EXPERIMENT NO: 1

AIM: Study of Open Source Relational Databases: MySQL

HARDWARE & SOFTWARE REQUIREMENT: MySQL Installer 8.0.33 Server, Any CPU with Pentium Processor or similar, 8 GB RAM or more, 500 GB Hard Disk or more.

THEORY:

What is an open source database?

An open source database is any database application with a codebase that is free to view, download, modify, distribute, and reuse. Open source licenses give developers the freedom to build new applications using existing database technologies.

A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.

Database Applications:

• Banking: transactions • Airlines: reservations, schedules • Universities: registration, grades • Sales: customers, products, purchases • Online retailers: order tracking, customized recommendations • Manufacturing: production, inventory, orders, supply chain • Human resources: employee records, salaries, tax deductions

How do open source database management systems work?

Open source database management systems provide a layer of abstraction developers can use to store information for organizations and their applications.

Databases are typically categorized into two groups:

- Relational databases: The traditional data storage approach in which key-value pairs are used to store structured data into tables consisting of columns and rows.

- NoSQL (non-relational) databases: Data stored using alternative data storage architectures, including document data store, column-oriented database, key-value store, and graph databases. Non-relational databases are the preferred choice for handling unstructured data.

Database management systems give you the software layer you need to control and manage your data for a multitude of purposes. For example, you can store business intelligence in a relational database for fast SQL queries or save unstructured image files in a graph database for an AI-powered analytics app.

Closed source vs. open source databases

Closed source databases are proprietary software. The source code cannot be accessed, modified, distributed, or reused. You may have to pay a subscription or licensing fees to use the database within your applications. The company that wrote the code maintains the codebase. That means you'll have to wait for the company to add new features or address any bugs in the database management system.

In contrast, anyone can view and access the source code for open source databases. There are no licensing fees so the total cost of ownership (TCO) is lower for open source databases than for commercial databases. You can download and modify source code to power your apps free of charge and without vendor lock-in. On the flip side, though, you're responsible for maintaining and securing your implementation of the open source database.

Examples of open source databases

Whether your goal is to store structured data for MySQL queries or unstructured data via JSON objects, there are plenty of open source database solutions to choose from on the web.

Examples of commonly used open source relational databases include:

- MySQL
- PostgreSQL
- MariaDB

Examples of commonly used open source NoSQL databases include:

- MongoDB
- CouchDB
- Cassandra

Data Models:

- Relational model
- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Semistructured data model (XML)
- Other older models: o Network model o Hierarchical model

A database is a means of storing information in such a way that information can be retrieved from it. In simplest terms, a relational database is one that presents information in tables with rows and columns. A table is referred to as a relation in the sense that it is a collection of objects of the same type (rows). Data in a table can be related according to common keys or concepts, and the ability to retrieve related data from a table is the basis for the term relational database. A Database Management System (DBMS) handles the way data is stored, maintained, and retrieved. In the case of a relational database, a Relational Database Management System (RDBMS) performs these tasks. A relational database management system (RDBMS) is a program that lets you create, update, and administer a relational database. Most commercial RDBMS's use the Structured Query Language (SQL) to access the database, although SQL was

invented after the development of the relational model and is not necessary for its use. MySQL open source RDBMS overview: MySQL is a popular open source relational database management system (RDBMS) choice for web-based applications. Developers, database administrators and DevOps teams use MySQL to build and manage next-generation web- and cloud-based applications. With most open source RDBMS options, MySQL is available in several different editions and runs on Windows, OS X, Solaris, FreeBSD and other variants of Linux and Unix:

- MySQL Classic Edition, available to only independent software vendors, OEMs and value added resellers, is designed to be an embeddable database for read-intensive applications.
- MySQL Community Editionis the free downloadable version of MySQL available under the GNU General Public License (GPL).
- MySQL Standard Editionis the entry-level RDBMS offering for online transaction processing
- MySQL Enterprise Editionadds advanced features, management tools (including OEM for MySQL) and technical support.
- MySQL Cluster Carrier Grade Editionis designed for Web and cloud development.

Data types supported by MySQL open source RDBMS: MySQL data types include numeric types, date and time types, string types (including binary, character and Binary Large Object), and spatial types. Additionally, MySQL will map certain data types from other DBMSes to MySQL data types for easier portability.

What is Relational Database?

Relational database means the data is stored as well as retrieved in the form of relations (tables). Table 1 shows the relational database with only one relation called **STUDENT** which stores **ROLL_NO**, **NAME**, **ADDRESS**, **PHONE** and **AGE** of students.

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

There are some important terminologies that are used in terms of relation.

Attribute: Attributes are the properties that define a relation. e.g.; ROLL_NO, NAME etc.

Tuple: Each row in the relation is known as tuple. The above relation contains 4 tuples, one of which is shown as:

1	RAM	DELHI	9455123451	18

Degree: The number of attributes in the relation is known as degree of the relation. The STUDENT relation defined above has degree 5.

Cardinality: The number of tuples in a relation is known as cardinality. The STUDENT relation defined above has cardinality 4.

Column: Column represents the set of values for a particular attribute. The column ROLL_NO is extracted from relation STUDENT.

ROLL_NO

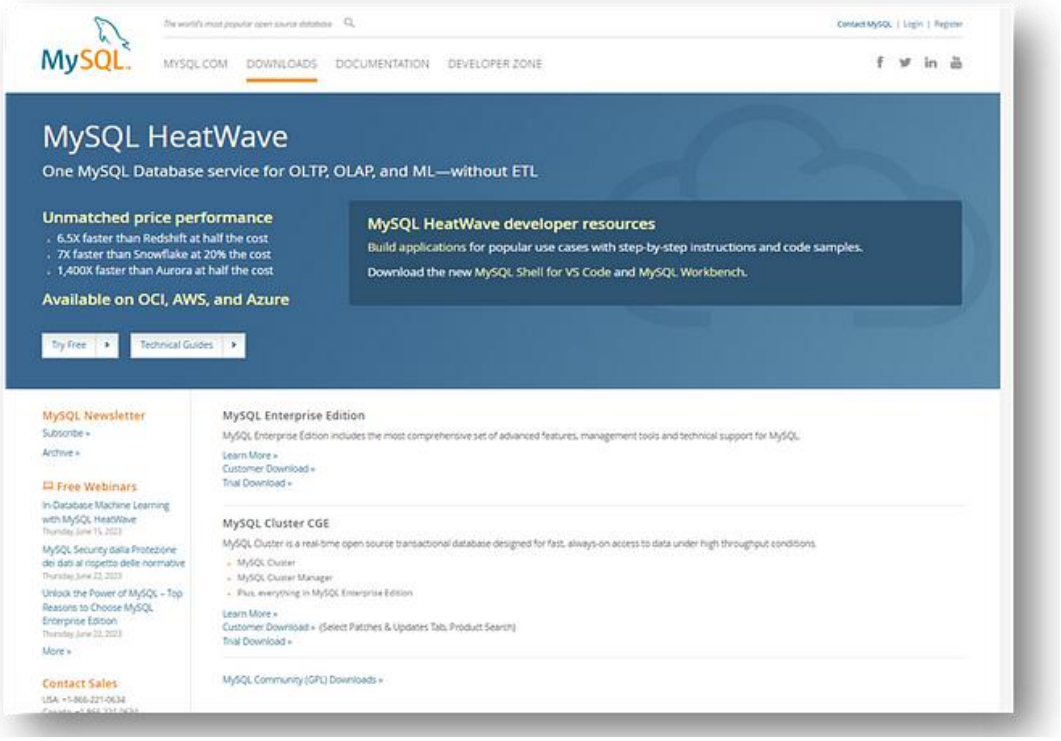
Installation Output:

Listed below are the steps to be followed to install MySQL Server:

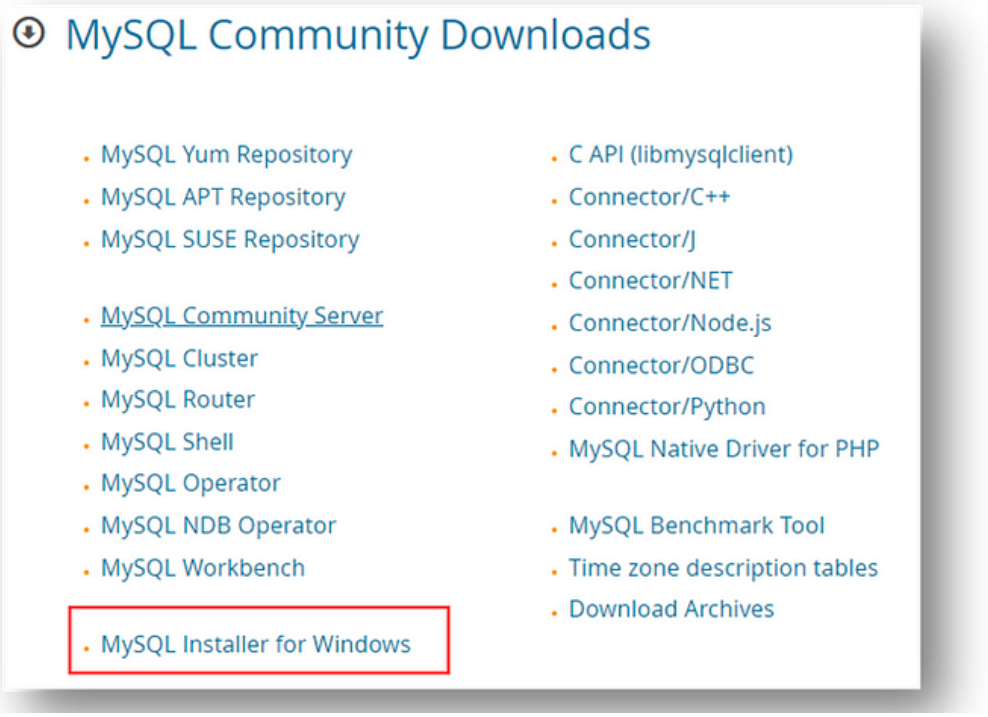
01. Go to any web browser. Then type as "mysql" on the browser.

Now press enter button. After pressing, you can see mysql official web link. Now click on it. Then you can see a window like this.

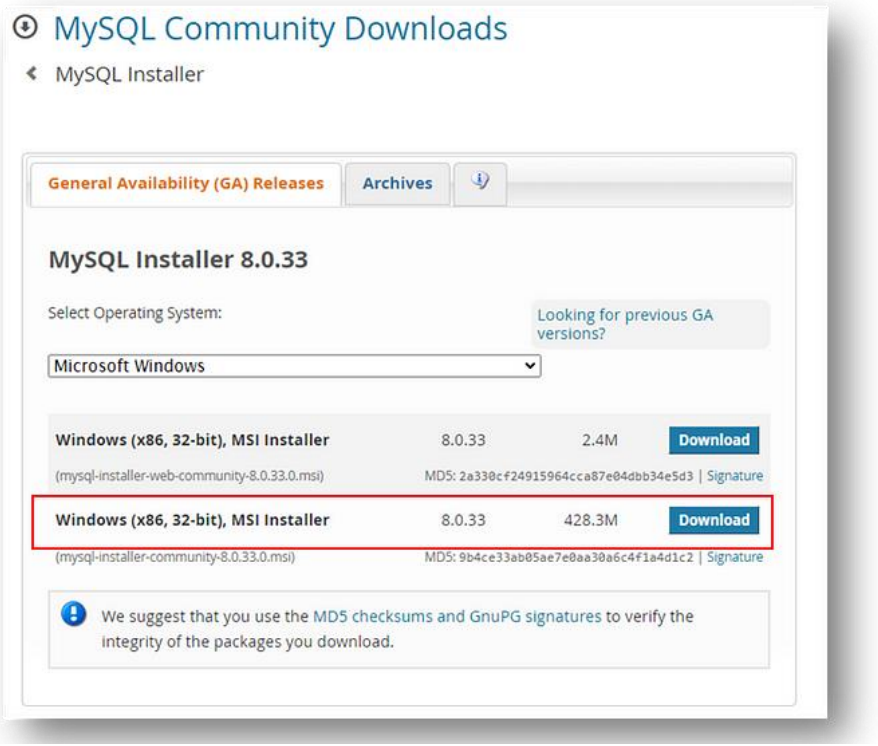
02. In here click on the "DOWNLOADS" tab. Then you can see a window like this



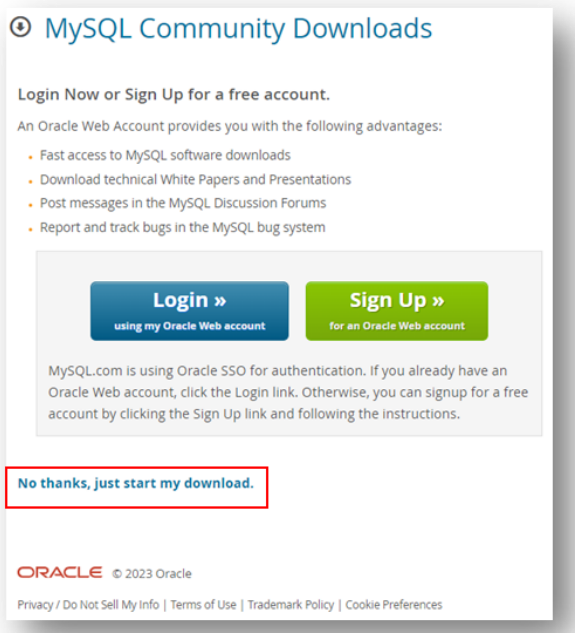
03. In here click on "MySQL Community (GPL) Downloads" Then you can see a window like this.



04. In here click on "MySQL Installer for Windows" and then you will find a window like this.

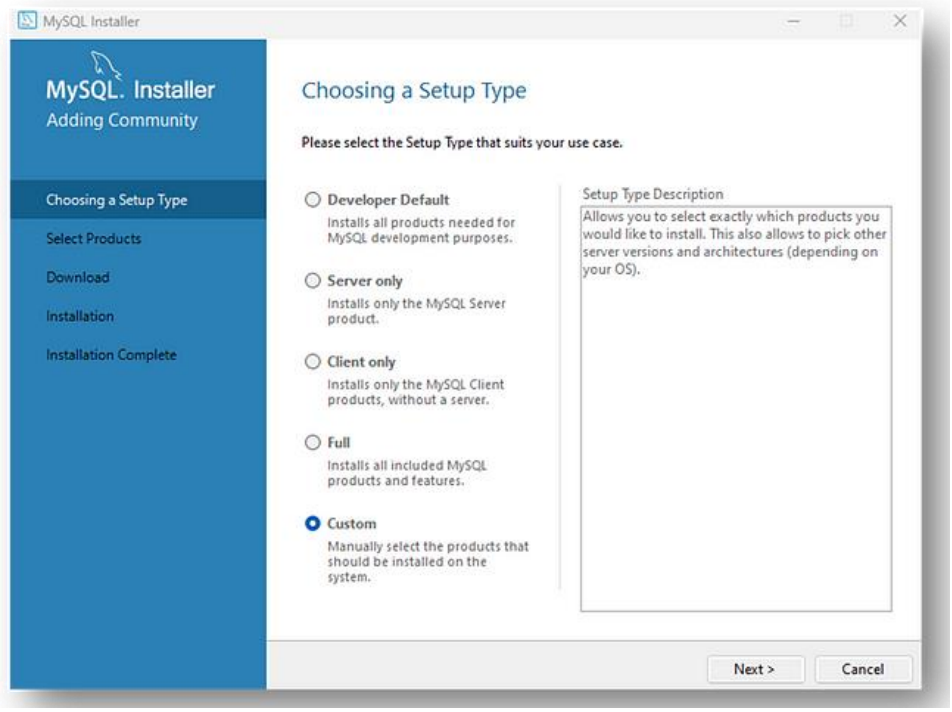


05. After clicking on the download button. You can see another window like this.



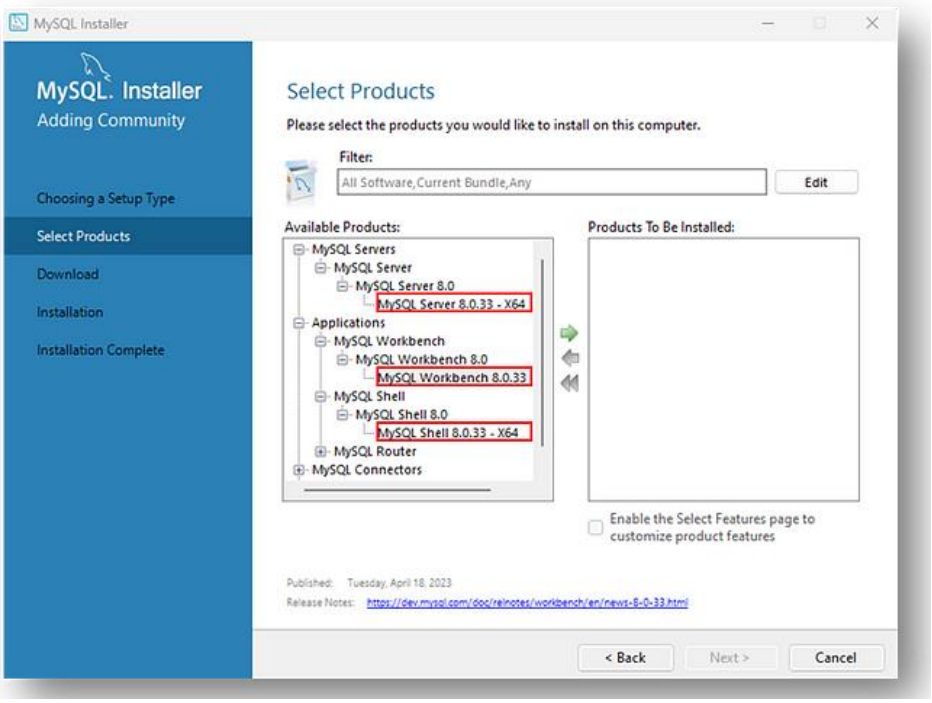
Here click on "**No thanks, just start my download.**"

