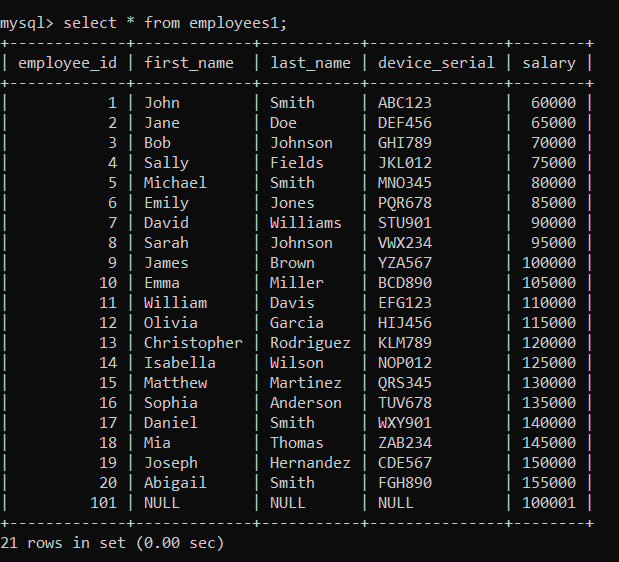
1. Create a view with name ‘v1’ using employees1 table which holds the value of employee\_id and salary of employee.
2. Do the insert and delete records from v1 table.



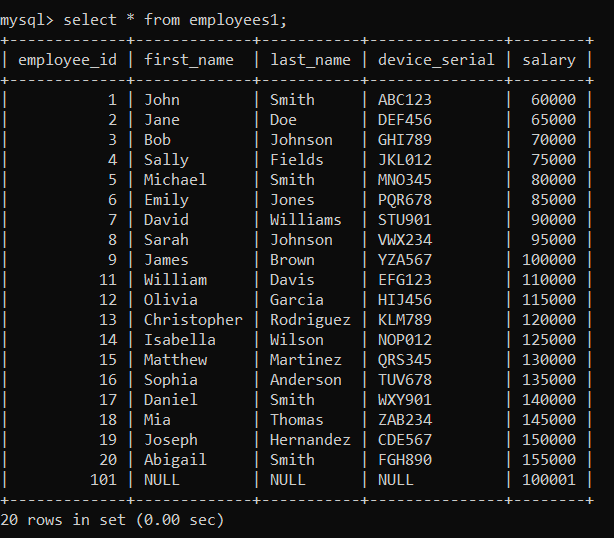








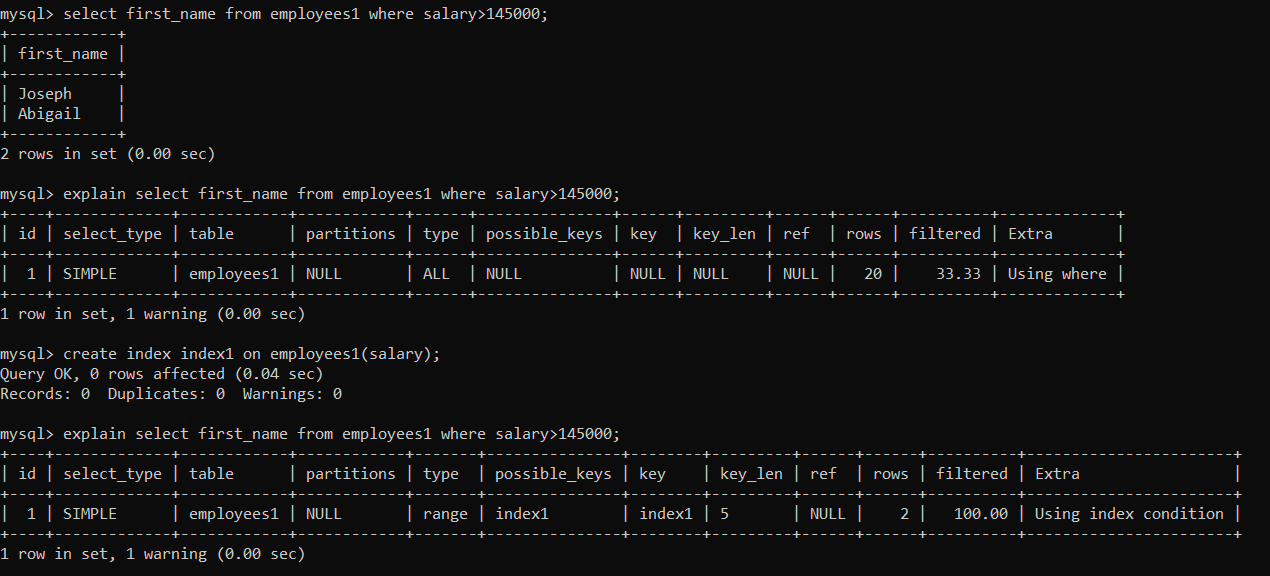


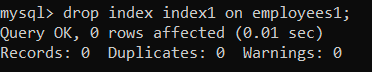


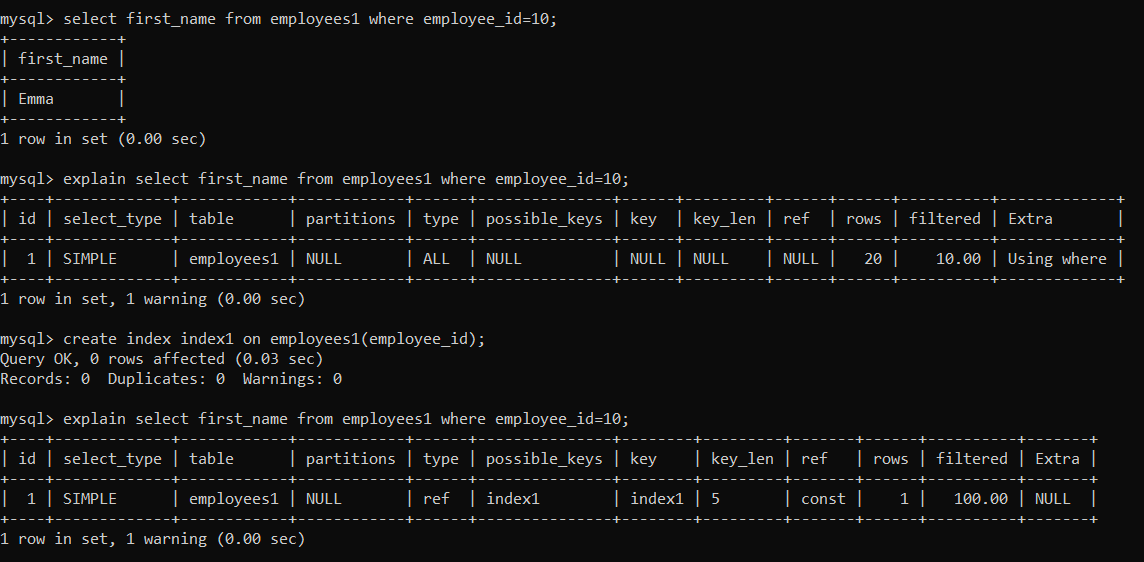
**INDEX:**

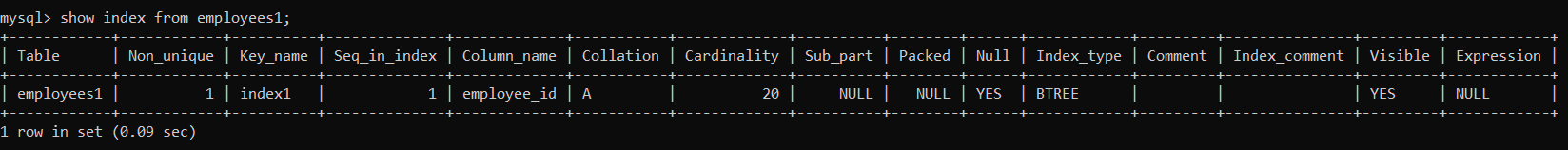
1. Create index1 for ‘salary’ attribute from employees1 relation and list the first name of the employees whose salary is above 145000 and explain the working principle of indexing and then drop the index1.
2. Create index1 for ‘employee\_id’ attribute and display the first name of an employee whose employee id is 10 and explain the working principle of index1.







