



DEPARTMENT OF COMPUTER SCIENCE
CENTRAL UNIVERSITY OF TAMILNADU
MSCP12 : Database Management System Lab.
M.Sc., Computer Science - Semester: I
Lab. Work Sheet – 1
Course Instructor: Appasami G.

Date: 25.07.2019

Time: 2.10 P.M. to 5.10 P.M.

-
- 1) Introduction to MYSQL.
 - 2) Study of DDL commands
 - 3) Creation of Databases
 - 4) Creation of Tables
-




DEPARTMENT OF COMPUTER SCIENCE
CENTRAL UNIVERSITY OF TAMILNADU
MSCP12 : Database Management System Lab.
M.Sc., Computer Science - Semester: I
Lab. Work Sheet – 2
Course Instructor: Appasami G.

Date: 01.08.2019

Time: 2.10 P.M. to 5.10 P.M.

-
- 1) Study of DML commands.
 - 2) Working with Tables using the following data manipulation commands.
 - a. Select
 - b. Insert
 - c. Update
 - d. Delete
-

	<p style="text-align: center;">DEPARTMENT OF COMPUTER SCIENCE CENTRAL UNIVERSITY OF TAMILNADU MSCP12 : Database Management System Lab. M.Sc., Computer Science - Semester: I Lab. Work Sheet – 2 Course Instructor: Appasami G.</p>
---	--


Date: 08.08.2019

Time: 2.10 P.M. to 5.10 P.M.

- 1) The project group for MSCP12 - Database Management System Laboratory is formed and no change in the project group will be entertained. You are asked to work with your project groups and construct the Entity-Relationship (ER) diagram as tabulated below:

S. No.	AD. No.	NAME	PROJECT
1		AFARI JESSI	Railway Reservation System
2	19130115	ALEENA PRAKASH	
3	19130121	ARUN KUMAR.B	
4	19130104	BEULAH EVANJALIN	
5	19130109	BHARATHI.B	Hotel management System
6	19130103	BOKKA MOUNIKA	
7	19130120	CHIRANJEEVI.S	
8	19130119	DHAVAKUMAR	
9	19130116	EZHILARASIA	Time table Management System
10	19130111	GUNDETI VEENA	
11	19130101	KAMIL KHAN	
12	19130118	KAPILRAJ.S	
13	19130110	KAYALVIZHI S	Library Management System
14	19130114	LINET M	
15	19130105	MADHUMITHA. K	
16	19130106	MANDA AMULYA	
17	19130117	NIVETHA R.S	ERP Information System
18	19130107	SRIDURGA.S	
19	19130108	THENMOZHI. K	
20	19130102	VESSESH KRISHNA	
21	19130112	YARRAPAGARI DAYANAND	

Note: All batches should submit their respective Project ERD with Hospital Management System.

	<p>DEPARTMENT OF COMPUTER SCIENCE CENTRAL UNIVERSITY OF TAMILNADU MSCP12 : Database Management System Lab. M.Sc., Computer Science - Semester: I Lab. Work Sheet – 3 Course Instructor: Appasami G.</p>
---	---

Date: 22.08.2019

Time: 2.10 P.M. to 5.10 P.M.

-
- 1) The respective (MSCP12 : Database Management System Lab.) project team will work on conversion of the ER diagram to Relation model.
 - 2) Create the tables for the converted Relation model.
-



DEPARTMENT OF COMPUTER SCIENCE
CENTRAL UNIVERSITY OF TAMILNADU
MSCP12 : Database Management System Lab.
M.Sc., Computer Science - Semester: I
Lab. Work Sheet – 4
Course Instructor: Appasami G.

Date: 29.08.2019

Time: 2.10 P.M. to 5.10 P.M.

1) Consider the Bank Database System with the following relations.

- a) branch(branch-name, branch-city, assets)
- b) customer(customer-name, customer-street, customer-city)
- c) account(account-number, branch-name, balance)
- d) loan(loan-number, branch-name, amount)
- e) depositor(customer-name, account-number)
- f) borrower(customer-name, loan-number)

2) Instances for the relations of Bank Database System are given below.

Branch Table

branch-name	branch-city	assets
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

Customer Table

customer-name	customer-street	customer-city
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

Account Table

account-number	branch-name	balance
A-101	Downtown	500
A-215	Mianus	700
A-102	Perryridge	400
A-305	Round Hill	350
A-201	Brighton	900
A-222	Redwood	700
A-217	Brighton	750