06. After downloading, double-click on that exe file. You see the window as follows.



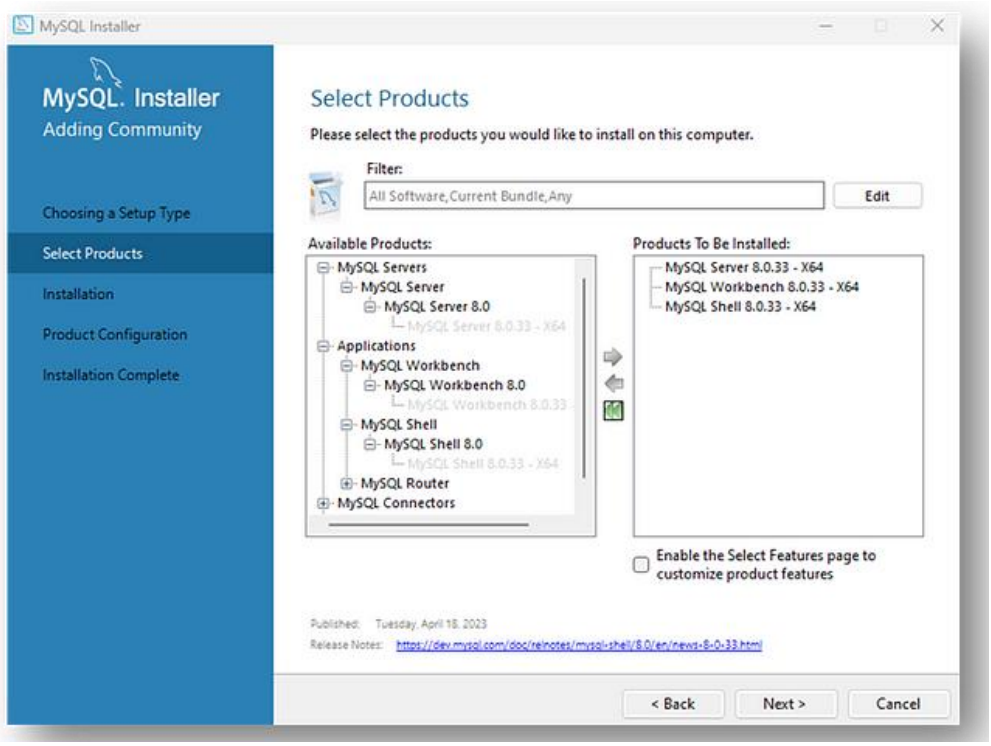
Here select “Custom” and click on the next button. Then you can see a window like this.

07. Now select the product list as follows.



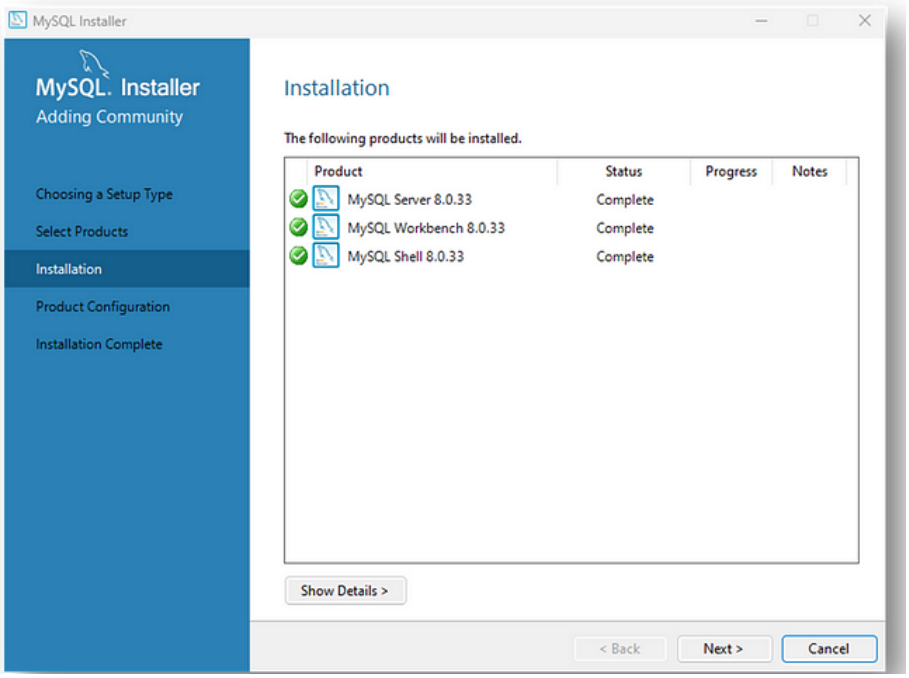
In here ensure to move the height lighted to the next box. Like this

08. The following window shows the moved product.



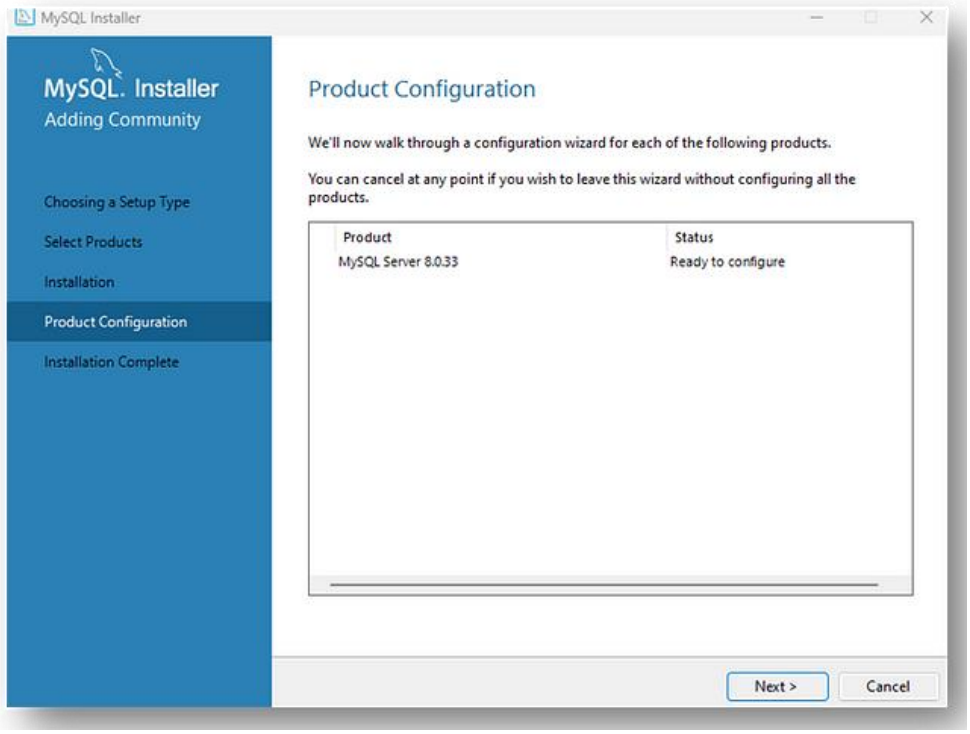
Then click on the next button and then click on the "**Execute**" button

09. In here, the installation process is going up.

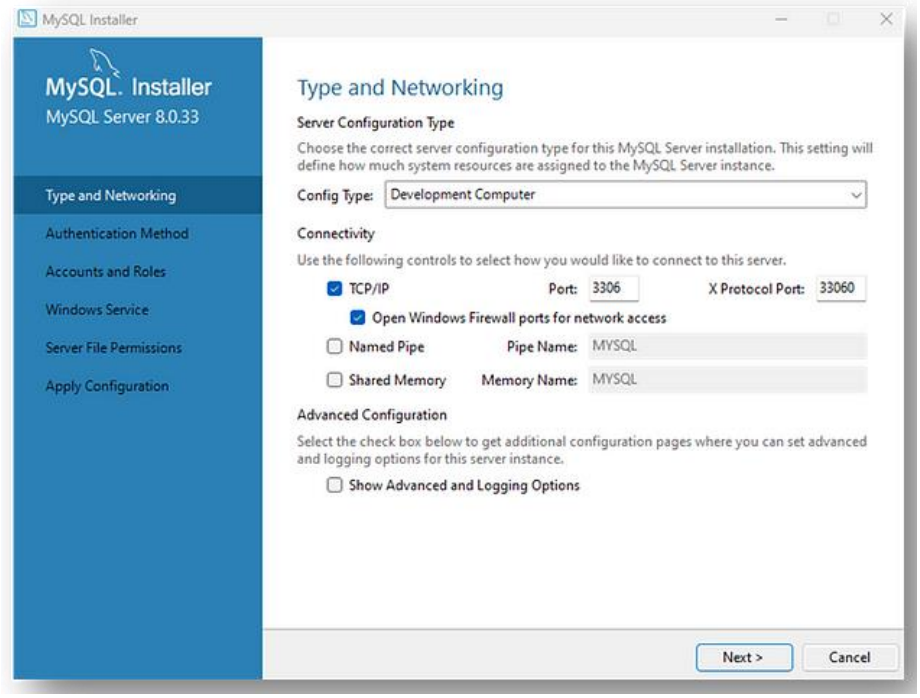


Here you should click on the "Next" button

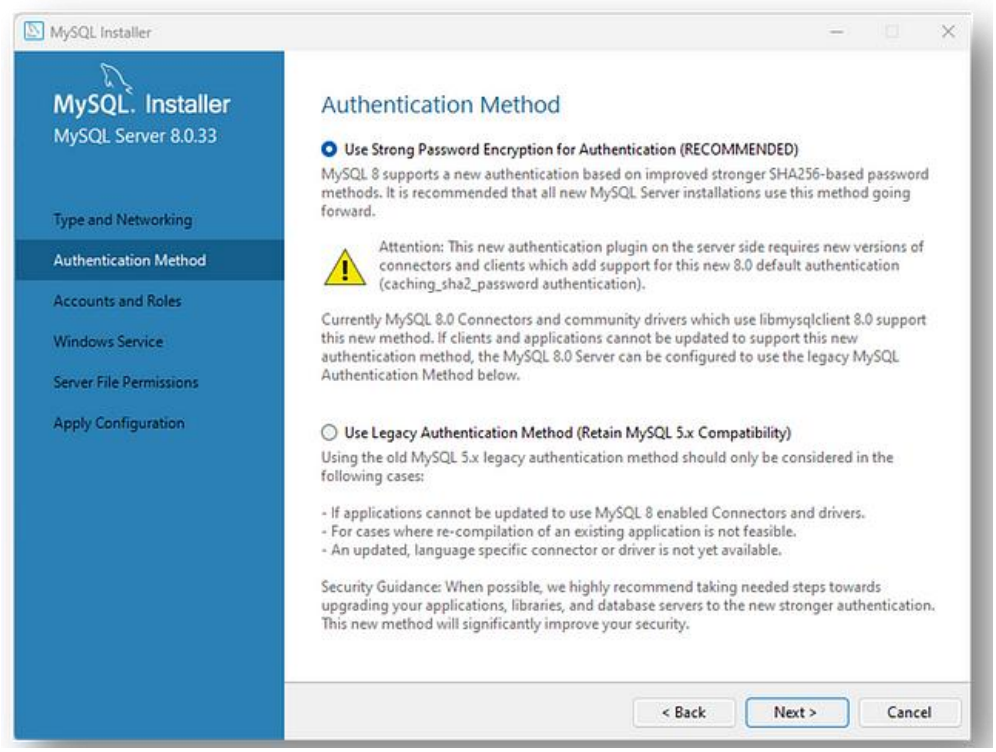
10. In the following window, click on the "Next" button.