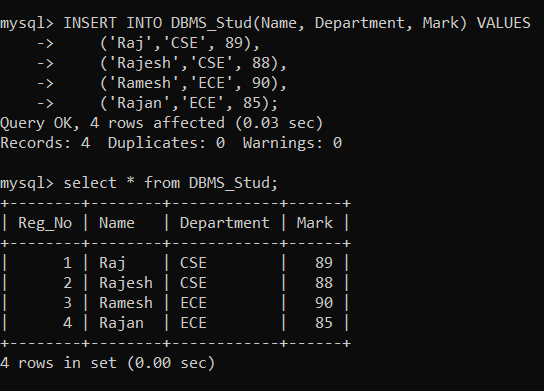


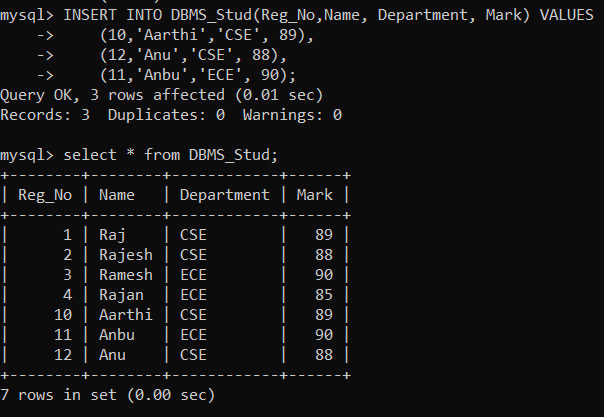


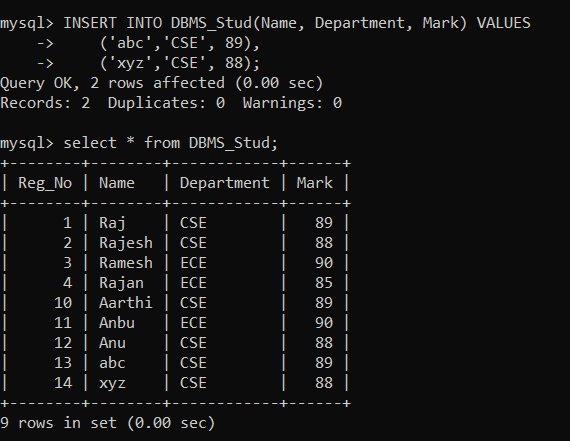
**SEQUENCE:**

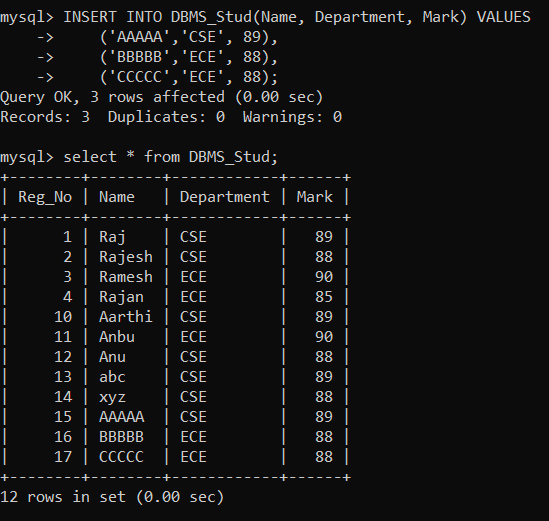
1. **Populateregister number using auto increment in DBMS\_Stud table.**
2. **Manually populate register number**
3. **Drop the auto increment.**

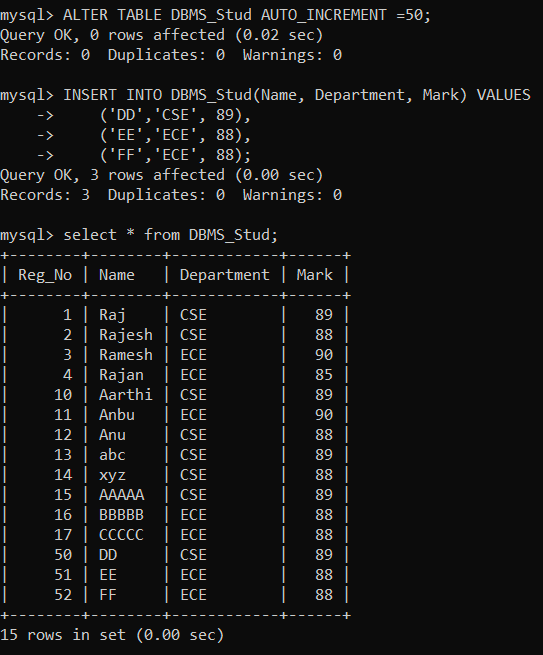


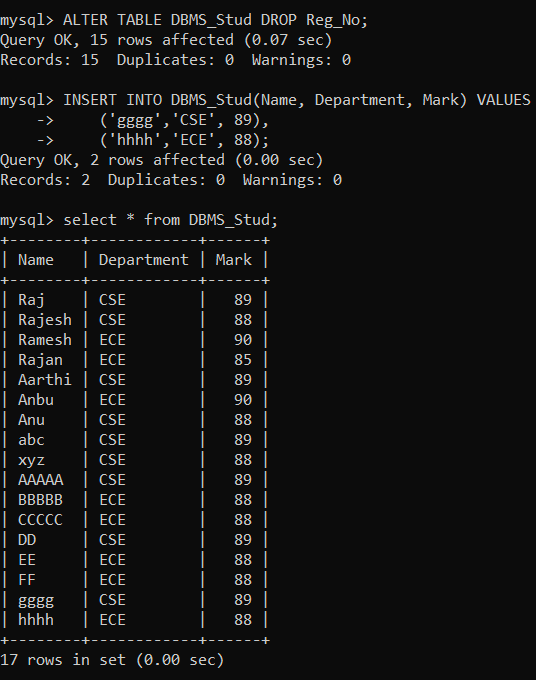


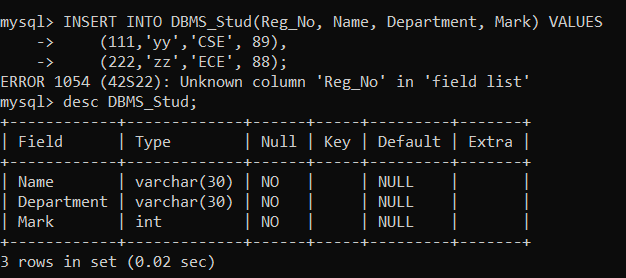












**RESULT**:

The records from the tables are displayed using view, index and sequence on the given relation.

Ex:No: 10

Date:

**Simple programming exercise using(REPEAT, WHILE )**

**Aim:**

To learn how to use various MySQL loop statements including while, repeat to run a block of code repeatedly based on a condition.

**Procedure:**

**WHILE loop**

The syntax of the WHILE statement is as follows:

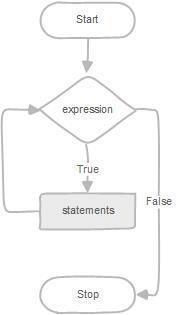
**WHILE** expression

**DO**statements

**END WHILE**

The WHILE loop checks the expression at the beginning of each iteration. If the expression valuates to TRUE, MySQL will execute statements between WHILE and END WHILE until the expression evaluates to FALSE. The WHILE loop is called pretest loop because it checks the expression before the statements execute.

The following flowchart illustrates the WHILE loop statement:



**Program1**

Write a function to build a string repeatedly until the value of the variable becomes s greater than 5. Then, we display the final string using a SELECT statement.

**Procedure:**

REPEAT loop

The syntax of the REPEAT loop statement is as follows:

**REPEAT**statements;

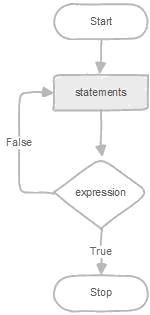
**UNTIL** expression

**END REPEAT**

First, MySQL executes the statements, and then it evaluates the expression. Ifthe expression evaluates to FALSE, MySQL executes the statements repeatedly untilthe expression evaluates to TRUE.

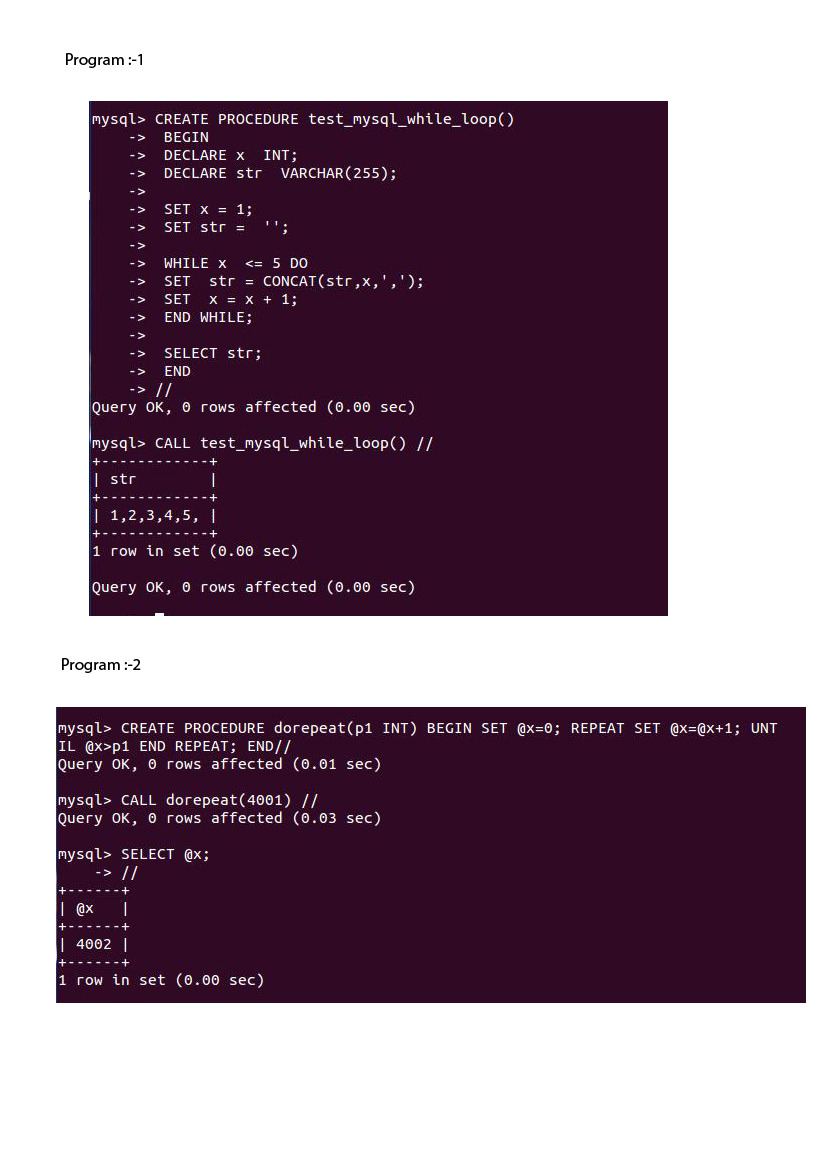
Because the REPEAT loop statement checks the expression after the execution of statements, the REPEAT loop statement is also known as the post-test loop.