Loan Table

loan-number	branch-name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

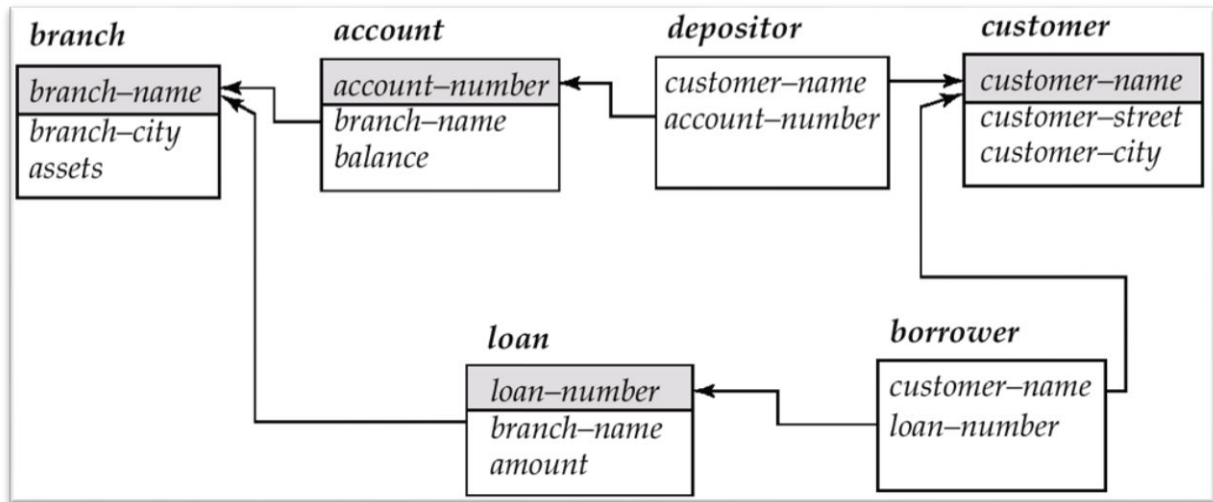
Depositor Table

customer-name	account-number
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

Borrower Table

customer-name	loan-number
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

3. Schema Diagram for the Banking Database System is given below.



4. Write the query expression in Relational Algebra (RA), Tuple Relational Calculus (TRC), and Domain Relational Calculus (DRC). Also write the output of the query expression for the Bank Database.

- 1) Find all loans over \$1200.
- 2) Find the loan number for each loan of an amount greater than \$1200.
- 3) Find the names of all customers who have a loan, an account, or both from the bank.
- 4) Find the names of all customers who have a loan and an account at the bank.
- 5) Find the names of all customers who have a loan at the Perryridge branch.
- 6) Find the names of all customers who have a loan at the Perryridge branch, but no account at any branch of the bank.
- 7) Find the names of all customers who have an account at the Downtown and Mianus branches.
- 8) Find the total amount each branch has in accounts.
- 9) Find the average loan amount of each customer
- 10) Find the names of all customers who have an account at every branch located in Brooklyn.



DEPARTMENT OF COMPUTER SCIENCE
CENTRAL UNIVERSITY OF TAMILNADU
MSCP12 : Database Management System Lab.
M.Sc., Computer Science - Semester: I
Lab. Work Sheet – 5
Course Instructor: Appasami G.

Date: 05.09.2019

Time: 2.10 P.M. to 5.10 P.M.

Try the following SQL queries with the syntax given below in the relations which is created on lab sheet 4.

ALTER TABLE

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

ALTER TABLE lets you add columns to a table in a database.

AND

```
SELECT column_name(s)  
FROM table_name  
WHERE column_1 = value_1  
      AND column_2 = value_2;
```

AND is an operator that combines two conditions. Both conditions must be true for the row to be included in the result set.

AS

```
SELECT column_name AS 'Alias'  
FROM table_name;
```

AS is a keyword in SQL that allows you to rename a column or table using an alias.

AVG()

```
SELECT AVG(column_name)  
FROM table_name;  
AVG() is an aggregate function that returns the average value for a numeric column.
```

BETWEEN

```
SELECT column_name(s)  
FROM table_name  
WHERE column_name BETWEEN value_1 AND value_2;
```

The BETWEEN operator is used to filter the result set within a certain range. The values can be numbers, text or dates.

CASE

```
SELECT column_name,  
CASE  
    WHEN condition THEN 'Result_1'  
    WHEN condition THEN 'Result_2'  
    ELSE 'Result_3'  
END  
FROM table_name;
```

CASE statements are used to create different outputs (usually in the SELECT statement). It is SQL's way of handling if-then logic.

COUNT()

```
SELECT COUNT(column_name)  
FROM table_name;
```

COUNT() is a function that takes the name of a column as an argument and counts the number of rows where the column is not NULL.

CREATE TABLE

```
CREATE TABLE table_name (  
    column_1 datatype,  
    column_2 datatype,  
    column_3 datatype  
);
```

CREATE TABLE creates a new table in the database. It allows you to specify the name of the table and the name of each column in the table.

DELETE

```
DELETE FROM table_name  
WHERE some_column = some_value;
```

DELETE statements are used to remove rows from a table.

GROUP BY

```
SELECT column_name, COUNT(*)  
FROM table_name  
GROUP BY column_name;
```

GROUP BY is a clause in SQL that is only used with aggregate functions. It is used in collaboration with the SELECT statement to arrange identical data into groups.

HAVING

```
SELECT column_name, COUNT(*)  
FROM table_name  
GROUP BY column_name  
HAVING COUNT(*) > value;
```

HAVING was added to SQL because the WHERE keyword could not be used with aggregate functions.

INNER JOIN

```
SELECT column_name(s)
FROM table_1
JOIN table_2
  ON table_1.column_name = table_2.column_name;
```

An inner join will combine rows from different tables if the join condition is true.

INSERT

```
INSERT INTO table_name (column_1, column_2, column_3)
VALUES (value_1, 'value_2', value_3);
```

INSERT statements are used to add a new row to a table.

IS NULL / IS NOT NULL

```
SELECT column_name(s)
FROM table_name
WHERE column_name IS NULL;
```

IS NULL and IS NOT NULL are operators used with the WHERE clause to test for empty values.

LIKE

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

LIKE is a special operator used with the WHERE clause to search for a specific pattern in a column.

LIMIT

```
SELECT column_name(s)
FROM table_name
LIMIT number;
```

LIMIT is a clause that lets you specify the maximum number of rows the result set will have.

MAX()

```
SELECT MAX(column_name)
FROM table_name;
```

MAX() is a function that takes the name of a column as an argument and returns the largest value in that column.

MIN()

```
SELECT MIN(column_name)
FROM table_name;
```

MIN() is a function that takes the name of a column as an argument and returns the smallest value in that column.

