EXPERIMENT NO.: 1 **DATE:** 07/09/2022

TITLE: Introduction to basic SQL commands

AIM: To study basic SQL queries.

THEORY:

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database. MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons −

- MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and works well even with large data sets.
- MySQL is very friendly to PHP, the most appreciated language for web development.
- MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).

Some of the SQL commands used are: -

1. CREATE
   This command is used to create database/schema. Create relations / views / triggers.
   - create database student;
   - create table student_details(name varchar(20),roll_no int(2));

2. SELECT
   It is used to fetch the data from a database table which returns this data in the form of a result table i.e
   - SELECT * FROM student_details;

8. SELECT-WHERE
   It is used to specify a condition while fetching the data from a single table or by joining with multiple tables. If the given condition is satisfied, then only it returns a specific value from the table. WHERE clause is used to filter the records and fetching only the necessary records. i.e
   - SELECT * FROM student_details where name='Sam';

9. SELECT-WHERE-AND

The AND operator displays a record if all the conditions separated by AND are TRUE i.e

- SELECT * FROM student_details where name='Jay' and roll_no=14;

10. SELECT-WHERE-OR

The OR operator displays a record if any of the conditions separated by OR is TRUE. i.e

- SELECT * FROM student_details where name='Andrew' or name='Harsh';

11. AGGREGATE FUNCTION

An aggregate function performs a calculation on a set of values, and returns a single value. Except for COUNT(*) , aggregate functions ignore null values. Aggregate functions are often used with the GROUP BY clause of the SELECT statement.

Various types of SQL aggregate functions are:

a. Count() - The COUNT() function returns the number of rows in a database table.
   Syntax: COUNT(*) or COUNT([ALL|DISTINCT] expression)

b. Sum() - The SUM() function returns the total sum of a numeric column.
   Syntax: SUM() or SUM([ALL|DISTINCT] expression)

c. Avg() - The AVG() function calculates the average of a set of values.
   Syntax: AVG() or AVG([ALL|DISTINCT] expression)

d. Min() - The MIN() aggregate function returns the lowest value (minimum) in a set of non-NULL values.
   Syntax:MIN() or MIN([ALL|DISTINCT] expression)

e. Max() - The MAX() aggregate function returns the highest value (maximum) in a set of non-NULL values.
   Syntax: AVG() or AVG([ALL|DISTINCT] expression)

12. LIKE Function

The LIKE command is used in a WHERE clause to search for a specified pattern in a column

You can use two wildcards with LIKE:

- % - Represents zero, one, or multiple characters
- _ - Represents a single character (MS Access uses a question mark (?) instead)

eg. The following SQL selects all customers with a CustomerName starting

with "a":
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';

14. Distinct function

The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

- SELECT DISTINCT Syntax

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

PROBLEM STATEMENT:

1) Create the following tables:

i) Employee

Name	Constraint	Data Type
Empno	Primary key	int
Empname	Not null	varchar(25)
Gender		varchar(1)
job		varchar(25)
managerno		int
hiredate		date
salary		Float(12,2)
commision		Float(10,2)
deptno		tinyint

ii) Dept

Name	Constraint	Data Type
Deptno	Primary key	int
dname	Not null	varchar(25)
location		varchar(20)

2) Insert atleast 15 rows in the employee table and atleast 5 rows in dept table0. (commission may be left null for major entries) (Insert data in such a way that every query below should generate a output).

3) Answer the following queries:

a) List all the information about employees in the Employee table.

b) List all the information about departments in the Department table.

c) List the employee number, name, job title and hired date of all employees.
d) List the employee number, name, job title and salary of all employees in department
10.
e) Select name and salary of employees who are clerks.
f) List the department number and name of all departments with department numbers
greater than or equal to 20.
g) List the name of the employees having salary less than 25000/-.
h) List the names of female employees working for deptno 5 earning salary greater than
30000/-.
i) List the name, salary and commission of employees whose commision is greater than
their salary.
j) List the employee number and name of the president.
k) List the employees who do not get commission.
l) Display the names of the employees whose job is either ANALYST or CLERK.
m) Display different kind of jobs available.
n) List names of all employees whose names are 4 characters long.
o) List names of all employees whose names end with letter 'R'.
p) List names of all employees whose names start with 'B' or 'M'.
q) Select all employees whose names fall between 'A' and 'G' alphabetical range.
r) List all the employees details in ascending order of salary.
s) List of the employees departmentwise(i.e. In the increasing order of the dept and all
employees of dept 1 should be displayed together) and within the dept list them in the alphabetical order.
t) List the employee names and hired date in the descending order of hiredate
u) List the names of employees in department 30 from employee table in the descending
order of their salary.
v) List name, salary and PF amount of all the employees. Pf is calculated as 10% of salary.
w) List the names of employees who are more than 2 years old in the organization.

EXPERIMENT NO.: 2 **DATE:** 14/09/2022

TITLE: Data Definition Language Commands

AIM: Execute DDL SQL statements to create and alter a schema.