The following flowchart illustrates the REPEAT loop statement:



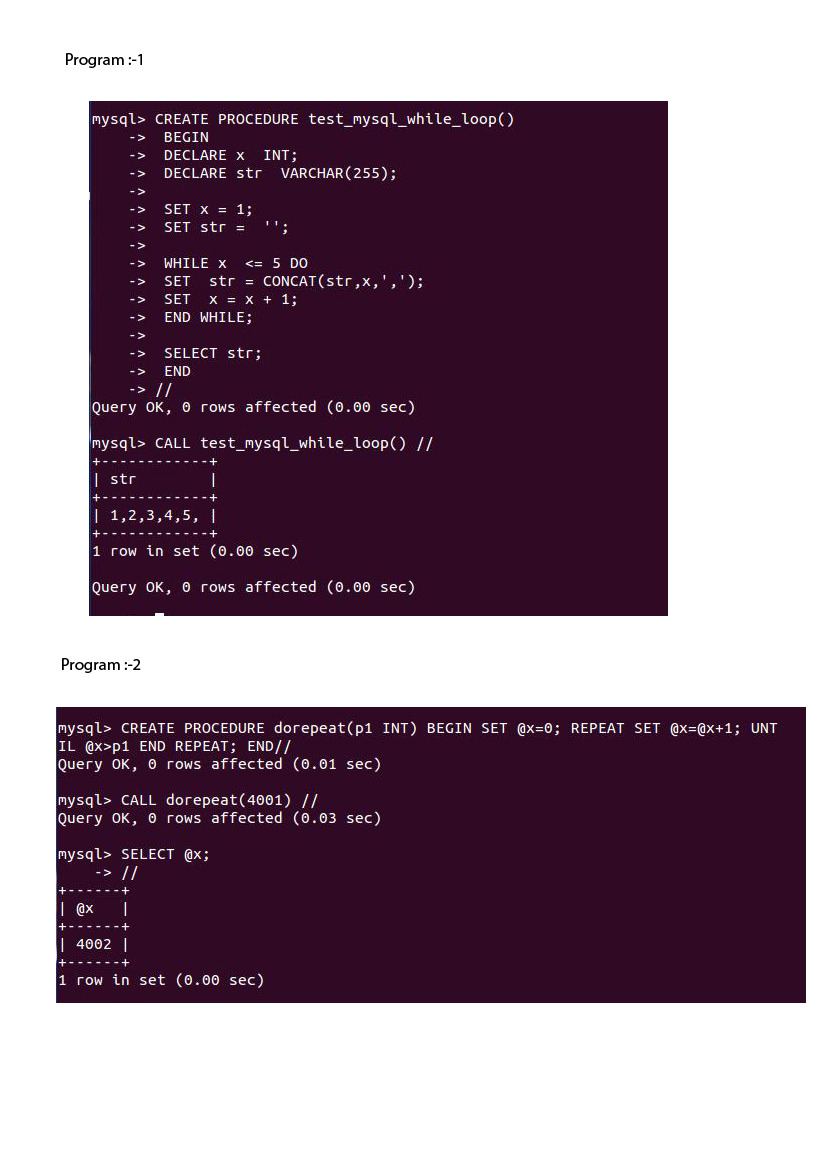
**Program1**

Write a function to build a string repeatedly until the value of the variable becomes **x greater than 5.**Then, we display the final string using a SELECT statement.



**Program 2:**

Write a function that uses REPEAT statement which would repeat the loopuntil income is greater than or equal to 4000, at which point the REPEAT loop would beterminated



**RESULT:**Thus the Simple programming exercise using(REPEAT, WHILE ) executed successfully.

Ex:No: 11

Date:

**Simple programming exercise using(CASE and LOOP)**

**Aim:**

To learn how to use various MySQL loop statements including case and loop to run a block of code repeatedly based on a condition.

**Procedure:**

In MySQL, the CASE statement has the functionality of an IF-THEN-ELSE statement and has 2 syntaxes that we will explore.

CASE Syntax

CASE case\_value

WHEN when\_value THEN statement\_list

[WHEN when\_value THEN statement\_list] ...

[ELSE statement\_list]

END CASE

**Program 1:**

Write a function that uses CASE statement where if monthly\_value is equal to or less than 4000, then income\_level will be set to 'Low Income'. If monthly\_value is equal to or less than 5000, then income\_level will be set to 'Avg Income'. Otherwise, income\_levelwill be set to 'High Income'.

**Procedure:**

LOOP Syntax

[begin\_label:] LOOP

statement\_list

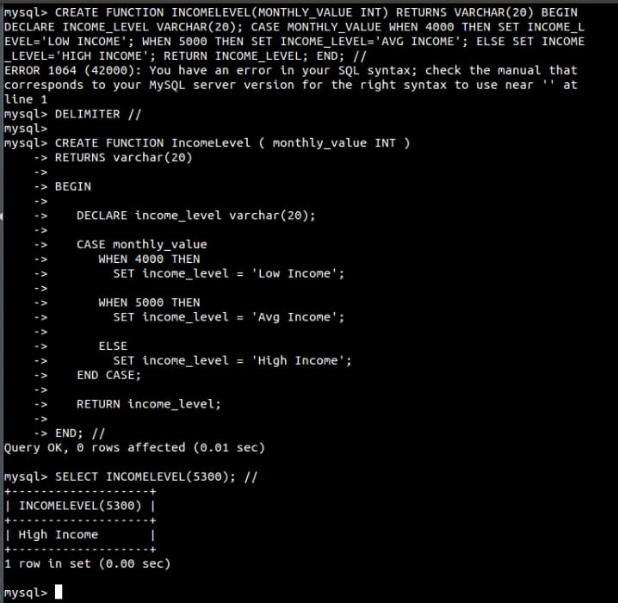
END LOOP [end\_label]

LOOP implements a simple loop construct, enabling repeated execution of the statement list, which consists of one or more statements, each terminated by a semicolon

* statement delimiter. The statements within the loop are repeated until the loop is terminated. Usually, this is accomplished with a LEAVE statement. Within a stored function, RETURN can also be used, which exits the function entirely.

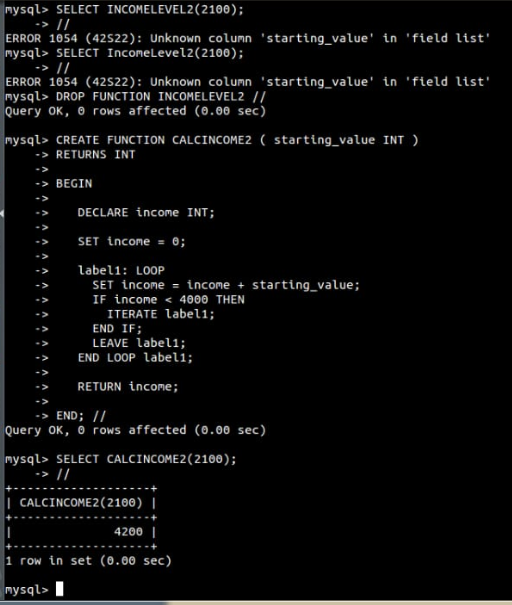
**Program 1:**

Write a function that uses CASE statement where if monthly\_value is equal to or less than 4000, then income\_level will be set to 'Low Income'. If monthly\_value is equal to or less than 5000, then income\_level will be set to 'Avg Income'. Otherwise, income\_level will be set to 'High Income'.



**Program 2:**

Write a function that will use ITERATE statement which would cause the loop to repeat while income is less than 4000. Once income is greater than or equal to 4000, would terminate the LOOP.



**RESULT:**Thus the Simple programming exercise using(CASE and LOOP) excuted successfully.

Ex:No: 12

Date:

**TCL COMMANADS**

**Aim:**

To learn how to use various TCL commands Commit, Rollback and Savepoint

SQL commands

**Procedure and Syntax:**

Transaction Control Language(TCL) commands are used to manage transactions in the database. These are used to manage the changes made to the data in a table by DML statements. It also allows statements to be grouped together into logical transactions.

**COMMIT command**

COMMIT command is used to permanently save any transaction into the database.

When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.

To avoid that, we use the COMMIT command to mark the changes as permanent

**SYNTAX;**

**COMMIT;**

**ROLLBACK command**

This command restores the database to last commited state. It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.

If we have used the UPDATE command to make some changes into the database, and realise that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not commited using the COMMIT command.

**Syntax:**

ROLLBACK TO savepoint\_name;

SAVEPOINT command

SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

**Syntax:**

SAVEPOINT savepoint\_name;

**Problem 1:**

Rollback to that state using the ROLLBACK command whenever required.

Create a following table Class and insert values into it in the order and create savepoints in between them. Try to rollback t the save point and check your output by giving select commands.

Let us use some SQL queries on the above table and see the results.