11. Next, in here click on the "Next" button.

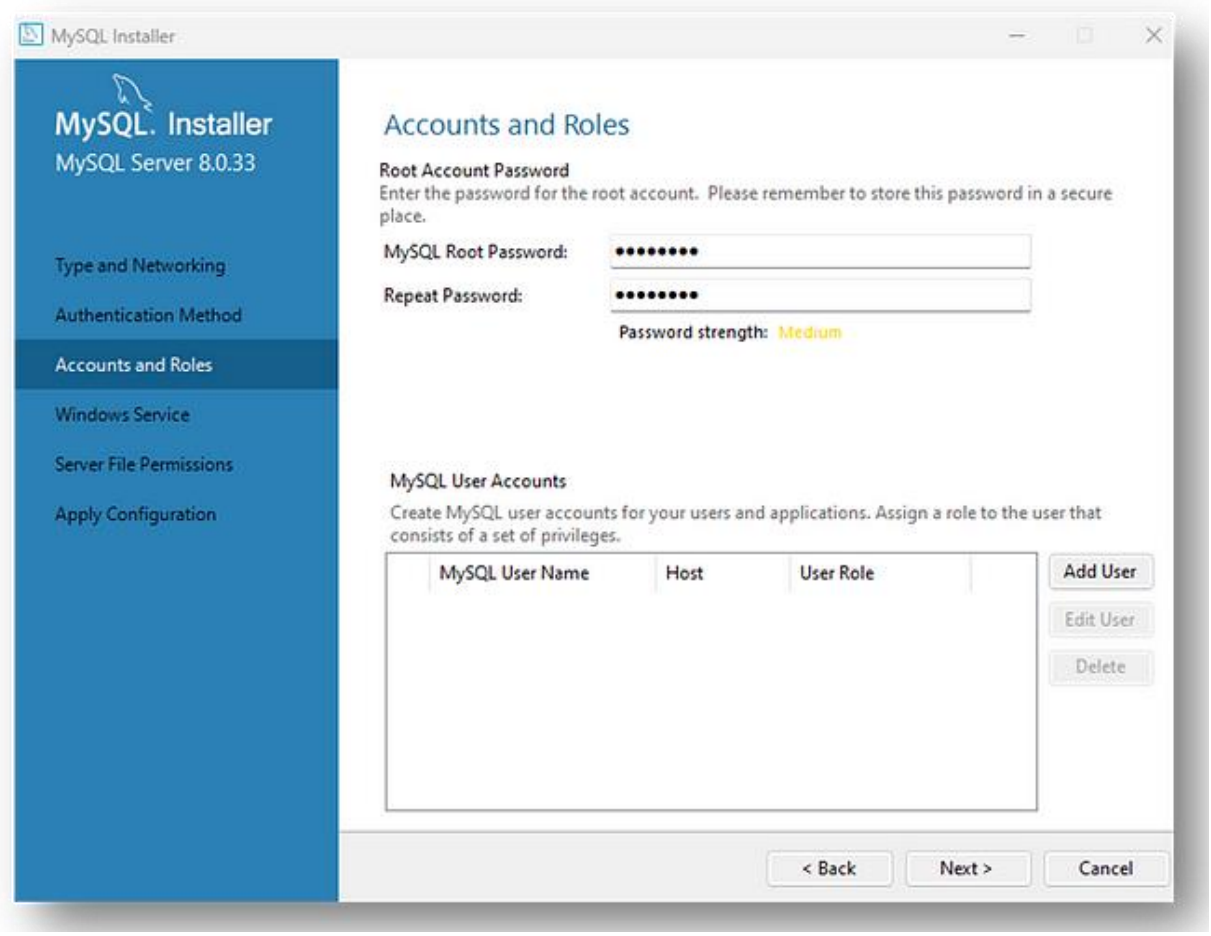


12. Next click on the “Next” button.

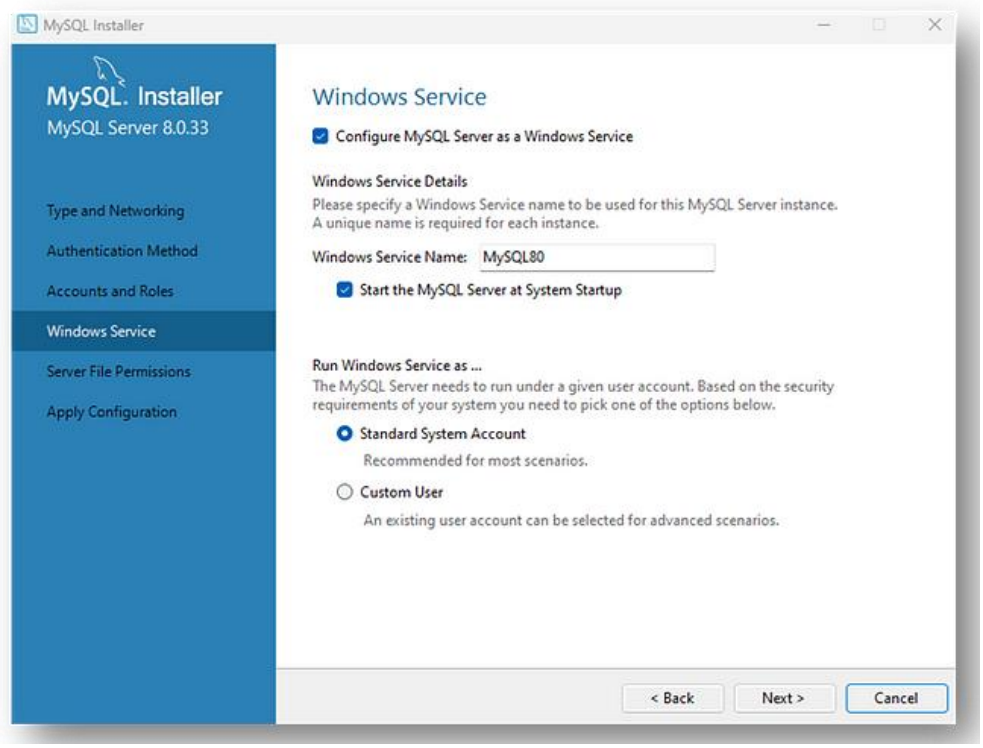


13. In here you should set MySQL Root Password.

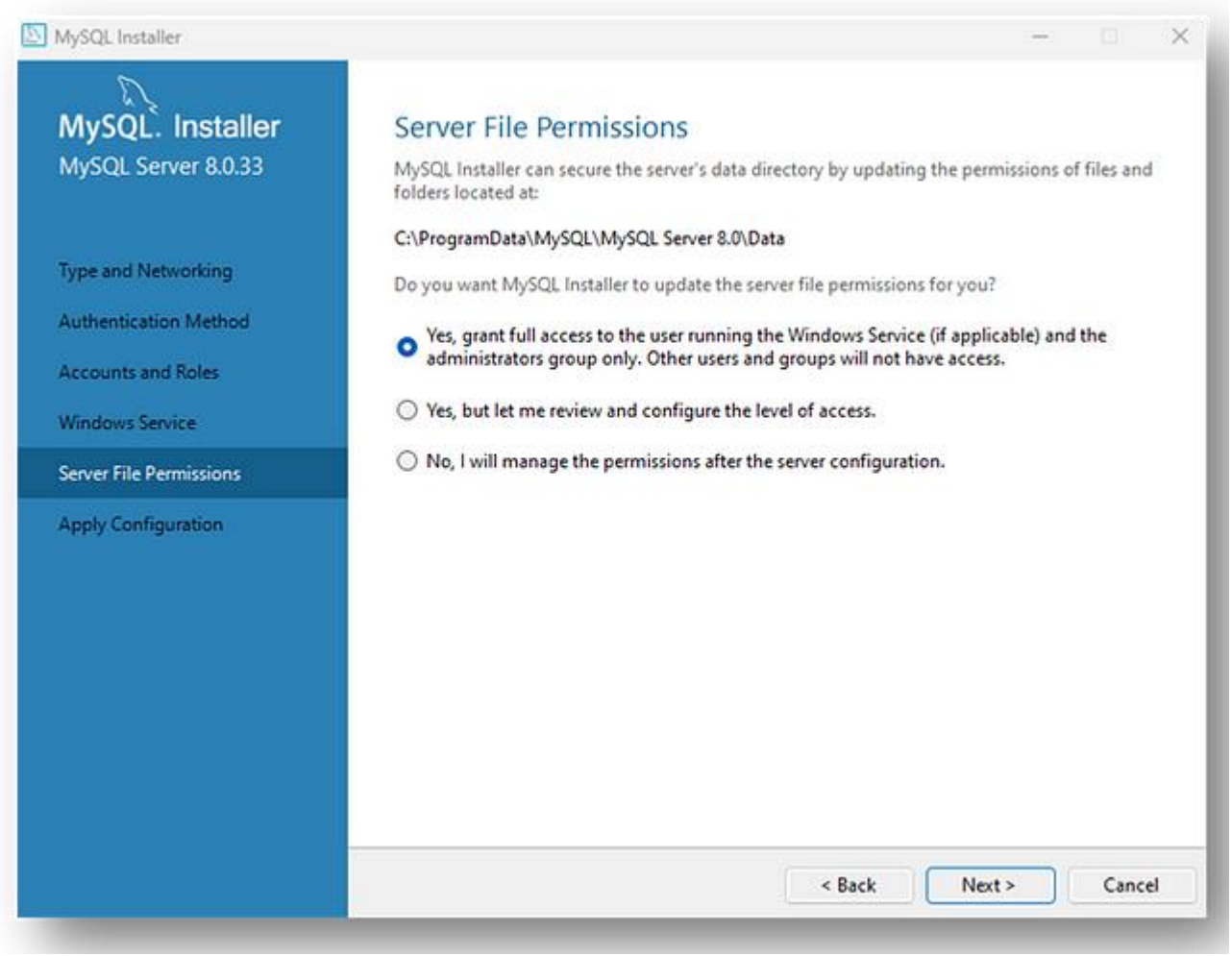
Note: — make sure to remember the password that you entered here. Then click on the "Next" button.



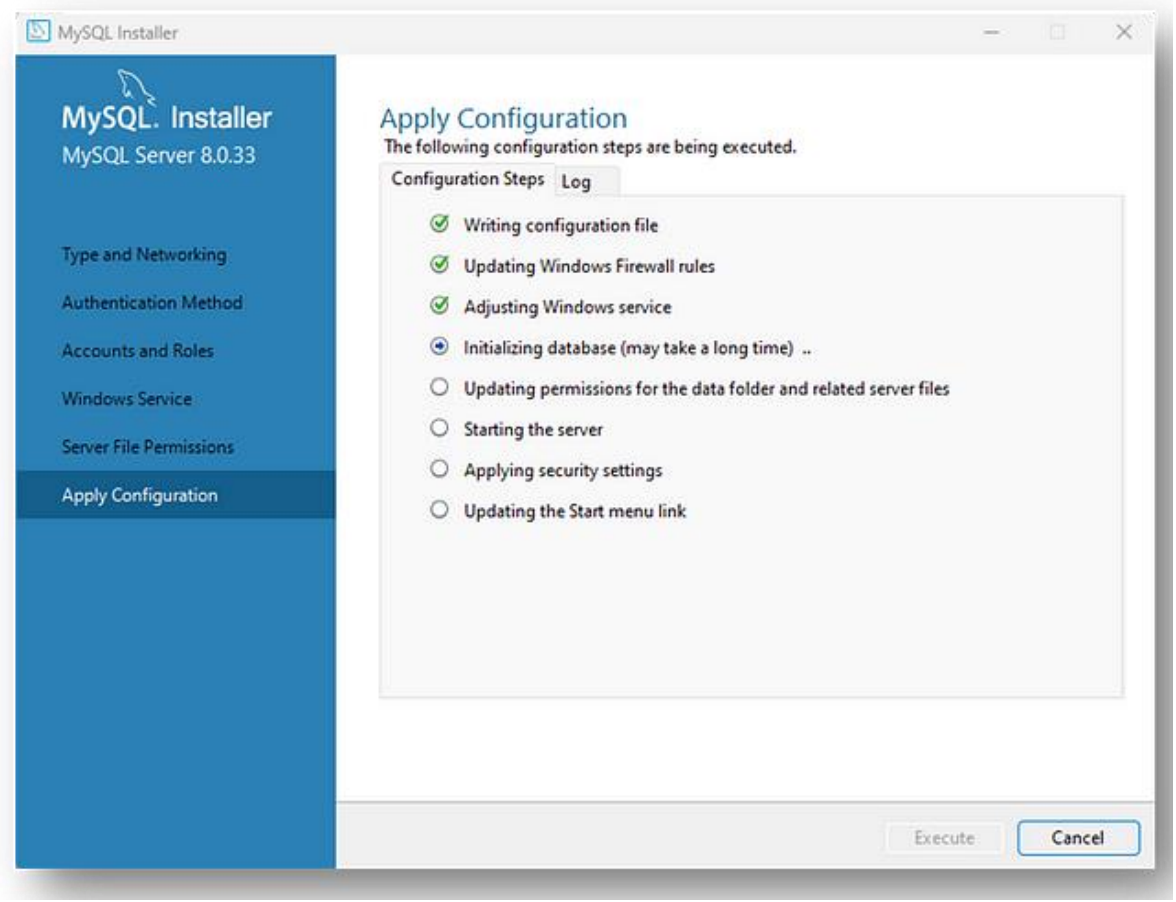
14. In here, you don't want to change anything. Keep this as default.



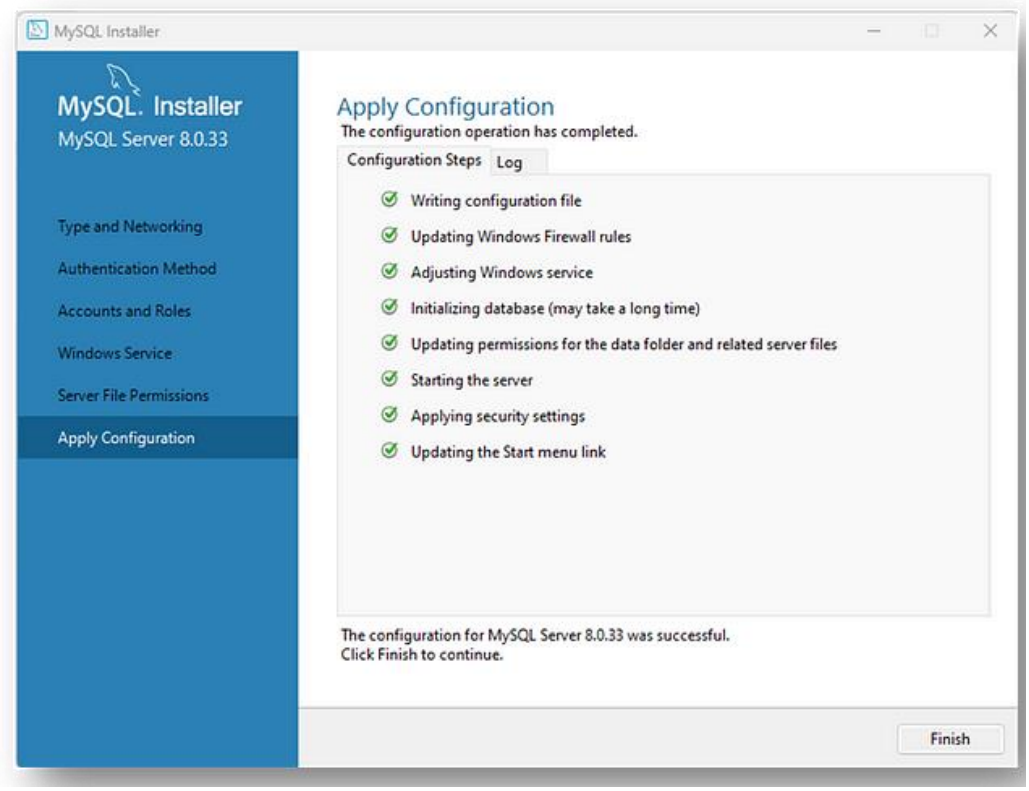
15. Same as the previous one, you don't want to change server file permission. Keep this as default. But you can choose anything as you wish.



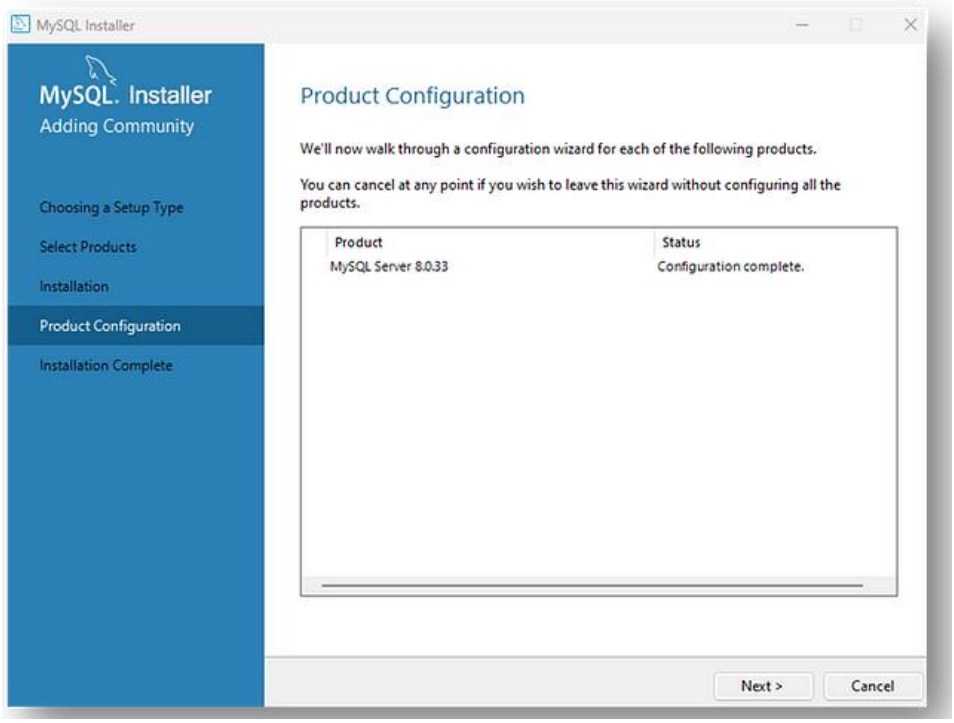
16. After clicking on the "Next" button, you can see a window like this. This is the window while applying the configuration.



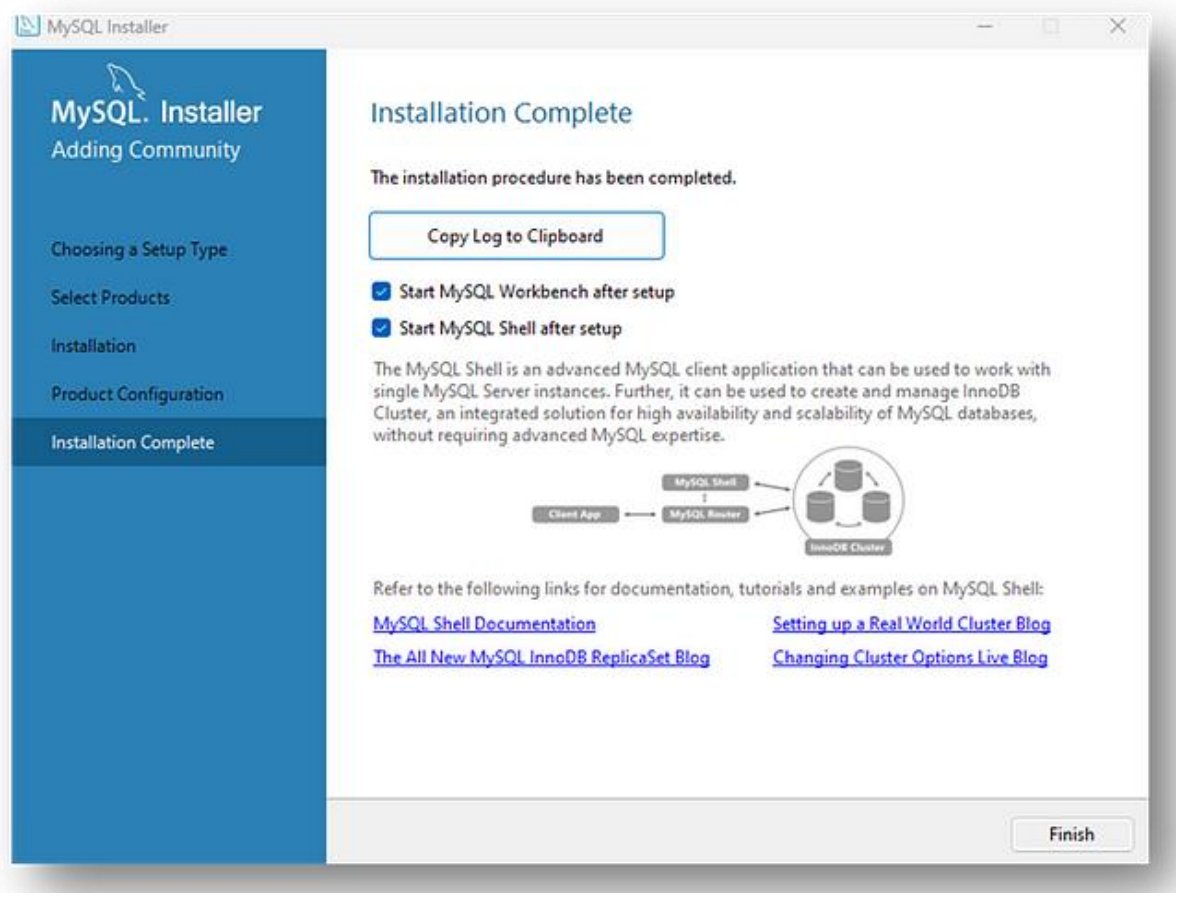
17. After completing the application configuration, you will see the following window. Here, you click on the "Finish" button.