THEORY:

Data definition language constraints include the following:

1. Create: This command builds a new table and has a predefined syntax.
   Syntax: CREATE TABLE [table name] ([column definitions]) [table parameters];

   It can be used with a few constraints too.
   i) Not null
   It is specified if NULL is not permitted for a particular attribute. This is always implicitly specified for the attributes that are part of the primary key of each relation, but it can be specified for any other attributes whose values are required not to be NULL.
   Syntax: field_name data_type not null
   ii) Unique
   The UNIQUE clause specifies alternate (unique) keys, also known as candidate keys. The UNIQUE clause can also be specified directly for a unique key if it is a single attribute.
   Syntax: field_name data_type unique or unique field_name
   iii) Check
   It is used to specify other table constraints or more general constraints.
   Syntax: field_name data_type check (condition)
   iv) Default
   It is used to give default values to an attribute when there is no value provided by the user.
   Syntax: field_name data_type default 'Default value'
   v) Primary Key
   The PRIMARY KEY clause specifies one or more attributes that make up the primary key of a relation. If a primary key has a single attribute, the clause can follow the attribute directly.
   Syntax: field_name data_type Primary key

2. Drop: A drop command is used to delete objects such as a table, index or view. A DROP statement cannot be rolled back, so once an object is destroyed, there's no way to recover it.
   Syntax: DROP object type object name;
   Dropping a column - ALTER TABLE table_name DROP COLUMN column_name;
   Dropping a constraint - ALTER TABLE table_name DROP constraint (column_name);

   A DROP statement in SQL removes a component from a relational database management system (RDBMS).

3. Alter: An alter command modifies an existing database table. This command can add up additional column, drop existing columns and even change the data type of columns involved in a database table.

Syntax: ALTER object type object name parameters;
Adding a column - ALTER TABLE table_name ADD new_column_name column_definition
Adding a constraint - ALTER TABLE table_name ADD constraint (column_name);

4. Truncate: Similar to DROP, the TRUNCATE statement is used to quickly remove all records from a table. However, unlike DROP that completely destroys a table, TRUNCATE preserves its full structure to be reused later.
   Syntax: TRUNCATE TABLE table_name;

5. Rename: Rename is used to rename the table or rename column in a table.
   Syntax:
   Rename a table - ALTER TABLE table_name RENAME TO new_table_name;
   Rename a column - ALTER TABLE table_name CHANGE COLUMN old_name new_name column_definition

6. Modify: Modifies the datatype, size, constraints.
   Syntax:
   ALTER TABLE table_name MODIFY column_name column_definition [FIRST | AFTER column_name];

PROBLEM STATEMENT:

1) Add the following table

i) Project

Name	Constraint	Data Type
Pno	Primary key	int
Pname	Not null, unique	varchar(25)
location		varchar(25)
dnum	Foreign key	tinyint

ii) works_on: primary key(eno,pno)

Name	Constraint	Data Type
eno	foreign key	int
pno	foreign key	int
hours_per_week		float

2) Make deptno in employee table as the foreign key referencing dept table.
3) Add a new column bdate to employee table with the data type date.
4) Make dname attribute of department unique.
5) Set hours_per_week to a default value 20.
6) Rename Pno attribute in Project table to Proj_num and its datatype to tinyint
7) Rename works_on table as emp_workson_project
8) Change the datatype of deptno to tinyint in department table.
9) Drop the newly added column bdate from the employee table .
10) Take the dump of mysql database.
11) Create a new database dummyemp from the dump file created.
12) Delete the dummyemp database.
13) Create a table dummyproject from the project table and check the data
14) Delete all the data from dummyproject table.
15) Insert all the data from project into dummyproject.(using single insert statement).
16) Create a new table sample with attributes no and name with Innodb engine.
17) Change the Innodb engine to myIsam for the newly created table sample.

EXPERIMENT NO.: 3 **DATE:** 21/09/2022

TITLE: Aggregate Functions

AIM: To execute aggregate functions on a database.

THEORY:

An aggregate function performs a calculation on a set of values, and returns a single value. They are used to summarize information from multiple tuples into a single-tuple summary. Except for COUNT(*) , aggregate functions ignore null values. Aggregate functions are often used with the GROUP BY clause of the SELECT statement.
The COUNT function returns the number of tuples or values as specified in a query. The functions SUM, MAX, MIN, and AVG can be applied to a set or multiset of numeric values and return, respectively, the sum, maximum value, minimum value, and average (mean) of those values

Various types of SQL aggregate functions are:

- Count() - The COUNT() function returns the number of rows in a database table.
  Syntax: COUNT(*) or COUNT([ALL|DISTINCT] expression)

- Sum() - The SUM() function returns the total sum of a numeric column.
  Syntax: SUM() or SUM([ALL|DISTINCT] expression)

- Avg() - The AVG() function calculates the average of a set of values.
  Syntax: AVG() or AVG([ALL|DISTINCT] expression)

- Min() - The MIN() aggregate function returns the lowest value (minimum) in a set of non-NULL values.
  Syntax:MIN() or MIN([ALL|DISTINCT] expression)

- Max() - The MAX() aggregate function returns the highest value (maximum) in a set of non-NULL values.
  Syntax: AVG() or AVG([ALL|DISTINCT] expression)

The select – where commands are used along with these aggregate functions to perform calculations on a specific part of the database. These functions can be used along with each other in a single query to get the desired results in one statement.

For e.g. select sum(Salary), avg(Salary) from Employee;

This query will give the sum of the salaries of all Employees and it’s average together.

PROBLEM STATEMENT:

1. Retrieve the number of employees working in the company.
2. List the number of jobs available in the company.
3. List the total salaries payable to employees.
4. List the minimum and maximum salary offered by the company
5. List the average salary paid to a ‘salesperson’.
6. List the total number of ‘clerks’ in the company.
7. List the number of employees working for department number 1 and minimum salary offered by the department.
8. List the earliest date and latest date on which someone was hired.
9. Count the number of employees whose commission is more than salary.
10. Count the number of projects controlled by department number 1 .