INSERTINTO class VALUES(5,'Rahul');

COMMIT;

UPDATE class SET name ='Abi' WHERE id ='5';

SAVEPOINT A;

INSERTINTO class VALUES(6,'Raj');

SAVEPOINT B;

INSERTINTO class VALUES(7,'Mary');

SAVEPOINT C;

The resultant table will look like,

Now let's use the ROLLBACK command to roll back the state of data to the savepoint B.

ROLLBACKTO B;

SELECT\*FROM class;

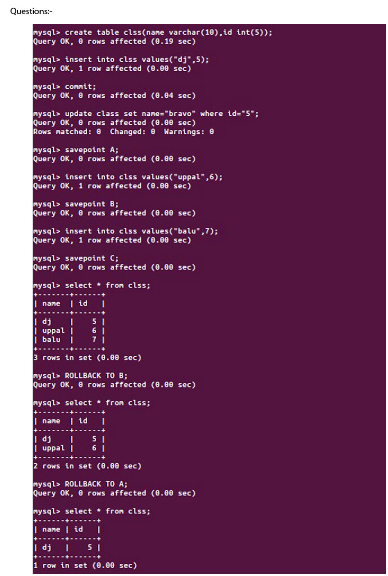
Now our class table will look like,

Now let's again use the ROLLBACK command to roll back the state of data to the savepoint A

ROLLBACKTO A;

SELECT\*FROM class;

Now the table will look like,



**Result:**

So now we know how the commands COMMIT, ROLLBACK and SAVEPOINT works.

Ex:No: 13

Date:

**DCL COMMANADS**

**Aim:**

To learn how to use various DCL commands GRANT and REVOKE SQL

commands

**Procedure and Syntax:**

Data Control Language(DCL) is used to control privileges in Database. To perform any operation in the database, such as for creating tables, sequences or views, a user needs privileges. Privileges are of two types,

**System:** This includes permissions for creating session, table, etc and all types of

other system privileges.

**Object:** This includes permissions for any command or query to perform any operation on the database tables.

In DCL we have two commands,

GRANT: Used to provide any user access privileges or other privileges for

the database.

REVOKE: Used to take back permissions from any user.

Allow a User to create session

When we create a user in SQL, it is not even allowed to login and create a session until and unless proper permissions/privileges are granted to the user.

Following command can be used to grant the session creating privileges.

GRANT CREATE SESSION TO username;

Allow a User to create table

To allow a user to create tables in the database, we can use the below command,

GRANT CREATE TABLE TO username;

Provide user with space on tablespace to store table

Allowing a user to create table is not enough to start storing data in that table. We also must provide the user with privileges to use the available tablespace for their table and data.

ALTER USER username QUOTA UNLIMITED ON SYSTEM;

The above command will alter the user details and will provide it access to unlimited tablespace on system.

NOTE: Generally unlimited quota is provided to Admin users.

Grant all privilege to a User

Sysdbais a set of priviliges which has all the permissions in it. So if we want toprovide all the privileges to any user, we can simply grant them the sysdba permission.

GRANT sysdba TO username

Grant permission to create any table

Sometimes user is restricted from creating come tables with names which are reserved for system tables. But we can grant privileges to a user to create any table using the below command,

GRANT CREATE ANY TABLE TO username

Grant permission to drop any table

As the title suggests, if you want to allow user to drop any table from the database, then grant this privilege to the user,

GRANT DROP ANY TABLE TO username

To take back Permissions

And, if you want to take back the privileges from any user, use the REVOKE command.

REVOKE CREATE TABLE FROM username

**RESULT:** Thus the DCL commands GRANT and REVOKE SQL executed successfully.

Ex:No: 14

Date:

**HIGH LEVEL PROGRAMMING EXTENSIONS**

**(PROCEDURES)**

**Aim:**

To implement procedures using program in MySQL.

**PROCEDURES:**

A procedure is a subprogram that performs a specific action.

**Creating a procedure**

We use the CREATE PROCEDURE statement to create a new stored procedure. We specify the name of stored procedure after the CREATE PROCEDURE statement. The DELIMITER command is used to change thestandard delimiter of MySQLcommands (i.e. ;). As the statements within the routines (functions, stored procedures or triggers) end with a semi-colon (;), to treat them as a compound statement we use DELIMITER.

Calling stored procedures(Executing a procedure)

In order to call a stored procedure, you use the following SQL command:

CALL stored\_procedure\_name();

**Program 1:**

Create a simple procedure to get all the records from the table ‘student\_info’ which have the following data:

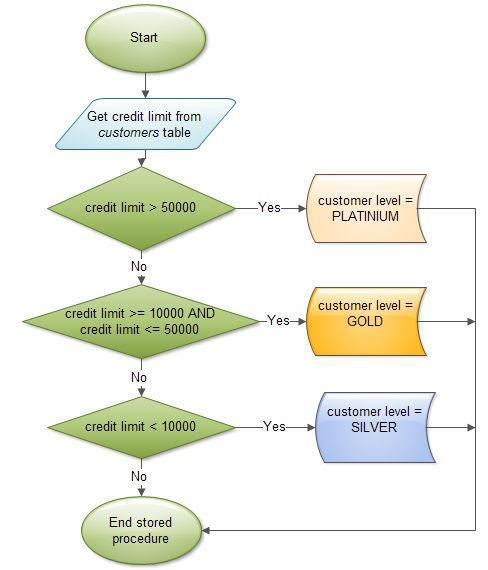
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| mysql> select \* from student\_info; | | | |  |
| +----- | +--------- | +------------ | +------------ | + |
| | id | | Name | | Address | | Subject | | |
| +----- | +--------- | +------------ | +------------ | + |
| | 100 | Aarav | | | Delhi | | Computers | | |
| | 101 | YashPal | Amritsar | History | | | | | |
| | 105 | Gaurav | Jaipur | | | | Literature | |  |
| | 110 | Rahul | | | Chandigarh | History | | | |
| +------ | +-------- | +------------ | +------------ | + |

**Program 2:**

Create a stored procedure GetCustomerLevel() that accepts two parameters customer number and customer level.

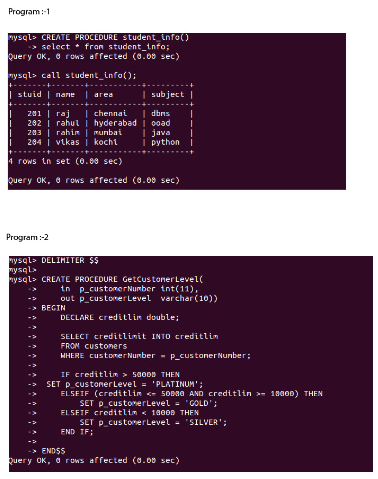
* First, it gets the credit limit from the customers table.
* Then, based on the credit limit, it determines the customer level: PLATINUM, GOLD, and SILVER.
* 
* The parameter p\_customerlevel stores the level of the customer and is used by the calling program.

The following flowchart demonstrates the logic of determining customer level.



The table ‘customers’ should have the following attributes:

customers(cno , cname, creditlimit)



Create a stored procedure GetCustomerLevel() that accepts two parameters customer number and customer level.

* First, it gets the credit limit from the customers table.
* Then, based on the credit limit, it determines the customer level: PLATINUM, GOLD, and SILVER.
* 
* The parameter p\_customerlevel stores the level of the customer and is used by the calling program.

The following flowchart demonstrates the logic of determining customer level.

**RESULT:**Thus the program in MySQL executed successfully.

Ex:No: 15

Date:

**HIGH LEVEL PROGRAMMING EXTENSIONS**

**(FUNCTIONS)**

**Aim:**

To implement Functions using program in MySQL.

**FUNCTIONS:**

A function is a subprogram that computes a value.

Creating a function

The CREATE FUNCTION statement is also used in MySQL to support UDFs (user-defined functions). A UDF can be regarded as an external stored function.

**MySQL stored function syntax**

CREATE FUNCTION function\_name(param1,param2,…)

RETURNS datatype

[NOT] DETERMINISTIC

statements

**Program 1:**

Create a function that returns the level of a customer based on credit limit.(Use the IF statement to determine the credit limit).

The table ‘customers’ should have the following attributes:

customers(cno , cname, creditlimit)

If credit limit > 50000 then customer\_level = PLATINUM

If credit limit >= 10000 AND credit limit <= 50000 then customer\_level = GOLD

If credit limit credit limit < 10000 then customer\_level = SILVER

RECURSION in Mysql Procedures

Mysql version should be >= 5.

Have to set system parameters. This means putting the recursion count limit.