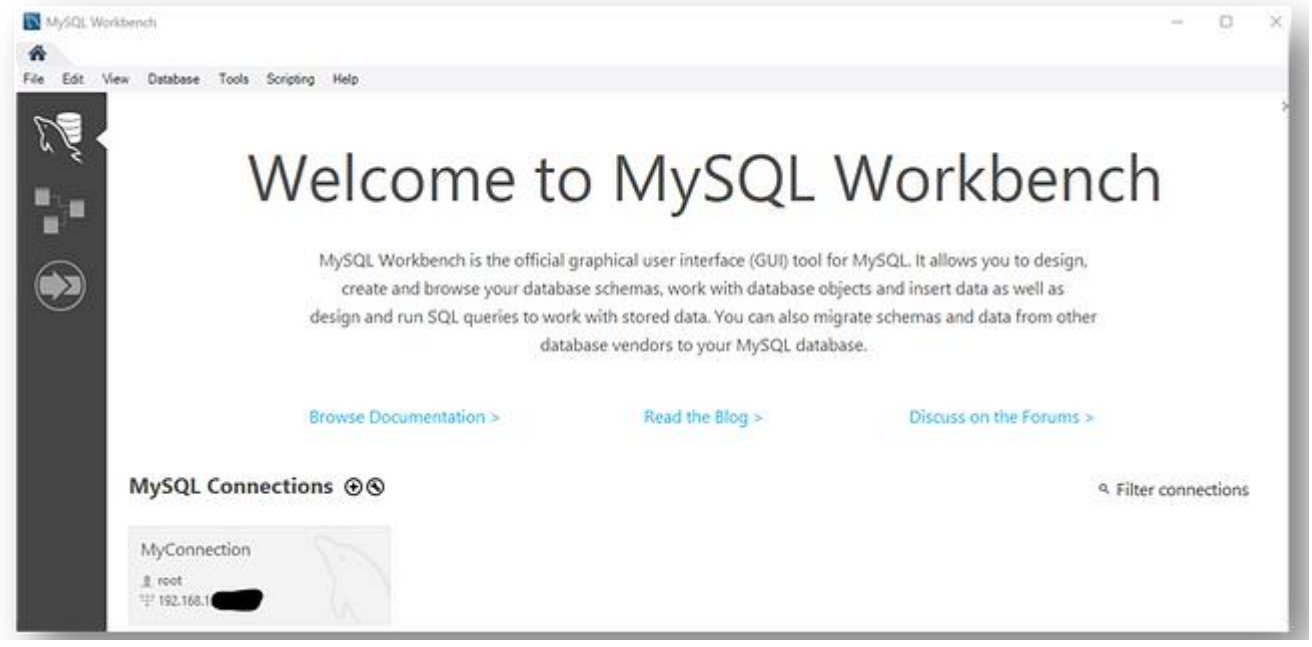
18. After finishing the above window, you can see the following window. Here, click on the "Next" button.



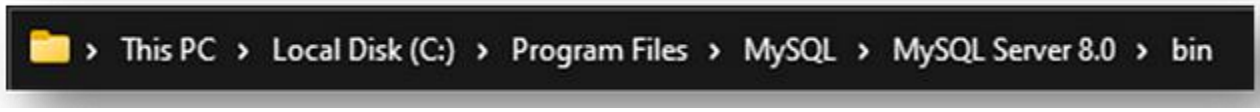
19. Now installation is completed. Click on the "Next" button.



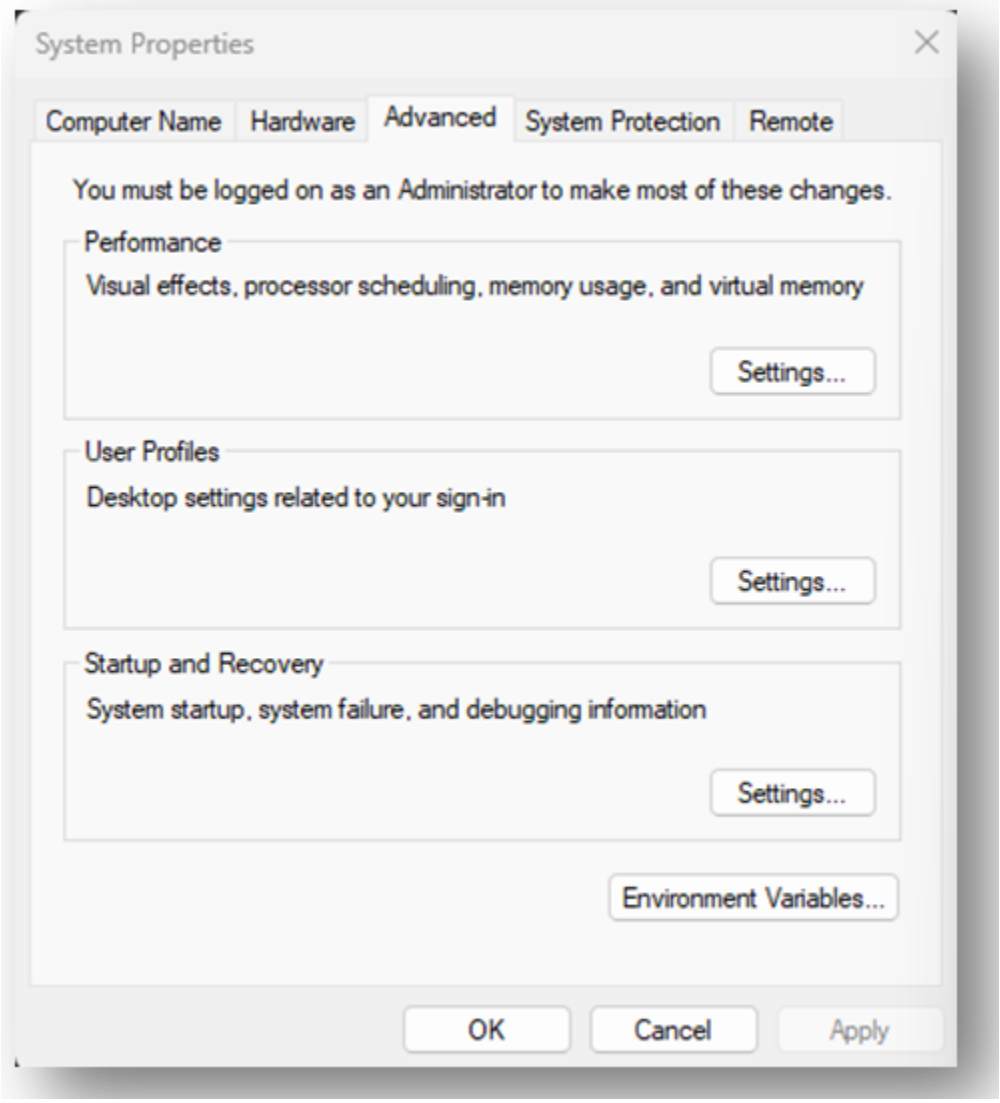
20. Then, the workbench will visible as follows.



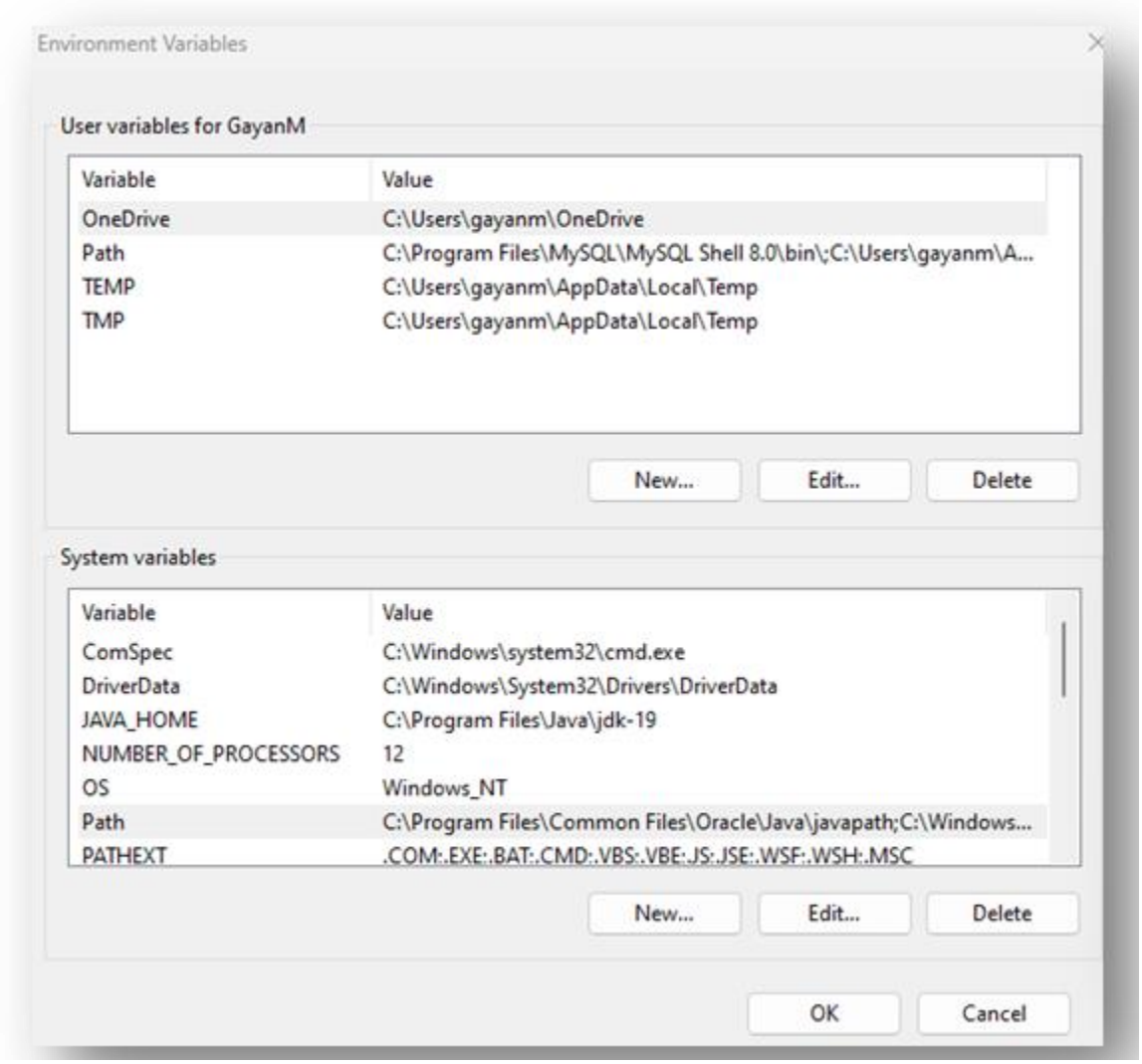
21. Now, I am going to copy the MySQL bin path as follows.



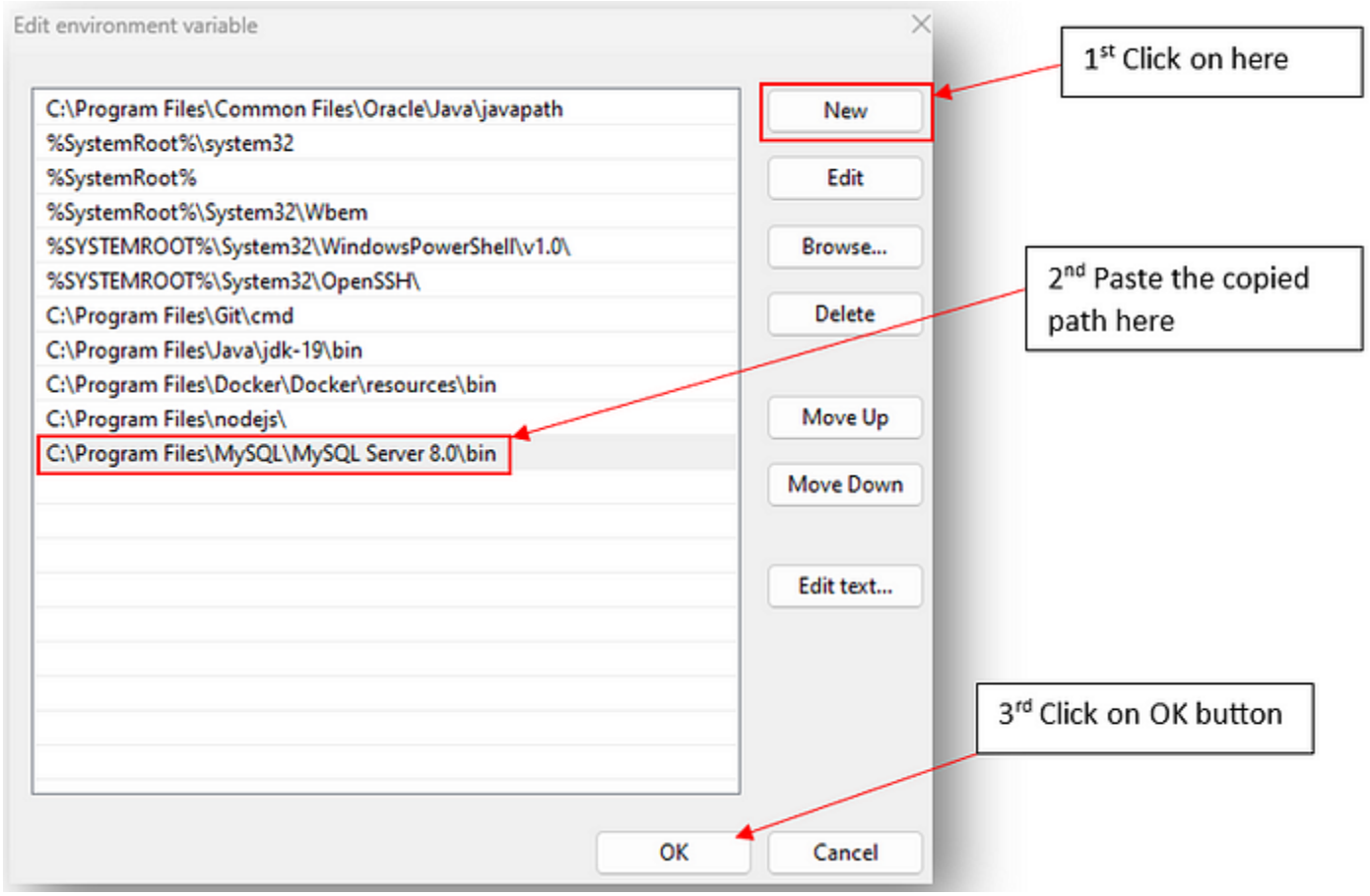
22. Now go to the environment variable and copy the path as follows.



23. Click on "Environment Variables" and then you will see another pop-up window like this.

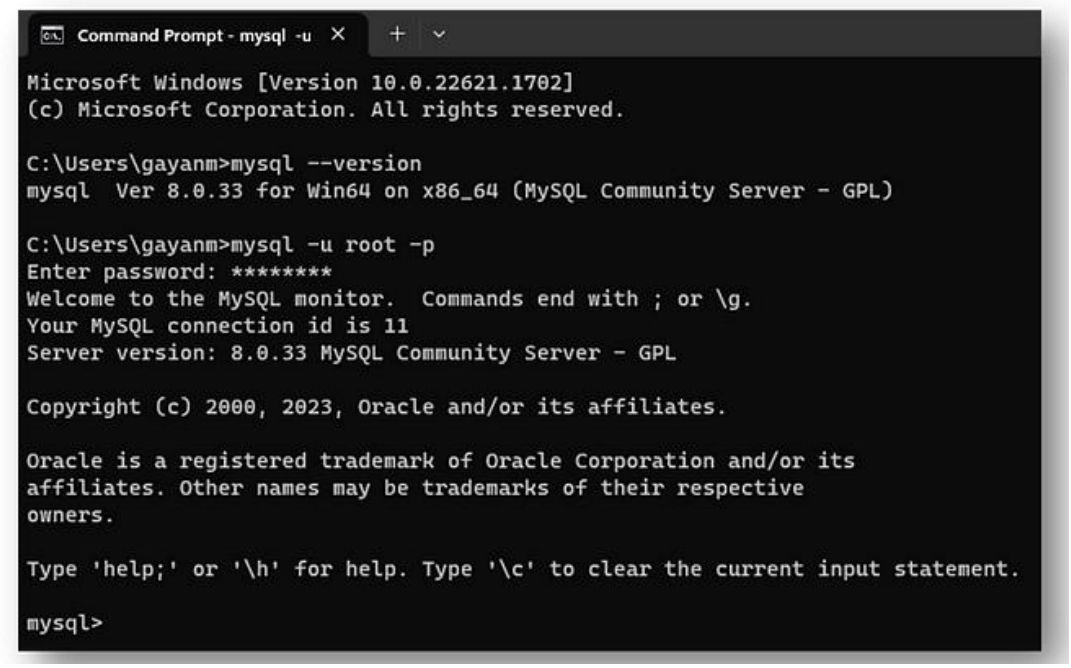


24. Select "path" and click on the "Edit" button under "System Variables" Then you will see another pop-up window as follows.



25. Finally, press the WIN key on your keyboard, then type "cmd" and press the Enter key, then you will see the command line interface. Here type as " mysql –version " and press enter key. Then you will see the mysql version. Next, go into the mysql using this command on CLI " mysql -u root -p " .

```
mysql –version
mysql -u root -p
```



Output

Conclusion:

Timely Submission (5)	Journal Presentation (5)	Performance (5)	Understanding (5)	Oral (5)	Total with Sign & Date

EXPERIMENT NO: 2

AIM: Study of Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym.

HARDWARE & SOFTWARE REQUIREMENT: SQL 2014 Server, Any CPU with Pentium Processor or similar, 8 GB RAM or more, 500 GB Hard Disk or more.

THEORY: MySQL is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc. MySQL is an **ANSI** (American National Standards Institute) standard language, but there are many different versions of the SQL language.

What is MySQL?

MySQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.

SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

Also, they are using different dialects, such as −

- MS SQL Server using T-SQL,
- Oracle using PL/SQL,
- MS Access version of SQL is called JET SQL (native format) etc.

Why SQL?

SQL is widely popular because it offers the following advantages −

- Allows users to access data in the relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views.

A Brief History of SQL

- **1970** − Dr. Edgar F. "Ted" Codd of IBM is known as the father of relational databases. He described a relational model for databases.
- **1974** − Structured Query Language appeared.
- **1978** − IBM worked to develop Codd's ideas and released a product named System/R.
- **1986** − IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software which later came to be known as Oracle.