OR

```
SELECT column_name
FROM table_name
WHERE column_name = value_1
  OR column_name = value_2;
```

OR is an operator that filters the result set to only include rows where either condition is true.

ORDER BY

```
SELECT column_name  
FROM table_name  
ORDER BY column_name ASC | DESC;
```

ORDER BY is a clause that indicates you want to sort the result set by a particular column either alphabetically or numerically.

OUTER JOIN

```
SELECT column_name(s)  
FROM table_1  
LEFT JOIN table_2  
ON table_1.column_name = table_2.column_name;
```

An outer join will combine rows from different tables even if the join condition is not met. Every row in the left table is returned in the result set, and if the join condition is not met, then NULL values are used to fill in the columns from the right table.

ROUND()

```
SELECT ROUND(column_name, integer)  
FROM table_name;
```

ROUND() is a function that takes a column name and an integer as arguments. It rounds the values in the column to the number of decimal places specified by the integer.

SELECT

```
SELECT column_name  
FROM table_name;
```

SELECT statements are used to fetch data from a database. Every query will begin with **SELECT**.

SELECT DISTINCT

```
SELECT DISTINCT column_name  
FROM table_name;
```

SELECT DISTINCT specifies that the statement is going to be a query that returns unique values in the specified column(s).

SUM

```
SELECT SUM(column_name)  
FROM table_name;
```

SUM() is a function that takes the name of a column as an argument and returns the sum of all the values in that column.

UPDATE

```
UPDATE table_name  
SET some_column = some_value  
WHERE some_column = some_value;
```

UPDATE statements allow you to edit rows in a table.

WHERE


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

WHERE is a clause that indicates you want to filter the result set to include only rows where the following condition is true.

WITH

```
WITH temporary_name AS (
    SELECT *
    FROM table_name)
SELECT *
FROM temporary_name
WHERE column_name operator value;
```

WITH clause lets you store the result of a query in a temporary table using an alias. You can also define multiple temporary tables using a comma and with one instance of the **WITH** keyword.

	<p style="text-align: center;">DEPARTMENT OF COMPUTER SCIENCE CENTRAL UNIVERSITY OF TAMILNADU MSCP12 : Database Management System Lab. M.Sc., Computer Science - Semester: I Lab. Work Sheet – 6 Course Instructor: Appasami G.</p>
---	--

Date: 12.09.2019

Time: 2.10 P.M. to 5.10 P.M.

1. Create the tables for the following relations where the primary keys are underlined.
employee (person-name, street, city)
works (person-name, company-name, salary)
company (company-name, city)
manages (person-name, manager-name)

2. Write SQL for the following queries:
 - a) Find the names of all employees who work for First Bank Corporation.
 - b) Find the names and cities of residence of all employees who work for First Bank Corporation.
 - c) Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than \$10,000 per annum.
 - d) Find the names of all employees in this database who live in the same city as the company for which they work.
 - e) Find the names of all employees who live in the same city and on the same street as do their managers.
 - f) Find the names of all employees in this database who do not work for First Bank Corporation.
 - g) Find the names of all employees who earn more than every employee of Small Bank Corporation.
 - h) Assume the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is located.



DEPARTMENT OF COMPUTER SCIENCE
CENTRAL UNIVERSITY OF TAMILNADU
MSCP12 : Database Management System Lab.
M.Sc., Computer Science - Semester: I
Lab. Work Sheet – 7
Course Instructor: Appasami G.

Date: 16.09.2019

Time: 2.10 P.M. to 5.10 P.M.

1. Execute the following SQL commands.

SQL Select
SQL Select Distinct
SQL Where
SQL And, Or, Not
SQL Order By
SQL Insert Into
SQL Null Values
SQL Update
SQL Delete
SQL Select Top
SQL Min and Max
SQL Count, Avg, Sum
SQL Like
SQL Wildcards
SQL In
SQL Between
SQL Aliases
SQL Joins
SQL Inner Join
SQL Left Join
SQL Right Join
SQL Full Join
SQL Self Join
SQL Union
SQL Group By
SQL Having
SQL Exists
SQL Any, All
SQL Select Into
SQL Insert Into Select
SQL Case
SQL Null Functions
SQL Stored Procedures
SQL Comments

2. Execute the following SQL Database commands

- SQL Create DB
- SQL Drop DB
- SQL Backup DB
- SQL Create Table
- SQL Drop Table
- SQL Alter Table
- SQL Constraints
- SQL Not Null
- SQL Unique
- SQL Primary Key
- SQL Foreign Key
- SQL Check
- SQL Default
- SQL Index
- SQL Auto Increment
- SQL Dates
- SQL Views
- SQL Injection
- SQL Hosting

3. Study of MySQL has many built-in functions.

MySQL String Functions

Function	Description
<u>ASCII</u>	Returns the ASCII value for the specific character
<u>CHAR_LENGTH</u>	Returns the length of a string (in characters)
<u>CHARACTER_LENGTH</u>	Returns the length of a string (in characters)
<u>CONCAT</u>	Adds two or more expressions together
<u>CONCAT_WS</u>	Adds two or more expressions together with a separator
<u>FIELD</u>	Returns the index position of a value in a list of values
<u>FIND_IN_SET</u>	Returns the position of a string within a list of strings
<u>FORMAT</u>	Formats a number to a format like "#,###,###.##", rounded to a specified number of decimal places
<u>INSERT</u>	Inserts a string within a string at the specified position and for a certain number of characters
<u>INSTR</u>	Returns the position of the first occurrence of a string in another string
<u>LCASE</u>	Converts a string to lower-case
<u>LEFT</u>	Extracts a number of characters from a string (starting from left)
<u>LENGTH</u>	Returns the length of a string (in bytes)
<u>LOCATE</u>	Returns the position of the first occurrence of a substring in a string
<u>LOWER</u>	Converts a string to lower-case
<u>LPAD</u>	Left-pads a string with another string, to a certain length
<u>LTRIM</u>	Removes leading spaces from a string
<u>MID</u>	Extracts a substring from a string (starting at any position)
<u>POSITION</u>	Returns the position of the first occurrence of a substring in a string
<u>REPEAT</u>	Repeats a string as many times as specified