SET @@GLOBAL.max\_sp\_recursion\_depth = 255;

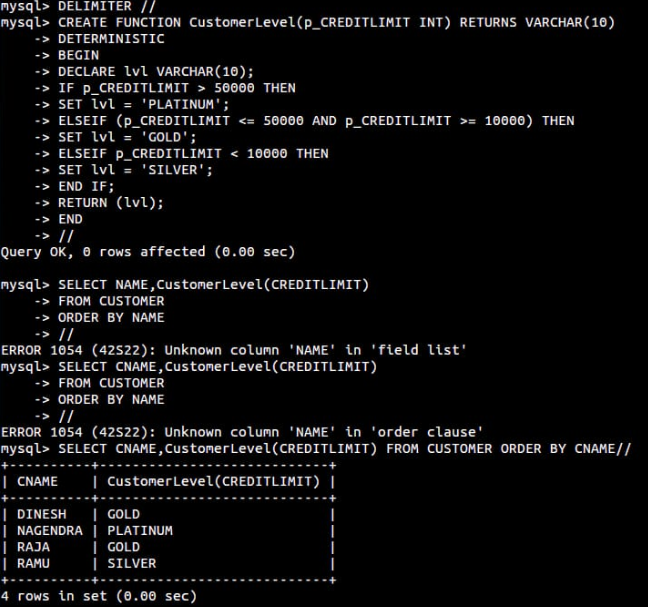
SET @@session.max\_sp\_recursion\_depth = 255;

**Program 2**

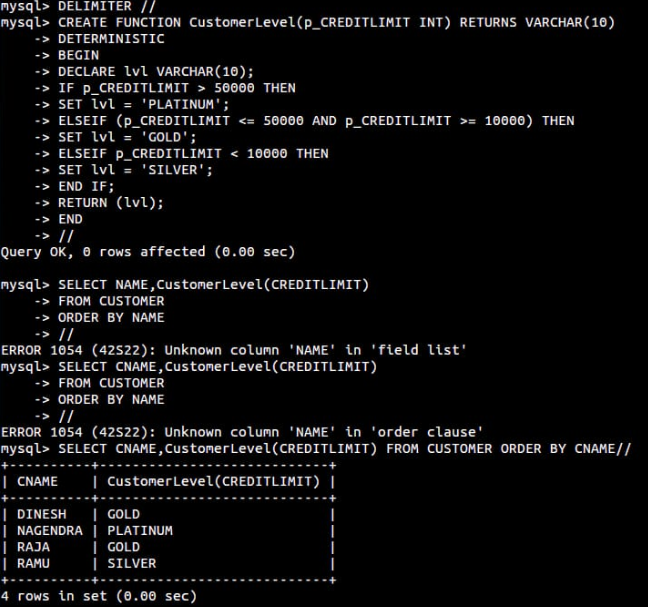
Write a recursive MySQL procedure compute the factorial of a number .

**OUTPUT**

1



2)



**RESULT:** Thus the Functions using program in MySQL executed successfully .

Ex.No: 16

Date:

**HIGH LEVEL LANGUAGE EXTENSION WITH CURSORS**

**Program 1**

Write a Cursor program using MySQL to retrieve the email-ids(build an email list) of employees from employees table.

**SOLUTION :**

create table employees(id integer, Name varchar(100), email varchar(100));

insert into employees(id, Name, email) values(1, "Harry Potter", "pharry@warnerbros.com");

insert into employees(id, Name, email) values(2, "Clark Kent","kclark@dccomics.com");

insert into employees(id, Name, email) values(3, "Tony Stark", "stony@marvel.com");

**Cursor** is a **Temporary Memory** or **Temporary Work Station**. It is Allocated by Database Server at the Time of Performing DML(Data Manipulation Language) operations on Table by User. Cursors are used to store Database Tables.

1. **Implicit Cursors:**  
   Implicit Cursors are also known as Default Cursors of SQL SERVER. These Cursors are allocated by SQL SERVER when the user performs DML operations.
2. **Explicit Cursors :**  
   Explicit Cursors are Created by Users whenever the user requires them. Explicit Cursors are used for **Fetching data from Table in Row-By-Row Manner**.

**How to create Explicit Cursor:**

1. **Declare Cursor Object.**  
   **Syntax :** DECLARE cursor\_name CURSOR FOR SELECT \* FROM table\_name

DECLARE s1 CURSOR FOR SELECT \* FROM studDetails

1. **Open Cursor Connection.**  
   **Syntax :** OPEN cursor\_connection

OPEN s1

1. **Fetch Data from cursor.**  
   There are total 6 methods to access data from cursor. They are as follows :  
   **FIRST** is used to fetch only the first row from cursor table.  
   **LAST** is used to fetch only last row from cursor table.  
   **NEXT** is used to fetch data in forward direction from cursor table.  
   **PRIOR** is used to fetch data in backward direction from cursor table.  
   **ABSOLUTE n** is used to fetch the exact nth row from cursor table.  
   **RELATIVE n** is used to fetch the data in incremental way as well as decremental way.  
   **Syntax :** FETCH NEXT/FIRST/LAST/PRIOR/ABSOLUTE n/RELATIVE n FROM cursor\_name

FETCH FIRST FROM s1

FETCH LAST FROM s1

FETCH NEXT FROM s1

FETCH PRIOR FROM s1

FETCH ABSOLUTE 7 FROM s1

FETCH RELATIVE -2 FROM s1

1. **Close cursor connection.**  
   **Syntax :** CLOSE cursor\_name

CLOSE s1

1. **Deallocate cursor memory.**  
   **Syntax :** DEALLOCATE cursor\_name

DELIMITER $$

CREATE PROCEDURE build\_email\_list (INOUT email\_listvarchar(4000))

BEGIN

DECLARE v\_finished INTEGER DEFAULT 0;

DECLARE v\_emailvarchar(100) DEFAULT "";

-- declare cursor for employee email

DECLARE**email\_cursor** CURSOR FOR

**SELECT email FROM employees;**

-- declare NOT FOUND handler

DECLARE CONTINUE HANDLER FOR

**NOT FOUND** SET v\_finished = 1;

OPEN email\_cursor;

get\_email: LOOP

FETCH email\_cursor INTO v\_email;

IF v\_finished = 1 THEN

LEAVE get\_email;

END IF;

-- build email list

SET email\_list = CONCAT(v\_email,";",email\_list);

END LOOP get\_email;

CLOSE email\_cursor;

END$$

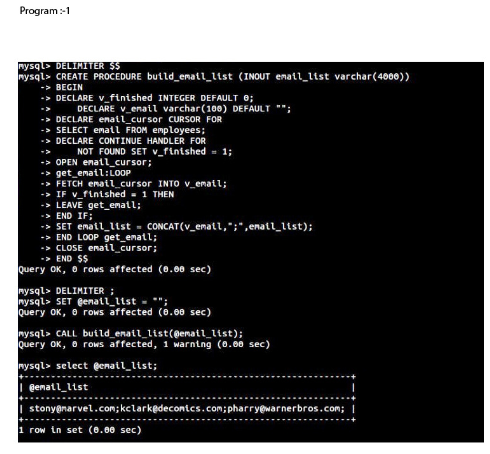
DELIMITER ;

-- Calling the procedure and getting the email list

SET @email\_list = "";

CALL build\_email\_list(@email\_list);

SELECT @email\_list;



**RESULT:**

stony@marvel.com;kclark@dccomics.com;pharry@warnerbros.com;

Ex:No:17

Date:

**TRIGGER**

**Aim:**

To implement trigger in MySQL.

A trigger or database trigger is a stored program **executed automatically**to respond to a specific event e.g., insert, update or delete occurred in a table.

**Create trigger syntax**

CREATE TRIGGER trigger\_nametrigger\_timetrigger\_event

ON table\_name

FOR EACH ROW

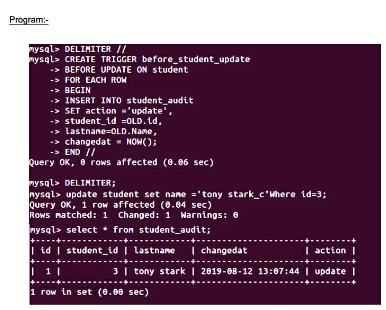
BEGIN

...

END;

**Program 1 :**

Create a trigger in MySQL to log the changes of the EMPLOYEES table with fields ID, Name and Email. Also create a new table named EMPLOYEES\_AUDIT to keep the changes of the employee table. Create a **BEFORE UPDATE trigger** that is invoked before a change is made to the employees table.



**RESULT:** Thus the trigger is executed successfully.

**Exp. No.:18**

**Date:**

**SQL String Functions- REPLACE, REPEAT, RIGHT, RPAD**

**AIM:**

**ASCII()**

This function returns the numeric value of the leftmost character of the string str. Returns 0 if str is the empty string. Returns NULL if str is NULL  
Syntax : ASCII(str)  
Example : SELECT ASCII('2');  
Output : 50  
Example : SELECT ASCII(2);  
Output : 50  
Example : SELECT ASCII(‘An’);  
Output : 65

**BIN()**

Returns a string representation of the binary value of N, where N is a longlong (BIGINT) number. Returns NULL if N is NULL.  
Syntax : BIN(N)  
Example : SELECT BIN(12);  
Output : 1100

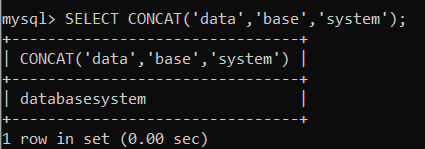
**BIT\_LENGTH()**

Returns the length of the string str in bits.  
Syntax : BIT\_LENGTH(str)  
Example : SELECT BIT\_LENGTH('text');  
Output : 32

**CHAR\_LENGTH()**

Returns the length of the string str, measured in characters. A multi-byte character counts as a single character. This means that for a string containing five 2-byte characters, LENGTH() returns 10, whereas CHAR\_LENGTH() returns 5.  
Syntax : CHAR\_LENGTH(str)  
Example : SELECT CHAR\_LENGTH('test string');  
Output : 11

**CONCAT()**

Returns the string that results from concatenating one or more arguments. If all arguments are nonbinary strings, the result is a nonbinary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent nonbinary string form.  
Syntax : CONCAT(str1,str2,...)  