SQL Process

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

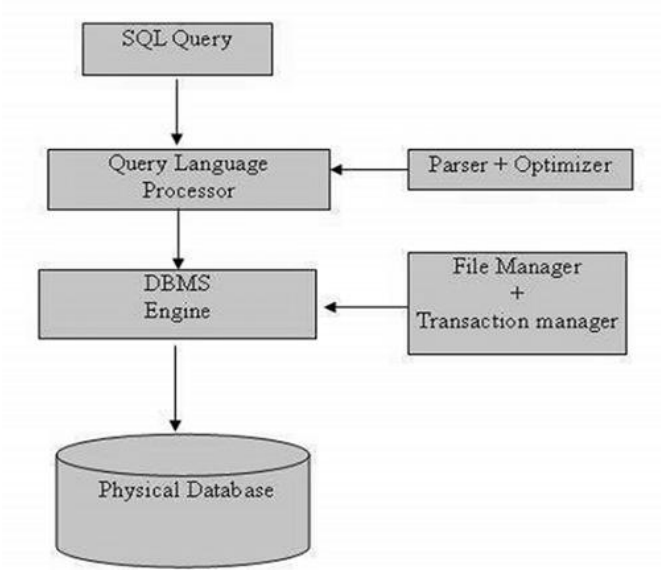
There are various components included in this process.

These components are −

- Query Dispatcher
- Optimization Engines
- Classic Query Engine
- SQL Query Engine, etc.

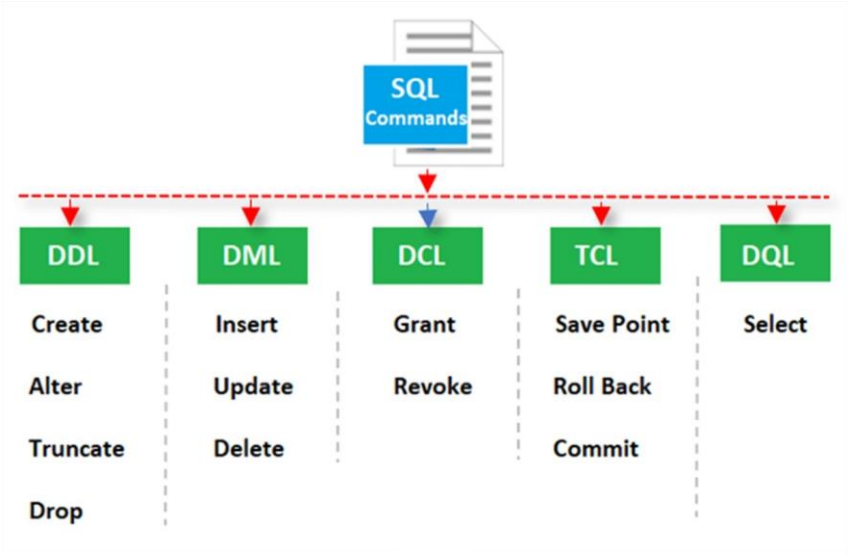
A classic query engine handles all the non-SQL queries, but a SQL query engine won't handle logical files.

Following is a simple diagram showing the SQL Architecture −



SQL Commands

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into the following groups based on their nature –



1. Data Definition Language (DDL):-

DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.

All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

CREATE- It is used to create a new table in the database.

There are two CREATE statements available in SQL:

1. CREATE DATABASE
2. CREATE TABLE

- **CREATE DATABASE**
- A **Database** is defined as a structured set of data. So, in SQL the very first step to store the data in a well structured manner is to create a database.
- The **CREATE DATABASE** statement is used to create a new database in SQL.

Syntax:

CREATE DATABASE database_name;

database_name: name of the database.

Example Query:

This query will create a new database in SQL and name the database as *my_database*.

CREATE DATABASE my_database;

CREATE TABLE

- To store the data we need a table to do that.
- The CREATE TABLE statement is used to create a table in SQL.
- A table comprises of rows and columns.
- So while creating tables we have to provide all the information to SQL about the names of the columns, type of data to be stored in columns, size of the data etc.

Syntax:

```
CREATE TABLE table_name
(
column1 data_type(size),
column2 data_type(size),
column3 data_type(size),
....
);
```

table_name: name of the table.

Column1: name of the first column.

data_type: Type of data we want to store in the particular column. For example, int for integer data.

size: Size of the data we can store in a particular column.

For example if for a column we specify the data_type as int and size as 10 then this column can store an integer number of maximum 10 digits.

ALTER – It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

ALTER TABLE is used to add, delete/drop or modify columns in the existing table. It is also used to add and drop various constraints on the existing table.

1. **ALTER TABLE – ADD:-** ADD is used to add columns into the existing table.

Syntax:

ALTER TABLE table_name

ADD (Columnname_1 datatype, Columnname_2 datatype, … Columnname_n datatype);

2. ALTER TABLE – DROP

- DROP COLUMN is used to drop column in a table. Deleting the unwanted columns from the table.

Syntax:

ALTER TABLE table_name

DROP COLUMN column_name;

3. ALTER TABLE-MODIFY

- It is used to modify the existing columns in a table. Multiple columns can also be modified at once.

Syntax (SQL Server):

ALTER TABLE table_name

ALTER COLUMN column_name column_type;

DROP - It is used to delete both the structure and record stored in the database.

Syntax:

DROP object object_name;

Examples:

DROP TABLE table_name;

table_name: Name of the table to be deleted.

DROP DATABASE database_name;

database_name: Name of the database to be deleted.

TRUNCATE - It is used to delete all the rows from the table and free the space containing the table. It removes all records from table permanently.

Syntax:

TRUNCATE TABLE table_name;

DROP vs TRUNCATE

1) Truncate is normally ultra-fast and its ideal for deleting data from a temporary table.
2) Truncate preserves the structure of the table for future use, unlike drop table where the table is deleted with its full structure.
3) Table or Database deletion using DROP statement cannot be rolled back, so it must be used wisely.

MySQL CREATE INDEX Statement: The CREATE INDEX statement is used to create indexes in tables.

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.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

CREATE INDEX *index_name*
ON *table_name* (*column1*, *column2*, ...);

e.g.

The SQL statement below creates an index named "idx_lastname" on the "LastName" column in the "Persons" table:

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```

If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

```
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
```

DROP INDEX Statement

The DROP INDEX statement is used to delete an index in a table.

ALTER TABLE *table_name*
DROP INDEX *index_name*;

CREATE UNIQUE INDEX Syntax

Creates a unique index on a table. Duplicate values are not allowed:

CREATE UNIQUE INDEX *index_name*
ON *table_name* (*column1*, *column2*, ...);

MySQL CREATE VIEW Statement: In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. We can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the CREATE VIEW statement.

CREATE VIEW Syntax

CREATE VIEW *view_name* AS
SELECT *column1*, *column2*, ...
FROM *table_name*
WHERE *condition*;

MySQL Updating a View

A view can be updated with the CREATE OR REPLACE VIEW statement.

CREATE OR REPLACE VIEW Syntax

CREATE OR REPLACE VIEW *view_name* AS
SELECT *column1*, *column2*, ...
FROM *table_name*
WHERE *condition*;

MySQL Dropping a View

A view is deleted with the DROP VIEW statement.

DROP VIEW Syntax

DROP VIEW *view_name*;

SEQUENCE: Sequences are database objects from which multiple users may generate unique integers. A sequence is a schema-bound, user-defined object which aids to generate a sequence of integers. This is most commonly used to generate values to identity columns in a table. We can create a sequence by using the CREATE SEQUENCE statement as shown below:

CREATE SEQUENCE serial_num START 100;

To get the next number 101 from the sequence, we use the nextval() method as shown below:
SELECT nextval('serial_num');

Also Let us understand it with the help of the following example. First, we need to create a new table and make sure that there is one column with the AUTO_INCREMENT attribute and that too, as PRIMARY KEY.

```
CREATE TABLE Insects (
  Id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (id),
 Name VARCHAR(30) NOT NULL,
 Type VARCHAR(30) NOT NULL,
 Origin VARCHAR(30) NOT NULL
  );
```

Now insert a few rows into this table where no need to provide the id for each row because it is auto-incremented by MySQL.

```
INSERT INTO Insects (Name, Type, Origin)
VALUES ('Cockroach', 'Crawling', 'Kitchen'),
('Mosquito', 'Flying', 'Driveway'),
('Spider' ,'Crawling', 'Court yard'),
('Grasshopper', 'Flying', 'Front yard');
```

Now execute the **SELECT statement** to verify the records:

```
SELECT * FROM Insects;
```

When we execute the INSERT query, we do not provide values for the Id column, but MySQL automatically generates a sequence for it.

SYNONYM: A Synonym provides another name for database object, referred to as original object, that may exist on a local or another server. A synonym belongs to schema; name of synonym should be unique. A synonym cannot be original object for an additional synonym and synonym cannot refer to user-defined function.

Syntax:

CREATE SYNONYM synonymname

FOR servername.databasename.schemaname.objectname;

GO

Example: Now, let us create synonym for Geektab table of GFG database, Geeeksh schema on server named Server1.

CREATE SYNONYM Geektable FOR erver1.GFG.Geeeksh.Geektab;

GO

Find the output in Server2 by using synonym.

SELECT ID, Name FROM Geektable;

Programs and Outputs:

Conclusion:_____

_____

_____

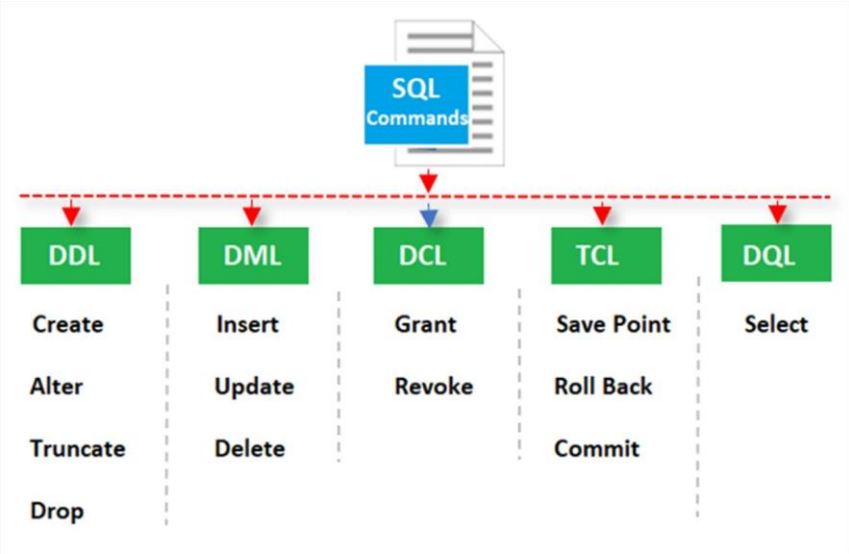
Timely Submission (5)	Journal Presentation (5)	Performance (5)	Understanding (5)	Oral (5)	Total with Sign & Date

EXPERIMENT NO: 3

AIM: Design and develop at least 5 SQL queries for suitable database application using SQL DML statements: Insert and Select with operators and functions.

HARDWARE & SOFTWARE REQUIREMENT: SQL 2014 Server, Any CPU with Pentium Processor or similar, 4 GB RAM or more, 500 GB Hard Disk or more.

THEORY:



DML (Data Manipulation Language):- Data Manipulation Language (DML) is a language that enables the user to access or manipulate data as organized by the appropriate data model.

For example- SELECT, UPDATE, INSERT, DELETE.

DML modifies the database instance by inserting, updating and deleting its data. DML is responsible for all froms data modification in a database. SQL contains the following set of commands in its DML section –

1. **SELECT:** The SELECT statement is used to select data from a database.

   Syntax: SELECT column_name FROM table_name;

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

   Syntax : SELECT DISTINCT column1, column2, ... FROM table_name;
2. **FROM:** FROM clause is used to specify the table to select or delete data from.

3. **WHERE:** The WHERE clause is used to filter records.

   Syntax: SELECT column_name FROM table_name WHERE condition;

4. **INSERT INTO:** The INSERT INTO statement is used to insert new records in a table.

   Syntax1: INSERT INTO table_name (column1, column2, column3)

   VALUES (value1, value2, value3);

   Syntax2: INSERT INTO table_name VALUES (value1, value2, value3);

5. **UPDATE:** The UPDATE statement is used to modify the existing records in a table.

   Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

   UPDATE Syntax

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

6. **DELETE:** The DELETE statement is used to delete existing records in a table.

   DELETE Syntax

   DELETE FROM *table_name* WHERE *condition*;

   Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

These basic constructs allow database programmers and users to enter data and information into the database and retrieve efficiently using a number of filter options.

FUNCTION:

A function is a special type of predefined command set that performs some operation and returns a single value. Functions operate on zero, one, two or more values that are provided to them. The values that are provided to functions are called parameters or arguments. The MySQL functions have been categorized into various categories, such as String functions, Mathematical functions, Date and Time Functions and so on.

MYSQL Math functions:

1. ABS() Function: The abs() is a Math function of MySQL. This function is used to get the absolute value of the given number.
   Syntax: select abs(num);

   e.g. SELECT ABS(-243.5);
   Output: 243.5
2. **CEIL() Function:** The CEIL() function returns the smallest value which is greater than or equal to the specified number.
   Syntax: select ceil(num);
   e.g. SELECT CEIL(25.75);
   Output: 26
3. **CEILING() Function:** The CEIL() function returns the smallest value which is greater than or equal to the specified number.
   Syntax: select ceil(num);
   e.g. SELECT CEIL(25.75);
   Output: 26
4. **FLOOR() Function:** This function is used to find the greatest integer which is equal to or less than the given number.
   Syntax: **select** floor(number);
   e.g.
   SELECT FLOOR(25.75);
   Output
   25
5. **GREATEST() Function:** The greatest() is a Math function of MySQL. This function is used to get the largest number from the list.
   Syntax: **select** greatest(exp1, exp2, exp3....);
   e.g.
   SELECT GREATEST(3, 12, 34, 8, 25);
   Output:
   34

6. **LEAST() Function:** The least() is a Math function of MySQL. This function is used to get the smallest number from the list.
   Syntax: select least(exp1, exp2, exp3.....);

   e.g.
   SELECT LEAST(3, 12, 34, 8, 25);
   Output:
   3
7. **RAND() Function:** The rand() is a Math function of MySQL. This function is used to generate the random number.
   Syntax: select rand();
   e.g.
   SELECT RAND();
   Output:
   0.7519901616051702
8. **ROUND() Function:** The round() is a Math function of MySQL. This function is used to round off the given number. Rounds a number to a specified number of decimal places.
   Syntax: select round(number);
   e.g.
   SELECT ROUND(135.375, 2);
   Output:
   135.38.
9. POWER() Function: The power() is a Math function of MySQL. This function is used to get the power of the given values.
   Syntax: select power(m,n);
   Parameter:
   m: It is base value in the calculation
   n: It is exponent value in the calculation
   Output: This function returns m raised to the nth power.
10. RANDOM:- RANDOM function can be used to return a random number between 0 and 1.

Syntax: RANDOM( )

The random function will return a value between 0 (inclusive) and 1 (exclusive), so value >= 0 and value < 1.

STRING FUNCTIONS: The string functions of MySQL can manipulate the text string in many ways. SQL String functions are the predefined functions that allow the database users for string manipulation. These functions only accept, process, and give results of the string data type.

1) **LENGTH:-**

   Length function returns the length of the specified string, expressed as the number of characters. Returns the length of a string in bytes

   Syntax: select length(E_name) FROM employee;

2) **UPPER/LOWER:-** UPPER/LOWER function converts all characters in the specifies string to uppercase/lowercase.

   Syntax & example:

   upper(string)

   Select UPPER('mahesh');

   lower(string)

   Select lower('MAHESH')

3) **REPLACE: -** Replaces all occurrences of a substring within a string, with a new substring.

   Syntax:- replace(string, from_substring, to_substring)

   ***Replace function is case sensitive.**

   REPLACE(string, substring, new_string)

   Example:-

   SELECT Customer_name,country, Replace (country,'UnitedStates','US') AS country new FROM customer;

4) **TRIM,LTRIM & RTRIM :-**

   TRIM function removes all specified characters either from the beginning or the end of a string. Also removes leading and trailing spaces from a string.

   RTRIM function removes all specified characters from the right-hand side of a string. Removes trailing spaces from a string.

LTRIM function removes all specified characters from the left-hand side of a string.

Removes leading spaces from a string.

Syntax:-

trim([leading | trailing | both] [trim_character] from string)

rtrim(string, trim_character)

ltrim(string, trim_character)

SELECT TRIM(' SQL Tutorial ') AS TrimmedString;

SELECT LTRIM(" SQL Tutorial") AS LeftTrimmedString;

SELECT RTRIM("SQL Tutorial ") AS RightTrimmedString;

5) **CONCAT:-** || operator allows you to concatenate 2 or more strings together.

Syntax: CONCAT(expression1, expression2, expression3,...)

Example:-

SELECT CONCAT("GANESH ", "GOKUL ", "WANI ");

SELECT CONCAT("GANESH ", "GOKUL ", "WANI ") AS FULLNAME;

DATE FUNCTIONS:

1) **ADDDATE() Function:** The ADDDATE() function adds a time/date interval to a date and then returns the date.

Syntax: ADDDATE(*date*, INTERVAL *value addunit*)

OR:

ADDDATE(*date*, *days*)

2) **ADDDATE() Function:** The ADDDATE() function adds a time/date interval to a date and then returns the date.

Syntax: ADDDATE(date, INTERVAL value addunit)

OR:

ADDDATE(date, days)

3) **CURDATE() Function:** The CURDATE() function returns the current date.