REPLACE	Replaces all occurrences of a substring within a string, with a new substring
REVERSE	Reverses a string and returns the result
RIGHT	Extracts a number of characters from a string (starting from right)
RPAD	Right-pads a string with another string, to a certain length
RTRIM	Removes trailing spaces from a string
SPACE	Returns a string of the specified number of space characters
STRCMP	Compares two strings
SUBSTR	Extracts a substring from a string (starting at any position)
SUBSTRING	Extracts a substring from a string (starting at any position)
SUBSTRING INDEX	Returns a substring of a string before a specified number of delimiter occurs
TRIM	Removes leading and trailing spaces from a string
UCASE	Converts a string to upper-case
UPPER	Converts a string to upper-case

MySQL Numeric Functions

Function	Description
ABS	Returns the absolute value of a number
ACOS	Returns the arc cosine of a number
ASIN	Returns the arc sine of a number
ATAN	Returns the arc tangent of one or two numbers
ATAN2	Returns the arc tangent of two numbers
AVG	Returns the average value of an expression
CEIL	Returns the smallest integer value that is \geq to a number
CEILING	Returns the smallest integer value that is \geq to a number
COS	Returns the cosine of a number
COT	Returns the cotangent of a number
COUNT	Returns the number of records returned by a select query
DEGREES	Converts a value in radians to degrees
DIV	Used for integer division
EXP	Returns e raised to the power of a specified number
FLOOR	Returns the largest integer value that is \leq to a number
GREATEST	Returns the greatest value of the list of arguments
LEAST	Returns the smallest value of the list of arguments
LN	Returns the natural logarithm of a number
LOG	Returns the natural logarithm of a number, or the logarithm of a number to a specified base
LOG10	Returns the natural logarithm of a number to base 10
LOG2	Returns the natural logarithm of a number to base 2
MAX	Returns the maximum value in a set of values
MIN	Returns the minimum value in a set of values

MOD	Returns the remainder of a number divided by another number
PI	Returns the value of PI
POW	Returns the value of a number raised to the power of another number
POWER	Returns the value of a number raised to the power of another number
RADIANS	Converts a degree value into radians
RAND	Returns a random number
ROUND	Rounds a number to a specified number of decimal places
SIGN	Returns the sign of a number
SIN	Returns the sine of a number
SQRT	Returns the square root of a number
SUM	Calculates the sum of a set of values
TAN	Returns the tangent of a number
TRUNCATE	Truncates a number to the specified number of decimal places

MySQL Date Functions


Function	Description
ADDDATE	Adds a time/date interval to a date and then returns the date
ADDTIME	Adds a time interval to a time/datetime and then returns the time/datetime
CURDATE	Returns the current date
CURRENT_DATE	Returns the current date
CURRENT_TIME	Returns the current time
CURRENT_TIMESTAMP	Returns the current date and time
CURTIME	Returns the current time
DATE	Extracts the date part from a datetime expression
DATEDIFF	Returns the number of days between two date values
DATE_ADD	Adds a time/date interval to a date and then returns the date
DATE_FORMAT	Formats a date
DATE_SUB	Subtracts a time/date interval from a date and then returns the date
DAY	Returns the day of the month for a given date
DAYNAME	Returns the weekday name for a given date
DAYOFMONTH	Returns the day of the month for a given date
DAYOFWEEK	Returns the weekday index for a given date
DAYOFYEAR	Returns the day of the year for a given date
EXTRACT	Extracts a part from a given date
FROM_DAYS	Returns a date from a numeric datevalue
HOUR	Returns the hour part for a given date
LAST_DAY	Extracts the last day of the month for a given date
LOCALTIME	Returns the current date and time
LOCALTIMESTAMP	Returns the current date and time
MAKEDATE	Creates and returns a date based on a year and a number of days value
MAKETIME	Creates and returns a time based on an hour, minute, and second value
MICROSECOND	Returns the microsecond part of a time/datetime

<u>MINUTE</u>	Returns the minute part of a time/datetime
<u>MONTH</u>	Returns the month part for a given date
<u>MONTHNAME</u>	Returns the name of the month for a given date
<u>NOW</u>	Returns the current date and time
<u>PERIOD_ADD</u>	Adds a specified number of months to a period
<u>PERIOD_DIFF</u>	Returns the difference between two periods
<u>QUARTER</u>	Returns the quarter of the year for a given date value
<u>SECOND</u>	Returns the seconds part of a time/datetime
<u>SEC_TO_TIME</u>	Returns a time value based on the specified seconds
<u>STR_TO_DATE</u>	Returns a date based on a string and a format
<u>SUBDATE</u>	Subtracts a time/date interval from a date and then returns the date
<u>SUBTIME</u>	Subtracts a time interval from a datetime and then returns the time/datetime
<u>SYSDATE</u>	Returns the current date and time
<u>TIME</u>	Extracts the time part from a given time/datetime
<u>TIME FORMAT</u>	Formats a time by a specified format
<u>TIME TO SEC</u>	Converts a time value into seconds
<u>TIMEDIFF</u>	Returns the difference between two time/datetime expressions
<u>TIMESTAMP</u>	Returns a datetime value based on a date or datetime value
<u>TO_DAYS</u>	Returns the number of days between a date and date "0000-00-00"
<u>WEEK</u>	Returns the week number for a given date
<u>WEEKDAY</u>	Returns the weekday number for a given date
<u>WEEKOFYEAR</u>	Returns the week number for a given date
<u>YEAR</u>	Returns the year part for a given date
<u>YEARWEEK</u>	Returns the year and week number for a given date