EXPERIMENT NO.: 4 **DATE:** 05/10/2022

TITLE: SQL: JOIN

AIM: To implement the JOIN operator in SQL queries

THEORY:

In real life, we store our data in multiple logical tables that are linked together by a common key value in relational databases like SQL Server and others. As a result, we constantly need to get data from two or more tables into the desired output based on some conditions. We can quickly achieve this type of data in SQL Server using the SQL JOIN clause. This article gives a complete overview of JOIN and its different types with an example.

The join clause allows us to retrieve data from two or more related tables into a meaningful result set. We can join the table using a SELECT statement and a join condition. It indicates how SQL Server can use data from one table to select rows from another table. In general, tables are related to each other using foreign key constraints.

In a JOIN query, a condition indicates how two tables are related:

- Choose columns from each table that should be used in the join. A join condition indicates a foreign key from one table and its corresponding key in the other table.
- Specify the logical operator to compare values from the columns like =, <, or >.

Types of Joins

1. Inner Join - This JOIN returns all records from multiple tables that satisfy the specified join condition. It is the simple and most popular form of join and assumes as a default join. If we omit the INNER keyword with the JOIN query, we will get the same output.

   Syntax:

   ```
   SELECT columns
   FROM table1
   INNER JOIN table2 ON condition1
   INNER JOIN table3 ON condition2
   ```

2. Self Join - A table is joined to itself using the SELF JOIN. It means that each table row is combined with itself and with every other table row. The SELF JOIN can be thought of as a JOIN of two copies of the same tables.

We can do this with the help of table name aliases to assign a specific name to each table's instance. The table aliases enable us to use the table's temporary name that we are going to use in the query. It's a useful way to extract hierarchical data and comparing rows inside a single table.
Syntax:

```
SELECT T1.col_name, T2.col_name...
FROM table1 T1, table1 T2
WHERE join_condition;
```

3. Cross Join - CROSS JOIN in SQL Server combines all of the possibilities of two or more tables and returns a result that includes every row from all contributing tables. It's also known as CARTESIAN JOIN because it produces the Cartesian product of all linked tables. The Cartesian product represents all rows present in the first table multiplied by all rows present in the second table.
   Syntax:

```
SELECT column_lists
FROM table1
CROSS JOIN table2;
```

4. Outer Join - OUTER JOIN in SQL Server returns all records from both tables that satisfy the join condition. In other words, this join will not return only the matching record but also return all unmatched rows from one or both tables.
   It can be further classified into:-

a. Left Outer Join - The LEFT OUTER JOIN retrieves all the records from the left table and matching rows from the right table. It will return NULL when no matching record is found in the right-side table. Since OUTER is an optional keyword, it is also known as LEFT JOIN.
   Syntax:

```
SELECT column_lists
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

b. Right Outer Join - The RIGHT OUTER JOIN retrieves all the records from the right-hand table and matched rows from the left-hand table. It will return NULL when no matching record is found in the left-hand table. Since OUTER is an optional keyword, it is also known as RIGHT JOIN.

Syntax:

```
SELECT column_lists
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
```

c. Full Outer Join - The FULL OUTER JOIN in SQL Server returns a result that includes all rows from both tables. The columns of the right-hand table return NULL when no matching records are found in the left-hand table. And if no matching records are found in the right-hand table, the left-hand table column returns NULL.

Syntax:

```
SELECT column_lists
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

PROBLEM STATEMENT:

1. List employee number, name, department number and the department name.
2. List employee name, department name and the department location.
3. List employee name, department name for all clerks in the company.
4. List employee name, job, his managers name and manager job.
5. Display the names of the employees who are working in sales or research department.
6. Display name and salary of the employee who is working in Bangalore.
7. Display the name and commission of female employees working for the admin department.
8. Display the names of the employees working on ‘Cloud computing’ project.
9. Display all the projects controlled by ‘admin’ department.
10. Display the names of the employees working on ‘DBMS’ project for more than 20 hours per week.
11. Display the names of the employees working on at least one project controlled by the same department to which they belong to.
12. List the names of the projects having the same location as of the department controlling it.
13. List the names of the managers with the project they are working on.
14. List the name and salary of the managers who are working on the ‘Computerization’ project for more than 15 hours per week.
15. Display the names of the employees working on the project located at ‘Mumbai’.

EXPERIMENT NO.: 5 **DATE:** 19/10/2022

TITLE: SQL : Grouping

AIM: To implement Grouping operations in SQL queries.

THEORY:
Grouping is used to create subgroups of tuples before summarization. Grouping and aggregation are required in many database applications.
In many cases we want to apply the aggregate functions to subgroups of tuples in a relation, where the subgroups are based on some attribute values. Here, we need to partition the relation into nonoverlapping subsets (or groups) of tuples. Each group (partition) will consist of the tuples that have the same value of some attribute(s), called the grouping attribute(s). We can then apply the function to each such group independently to produce summary information about each group. SQL has a GROUP BY clause for this purpose. The GROUP BY clause specifies the grouping attributes, which should also appear in the SELECT clause, so that the value resulting from applying each aggregate function to a group of tuples appears along with the value of the grouping attribute(s).
If NULLs exist in the grouping attribute, then a separate group is created for all tuples with a NULL value in the grouping attribute.

SQL provides a HAVING clause, which can appear in conjunction with a GROUP BY clause, for this purpose. HAVING provides a condition on the summary information regarding the group of tuples associated with each value of the grouping attributes. Only the groups that satisfy the condition are retrieved in the result of the query.