The date is returned as "YYYY-MM-DD" (string) or as YYYYMMDD (numeric).

This function equals the CURRENT_DATE() function.

Example 1: Return the current date:

SELECT CURDATE();

4) **CURRENT_TIME() Function:**

The CURRENT_TIME() function returns the current time.

The time is returned as "HH-MM-SS" (string) or as HHMMSS.uuuuuu (numeric).

This function equals the CURTIME() function.

Syntax

CURRENT_TIME()

MySQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

MIN() Syntax

SELECT MIN(*column_name*)

FROM *table_name*

WHERE *condition*;

Select max(SALARY) from EMPLOYEE where city=’PUNE’;

e.g. 1

SELECT MIN(Price) FROM Products;

e.g. 2

SELECT MIN(Price) AS SmallestPrice

FROM Products;

Select min(SALARY) AS smallestsalary from EMPLOYEE;

MAX() Syntax

SELECT MAX(*column_name*)

FROM *table_name*

WHERE *condition*;

e.g. 1

SELECT MAX(Price) FROM Products;

e.g. 2

SELECT MAX(Price) AS LargestPrice

FROM Products;

MySQL COUNT(), AVG() and SUM() Functions

COUNT() Function

The COUNT() function returns the number of rows that matches a specified criterion.

Note: NULL values are not counted.

COUNT() Syntax

SELECT COUNT(*column_name*)

FROM *table_name*

WHERE *condition*;

Select count(DEPT) from EMPLOYEE;

Select count(DEPT) from EMPLOYEE where DEPT=’DA’;

OR

SELECT COUNT(*column_name*)

FROM *table_name*;

e.g. The following SQL statement finds the number of products:

SELECT COUNT(ProductID)

FROM Products;

Note: NULL values are not counted.

AVG() Function

The AVG() function returns the average value of a numeric column.

Note: NULL values are ignored.

AVG() Syntax

SELECT AVG(*column_name*)

FROM *table_name*

WHERE *condition*;

OR

SELECT AVG(*column_name*)

FROM *table_name*;

e.g.

The following SQL statement finds the average price of all products:

SELECT AVG(Price)

FROM Products;

NULL values are ignored.

SUM() Function

The SUM() function returns the total sum of a numeric column.

Note: NULL values are ignored.

SUM() Syntax

SELECT SUM(*column_name*)

FROM *table_name*

WHERE *condition*;

OR

SELECT SUM(column_name)

FROM table_name;

e.g

SELECT SUM(Quantity)

FROM OrderDetails;

NULL values are ignored.

OPERATORS:

The following operators can be used in the WHERE clause:

= Equal

> Greater than

< Less than

>= Greater than or equal

<= Less than or equal

<> Not equal. Note: In some versions of SQL this operator may be written as !=

BETWEEN Between a certain range

LIKE Search for a pattern

IN To specify multiple possible values for a column

The SQL AND, OR, NOT, IN and Not IN Operators

- The WHERE clause can be combined with AND, OR, and NOT operators.
- The AND and OR operators are used to filter records based on more than one condition.
- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.
- The NOT operator displays a record if the condition(s) is NOT TRUE.

Programs & Outputs:-

Conclusion:

_____

_____

_____

Timely Submission (5)	Journal Presentation (5)	Performance (5)	Understanding (5)	Oral (5)	Total with Sign & Date

EXPERIMENT NO: 4

AIM: Design and develop at least 5 SQL queries for suitable database application using SQL DML statements: Update and Delete with operators and functions.

HARDWARE & SOFTWARE REQUIREMENT: SQL 2014 Server, Any CPU with Pentium Processor or similar, 4 GB RAM or more, 500 GB Hard Disk or more.

THEORY:

DML commands are used to modify the database. It is responsible for all form of changes in the database.

The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

The MySQL UPDATE Statement: - The UPDATE statement is used to modify the existing records i.e to update or modify the value of a column in a table.

Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

UPDATE Syntax

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

e.g.

UPDATE Customers SET ContactName = 'Mahesh', City = 'Pune' WHERE CustomerID = 7;

UPDATE Multiple Records: - It is the WHERE clause that determines how many records will be updated. The following SQL statement will update the Postal Code to 410507 for all records where City is "Talegaon".

e.g.

```
UPDATE Customers
SET PostalCode = 410507
WHERE City = 'Talegaon';
```

Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!

e.g.

```
UPDATE Customers
SET PostalCode = 410507;
```

The MySQL DELETE Statement: - The DELETE statement is used to delete existing records in a table. It is used to remove one or more row from a table.

DELETE Syntax

DELETE FROM *table_name* WHERE *condition*;

Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

e.g.

```
DELETE FROM Customers WHERE CustomerName='SUNIL';
```

Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

Syntax

DELETE FROM *table_name*;

e.g

The following SQL statement deletes all rows in the "Customers" table, without deleting the table

DELETE FROM Customers;

The SQL AND, OR, NOT, IN and Not IN Operators:-

- The WHERE clause can be combined with AND, OR, and NOT operators.
- The AND and OR operators are used to filter records based on more than one condition.
- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.
- The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

e.g.

```
SELECT * FROM Customers
WHERE Country='INDIA' AND City='PUNE';
```

OR Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

e.g.

```
SELECT * FROM Customers
WHERE City='Mumbai' OR City='Pune';
```

NOT Syntax

SELECT column1, column2, ...

FROM table_name

WHERE NOT condition;

e.g.

SELECT * FROM Customers

WHERE NOT Country='PAKISTAN';

Combining AND, OR and NOT

We can also combine the AND, OR and NOT operators.

e.g. 1

SELECT * FROM Customers

WHERE Country = 'INDIA' AND (City = 'PUNE' OR City = 'NASHIK');

e.g. 2

SELECT * FROM Customers

WHERE NOT Country = 'PAKISTAN' AND NOT Country = 'CHINA';

MySQL Wildcard Characters

- A wildcard character is used to substitute one or more characters in a string.
- Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

Wildcard Characters in MySQL

Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit

The MySQL LIKE Operator

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator:
- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character
- The percent sign and the underscore can also be used in combinations!
- We can also combine any number of conditions using AND or OR operators.
- Here are some examples showing different LIKE operators with '%' and '_' wildcards

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"
WHERE CustomerName LIKE 'a_%_%'	Finds any values that starts with "a" and are at least 3 characters in length

e.g. **Using the % Wildcard**

The following SQL statement selects all customers with a CustomerName starting with "a":

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

Select * from employee where E_NAME not like '_a_ _s_'

The following SQL statement selects all customers with a CustomerName ending with "a":

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%a';
```

The following SQL statement selects all customers with a CustomerName that have "or" in any position:

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%or%';
```

The following SQL statement selects all customers with a CustomerName that have "r" in the second position:

```
SELECT * FROM Customers
WHERE CustomerName LIKE '_r%';
```

The following SQL statement selects all customers with a CustomerName that starts with "a" and are at least 3 characters in length:

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a__%';
```

The following SQL statement selects all customers with a ContactName that starts with "a" and ends with "o":

```
SELECT * FROM Customers
WHERE ContactName LIKE 'a%o';
```

The following SQL statement selects all customers with a City starting with "ber":

```
SELECT * FROM Customers
WHERE City LIKE 'ber%';
```

The following SQL statement selects all customers with a City containing the pattern "es":

```
SELECT * FROM Customers
WHERE City LIKE '%es%';
```

The following SQL statement selects all customers with a CustomerName that does NOT start with "a":

```
SELECT * FROM Customers
WHERE CustomerName NOT LIKE 'a%';
```

Using the _ Wildcard

The following SQL statement selects all customers with a City starting with any character, followed by "_ondon":

```
SELECT * FROM Customers
WHERE City LIKE '_ondon';
```

The following SQL statement selects all customers with a City starting with "L", followed by any character, followed by "n", followed by any character, followed by "on":

```
SELECT * FROM Customers
WHERE City LIKE 'L_n_on';
```

The SQL IN Operator:- The IN operator allows you to specify multiple values in a WHERE clause. The IN operator is a shorthand for multiple OR conditions.

IN Syntax

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

OR

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

NOT IN Syntax

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

OR

```
SELECT column_name(s)
FROM table_name
WHERE column_name NOT IN (SELECT STATEMENT);
```

BETWEEN Syntax