MySQL Advanced Functions

Function	Description
<u>BIN</u>	Returns a binary representation of a number
<u>BINARY</u>	Converts a value to a binary string
<u>CASE</u>	Goes through conditions and return a value when the first condition is met
<u>CAST</u>	Converts a value (of any type) into a specified datatype
<u>COALESCE</u>	Returns the first non-null value in a list
<u>CONNECTION_ID</u>	Returns the unique connection ID for the current connection
<u>CONV</u>	Converts a number from one numeric base system to another
<u>CONVERT</u>	Converts a value into the specified datatype or character set
<u>CURRENT_USER</u>	Returns the user name and host name for the MySQL account that the server used to authenticate the current client
<u>DATABASE</u>	Returns the name of the current database
<u>IF</u>	Returns a value if a condition is TRUE, or another value if a condition is FALSE
<u>IFNULL</u>	Return a specified value if the expression is NULL, otherwise return

	the expression
<u>ISNULL</u>	Returns 1 or 0 depending on whether an expression is NULL
<u>LAST_INSERT_ID</u>	Returns the AUTO_INCREMENT id of the last row that has been inserted or updated in a table
<u>NULLIF</u>	Compares two expressions and returns NULL if they are equal. Otherwise, the first expression is returned
<u>SESSION_USER</u>	Returns the current MySQL user name and host name
<u>SYSTEM_USER</u>	Returns the current MySQL user name and host name
<u>USER</u>	Returns the current MySQL user name and host name
<u>VERSION</u>	Returns the current version of the MySQL database

	<p style="text-align: center;">DEPARTMENT OF COMPUTER SCIENCE CENTRAL UNIVERSITY OF TAMILNADU MSCP12 : Database Management System Lab. M.Sc., Computer Science - Semester: I Lab. Work Sheet – 8 (Internal Test 1) Course Instructor: Appasami G.</p>
---	--

Date: 23.09.2019

Time: 2.10 P.M. to 5.10 P.M.

1. Draw the ERD for employee management system. Also construct schema relationship diagram.
2. Create the tables for the following relations where the primary keys are underlined.
employee (person-name, street, city)
works (person-name, company-name, salary)
company (company-name, city)
manages (person-name, manager-name)
3. Write SQL for the following queries:
 - a) Find the company name which is located in Chennai city.
 - b) Find the person name who is getting more salary.
 - c) Find the names of all employees who work for First Bank Corporation.
 - d) Find the names and cities of residence of all employees who work for First Bank Corporation.
 - e) Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than 10,000 per annum.
 - f) Find the names of all employees in this database who live in the same city as the company for which they work.
 - g) Find the names of all employees who live in the same city and on the same street as do their managers.
 - h) Find the names of all employees in this database who do not work for First Bank Corporation.
 - i) Find the names of all employees who earn more than every employee of Small Bank Corporation.
 - j) Assume the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is located.



DEPARTMENT OF COMPUTER SCIENCE
CENTRAL UNIVERSITY OF TAMILNADU
MSCP12 : Database Management System Lab.
M.Sc., Computer Science - Semester: I
Lab. Work Sheet – 9
Course Instructor: Appasami G.

Date: 27.09.2019

Time: 2.10 P.M. to 5.10 P.M.

1. Create a table with following values:

emp_id	emp_name	emp_salary
E00001	Alice	50000
E00002	Bob	90000
E00003	Carol	145000
E00004	Dave	12000
E00005	Eve	6000

2. Create a user defined functions for the following:

- a) Create a user defined function which classify the employee as Group A employee and Group B employee. If the salary of employee is greater than 99,999 then they are classified as Group A else Group B.
- b) Create a user defined function to show maximum, minimum, average, and sum of employee salaries.

USER DEFINED FUNCTIONS

Stored functions

Stored functions are just like built in functions except that you have to define the stored function yourself. Once a stored function has been created, it can be used in SQL statements just like any other function. The basic syntax for creating a stored function is as shown below

```
CREATE FUNCTION sf_name ([parameter(s)])  
RETURNS data type  
DETERMINISTIC  
STATEMENTS
```

HERE


1. **"CREATE FUNCTION sf_name ([parameter(s)]) "** is mandatory and tells MySQL server to create a function named 'sf_name' with optional parameters defined in the parenthesis.
2. **"RETURNS data type"** is mandatory and specifies the data type that the function should return.
3. **"DETERMINISTIC"** means the function will return the same values if the same arguments are supplied to it.
4. **"STATEMENTS"** is the procedural code that the function executes.

Sample User Defined Function

```
DELIMITER |  
CREATE FUNCTION classify_emp (emp_salary int)  
RETURNS VARCHAR(20)  
DETERMINISTIC  
BEGIN  
DECLARE a VARCHAR(20);  
IF emp_salary > =100000  
THEN SET a = 'Group A';  
ELSEIF emp_salary < 100000  
THEN SET a = 'Group B';  
END IF;  
RETURN a;  
END |
```

After make sure that the above doesn't have any error, type the following
select emp_id,emp_name,emp_salary,classify_emp(emp_salary) from emp_data;

emp_id	emp_name	emp_salary	classify_emp(emp_salary)
E00001	Alice	50000	Group B
E00002	Bob	90000	Group B
E00003	Carol	145000	Group A
E00004	Dave	12000	Group B
E00005	Eve	6000	Group B

	<p>DEPARTMENT OF COMPUTER SCIENCE CENTRAL UNIVERSITY OF TAMILNADU MSCP12 : Database Management System Lab. M.Sc., Computer Science - Semester: I Lab. Work Sheet – 10 Course Instructor: Appasami G.</p>
---	--

Date: 03.10.2019

Time: 2.10 P.M. to 5.10 P.M.