The GROUP BY Statement in SQL is used to arrange identical data into groups with the help of some functions i.e., if a particular column has same values in different rows then it will arrange these rows in a group.

Important Points:

- GROUP BY clause is used with the SELECT statement.
- In the query, GROUP BY clause is placed after the WHERE clause.
- In the query, GROUP BY clause is placed before ORDER BY clause if used any.
- In the query , Group BY clause is placed before Having clause .
- Place condition in the having clause

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

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
```

```
GROUP BY column_name(s)
ORDER BY column_name(s);
```

The SELECT clause lists the attributes or functions to be retrieved. The FROM clause specifies all relations (tables) needed in the query, including joined relations, but not those in nested queries. The WHERE clause specifies the conditions for selecting the tuples from these relations, including join conditions if needed. GROUP BY specifies grouping attributes, whereas HAVING specifies a condition on the groups being selected rather than on the individual tuples. The built-in aggregate functions COUNT, SUM, MIN, MAX, and AVG are used in conjunction with grouping, but they can also be applied to all the selected tuples in a query without a GROUP BY clause. Finally, ORDER BY specifies an order for displaying the result of a query.

Note that the SELECT clause includes only the grouping attribute and the aggregate functions to be applied on each group of tuples.

Having Clause:

We know that WHERE clause is used to place conditions on columns but what if we want to place conditions on groups? This is where HAVING clause comes into use. We can use HAVING clause to place conditions to decide which group will be the part of final result-set. The HAVING clause enables you to specify conditions that filter which group results appear in the results. The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

Advantages of Group by:

1. Aggregations can be filtered using the HAVING clause
   The WHERE clause cannot be used on aggregations. This is because the where statement is evaluated before any aggregations take place. The alternate having is placed after the group by and allows you to filter the returned data by an aggregated column.

2. An alternative to SELECT DISTINCT
   The GROUP BY can be used as an alternative to the SELECT DISTINCT structure. The only difference would be that the data would be sorted in a different way by both the methods.

3. Data Aggregation
   The GROUP BY clause is often used in SQL statements which retrieve numerical data. It is commonly used with SQL functions like COUNT, SUM, AVG, MAX and MIN and is used mainly to aggregate data. Data aggregation allows values from multiple rows to be grouped together to form a single row.

PROBLEM STATEMENT:

1. Find out the number of employees in every dept. display the dept name and the count. Also display the maximum, minimum and the average salary offered by each dept.
2. List the total salary payable to each dept. display the dept name and the total salary.
3. List the jobs and the number of employees in each job. The result should be in the descending order of the number of jobs.
4. List the jobwise total salary, average salary and minimum salary of employees.
5. List the total salary of employees jobwise for 'research' dept.
6. List the job wise total salary, average salary of employees of 'admin' dept. Display only those jobs having average salary greater than 50000.
7. Retrieve the total number of projects controlled by every dept. Display the dept name and the count.
8. Retrieve the total number of employees working on every project. Display the project no., name and the count.
9. Retrieve the total time (in hours) spent on every project.
10. Retrieve the details of project with more than three employees working on it. Also display the average salary offered for that project.
11. For each dept having more than two female employees, retrieve the maximum and minimum salary paid to the female employee.
12. For each employee display the total number of projects he is working on, and also display the total time (hours per week) he spends on all projects.

EXPERIMENT NO.: 6 **DATE:** 31/10/2022

TITLE: Nested Subqueries

AIM: To study nested subqueries in SQL

THEORY:

A subquery/ inner query or a nested query is a query within another SQL query and embedded within the WHERE clause. A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved. Basically, in nested queries, a query is written inside a query and the result is used in execution of outer query.

Subqueries can be used with the SELECT, INSERT, UPDATE and DELETE statements along with operators like =, <, >, >=, <=, IN, BETWEEN, etc.

Rules to be followed:

1. Subqueries must be enclosed within parenthesis.
2. A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
3. An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
4. Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
5. A subquery cannot be immediately enclosed in a set function.
6. The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

Types of Nested Queries:

1. Independent Nested Queries: In independent nested queries, query execution starts from innermost query to outermost queries. The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query. Various operators like IN, NOT IN, ANY, ALL etc., are used in writing independent nested queries.
2. Co-related Nested Queries: In co-related nested queries, the output of inner query depends on the row which is being currently executed in outer query.

The IN, ANY & AND Operators:

1. IN - The SQL IN condition (sometimes called the IN operator) allows you to easily test if an expression matches any value in a list of values. It is used to help reduce the need for multiple OR conditions in a SELECT, INSERT, UPDATE, or DELETE statement. It compares a value v with a set (or multiset) of values V and evaluates to TRUE if v is one of the elements in V.

2. ANY - ANY compares a value to each value in a list or results from a query and evaluates to true if the result of an inner query contains at least one row. ANY returns true if any of the subqueries values meet the condition. ANY must be preceded by comparison operators >, >=, <, <=, = or !=. If the subquery returns no row, the condition evaluates to false.

3. ALL - ALL operator is used to select all tuples of SELECT STATEMENT. It is also used to compare a value to every value in another value set or result from a subquery. The ALL operator returns TRUE if and only if all of the subqueries values meet the condition. The ALL must be preceded by comparison operators >, >=, <, <=, = or !=. It evaluates to true if all of the subqueries values meet the condition. ALL is used with SELECT, WHERE, HAVING statement.

PROBLEM STATEMENT:

1. List all the employees who have a same job as 'Smith'.
2. Display the names of the employees who are working in sales or research department.
3. Display the name and salary of the employee who is working in 'Bangalore'.
4. List the employee names whose salary is greater than the lowest salary of the employees of 'research' department.
5. List the employee names whose salary is greater than the highest salary of the employees of 'admin' department.
6. List the name of the employee earning the highest salary.
7. List the names of the employees earning highest salary in every department.
8. List the details about employees whose salary is second highest in the company.
9. List the details about employees who have maximum number of people reporting to them.
10. List the employees who earn more than the average salary in their own department.
11. List the employee and his managers details, where the employee’s salary is greater than the managers salary.
12. Retrieve the name of the department which is offering average salary less than the average salary offered by the company.
13. For each department that has more than 2 employees, retrieve the department number and number of its employee’s earning salary above 40k. Also retrieve the name of the department.
14. Find the employee who spends maximum number of hours on projects.
15. Find an employee who works on all the projects controlled by department number 2.
16. Find the employee who works for maximum hours on a single project.
17. Find the employee who earns the maximum salary in every dept.

EXPERIMENT NO.: 7 **DATE:** 14/11/2022

TITLE: Views and Temporary Tables

AIM: To implement views and temporary tables in SQL

THEORY:

A view in SQL terminology is a single table that is derived from other tables. These other tables can be base tables or previously defined views. A view does not necessarily exist in physical form; it is considered to be a virtual table, in contrast to base tables, whose tuples are always physically stored in the database. This limits the possible update operations that can be applied to views, but it does not provide any limitations on querying a view. We can think of a view as a way of specifying a table that we need to reference frequently, even though it may not exist physically.

Syntax:

CREATE VIEW *view_name*
AS SELECT *attributes*
FROM t*able_name*
WHERE *conditions*;

A view is supposed to be always up-to-date; if we modify the tuples in the base tables on which the view is defined, the view must automatically reflect these changes. Hence, the view does not have to be realized or materialized at the time of view definition but rather at the time when we specify a query on the view. It is the responsibility of the DBMS and not the user to make sure that the view is kept up-to date.