```
SELECT column_name(s)
```

FROM table_name

WHERE column_name **BETWEEN** value1 **AND** value2;

NOT BETWEEN:- To display the products outside the range of the previous example, use **NOT BETWEEN**

SELECT column_name(s)

FROM table_name

*WHERE column_name **NOT BETWEEN** value1 **AND** value2;*

Program & Output:

Conclusion:

Timely Submission (5)	Journal Presentation (5)	Performance (5)	Understanding (5)	Oral (5)	Total with Sign & Date

EXPERIMENT NO: 5

AIM: Design SQL queries for suitable database application using SQL DML statements: all types of Joins and Sub-Query.

HARDWARE & SOFTWARE REQUIREMENT: MySQL Installer 8.0.33 Server, Any CPU with Pentium Processor or similar, 8 GB RAM or more, 500 GB Hard Disk or more.

THEORY:

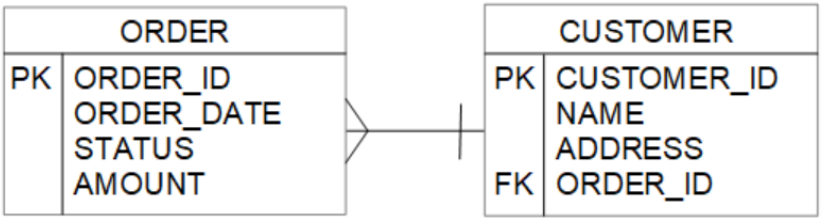
An SQL Join statement is used to combine data or rows from two or more tables based on a common field between them.

A subquery is a query that is nested inside a SELECT, INSERT, UPDATE, or DELETE statement, or inside another subquery.

Joins and subqueries are both used to combine data from different tables into a single result.

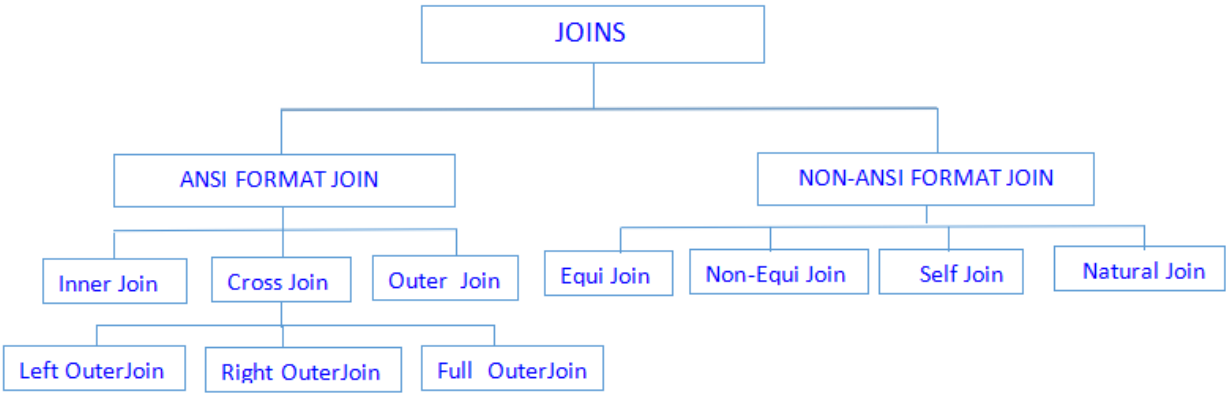
Primary Keys and Foreign Keys

A primary key is a column in a table that's used to uniquely identify a row in that table.
A foreign key is used to form a relationship between two tables. For example, let's say you have a one-to-many relationship between customers in a CUSTOMER table and orders in an ORDER table. To relate the two tables, you would define an ORDER_ID column in the CUSTOMER table that corresponds with the ORDER_ID column in the ORDER table.



The values defined in a foreign key column MUST match the values defined in the primary key column. This is referred to as referential integrity and is enforced by the RDBMS. If the primary key for a table is a composite key, the foreign key must also be a composite key.

Classification of Joins

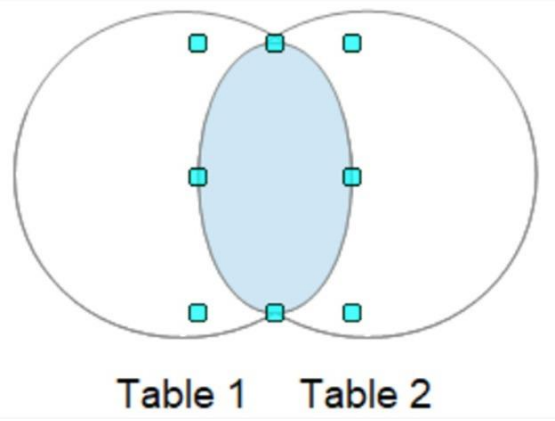


When we retrieve the data from multiple tables with on keyword condition then this is called as ANSI format joins.

When we retrieve data from multiple tables based on where keyword condition then it is called as NON-ANSI format Joins.

Inner Joins

Inner joins are used to combine related information from multiple tables. An inner join retrieves matching rows between two tables. Matches are normally made based on the primary key/foreign key relationships that exist. If there's a match, a row is included in the results. Otherwise, it's not.



Inner Join is used for retrieving data from multiple Tables. When we use inner Join we should use a common column name and data type is also same in the Table.

Syntax:

SELECT <columns>

FROM <table1>

JOIN <table2>

ON <table1>.<columnA> = <table2>.<columnB>

Using Table Aliases

Alias Name:- Alias Name is a duplicate or Alternative Name.

We can define alias name in two levels,

1. Column-level alias name
2. Table-level alias name

When we create a duplicate name for a column then it is called column level alias name.

Syntax: <Column Name> AS <column alias Name>

When we create the alternative name for the tables in the database then it is called table level alias name.

Syntax: <Table Name> AS <Table alias Name>

Note: Alias names are mostly implemented in Joins.

Syntax:

SELECT <columns> FROM <table1> <alias1>, <table2> <alias2> JOIN <table2> ON <alias1>.<columnA> = <alias2>.<columnB>

Outer Join

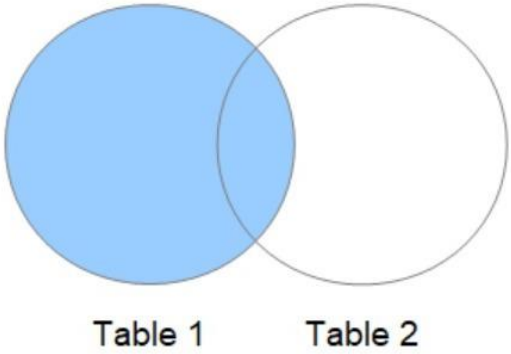
Outer Join is an extension of the Inner Join. In Inner Join mechanisms, the user will get matching data from the Tables and leave unmatching data from the Tables.

To overcome the drawback we use the Outer Join. By using outer we can retrieve matching data and also unmatching data from the tables at the same time.

What happens if you're doing an inner join and one of the records in a table doesn't have a matching record in the other table?
Then no row appears in the result set. However, you may still wish to see the row in your report. For example, you may have a customer, but that customer hasn't placed any orders yet. You still want to list the customer. That's where outer joins come to the rescue.

Left Outer Joins

A left outer join returns all records from the table on the left and the records that match from the table on the right.
If there is no matching record, then NULL is returned for any columns selected from the table on the right.
It is the equivalent of an inner join plus the unmatched rows from the table on the left.
It retrieves matching data from multiple tables and also unmatching data from the left-hand side table only.



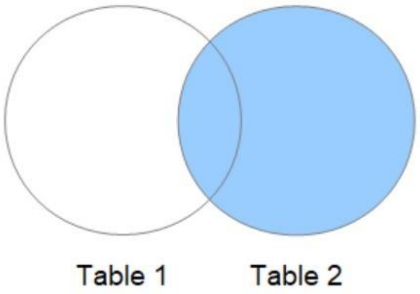
Syntax:

SELECT <columns> FROM <table1> LEFT JOIN <table2> ON <table1>.<columnA> = <table2>.<columnB>

Right Outer Join

It retrieves matching data from multiple tables and also unmatching data from the right-hand side table only.

A right outer join is the opposite of a left outer join. A right outer join returns all records from the table on the right and the records that match from the table on the left. If there is no matching record, then NULL is returned for any columns selected from the table on the left.
It is the equivalent of an inner join plus the unmatched rows from the table on the right.

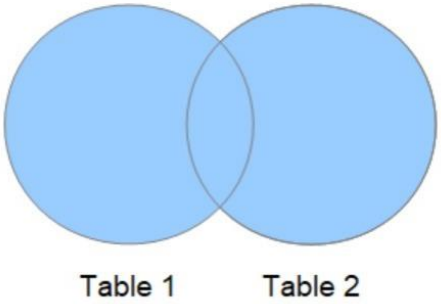


Syntax:

SELECT <columns> FROM <table1> RIGHT JOIN <table2> ON <table1>.<columnA> = <table2>.<columnB>

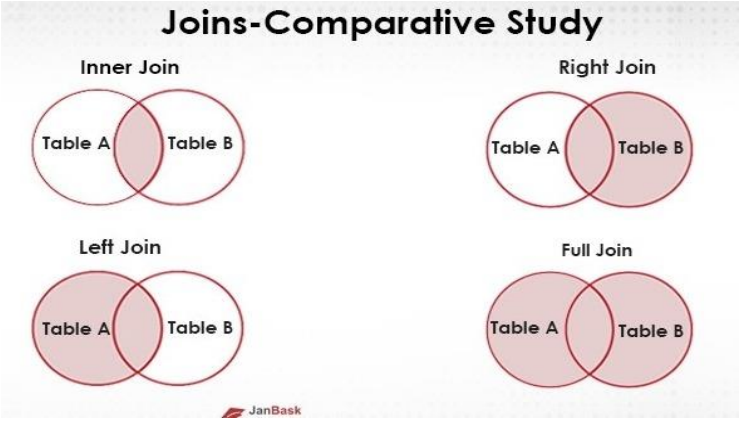
Full Outer Joins

A full outer join returns all matching records between the table on the left and the table on the right, as well as all non-matching records on both sides. It is the equivalent of an inner join plus the unmatched rows from the table on the left and the unmatched rows from the table on the right.



Syntax:

SELECT <columns>FROM <table1>FULL JOIN <table2> ON <table1>.<columnA> = <table2>.<columnB>



Cross Join

When we Join the two table's information without any condition it is known as Cross Join

In the Cross Join mechanism, each record of a first Table is joined with each record of the second Table.

For example, if the first Table contains A number of records and the second Table contains B number of records then we will get the cross product A*B records.

Non-Equi Join

When we retrieve the data from multiple tables based on any condition except an Equality condition, it is known as Non-Equi Join.

When we implement Non-Equi Join there is no requirement to maintain a common column in the Table. It supports all operators.

Self Join

A Table joining by itself is known as a Self Join. Self Join can be implemented when any 2 columns have some relationship within the same Table.

It can be worked on a single Table only. When we use Self Join on a Table then we should create alias names for the Table. Without alias names, we cannot implement Self Join.

A Table contains any number of alias names.

Natural Join

Natural Join is used for avoiding the duplicate columns from a Result set.

Advantages of Joins:

- The advantage of a join includes that it executes faster.
- The retrieval time of the query using joins almost always will be faster than that of a subquery.
- By using joins, you can maximize the calculation burden on the database i.e., instead of multiple queries using one join query. This means you can make better use of the database's abilities to search through, filter, sort, etc.

Disadvantages of Joins:

- Disadvantage of using joins includes that they are not as easy to read as subqueries.
- More joins in a query means the database server has to do more work, which means that it is more time consuming process to retrieve data
- As there are different types of joins, it can be confusing as to which join is the appropriate type of join to use to yield the correct desired result set.
- Joins cannot be avoided when retrieving data from a normalized database, but it is important that joins are performed correctly, as incorrect joins can result in serious performance degradation and inaccurate query results.

What is a Sub query (Nested Query)?

A subquery is a nested query (inner query) that's used to filter the results of the outer query. Subqueries can be used as an alternative to joins. A subquery is typically nested inside the WHERE clause.

Syntax:

SELECT <columns> FROM <table> WHERE <column> <operator> (SELECT <columns> FROM <table>)

Subqueries must always be enclosed within parentheses.

The operator can correspond with one of the following values:

IN, =, <>, <, >, >=, <=

If the subquery returns more than one result, then you can only use the IN operator

The table that's specified in the subquery is typically different than the one in the outer query, but it can be the same.

Regular Subquery vs. Correlated Subquery

The type of subquery we have studied so far, which we'll refer to as a regular subquery, is independent of the outer query and gets executed once. Its results are used by the outer query.

On the other hand, a correlated subquery depends on the outer query and gets executed once for each row returned by the outer query. It's also referred to as a repeating subquery.

Because correlated subqueries get executed multiple times, they can be slow.

Advantages of Subquery:

- Subqueries divide the complex query into isolated parts so that a complex query can be broken down into a series of logical steps.
- It is easy to understand and code maintenance is also at ease.
- Subqueries allow you to use the results of another query in the outer query.
- In some cases, subqueries can replace complex joins and unions.

Disadvantages of Subquery:

- The optimizer is more mature for MYSQL for joins than for subqueries, so in many cases a statement that uses a subquery can be executed more efficiently if you rewrite it as join.
- We cannot modify a table and select from the same table within a subquery in the same SQL statement.

Programs & Outputs:

Conclusion: _____

_____

_____

_____

Timely Submission (5)	Journal Presentation (5)	Performance (5)	Understanding (5)	Oral (5)	Total with Sign & Date

EXPERIMENT NO: 6

AIM: Write a PL/SQL block of code for the following requirements: -

Schema:

1. Borrower (Roll no., Name, Date of Issue, Name of Book, Status)
2. Fine (Roll no, Date, Amt.)

- Accept roll no. & name of book from user.
- Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day.
- If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.
- After submitting the book, status will change from I to R.
- If condition of fine is true, then details will be stored into fine table.

HARDWARE & SOFTWARE REQUIREMENT: MySQL Installer 8.0.33 Server, Any CPU with Pentium Processor or similar, 8 GB RAM or more, 500 GB Hard Disk or more**.**

THEORY:

PL/SQL – Procedural Language/Structured Query Language. PL/SQL is a block structured language that enables developers to combine the power of SQL with procedural statements. All the statements of a block are passed to oracle engine all at once which increases processing speed and decreases the traffic. The development of database applications typically requires language constructs similar to those that can be found in programming languages such as C, C++, or Pascal. These constructs are necessary in order to implement complex data structures and algorithms. A major restriction of the database language SQL, however, is that many tasks cannot be accomplished by using only the provided language elements.

PL/SQL (Procedural Language/SQL) is a procedural extension of Oracle-SQL that offers language constructs similar to those in imperative programming languages. OR

A PL/SQL is a procedural language extension to the SQL in which you can declare and use the variables, constants, do exception handling and you can also write the program modules in the form of PL/SQL subprograms.PL/SQL combines the features of a procedural language with structured query language.PL/SQL allows users and designers to develop complex database applications that require the usage of control structures and procedural elements such as procedures, functions, and modules.

The basic construct in PL/SQL is a block. Blocks allow designers to combine logically related

(SQL-) statements into units. In a block, constants and variables can be declared, and variables can be used to store query results. Statements in a PL/SQL block include SQL statements, control structures (loops), condition statements (if-then-else), exception handling, and calls of other

PL/SQL blocks. PL/SQL blocks that specify procedures and functions can be grouped into packages. A package is similar to a module and has an interface and an implementation part. Oracle offers several predefined packages, for example, input/output routines, file handling, job scheduling etc. (see directory $ORACLE HOME/rdbms/admin).

Another important feature of PL/SQL is that it offers a mechanism to process query results in a tuple-oriented way, that is, one tuple at a time. For this, cursors are used. A cursor basically is a pointer to a query result and is used to read attribute values of selected tuples into variables. A cursor typically is used in combination with a loop construct such that each tuple read by the cursor can be processed individually.

In summary, the major goals of PL/SQL are to

• Increase the expressiveness of SQL,

• Process query results in a tuple-oriented way,

• Optimize combined SQL statements,

• Develop modular database application programs,

• Reuse program code, and

• Reduce the cost for maintaining and changing applications

Advantages of PL/SQL: Following are some advantages of Pl/SQL

1) Support for SQL: -PL/SQL is the procedural language extension to SQL supports all the functionalities of SQL.

2) Improved performance: - In SQL every statement individually goes to the ORACLE server, get processed and then execute. But in PL/SQL an entire block of statements can be sent to ORACLE server at one time, where SQL statements are processed one at atime.PL/SQL block statements drastically reduce communication between the application and ORACLE. This helps in improving the performance.

3) Higher Productivity: - Users use procedural features to build applications.PL/SQL code is written in the form of PL/SQL block.PL/SQL blocks can also use in other ORACLE Forms, ORACLE reports. This code reusability increases the programmer's productivity.

4) Portability: - Applications written in PL/SQL are portable. We can port them from one environment to any computer hardware and operating system environment running ORACLE.

5) Integration with ORACLE: -Both PL/SQL and ORACLE are SQL based.PL/SQL variables have datatypes native to the oracle RDBMS dictionary. This gives tight integration with ORACLE.

Features of PL/SQL: -

1) We can define and use variables and constants in PL/SQL.

2) PL/SQL provides control structures to control the flow of a program. The control structures supported by PL/SQL are if,Then, loop, for..loop and others.

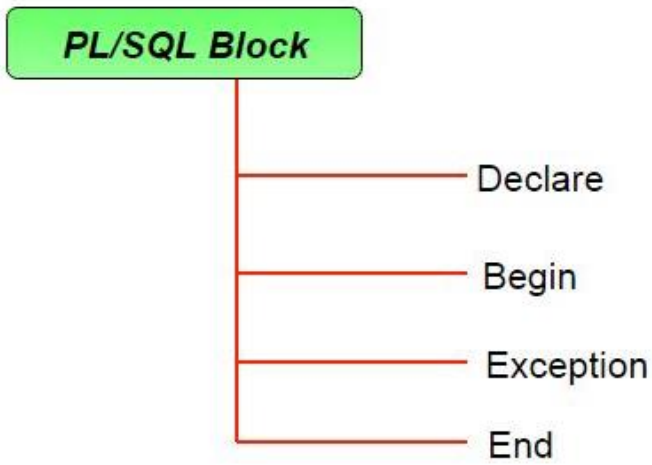
3) We can do row by row processing of data in PL/SQL.PL/SQL supports row by row processing using the mechanism called cursor.

4) We can handle pre-defined and user-defined error situations. Errors are warnings and called as exceptions in PL/SQL.

5) We can write modular application by using sub programs.

The structure of PL/SQL program: -

PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is more powerful than SQL. The basic unit in PL/SQL is a block. All PL/SQL programs are made up of blocks, which can be nested within each other.



The basic unit of code in any PL/SQL program is a block. All PL/SQL programs are composed of blocks. These blocks can be written sequentially.

The structure of PL/SQL block is: -

Declaration **e.g a int**

Declaration section

Executable code

Executable section

begin

code

end

exceptional handling

Exception handling section

END;

Where

1) Declaration section: PL/SQL variables, types, cursors, and local subprograms are defined here.

2) Executable section: Procedural and SQL statements are written here. This is the main section of the block. This section is required.

3) Exception handling section: Error handling code is written here.This section is optional whether it is defined within body or outside body of program.

Delimiters in SQL

Delimiters are used when we need to define the stored procedures as well as to create triggers. Default delimiter is semicolon. Others are $, //

Procedures in PL/SQL

A stored procedure in PL/SQL is nothing but a series of declarative SQL statements which can be stored in the database catalogue. A procedure can be thought of as a function or a method.

1. Cursor in PL/SQL:

A cursor can be basically referred to as a pointer to the context area. Context area is a memory area that is created by Oracle when SQL statement is processed. The cursor is thus responsible for holding the rows that have been returned by a SQL statement. Thus the PL/SQL controls the context area by the help of cursor.An Active set is basically the set of rows that the cursor holds. The cursor can be of two types: Implicit Cursor, and Explicit Cursor.

Advantages of Cursor:

They are helpful in performing the row by row processing and also row wise validation on each row.

1. Better concurrency control can be achieved by using cursors.

2. Cursors are faster than while loops.

Disadvantages of Cursor:

1. They use more resources each time and thus may result in network round trip.
2. More number of network round trips can degrade the performance and reduce the speed.

2. Trigger in PL/SQL:

A Trigger is basically a program which gets automatically executed in response to some events such as modification in the database. Some of the events for their execution are DDL statement, DML statement or any Database operation. Triggers are thus stored within the database and come into action when specific conditions match. Hence, they can be defined on any schema, table, view etc. There are six types of triggers: BEFORE INSERT, AFTER INSERT, BEFORE UPDATE, AFTER UPDATE, BEFORE DELETE, and AFTER DELETE.

Conditional statements and Loops used in PL/SQL:

Conditional statements check the validity of a condition and accordingly execute a set of statements. The conditional statements supported by Pl/SQL is

1) IF..THEN

2) IF..THEN..ELSE

3) IF..THEN..ELSIF

1) IF...THEN

Syntax1: -

If condition THEN

Statement list

END IF;

2) IF..THEN..ELSE

Syntax 2:-

IF condition THEN

Statement list

ELSE

Statements

END IF;

3) IF..THEN..ELSIF

Syntax 3: -

If condition, THEN

Statement list

ELSIF condition THEN

Statement list

ELSE

Statement list

END IF;

END IF;

2) CASE Expression :CASE expression can also be used to control the branching logic within PL/SQL blocks. The general syntax is

CASE

WHEN <expression> THEN <statements>;

WHEN <expression> THEN <statements>;

.

ELSE

<statements>;

END CASE;

Here expression in WHEN clause is evaluated sequentially. When result of expression is TRUE, then corresponding set of statements are executed and program flow goes to END CASE.

ITERATIVE Constructs : Iterative constructs are used to execute a set of statements

respectively. The iterative constructs supported by PL/SQL are follows:

1) SIMPLE LOOP

2) WHILE LOOP

3) FOR LOOP

1) The Simple LOOP : It is the simplest iterative construct and has syntax like:

LOOP

Statements

END LOOP;

The LOOP does not facilitate a checking for a condition and so it is an endless loop. To end the iterations, the EXIT statement can be used.

LOOP

<statement list>
IF condition THEN
EXIT;
END IF;
END LOOP;
The statements here is executable statements,which will be executed repeatedly until the condition given if IF..THEN evaluates TRUE.

2) THE WHILE LOOP

The WHILE…LOOP is a condition driven construct i.e the condition is a part of the loop construct and not to be checked separately. The loop is executed as long as the condition evaluates to TRUE.
The syntax is: -
WHILE condition LOOP
Statements
END LOOP;
The condition is evaluated before each iteration of loop. If it evaluates to TRUE, sequence of statements are executed. If the condition is evaluated to FALSE or NULL, the loop is finished and the control resumes after the END LOOP statement.

3) THE FOR LOOP: The number of iterations for LOOP and WHILE LOOP is not known in advance. THE number of iterations depends on the loop condition. The FOR LOOP can be used to
The syntax is: -
For loop counter IN [REVERSE] Low bound. High bound LOOP
Statements;
End loop;
Where

- loop counter –is the implicitly declared index variable as BINARY_INTEGER.
- Low bound and high bound specify the number of iteration.
- Statements: -Are the contents of the loop

EXCEPTIONS: - Exceptions are errors or warnings in a PL/SQL program.PL/SQL implements

error handling using exceptions and exception handler.

Exceptions are the run time error that a PL/SQL program may encounter.

There are two types of exceptions

1) Predefined exceptions

2) User defined exceptions

1) Predefined exceptions: - Predefined exceptions are the error condition that are defined by

ORACLE. Predefined exceptions cannot be changed. Predefined exceptions correspond to

common SQL errors. The predefined exceptions are raised automatically whenever a PL/SQL program violates an ORACLE rule.

2)User defined Exceptions: - A user defined exceptions is an error or a warning that is defined by the program. User defined exceptions can be define in the declaration section of PL/SQL block. User defined exceptions are declared in the declarative section of a PL/SQL block. Exceptions have a type Exception and scope.

Syntax :

DECLARE

<Exception Name> EXCEPTION;

BEGIN

….

RAISE <Exception Name>

…

EXCEPTION

WHEN <Exception name> THEN

<Action>

END; have a definite numbers of iterations.

Exception Handling

A PL/SQL block may contain statements that specify exception handling routines. Each error or warning during the execution of a PL/SQL block raises an exception. One can distinguish between two types of exceptions:

• System defined exceptions

• User defined exceptions (which must be declared by the user in the declaration part of a block where the exception is used/implemented)

System defined exceptions are always automatically raised whenever corresponding errors or warnings occur. User defined exceptions, in contrast, must be raised explicitly in a sequence of statements using raise <exception name>. After the keyword exception at the end of a block, user defined exception handling routines are implemented. An implementation has the pattern when <exception name> then <sequence of statements>;

Syntax: - <Exception_name>Exception;

Handling Exceptions: - Exceptions handlers for all the exceptions are written in the exception

handling section of a PL/SQL block.

Syntax: -