1. Write a triggers for Insert and Update. (Kindly refer the attachment which explains in detail about triggers)
2. Write a procedure to compute age from date of birth.
3. Write a function to return maximum, average and minimum of salary.

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events

- ☐ A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- ☐ A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- ☐ A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

Triggers can be written for the following purposes –

- ☐ Generating some derived column values automatically
- ☐ Enforcing referential integrity
- ☐ Event logging and storing information on table access
- ☐ Auditing
- ☐ Synchronous replication of tables
- ☐ Imposing security authorizations
- ☐ Preventing invalid transactions

For creating a new trigger, we need to use CREATE TRIGGER statement. Its syntax is as follows:

```
CREATE TRIGGER trigger_name trigger_time trigger_event
```

```
ON table_name
```

```
FOR EACH ROW
```

```
BEGIN
```

```
...
```

```
END;
```

Here,

1. **Trigger_name** is the name of the trigger which must be put after CREATE TRIGGER statement. The naming convention for trigger_name can be like [trigger time]_[table name]_[trigger event]. For example, before_student_update or after_student_insert can be a name of the trigger.
2. **Trigger_time** is the time of trigger activation and it can be BEFORE or AFTER. We must have to specify the activation time while defining a trigger. We must to use BEFORE if we want to process action prior to the change made on the table and AFTER if we want to process action post to the change made on the table.
3. **Trigger_event** can be INSERT, UPDATE or DELETE. This event causes the trigger to be invoked. A trigger only can be invoked by one event. To define a trigger that is invoked by multiple events, we have to define multiple triggers, one for each event.
4. **Table_name** is the name of the table. Actually, a trigger is always associated with a specific table. Without a table, a trigger would not exist hence we have to specify the table name after the 'ON' keyword.
5. **BEGIN...END** is the block in which we will define the logic for the trigger.

Example

Suppose we want to apply trigger on the table Student_age which is created as follows:

```
mysql> Create table Student_age(age INT, Name Varchar(35));
```

Query OK, 0 rows affected (0.80 sec)

Now, the following trigger will automatically insert the age = 0 if someone try to insert age < 0.

```
mysql> DELIMITER //
```

```
mysql> Create Trigger before_inser_studentage BEFORE INSERT ON student_age FOR  
EACH ROW
```

```
BEGIN
```

```
IF NEW.age < 0 THEN SET NEW.age = 0;
```

```
END IF;
```

```
END //
```

Query OK, 0 rows affected (0.30 sec)

Now, for invoking this trigger, we can use the following statements:

```
mysql> INSERT INTO Student_age(age, Name) values(30, 'Rahul');
```

Query OK, 1 row affected (0.14 sec)

```
mysql> INSERT INTO Student_age(age, Name) values(-10, 'Harshit');
```

Query OK, 1 row affected (0.11 sec)

```
mysql> Select * from Student_age;
```

```
+-----+-----+
| age   | Name   |
+-----+-----+
| 30    | Rahul  |
| 0     | Harshit|
+-----+-----+
2 rows in set (0.00 sec)
```

The above result set shows that on inserting the negative value in the table will lead to insert 0 by a trigger.

The above was the example of a trigger with trigger_event as INSERT and trigger_time are BEFORE.

UPDATE TRIGGER

```
CREATE TABLE Student_age_audit
```

```
(
    id INT AUTO_INCREMENT PRIMARY KEY,
    age INT NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    changedat DATETIME DEFAULT NULL,
    action VARCHAR(50) DEFAULT NULL
);
```

```
DELIMITER $$
```

```
CREATE TRIGGER before_Student_age
    BEFORE UPDATE ON Student_age
    FOR EACH ROW
```

```
BEGIN

    INSERT INTO Student_age_audit

    SET action = 'update',

    age = OLD.age,

        last_name = OLD.name,

        changedat = NOW();

END$$

DELIMITER ;

UPDATE Student_age

SET

    age = 25

WHERE

    name = 'Arun';
```



DEPARTMENT OF COMPUTER SCIENCE
CENTRAL UNIVERSITY OF TAMILNADU
MSCP12 : Database Management System Lab.
M.Sc., Computer Science - Semester: I
Lab. Work Sheet – 11
Course Instructor: Appasami G.

Date: 10.10.2019

Time: 2.10 P.M. to 5.10 P.M.