However, issuing an INSERT, DELETE, or UPDATE command on a view table is in many cases not possible. In general, an update on a view defined on a single table without any aggregate functions can be mapped to an update on the underlying base table under certain conditions. For a view involving joins, an update operation may be mapped to update operations on the underlying base relations in multiple ways. Hence, it is often not possible for the DBMS to determine which of the updates is intended.

Temporary Tables are created in the Tempdb and are automatically deleted as soon as the last connection is terminated. Temporary Tables helps us to store and process intermediate results. Temporary tables are very useful when we need to store temporary data.

Syntax:

CREATE TEMPORARY TABLE temporary_table_name (*subquery*) LIMIT *no_of_rows*

The above is the syntax to create temporary tables. The subquery defines the table. The structure of the table is copied as it is. The LIMIT keyword specifies the number of rows that you want to be copied from the original table.

There are 2 types of Temporary Tables: Local Temporary Table and Global Temporary Table. These are explained as following below.

1. Local Temporary Table:
   A Local temporary table is available only for the session that has created it. It is automatically dropped (deleted) when the connection that has created it, is closed. Also, the user can drop this temporary table by using the "DROP TABLE EmpDetails" query. There will be random numbers are appended to the name of table name. If the temporary table is created inside the stored procedure, it gets dropped automatically upon the completion of stored procedure execution.
2. Global Temporary Table:
   Global temporary tables are visible to all connections and dropped when the last connection referencing the table is closed. Global table name must have a unique table name. There will be no random numbers suffixed at the end of the table name.

PROBLEM STATEMENT:

1. Create a view containing the information about all employees belonging to admin department check if view is updatable.
2. Create a view for every project containing project no, pname, pbudget, status, plocation, duration and the name, department name, salary of employees working on that project and the no. of hours employee works on the project and check if the view is updatable
3. Create a view to store the number of employees belonging to every department ,their average salary and name of the department and check if the view is updatable
4. Create a view to store name, empid, salary and also the details of project he works on like pno, pname, hours and status
5. Using the view created above create another view to store the no. of employees working on every project and total hours spent on that project
6. Using the view created in question 4 create another view to store the no. of projects each employee works on and the total hours
7. Create a temporary table to store the details of project located at Bangalore along with employees' details working on it.
8. Using the temporary table above find the employees working all projects located at Bangalore.
9. Using the views created above answer the following questions
   a) Find the project with a minimum no. of employees working on it.
   b) Find the employees who works on the maximum projects
   c) Find the department with average salary greater than average offered by company

EXPERIMENT NO.: 8 **DATE:** 22/11/2022

TITLE: Users and Authorization

AIM: To implement multiple users and authorization operations in SQL

THEORY:

1. Users

The MySQL user is a record in the **USER** table of the MySQL server that contains the login information, account privileges, and the host information for MySQL account. It is essential to create a user in MySQL for accessing and managing the databases.

The MySQL Create User statement allows us to create a new user account in the database server. It provides authentication, SSL/TLS, resource-limit, role, and password management properties for the new accounts. It also enables us to control the accounts that should be initially locked or unlocked.

Syntax:

- To create a new user and set a password for the user at the same time:
  CREATE USER '*username*'@'localhost' identified by '*password*';

- To login as a user:
  mysql -u *username* -p
  When you are prompted, enter the password for the user.

- To show all users:
  Select user from mysql.user;

- To show current user:
  Select user();

- To drop user:
  drop user '*username*'

2. Privileges

In MySQL, a **privilege** is a right to perform an action on a database that must be granted to users. This effectively defines a user's access level on a database and what they can do within it. We can organize these privileges by scope into levels:

- Global privileges
  Apply to all databases on the server. Administrative privileges fall into the global group because they enable a user to manage operations of the MySQL server and aren't specific to a particular database.
- Database privileges

Apply to specific databases in your MySQL instance and all of the objects within those databases (e.g., tables, columns, and views). You can also grant database privileges globally.

- Proxy privileges
  Allow a user to act as if they have the privileges granted to another user.
- Privileges for database objects (tables, columns, stored routines, views, etc.)
  Can apply to all objects of one type within a particular database or to specific objects, such as a certain table or view. You can also grant database object privileges globally.

We can choose access right from the below list on which privileges can be applied.

1. SELECT: It enables us to view the result set from a specified table.
2. INSERT: It enables us to add records in a given table.
3. DELETE: It enables us to remove rows from a table.
4. CREATE: It enables us to create tables/schemas.
5. ALTER: It enables us to modify tables/schemas.
6. UPDATE: It enables us to modify a table.
7. DROP: It enables us to drop a table.
8. INDEX: It enables us to create indexes on a table.
9. ALL: It enables us to give ALL permissions except GRANT privilege.
10. GRANT: It enables us to change or add access rights.

Syntax:
To grant privilege:

GRANT privilege_name(s) **ON** database_name.table_name **TO** user_account_name;

To grant privileges to all databases or tables we use a *.

To show grants for user:

SHOW GRANTS **FOR** user_account_name;

To revoke privilege:

REVOKE privilege [privilege] ON database_name.table_name privilege_level **FROM** user1 [, user2] ..;

3. Indexes

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

Syntax:

- To create index
  CREATE INDEX *index_name* ON *table_name* (*column1*, *column2*, ...);

- To drop index

ALTER TABLE *table_name* DROP INDEX *index_name*;

PROBLEM STATEMENT:

1. Create 2 users emp1, emp2 and emp3.
2. Give all privileges on the view created in question 1(expt7)to emp1.
3. Give select and update privilege on name and dno attribute of employee table to emp2 with grant option.
4. Let emp2 grant these privileges to emp1.
5. Grant all privileges on all the databases on the MySQL server to emp3.
6. Login and check the privileges.
7. Revoke the view privileges from emp1.
8. Provide a list of various privileges available to the user.
9. Create an index on empid attribute of employee table.

EXPERIMENT NO.: 9 **DATE:** 05/12/2022

TITLE: Procedures

AIM: To implement procedures in SQL

THEORY:

A procedure (often called a stored procedure) is a collection of pre-compiled SQL statements stored inside the database. It is a subroutine or a subprogram in the regular computing language. A procedure always contains a name, parameter lists, and SQL statements**.** We can invoke the procedures by using triggers, other procedures and applications such as Java, Python, PHP, etc. It was first introduced in MySQL version 5**.** Presently, it can be supported by almost all relational database systems.

If we consider the enterprise application, we always need to perform specific tasks such as database clean up, processing payroll, and many more on the database regularly. Such tasks involve multiple SQL statements for executing each task. This process might easy if we group these tasks into a single task. We can fulfill this requirement in MySQL by creating a stored procedure in our database.

A procedure is called a recursive stored procedure when it calls itself. Most database systems support recursive stored procedures but, it is not supported well in MySQL.

Parameter Types:

1. IN parameter

It is the default mode. It takes a parameter as input, such as an attribute. When we define it, the calling program has to pass an argument to the stored procedure. This parameter's value is always protected.

2. OUT parameters

It is used to pass a parameter as output. Its value can be changed inside the stored procedure, and the changed (new) value is passed back to the calling program. It is noted that a procedure cannot access the OUT parameter's initial value when it starts.

3. INOUT parameters

It is a combination of IN and OUT parameters. It means the calling program can pass the argument, and the procedure can modify the INOUT parameter, and then passes the new value back to the calling program.

Syntax:

To create procedures:

```
DELIMITER &&
CREATE PROCEDURE procedure_name [[IN | OUT | INOUT] parameter_name datatype [
, parameter datatype]) ]
BEGIN
   Declaration_section
   Executable_section
```

END &&

DELIMITER ;

To call procedure:

CALL procedure_name (parameter(s))

To delete procedure:

DROP PROCEDURE [IF EXISTS] procedure_name;

PROBLEM STATEMENT:

1. Write a procedure to get details of all projects.
2. Write a procedure which accepts 'empid' as parameter and display his details.
3. Write a procedure which accepts 'project_no' as parameter and display the controlling dept_no and no. of employees working on that project.
4. Write procedure which accepts empid as an input and finds the salary and no. of projects he is working on if sal>50000 and no of projects his working on is >2 increase the salary by 5% if salary is between 50000>=60000 and no. of projects >2 increase salary by 2%,if sal>100000 and no of projects >=1 increase salary by 1%. Display empid, sal, no. of projects, increased salary.
5. Write a procedure which executes the same task as in procedure 4 but for all the employees in the database

EXPERIMENT NO.: 10 **DATE:** 12/12/2022

TITLE: Triggers

AIM: To implement triggers in SQL.

THEORY:

A trigger is a set of SQL statements that reside in system memory with unique names. It is a specialized category of stored procedure that is called automatically when a database server event occurs. Each trigger is always associated with a table.

Trigger is stored into database and invoked repeatedly, when specific condition match. Triggers are stored programs, which are automatically executed or fired when some event occurs.

A trigger is called a special procedure because it cannot be called directly like a stored procedure. The key distinction between the trigger and procedure is that a trigger is called automatically when a data modification event occurs against a table. A stored procedure, on the other hand, must be invoked directly.

The following are the main characteristics that distinguish triggers from stored procedures:

- We cannot manually execute/invoked triggers.
- Triggers have no chance of receiving parameters.
- A transaction cannot be committed or rolled back inside a trigger.

Syntax:

- To Create a Trigger

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[FOR EACH ROW]
WHEN (condition)
DECLARE
  Declaration-statements
BEGIN
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;
```