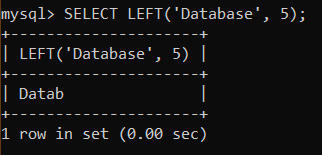
**INSTR()**

MySQL INSTR() takes a string and a substring of it as arguments, and returns an integer which indicates the position of the first occurrence of the substring within the string  
Syntax : INSTR(str,substr)  
Example : SELECT INSTR('myteststring','st');  
Output : 5

**LCASE()**

MySQL LCASE() converts the characters of a string to lower case characters.  
Syntax : LCASE(str)  
Example : SELECT LCASE('MYTESTSTRING');  
Output : myteststring

**LEFT()**

MySQL LEFT() returns a specified number of characters from the left of a given string. Both the number and the string are supplied in the arguments as str and len of the function.  
Syntax : LEFT(str,len)  


**LENGTH()**

MySQL LENGTH() returns the length of a given string.  
Syntax : LENGTH(str)  
Example : SELECT LENGTH('text');  
Output : 4

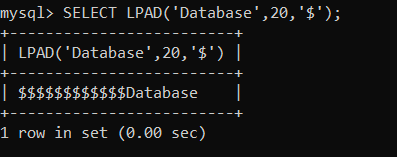
**LOCATE()**

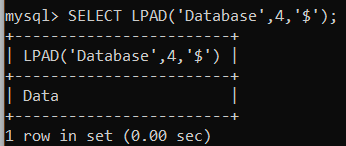
MySQL LOCATE() returns the position of the first occurrence of a string within a string. Both of these strings are passed as arguments. An optional argument may be used to specify from which position of the string (i.e. string to be searched) searching will start. If this position is not mentioned, searching starts from the beginning.  
Syntax : LOCATE(substr,str,pos)  
Example : SELECT LOCATE('st','myteststring');  
Output : 5

**LOWER()**

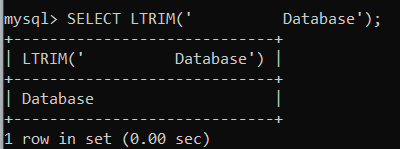
MySQL LOWER() converts all the characters in a string to lowercase characters.  
Syntax : LOWER(str)  
Example : SELECT LOWER('MYTESTSTRING');  
Output : myteststring

**LPAD()**

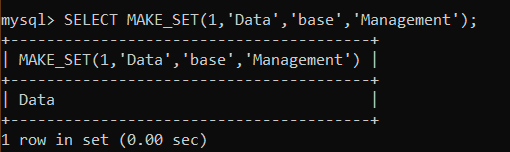
MySQL LPAD() left pads a string with another string. The actual string, a number indicating the length of the padding in characters (optional) and the string to be used for left padding - all are passed as arguments.  
Syntax : LPAD(str,len,padstr)  




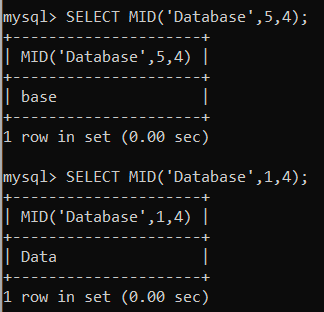
**LTRIM(str)**

MySQL LTRIM() removes the leading space characters of a string passed as argument.  
Syntax : LTRIM(str)  


**MAKE\_SET()**

MySQL MAKE\_SET() returns a set value (a string containing substrings separated by “,” characters) consisting of the strings that have the corresponding bit in the first argument.  
Syntax : MAKE\_SET(bits,str1,str2,...)  


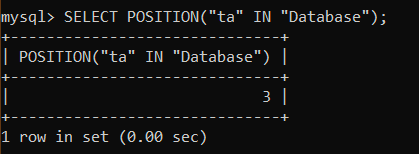
**MID()**

MySQL MID() extracts a substring from a string. The actual string, position to start extraction and length of the extracted string - all are specified as arguments.  
Syntax : MID(str,pos,len)  


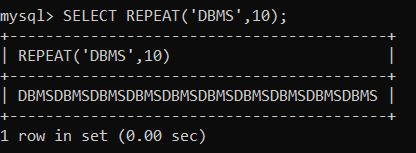
**OCT()**

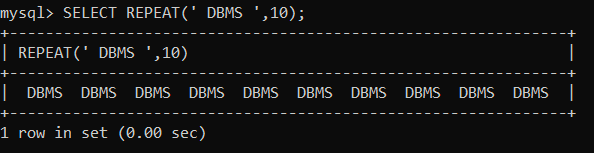
Returns a string representation of the octal value of N, where N is a longlong (BIGINT) number. Returns NULL if N is NULL.  
Syntax : OCT(N)  
Example : SELECT OCT(12);  
Output : 14

**POSITION()**

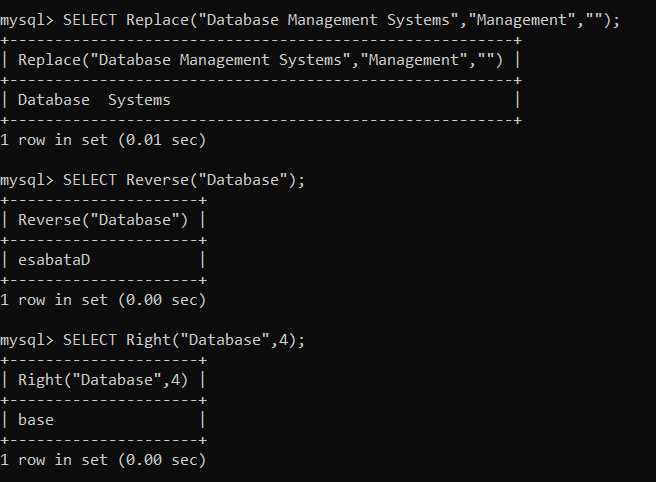
MySQL POSITION() returns the position of a substring within a string..  
Syntax : POSITION(substr IN str)  


**REPEAT()**

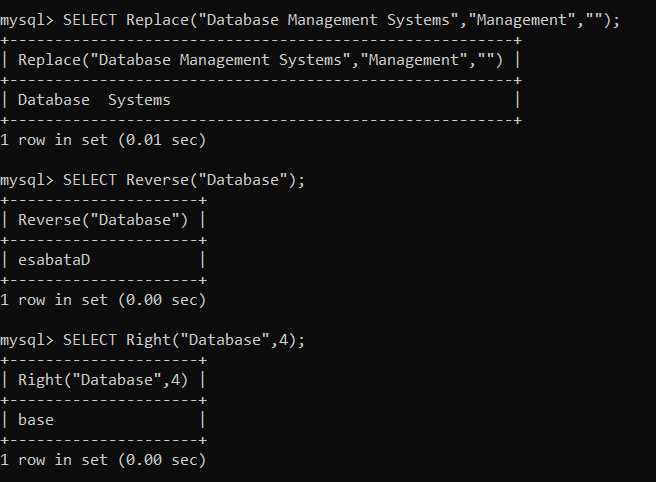
MySQL REPEAT() repeats a string for a specified number of times.  
The function returns NULL either any either of the arguments are NULL.  
Syntax : REPEAT(str,count)  




**REPLACE()**

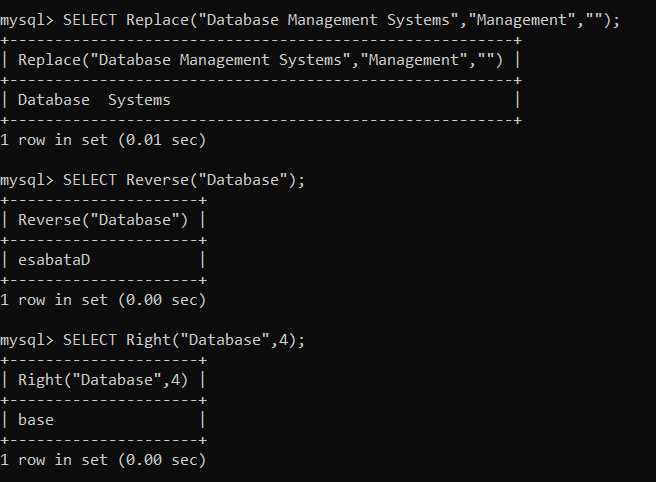
MySQL REPLACE() replaces all the occurrences of a substring within a string.  
Syntax : REPLACE(str,from\_str,to\_str)  


**REVERSE()**

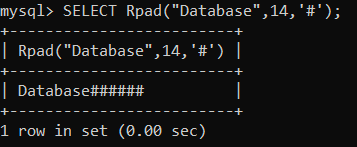
Returns a given string with the order of the characters reversed.  
Syntax : REVERSE(str)  


**RIGHT()**

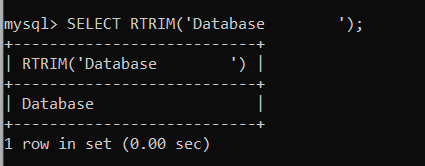
MySQL RIGHT() extracts a specified number of characters from the right side of a given string.  
Syntax : RIGHT(str,len)



**RPAD()**

MySQL RPAD() function pads strings from right. The actual string which is to be padded as str, length of the string returned after padding as len and string which is used for padding as padstr is used as a parameters within the argument.  
Syntax : RPAD(str,len,padstr)  


**RTRIM()**

MySQL RTRIM() removes the trailing spaces from a given string.  
Syntax : RTRIM(str)  