1. Working with QBE.
2. QBE using MS-Access.
3. Back end (Database) integration with front end.

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events

- ☐ A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- ☐ A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- ☐ A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

Triggers can be written for the following purposes –

- ☐ Generating some derived column values automatically
- ☐ Enforcing referential integrity
- ☐ Event logging and storing information on table access
- ☐ Auditing
- ☐ Synchronous replication of tables
- ☐ Imposing security authorizations
- ☐ Preventing invalid transactions

For creating a new trigger, we need to use CREATE TRIGGER statement. Its syntax is as follows:

```
CREATE TRIGGER trigger_name trigger_time trigger_event
```

```
ON table_name
```

```
FOR EACH ROW
```

```
BEGIN
```

```
...
```

```
END;
```

Here,

1. **Trigger_name** is the name of the trigger which must be put after CREATE TRIGGER statement. The naming convention for trigger_name can be like [trigger time]_[table name]_[trigger event]. For example, before_student_update or after_student_insert can be a name of the trigger.
2. **Trigger_time** is the time of trigger activation and it can be BEFORE or AFTER. We must have to specify the activation time while defining a trigger. We must to use BEFORE if we want to process action prior to the change made on the table and AFTER if we want to process action post to the change made on the table.
3. **Trigger_event** can be INSERT, UPDATE or DELETE. This event causes the trigger to be invoked. A trigger only can be invoked by one event. To define a trigger that is invoked by multiple events, we have to define multiple triggers, one for each event.
4. **Table_name** is the name of the table. Actually, a trigger is always associated with a specific table. Without a table, a trigger would not exist hence we have to specify the table name after the 'ON' keyword.
5. **BEGIN...END** is the block in which we will define the logic for the trigger.

Example

Suppose we want to apply trigger on the table Student_age which is created as follows:

```
mysql> Create table Student_age(age INT, Name Varchar(35));
```

Query OK, 0 rows affected (0.80 sec)

Now, the following trigger will automatically insert the age = 0 if someone try to insert age < 0.

```
mysql> DELIMITER //
```

```
mysql> Create Trigger before_inser_studentage BEFORE INSERT ON student_age FOR  
EACH ROW
```

```
BEGIN
```

```
IF NEW.age < 0 THEN SET NEW.age = 0;
```

```
END IF;
```

```
END //
```

Query OK, 0 rows affected (0.30 sec)

Now, for invoking this trigger, we can use the following statements:

```
mysql> INSERT INTO Student_age(age, Name) values(30, 'Rahul');
```

Query OK, 1 row affected (0.14 sec)

```
mysql> INSERT INTO Student_age(age, Name) values(-10, 'Harshit');
```

Query OK, 1 row affected (0.11 sec)

```
mysql> Select * from Student_age;
```

```
+-----+-----+
| age   | Name   |
+-----+-----+
| 30    | Rahul  |
| 0     | Harshit|
+-----+-----+
2 rows in set (0.00 sec)
```

The above result set shows that on inserting the negative value in the table will lead to insert 0 by a trigger.

The above was the example of a trigger with trigger_event as INSERT and trigger_time are BEFORE.

UPDATE TRIGGER

```
CREATE TABLE Student_age_audit
```

```
(
    id INT AUTO_INCREMENT PRIMARY KEY,
    age INT NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    changedat DATETIME DEFAULT NULL,
    action VARCHAR(50) DEFAULT NULL
);
```

```
DELIMITER $$
```

```
CREATE TRIGGER before_Student_age
    BEFORE UPDATE ON Student_age
    FOR EACH ROW
```

```
BEGIN

    INSERT INTO Student_age_audit

    SET action = 'update',

    age = OLD.age,

        last_name = OLD.name,

        changedat = NOW();

END$$

DELIMITER ;

UPDATE Student_age

SET

    age = 25

WHERE

    name = 'Arun';
```



DEPARTMENT OF COMPUTER SCIENCE
CENTRAL UNIVERSITY OF TAMILNADU
MSCP12 : Database Management System Lab.
M.Sc., Computer Science - Semester: I
Lab. Work Sheet – 12
Course Instructor: Appasami G.

Date: 17.10.2019

Time: 2.10 P.M. to 5.10 P.M.

1. Mini Project presentations by individual.
2. Report submission.

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events

- ☐ A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- ☐ A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- ☐ A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

Triggers can be written for the following purposes –

- ☐ Generating some derived column values automatically
- ☐ Enforcing referential integrity
- ☐ Event logging and storing information on table access
- ☐ Auditing
- ☐ Synchronous replication of tables
- ☐ Imposing security authorizations
- ☐ Preventing invalid transactions

For creating a new trigger, we need to use CREATE TRIGGER statement. Its syntax is as follows:

```
CREATE TRIGGER trigger_name trigger_time trigger_event
```

```
ON table_name
```

```
FOR EACH ROW
```

```
BEGIN
```

```
...
```

```
END;
```

Here,

1. **Trigger_name** is the name of the trigger which must be put after CREATE TRIGGER statement. The naming convention for trigger_name can be like [trigger time]_[table name]_[trigger event]. For example, before_student_update or after_student_insert can be a name of the trigger.
2. **Trigger_time** is the time of trigger activation and it can be BEFORE or AFTER. We must have to specify the activation time while defining a trigger. We must use BEFORE if we want to process action prior to the change made on the table and AFTER if we want to process action post to the change made on the table.
3. **Trigger_event** can be INSERT, UPDATE or DELETE. This event causes the trigger to be invoked. A trigger only can be invoked by one event. To define a trigger that is invoked by multiple events, we have to define multiple triggers, one for each event.
4. **Table_name** is the name of the table. Actually, a trigger is always associated with a specific table. Without a table, a trigger would not exist hence we have to specify the table name after the 'ON' keyword.
5. **BEGIN...END** is the block in which we will define the logic for the trigger.