- CREATE TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise, the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

➢ Viewing a Trigger

To view all the triggers created by user, a data dictionary USER_TRIGGERS could be used. Use select statement on USER_TRIGGERS as shown below:
Select trigger_name from user_triggers;

➢ Deleting a trigger

If trigger is not required, then it can be deleted using the DDL drop command as shown below:
Drop trigger <trigger_name>;

PROBLEM STATEMENT:

1. Create 4 tables with attributes mentioned in the brackets: table1(a1), table2(a2), table3(a3), table4(a4,b4)
   Create a trigger on table1 to insert, delete, update tables table2,table3,table4 respectively. Whenever we insert a value in table1(a1) the same value should be inserted in table2; the number is present in table3, it should be deleted from table3 and if the number is present in table4(a4) then b4 should be updated(incremented by 1).
2. Create a table emp_proj with empid, number of projects he is working on, total number of hrs. Create another table proj_info with project number, number of employees working on that project, total number of hours spent on that project. Whenever we delete an entry from the works_on table respective changes must be done to emp_proj and proj_info tables.

GOA COLLEGE OF ENGINEERING

ASSIGNMENT NO: 1 **DATE:** 03/09/2022

1) List the features, advantages and disadvantages of the various software stacks available for MySQL .(ex. XAMPP, LAMPP,WAMP etc)

Ans:

a. XAMPP

Features

- Supporting Platforms: It is a cross-platform software package supported by platforms like Linux, Windows, and Mac OS.
- Programming Languages: The programming/ scripting languages used for development in XAMPP are Perl and PHP.
- Database: XAMPP uses MariaDB, which is a relational database management system. It was developed by MySQL.
- Servers: Apache Server is used for testing and running webpages of local hosts.

Advantages:

- It is free and easy to use and easily available for Windows, Linux and Mac OS.
- It is a beginner friendly solution package for full stack web development.
- It is an open-source software package which gives a easy installation experience.

Disadvantages:

- No Password for the Database Administrator
- MySQL Can Be Accessed Over a Network
- The Local Mail Server Is Not Secure

b. MAMPP

Features:

- Supporting Platforms: This stack of software is only for the MAC operating system.
- Programming Languages: The coding for the development and testing is done by using PHP in the MAMP server.
- Database: MAMP stores its data in a relational database. It uses MySQL for data storage and retrieval.
- Server: It uses Apache webserver.

Advantages:

- MAMP is much easier to use. Installation is just a few clicks away in the MAMP software.
- One of the advantages of using the MAMP server is that it provides various tools to the user, like the ability to set up the Nginx server, mobile testing tools and built-in editors.
- It provides a lot more tools, including the WordPress development tools.

Disadvantages:

- Launching multiple local projects is complicated
- There's no compartmentalization between local projects
- You can't modify the MAMP software stack.

c. WAMP

Features:

- Supporting Platforms: WAMP local server is only supported by Windows Operating system.
- Programming Languages: WAMP uses PHP (a script-based programming language) for development and testing.
- Database: Just like MAMP, WAMP uses MySQL, which is an RDBMS for storing and retrieving operations on data.
- Server: WAMP uses the Apache Web server.

Advantages:

- It is easy to Use. (Changing Configuration)
- WAMP makes easy to code PHP and Creating Databases (in MySQL) in Windows platform.
- WAMP is Available for both 64 bit and 32-bit system.

Disadvantages:

- It is not easy to install as compared to XAMPP.

d. LAMPP

Features:

- Supporting Platforms: LAMP is supported by a single platform i.e., Linux based systems
- Programming Languages: Unlike other similar local servers, LAMP is multi-lingual in terms of development. It supports coding done in PHP, Perl, and Python.
- Database: LAMP supports its data storage function and other data-based operations using MySQL RDBMS.
- Server: LAMP, like the other local servers, uses Apache Web server.

Advantages:

- Scalability
- Highly secure
- Fast development

Disadvantages:

- Lack of Separation
- Library Captivity
- Limited Scalability of Infrastructure

2) List and explain the features of top five relational databases.

Ans:

a. Oracle

- Scalability and Performance: Features like Real Application Clustering and Portability make a database scalable according to its usage. In a multiuser database, it is required to control data consistency and concurrency which Oracle contemplates.
- Availability: Real-time applications require high data availability. High-performing computing environments are configured to provide all-time data availability. Data is available during the time of planned or unplanned downtimes and failures.
- Backup and Recovery: Its layout has complete recovery features to recover data from almost all kinds of failures. In case of failure, the database needs to be recovered within no time for high availability. Unaffected parts of data are available while the affected ones are getting recovered.
- Security: Securing the data is always the top priority. It provides mechanisms to control data access and usage. Implementing authorization and editing user actions can prevent unauthorized access and allow distinct access to the users.