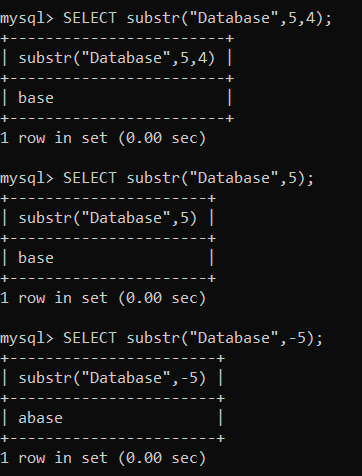
**SPACE()**

MySQL SPACE() returns the string containing a number of spaces as specified in the argument.  
Syntax : SPACE(N)  

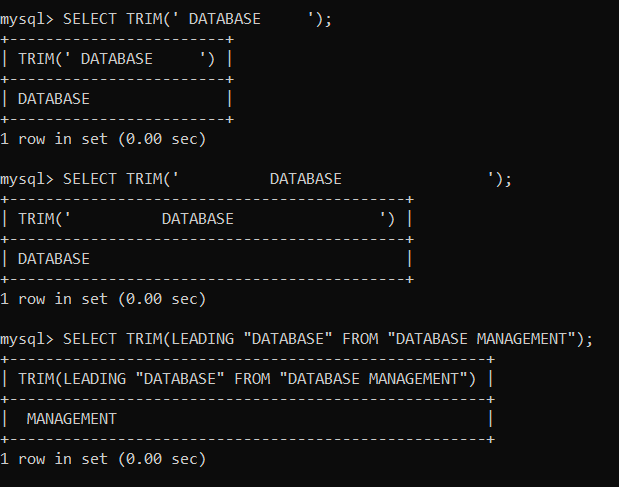

**SUBSTRING() / SUBSTR()**

**MySQL SUBSTRING()** returns a specified number of characters from a particular position of a given string.

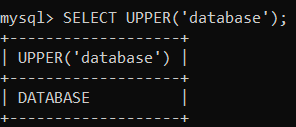
Syntax : SUBSTRING(str,pos,len)

**MySQL SUBSTR()** returns the specified number of characters from a particular position of a given string. SUBSTR() is a synonym for SUBSTRING().  
Syntax : SUBSTR(str,pos,len)  
  


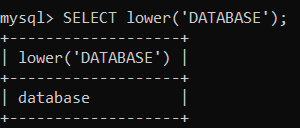
**TRIM()**

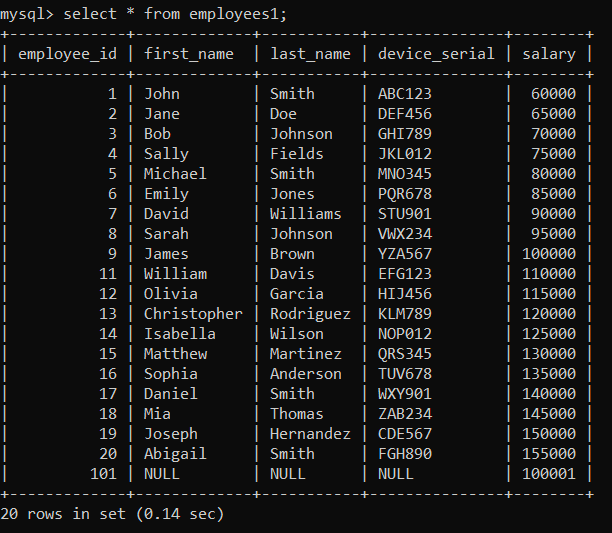
MySQL TRIM() function returns a string after removing all prefixes or suffixes from the given string.  
Syntax : TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] str)  


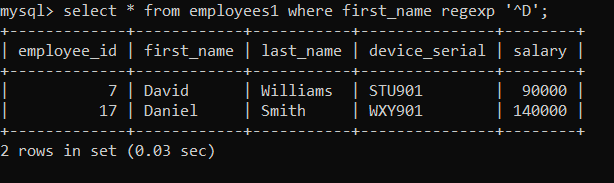
**UPPER()**

MySQL UPPER() converts all the characters in a string to uppercase characters.  
Syntax : UPPER(str)  


**LOWER()**

MySQL **LOWER**() converts all the characters in a string to lowercase characters.  
Syntax: **LOWER** (str);  






**Exp No:20**

**Date:**

**DATABASECONNECTIVITYUSINGPHP AND MYSQL**

**AIM:**

ToconnecttheMysqlusingPHPandMYSQLandtoexecutetheCREATE,INSERT,SELECTcommandinMySQL

**PROGRAM:**

<?php

$host=

$password=””;

$conn=mysqli\_connect($host,$user,$password);

if(!$conn)

{

Die(‘couldnot connect:’,mysql\_connect\_error());

}

echo”connectsuccessfully(‘br/>’)”;

$sql=’Create database mydb’;

$sql="create table emp(id int,namevarchar(10) NOT NULL,empsalary INT NOT NULL,primary key(id))";

$sql=”insert into emp(id,name,empsalary) values(312,RANA,200000)”;

$sql=”delete from emp where id=1”;

$sql=”updateemp set empsalary=9000000 where id=312”;

if(mysqli\_query($conn,$sql))

{echo ”operations failed failed”,mysqli\_error($conn);

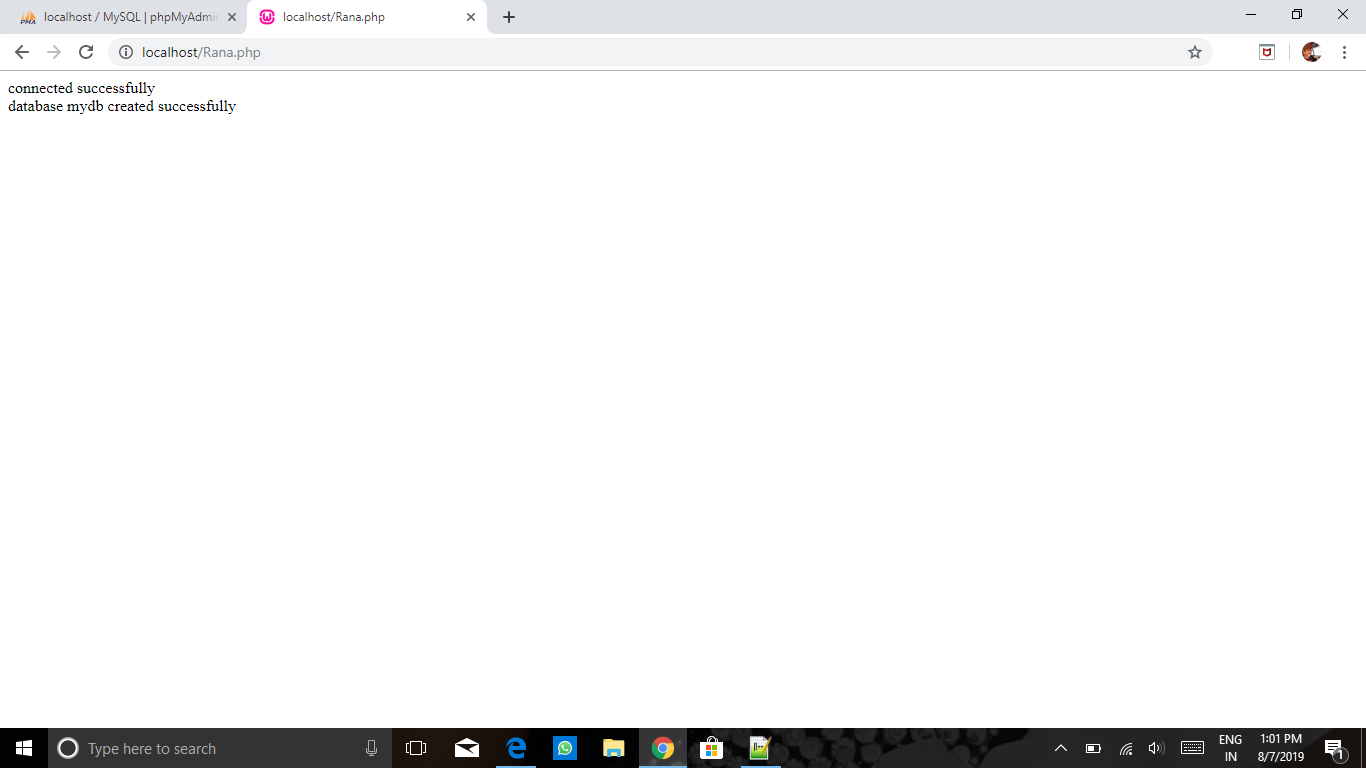
}

mysqli\_close($conn);