Example

Suppose we want to apply trigger on the table Student_age which is created as follows:

```
mysql> Create table Student_age(age INT, Name Varchar(35));
```

Query OK, 0 rows affected (0.80 sec)

Now, the following trigger will automatically insert the age = 0 if someone try to insert age < 0.

```
mysql> DELIMITER //
```

```
mysql> Create Trigger before_inser_studentage BEFORE INSERT ON student_age FOR  
EACH ROW
```

```
BEGIN
```

```
IF NEW.age < 0 THEN SET NEW.age = 0;
```

```
END IF;
```

```
END //
```

Query OK, 0 rows affected (0.30 sec)

Now, for invoking this trigger, we can use the following statements:

```
mysql> INSERT INTO Student_age(age, Name) values(30, 'Rahul');
```

Query OK, 1 row affected (0.14 sec)

```
mysql> INSERT INTO Student_age(age, Name) values(-10, 'Harshit');
```

Query OK, 1 row affected (0.11 sec)

```
mysql> Select * from Student_age;
```

```
+-----+-----+
| age   | Name   |
+-----+-----+
| 30    | Rahul  |
| 0     | Harshit|
+-----+-----+
2 rows in set (0.00 sec)
```

The above result set shows that on inserting the negative value in the table will lead to insert 0 by a trigger.

The above was the example of a trigger with trigger_event as INSERT and trigger_time are BEFORE.

UPDATE TRIGGER

```
CREATE TABLE Student_age_audit
```

```
(
    id INT AUTO_INCREMENT PRIMARY KEY,
    age INT NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    changedat DATETIME DEFAULT NULL,
    action VARCHAR(50) DEFAULT NULL
);
```

```
DELIMITER $$
```

```
CREATE TRIGGER before_Student_age
    BEFORE UPDATE ON Student_age
    FOR EACH ROW
```

```
BEGIN

    INSERT INTO Student_age_audit

    SET action = 'update',

    age = OLD.age,

        last_name = OLD.name,

        changedat = NOW();

END$$

DELIMITER ;

UPDATE Student_age

SET

    age = 25

WHERE

    name = 'Arun';
```



DEPARTMENT OF COMPUTER SCIENCE
CENTRAL UNIVERSITY OF TAMILNADU
MSCP12 : Database Management System Lab.
M.Sc., Computer Science - Semester: I
Lab. Work Sheet – 13
Course Instructor: Appasami G.

Date: 25.10.2019

Time: 2.10 P.M. to 5.10 P.M.

1. Internal Lab. test II.

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events

- ☐ A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- ☐ A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- ☐ A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

Triggers can be written for the following purposes –

- ☐ Generating some derived column values automatically
- ☐ Enforcing referential integrity
- ☐ Event logging and storing information on table access
- ☐ Auditing
- ☐ Synchronous replication of tables
- ☐ Imposing security authorizations
- ☐ Preventing invalid transactions

For creating a new trigger, we need to use CREATE TRIGGER statement. Its syntax is as follows:

```
CREATE TRIGGER trigger_name trigger_time trigger_event
```

```
ON table_name
```

```
FOR EACH ROW
```

```
BEGIN
```

```
...
```

```
END;
```

Here,

1. **Trigger_name** is the name of the trigger which must be put after CREATE TRIGGER statement. The naming convention for trigger_name can be like [trigger time]_[table name]_[trigger event]. For example, before_student_update or after_student_insert can be a name of the trigger.
2. **Trigger_time** is the time of trigger activation and it can be BEFORE or AFTER. We must have to specify the activation time while defining a trigger. We must to use BEFORE if we want to process action prior to the change made on the table and AFTER if we want to process action post to the change made on the table.
3. **Trigger_event** can be INSERT, UPDATE or DELETE. This event causes the trigger to be invoked. A trigger only can be invoked by one event. To define a trigger that is invoked by multiple events, we have to define multiple triggers, one for each event.
4. **Table_name** is the name of the table. Actually, a trigger is always associated with a specific table. Without a table, a trigger would not exist hence we have to specify the table name after the 'ON' keyword.
5. **BEGIN...END** is the block in which we will define the logic for the trigger.

Example

Suppose we want to apply trigger on the table Student_age which is created as follows:

```
mysql> Create table Student_age(age INT, Name Varchar(35));
```

Query OK, 0 rows affected (0.80 sec)

Now, the following trigger will automatically insert the age = 0 if someone try to insert age < 0.

```
mysql> DELIMITER //
```

```
mysql> Create Trigger before_inser_studentage BEFORE INSERT ON student_age FOR  
EACH ROW
```

```
BEGIN
```

```
IF NEW.age < 0 THEN SET NEW.age = 0;
```

```
END IF;
```

```
END //
```

Query OK, 0 rows affected (0.30 sec)

Now, for invoking this trigger, we can use the following statements:

```
mysql> INSERT INTO Student_age(age, Name) values(30, 'Rahul');
```

Query OK, 1 row affected (0.14 sec)

```
mysql> INSERT INTO Student_age(age, Name) values(-10, 'Harshit');
```

Query OK, 1 row affected (0.11 sec)

```
mysql> Select * from Student_age;
```

```
+-----+-----+
| age   | Name   |
+-----+-----+
| 30    | Rahul  |
| 0     | Harshit|
+-----+-----+
2 rows in set (0.00 sec)
```

The above result set shows that on inserting the negative value in the table will lead to insert 0 by a trigger.

The above was the example of a trigger with trigger_event as INSERT and trigger_time are BEFORE.

UPDATE TRIGGER

```
CREATE TABLE Student_age_audit
```

```
(
    id INT AUTO_INCREMENT PRIMARY KEY,
    age INT NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    changedat DATETIME DEFAULT NULL,
    action VARCHAR(50) DEFAULT NULL
);
```

```
DELIMITER $$
```

```
CREATE TRIGGER before_Student_age
    BEFORE UPDATE ON Student_age
    FOR EACH ROW
```



```
BEGIN

    INSERT INTO Student_age_audit

    SET action = 'update',

    age = OLD.age,

        last_name = OLD.name,

        changedat = NOW();

END$$

DELIMITER ;

UPDATE Student_age

SET

    age = 25

WHERE

    name = 'Arun';
```