b. MySQL

- Open-Source: This software can be downloaded, used and modified by anyone. It is free-to-use and easy-to-understand. The source code of MySQL can be studied, and changed based on the requirements.
- Quick and Reliable: MySQL stores data efficiently in the memory ensuring that data is consistent, and not redundant. Hence, data access and manipulation using MySQL is quick.
- Scalable: Scalability refers to the ability of systems to work easily with small amounts of data, large amounts of data, clusters of machines, and so on. MySQL server was developed to work with large databases.
- Data Types: It contains multiple data types such as unsigned integers, signed integers, float (FLOAT), double (DOUBLE), character (CHAR), variable character (VARCHAR), text, blob, date, time, datetime, timestamp, year, and so on.

c. Microsoft SQL Server

- Run SQL Server anywhere: Use SQL Server with Windows and Linux containers, plus deploy and manage your deployments using Kubernetes.
- Always encrypted data enclaves: Encrypt sensitive data and execute rich computations on encrypted data, plus enable customised data access based on role with complex row filtering.
- Maximum availability: Gain mission-critical uptime, fast failover and improved TCO using cloud disaster recovery replicas and always on availability groups.

d. PostgreSQL

- Highly extensible: PostgreSQL is highly extensible in several phases which are as following:
  - It supports procedural Languages such as Perl, PL/PGSQL, and Python, etc.
  - JSON/SQL path expressions
  - Stored procedures and functions.
- Compatible with Data Integrity: It supports data integrity which includes the following:
  - Primary Keys
  - UNIQUE, NOT NULL
  - Foreign Keys
  - Explicit Locks, Advisory Locks
- Compatible with multiple data types: PostgreSQL support various data types such as:
  - Structured: Array, Date and Time, UUID (Universally Unique Identifier), Array, Range.
  - Primitives: String, Integer, Boolean, Numeric.
  - Customizations: Custom Types, Composite.

e. IBM Db2

- Advanced protection: Advanced authorization, encryption at rest and in transit, and comprehensive security controls for managing GDPR compliance
- Scaling: Elastic scaling up to 128 machines in multi- cloud and hybrid environments to reduce storage costs; data federation eliminates data silos
- Continuous availability: Auto resynchronization and recovery plus clustering with IBM pureScale® to keep business running 24x7

3) Explain the following NOSQL types and explain one example in each type: a) Document store b) Key-value store c) Graph databases d) Time series databases.

Ans: a. Document store

- A document-oriented database or a NoSQL document store is a modern way to store data in JSON format rather than simple rows and columns. It allows you to express data in its natural form the way it's meant to be.
- In contrast to rows and columns, NoSQL databases keep data in documents. These documents follow a minimum of standard format rules (so that the database can understand it for post processing). The format used could be JSON, XML, YAML etc. The JSON format is the format of choice for NoSQL databases, and for good reason. A JSON document is simply more compact and more readable.

Example: MongoDB

MongoDB is a scalable, flexible NoSQL document database platform designed to overcome the relational databases approach and the limitations of other NoSQL solutions. MongoDB is well known for its horizontal scaling and load balancing capabilities, which has given application developers an unprecedented level of flexibility and scalability.

b. Key Value Store

- This specific type of NoSQL database uses the key-value method and represents a collection of numerous key-value pairs. The keys are unique identifiers for the values. The values can be any type of object - a number or a string, or even another key-value pair in which case the structure of the database grows more complex.
- Key names can range from as simple as numbering to specific descriptions of the value that is about to follow. A key-value database can be thought of as a dictionary or a directory. Dictionaries have words as keys and their meanings as values.

Example: Amazon DynamoDB

Amazon DynamoDB is a NoSQL managed database service provided by Amazon that stores semi-structured data like key-value pairs. A DynamoDB table consists of items. Each item consists of one partition key and one or more attributes. A partition key is used to differentiate between items. A query operation in DynamoDB finds items based on primary key values. The name of the partition key attribute and a single value for that attribute must be provided. The query returns all items searched against that partition key value.

c. Graph Databases

- A graph database is a type of database used to represent the data in the form of a graph. It has three components: nodes, relationships, and properties. These components are used to model the data.
- The concept of a Graph Database is based on the theory of graphs. They are commonly referred to NoSql databases as data is stored using nodes, relationships and properties instead of traditional databases.
- A graph database is very useful for heavily interconnected data. Here relationships between data are given priority and therefore the relationships can be easily visualized. They are flexible as new data can be added without hampering the old ones. They are useful in the fields of social networking, fraud detection, AI Knowledge graphs etc.

Example: Neo4j

Neo4j is the world's leading open-source Graph Database which is developed using Java technology. It is highly scalable and schema free (NoSQL). Neo4j is a native graph database, which means that it implements a true graph model all the way down to the storage level.

The data isn't stored as a "graph abstraction" on top of another technology, it's stored just as you whiteboard it. This is important because it's the reason why Neo4j outperforms other graphs and stays so flexible.

d. Time Series Databases

- A time-series database (TSDB) is a computer system that is designed to store and retrieve data records that are part of a "time series," which is a set of data points that are associated with timestamps. The timestamps provide a critical context for each of the data points in how they are related to others.
- Time series data is often a continuous flow of data like measurements from sensors and intraday stock prices. A time-series database lets you store large volumes of timestamped data in a format that allows fast insertion and fast retrieval to support complex analysis on that data.

Example: TimescaleDB

TimescaleDB is an open-source database designed to make SQL scalable for time-series data. It is engineered up from PostgreSQL and packaged as a PostgreSQL extension, providing automatic partitioning across time and space (partitioning key), as well as full SQL support.