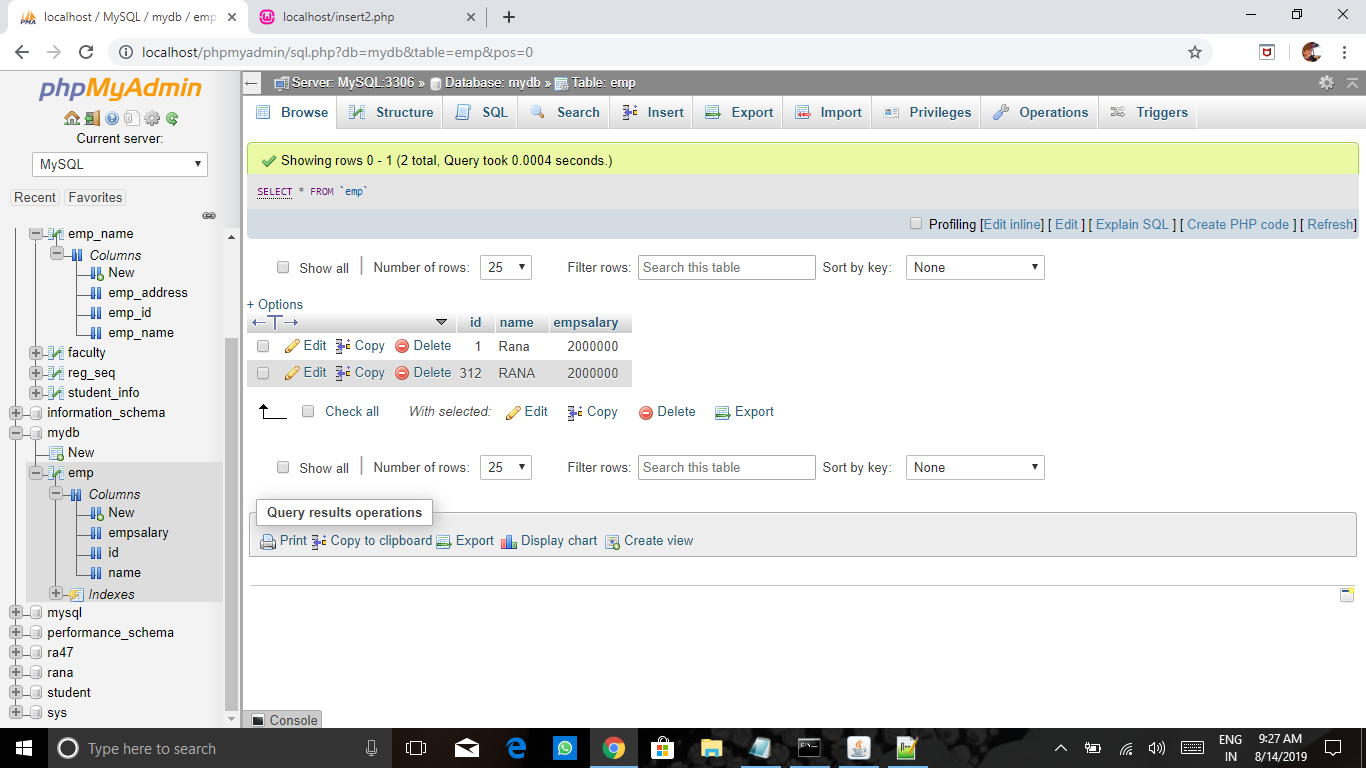
**?>**

**OUTPUT:**

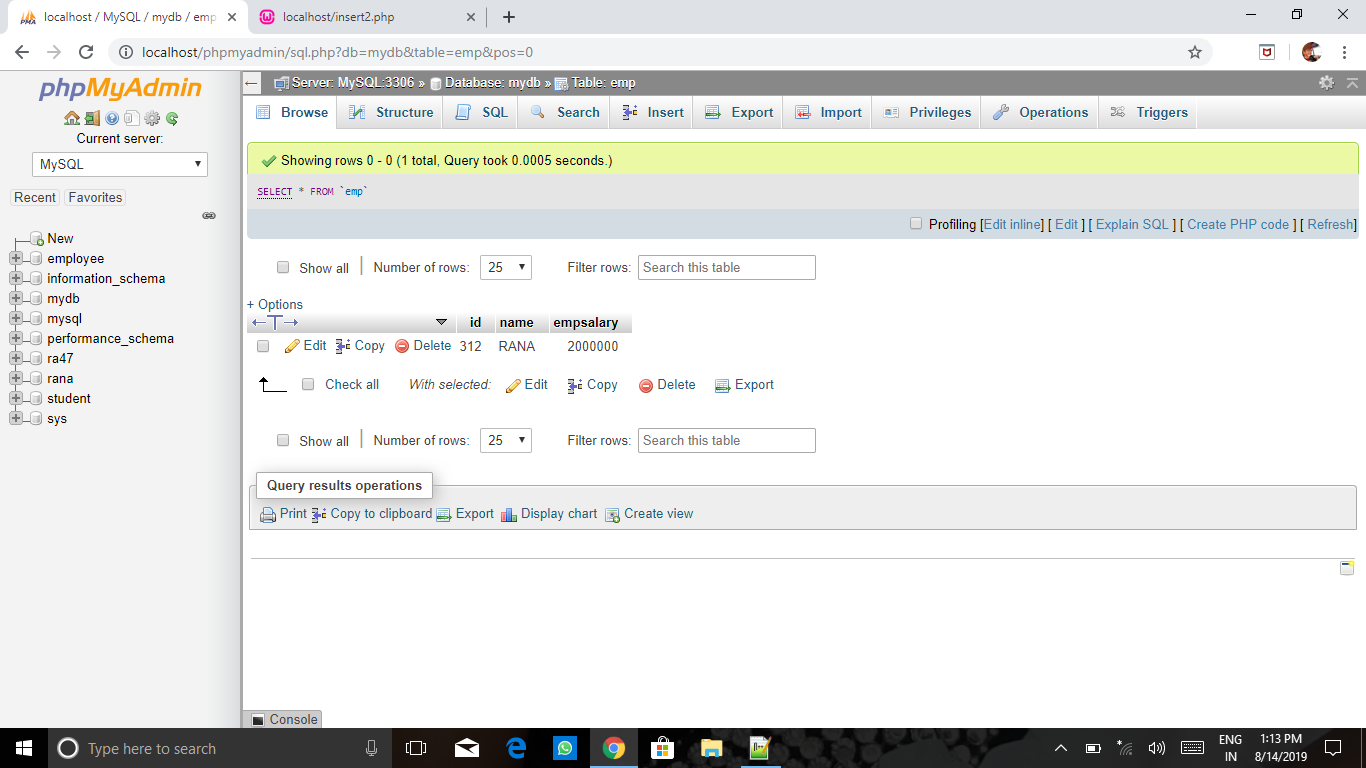
After creation of database and emp table

****

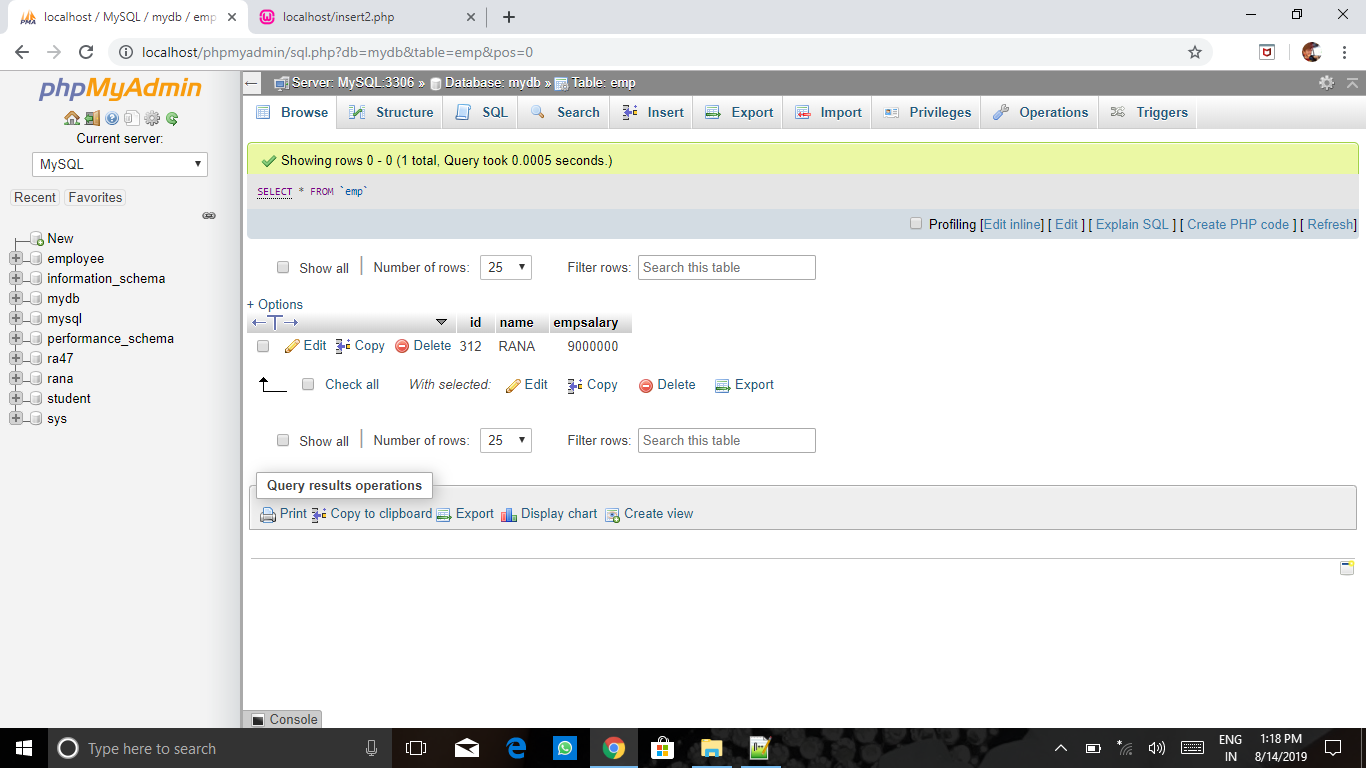
After insertion:



After deletion:



After updation:



**RESULT:**ThustheMysqlconnectedusingPHP and MYSQLandexecutedtheCREATE,INSERT,SELECTcommandinMySQL.