```
Exception
    When exception_name then
        Sequence_of_statements1;
    When exception_name then
        Sequence_of_statements2;
    When exception_name then
        Sequence_of_statements3;
End;
```

Conclusion: _____

_____

_____

_____

Programs & Outputs:

Timely Submission (5)	Journal Presentation (5)	Performance (5)	Understanding (5)	Oral (5)	Total with Sign & Date

EXPERIMENT NO: 7

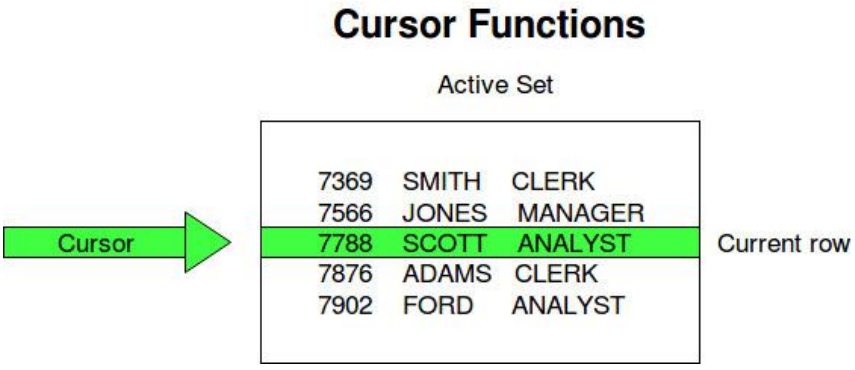
AIM: Write a PL/SQL block of code using parameterized Cursor that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table, then that data should be skipped.

HARDWARE & SOFTWARE REQUIREMENT: MySQL Installer 8.0.33 Server, Any CPU with Pentium Processor or similar, 8 GB RAM or more, 500 GB Hard Disk or more**.**

THEORY: What is Cursor in PL/SQL: - When an SQL statement is processed, Oracle creates a memory area known as context area. A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor. It contains all information needed for processing the statement. In PL/SQL, the context area is controlled by Cursor. A cursor contains information on a select statement and the rows of data accessed by it. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set. We can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time.

To execute SQL statements, a work area is used by the Oracle engine for its internal processing and storing the information. This work area is private to SQL's operations. The 'Cursor' is the PL/SQL construct that allows the user to name the work area and access the stored information in it.

Use of Cursor: - The major function of a cursor is to retrieve data, one row at a time, from a result set, unlike the SQL commands which operate on all the rows in the result set at one time. Cursors are used when the user needs to update records in a singleton fashion or in a row by row manner, in a database table. The Data that is stored in the Cursor is called the Active Data Set. Oracle DBMS has another predefined area in the main memory Set, within which the cursors are opened. Hence the size of the cursor is limited by the size of this pre-defined area.



There are two types of cursors −

1. Implicit cursors
2. Explicit cursors

1) Implicit Cursors

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

These are created by default to process the statements when DML statements like INSERT, UPDATE, DELETE etc. are executed. Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

In PL/SQL, you can refer to the most recent implicit cursor as the SQL cursor, which always has attributes such as %FOUND, %ISOPEN, %NOTFOUND, and %ROWCOUNT. The SQL cursor has additional attributes, %BULK_ROWCOUNT and %BULK_EXCEPTIONS, designed for use with the FORALL statement.

The following table provides the description of the most used attributes –

Sr. No	Attribute & Description
1	%FOUND Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
2	%NOTFOUND The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
3	%ISOPEN Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
4	%ROWCOUNT Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Any SQL cursor attribute will be accessed as sql%attribute_name as shown below in the example.

Example: We will be using the CUSTOMERS table

Select * from customers;

```
Select * from customers;

+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
+----+----------+-----+-----------+----------+
```

The following program will update the table and increase the salary of each customer by 500 and use the SQL%ROWCOUNT attribute to determine the number of rows affected –

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE customers
  SET salary = salary + 500;
  IF sql%notfound THEN
    dbms_output.put_line('no customers selected');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers selected ');
  END IF;
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result

```
Select * from customers;

+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2500.00 |
|  2 | Khilan   |  25 | Delhi     |  2000.00 |
|  3 | kaushik  |  23 | Kota      |  2500.00 |
|  4 | Chaitali |  25 | Mumbai    |  7000.00 |
|  5 | Hardik   |  27 | Bhopal    |  9000.00 |
|  6 | Komal    |  22 | MP        |  5000.00 |
+----+----------+-----+-----------+----------+
```

If you check the records in customers table, you will find that the rows have been updated

```
Select * from customers;

+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2500.00 |
|  2 | Khilan   |  25 | Delhi     |  2000.00 |
|  3 | kaushik  |  23 | Kota      |  2500.00 |
|  4 | Chaitali |  25 | Mumbai    |  7000.00 |
|  5 | Hardik   |  27 | Bhopal    |  9000.00 |
|  6 | Komal    |  22 | MP        |  5000.00 |
+----+----------+-----+-----------+----------+
```

2) Explicit Cursors

Explicit cursors are programmer-defined cursors for gaining more control over the context area. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

The syntax for creating an explicit cursor is

CURSOR cursor_name IS select_statement;

Working with an explicit cursor includes the following steps

i. Declaring the cursor for initializing the memory
ii. Opening the cursor for allocating the memory
iii. Fetching the cursor for retrieving the data
iv. Closing the cursor to release the allocated memory

i) **Declaring the Cursor:** Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example

```
CURSOR c_customers IS
    SELECT id, name, address FROM customers;
```

ii) **Opening the Cursor:** Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows

```
OPEN c_customers;
```

iii) **Fetching the Cursor:** Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows −

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

iv) Closing the Cursor: Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows −

```
CLOSE c_customers;
```

Example

Following is a complete example to illustrate the concepts of explicit cursors

```
DECLARE
   c_id customers.id%type;
   c_name customers.name%type;
   c_addr customers.address%type;
   CURSOR c_customers is
      SELECT id, name, address FROM customers;
BEGIN
   OPEN c_customers;
   LOOP
   FETCH c_customers into c_id, c_name, c_addr;
      EXIT WHEN c_customers%notfound;
      dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
   END LOOP;
   CLOSE c_customers;
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result

```
1 Ramesh Ahmedabad
2 Khilan Delhi
3 kaushik Kota
4 Chaitali Mumbai
5 Hardik Bhopal
6 Komal MP

PL/SQL procedure successfully completed.
```

PL/SQL Cursor FOR LOOP:

The cursor FOR LOOP statement is an elegant extension of the numeric FOR LOOP statement.

The numeric FOR LOOP executes the body of a loop once for every integer value in a specified range. Similarly, the cursor FOR LOOP executes the body of the loop once for each row returned by the query associated with the cursor.

A nice feature of the cursor FOR LOOP statement is that it allows you to fetch every row from a cursor without manually managing the execution cycle i.e., OPEN, FETCH, and CLOSE.

The cursor FOR LOOP implicitly creates its loop index as a record variable with the row type in which the cursor returns and then opens the cursor.

In each loop iteration, the cursor FOR LOOP statement fetches a row from the result set into its loop index. If there is no row to fetch, the cursor FOR LOOP closes the cursor.

The cursor is also closed if a statement inside the loop transfers control outside the loop, e.g., EXIT and GOTO, or raises an exception.

The following illustrates the syntax of the cursor FOR LOOP statement:

```
FOR record IN cursor_name
LOOP
   process_record_statements;
END LOOP;
```

1) record: The record is the name of the index that the cursor FOR LOOP statement declares implicitly as a %ROWTYPE record variable of the type of the cursor.

The record variable is local to the cursor FOR LOOP statement. It means that you can only reference it inside the loop, not outside. After the cursor FOR LOOP statement execution ends, the record variable becomes undefined.

2) cursor_name: The cursor_name is the name of an explicit cursor that is not opened when the loop starts. Note that besides the cursor name, you can use a SELECT statement as shown below:

```
FOR record IN (select_statement)
LOOP
   process_record_statements;
END LOOP;
```

In this case, the cursor FOR LOOP declares, opens, fetches from, and closes an implicit cursor. However, the implicit cursor is internal; therefore, you cannot reference it.

Note that Oracle Database automatically optimizes a cursor FOR LOOP to work similarly to a BULK COLLECT query. Although your code looks as if it fetched one row at a time, Oracle Database fetches multiple rows at a time and allows you to process each row individually.

PL/SQL cursor FOR LOOP examples

Let's look at some examples of using the cursor FOR LOOP statement.

1) PL/SQL cursor FOR LOOP example

```
DECLARE
  CURSOR c_product
  IS
    SELECT
        product_name, list_price
    FROM
        products
    ORDER BY
        list_price DESC;
BEGIN
  FOR r_product IN c_product
  LOOP
    dbms_output.put_line( r_product.product_name || ': $' ||  r_product.list_price );
  END LOOP;
END;
```

3) Cursor FOR LOOP with a SELECT statement example

The following example is equivalent to the example above but uses a query in a cursor FOR LOOP statement.

```
BEGIN
  FOR r_product IN (
        SELECT
            product_name, list_price
        FROM
            products
        ORDER BY list_price DESC
    )
  LOOP
     dbms_output.put_line( r_product.product_name ||
        ': $' ||
        r_product.list_price );
  END LOOP;
END;
```

PARAMETERIZED CURSORS (PL/SQL):

Parameterized cursors are static cursors that can accept passed-in parameter values when they are opened.

A static cursor is a cursor whose associated query is fixed at compile time. Declaring a cursor is a prerequisite to using it. Declarations of static cursors using PL/SQL syntax within PL/SQL contexts are supported by the data server.

Syntax

CURSOR

cursor-name

IS

query

Description

cursor-name

Specifies an identifier for the cursor that can be used to reference the cursor and its result set.

query

Specifies a SELECT statement that determines a result set for the cursor.

Example

The following example shows a procedure that contains multiple static cursor declarations:

```
CREATE OR REPLACE PROCEDURE cursor_example
IS
      CURSOR emp_cur_1 IS SELECT * FROM emp;

      CURSOR emp_cur_2 IS SELECT empno, ename FROM emp;

      CURSOR emp_cur_3 IS SELECT empno, ename
                          FROM emp
                          WHERE deptno = 10
                          ORDER BY empno;
BEGIN
     OPEN emp_cur_1;
          ...
END;
```

The following example includes a parameterized cursor. The cursor displays the name and salary of each employee in the EMP table whose salary is less than that specified by a passed-in parameter value.

```
DECLARE
    my_record       emp%ROWTYPE;
    CURSOR c1 (max_wage NUMBER) IS
        SELECT * FROM emp WHERE sal < max_wage;
BEGIN
    OPEN c1(2000);
    LOOP
        FETCH c1 INTO my_record;
        EXIT WHEN c1%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Name = ' || my_record.ename || ', salary = '
            || my_record.sal);
    END LOOP;
    CLOSE c1;
END;
```

If 2000 is passed in as the value of max_wage, only the name and salary data for those employees whose salary is less than 2000 is returned:

```
Name = SMITH, salary = 800.00
Name = ALLEN, salary = 1600.00
Name = WARD, salary = 1250.00
Name = MARTIN, salary = 1250.00
Name = TURNER, salary = 1500.00
Name = ADAMS, salary = 1100.00
Name = JAMES, salary = 950.00
Name = MILLER, salary = 1300.00
```

Parameterized cursors can only reference its own parameters. Parameterized cursors cannot reference local variables. In this example, cursor_id must be used in the select statement because in_id is not within the scope of the cursor.

```
CREATE OR REPLACE PROCEDURE myproc (in_id IN NUMBER) IS
  CURSOR c(cursor_id in NUMBER) IS
  SELECT id,emp_name FROM employee WHERE id = cursor_id;
  empName VARCHAR2(100);

  BEGIN
    FOR r IN c(in_id) LOOP
      empName := r.emp_name;
      DBMS_OUTPUT.PUT_LINE(empName);
    END LOOP;
  END;
```

Conclusion: _____

_____

_____

_____

Programs & Outputs:

Timely Submission (5)	Journal Presentation (5)	Performance (5)	Understanding (5)	Oral (5)	Total with Sign & Date

EXPERIMENT NO: 8

AIM: Write a Stored Procedure namely proc_Grade for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and900 category is first class, if marks 899 and 825 category is Higher Second Class Write a PL/SQL block for using procedure created with above requirement. Stud_Marks(name, total_marks) Result(Roll,Name, Class).

HARDWARE & SOFTWARE REQUIREMENT: MySQL Installer 8.0.33 Server, Any CPU with Pentium Processor or similar, 8 GB RAM or more, 500 GB Hard Disk or more**.**

THEORY: Stored Procedure and Function in PL/SQL: -

Stored procedure and Function, both can be defined as a set of logically written statements, stored in the database and are executed when called, to perform a specific task.

Both function as well as stored procedure have a unique named block of code which is compiled and stored in the database.

Any stored procedure or function created, remains useless unless it is called. Therefore, after creating a stored procedure or function it is necessary to make a call for that stored procedure or function from another PL/SQL block in order to serve the purpose for which it has been created.

A stored procedure in PL/SQL is nothing but a series of declarative SQL statements which can be stored in the database catalogue. A procedure can be thought of as a function or a method. They can be invoked through triggers, other procedures, or applications on Java, PHP etc.

All the statements of a block are passed to Oracle engine all at once which increases processing speed and decreases the traffic.

Advantages:

- They result in performance improvement of the application. If a procedure is being called frequently in an application in a single connection, then the compiled version of the procedure is delivered.
- They reduce the traffic between the database and the application, since the lengthy statements are already fed into the database and need not be sent again and again via the application.
- They add to code reusability, similar to how functions and methods work in other languages such as C/C++ and Java.

Disadvantages:

- Stored procedures can cause a lot of memory usage. The database administrator should decide an upper bound as to how many stored procedures are feasible for a particular application.
- MySQL does not provide the functionality of debugging the stored procedures.

PL/SQL Procedure

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

The procedure contains a header and a body.

Header: The header contains the name of the procedure and the parameters or variables passed to the procedure.

Body: The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

How to pass parameters in procedure.

When you want to create a procedure or function, you have to define parameters. There is three ways to pass parameters in procedure:

IN parameters: The IN parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or the function.

OUT parameters: The OUT parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.

INOUT parameters: The INOUT parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

A procedure may or may not return any value.

PL/SQL Create Procedure: A procedure is created with the CREATE OR REPLACE PROCEDURE statement.

Syntax for creating procedure is

```
CREATE [OR REPLACE] PROCEDURE procedure_name
   [ (parameter [,parameter]) ]
IS
   [declaration_section]
BEGIN
   executable_section
[EXCEPTION
   exception_section]
END [procedure_name];
```

Where,

- procedure-name specifies the name of the procedure.
- [OR REPLACE] option allows the modification of an existing procedure.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- procedure-body contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone procedure.

Create procedure example 1

The following example creates a simple procedure that displays the string 'Hello World!' on the screen when executed.

```
CREATE OR REPLACE PROCEDURE greetings
AS
BEGIN
   dbms_output.put_line('Hello World!');
END;
/
```

When the above code is executed using the SQL prompt, it will produce the following result

```
Procedure created.
```

Executing a Standalone Procedure: A standalone procedure can be called in two ways

1. Using the EXECUTE keyword
2. Calling the name of the procedure from a PL/SQL block

The above procedure named 'greetings' can be called with the EXECUTE keyword as

```
EXECUTE greetings;
```

The above call will display

```
Hello World

PL/SQL procedure successfully completed.
```

The procedure can also be called from another PL/SQL block

```
BEGIN
   greetings;
END;
/
```

The above call will display

```
Hello World

PL/SQL procedure successfully completed.
```

Deleting a Standalone Procedure

A standalone procedure is deleted with the DROP PROCEDURE statement. Syntax for deleting a procedure is

```
DROP PROCEDURE procedure-name;
```

You can drop the greetings procedure by using the following statement

```
DROP PROCEDURE greetings;
```

Create procedure example 2

In this example, we are going to insert record in user table. So we need to create user table first.

Table creation:

```
create table user(id number(10) primary key,name varchar2(100));
```

Now write the procedure code to insert record in user table.

Procedure Code:

```
create or replace procedure "INSERTUSER"
(id IN NUMBER,
name IN VARCHAR2)
is
begin
insert into user values(id,name);
end;
/
```

Output:

```
Output:

Procedure created.
```

PL/SQL program to call procedure

Let's see the code to call above created procedure.

```
BEGIN
  insertuser(101,'Rahul');
  dbms_output.put_line('record inserted successfully');
END;
/
```

Now, see the "USER" table, you will see one record is inserted.

ID	Name
101	Rahul

PL/SQL Drop Procedure

Syntax for drop procedure

DROP PROCEDURE procedure_name;

Example of drop procedure

DROP PROCEDURE pro1;

PL/SQL FUNCTION

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too.

A standalone function is created using the CREATE FUNCTION statement. The simplified syntax for the CREATE OR REPLACE FUNCTION statement is as follows

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
  < function_body >
END [function_name];
```

Where,

- function-name specifies the name of the function.
- [OR REPLACE] option allows the modification of an existing function.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- The function must contain a return statement.
- The RETURN clause specifies the data type you are going to return from the function.
- function-body contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

The function must contain a return statement.

- RETURN clause specifies that data type you are going to return from the function.
- Function_body contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

Stored Procedure or function's block of code in PL/SQL is made up of the following three sections:

1. **Declarative section:** In this section, variables, constants, cursor or exceptions that are going to be used by procedure or function are declared.
2. **Executable section:** In this section, the definition of procedure or function created is written. This section also contains the SQL or PL/SQL statements assigning values, controlling execution and manipulating data.
3. **Exception Handling section:** In this section, the expected exceptions are written which may arise during execution of code written in executable part. This section is optional.

Example

The following example illustrates how to create and call a standalone function. This function returns the total number of CUSTOMERS in the customers table.

```
Select * from customers;

+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
+----+----------+-----+-----------+----------+
```

```
CREATE OR REPLACE FUNCTION totalCustomers
RETURN number IS
   total number(2) := 0;
BEGIN
   SELECT count(*) into total
   FROM customers;

   RETURN total;
END;
/
```

When the above code is executed using the SQL prompt, it will produce the following result −

Function created.

Calling a Function: While creating a function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. When a program calls a function, the program control is transferred to the called function.

A called function performs the defined task and when its return statement is executed or when the last end statement is reached, it returns the program control back to the main program.

To call a function, you simply need to pass the required parameters along with the function name and if the function returns a value, then you can store the returned value. Following program calls the function totalCustomers from an anonymous block

```
DECLARE
   c number(2);
BEGIN
   c := totalCustomers();
   dbms_output.put_line('Total no. of Customers: ' || c);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result

```
Total no. of Customers: 6

PL/SQL procedure successfully completed.
```

Conclusion: _____

_____

_____

_____

Programs & Outputs:

Timely Submission (5)	Journal Presentation (5)	Performance (5)	Understanding (5)	Oral (5)	Total with Sign & Date

EXPERIMENT NO: 9

AIM: Write a database trigger on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library_Audit table.

HARDWARE & SOFTWARE REQUIREMENT: MySQL Installer 8.0.33 Server, Any CPU with Pentium Processor or similar, 8 GB RAM or more, 500 GB Hard Disk or more**.**

THEORY:

Triggers are stored programs, which are automatically executed or fired when some events occur.

Trigger is invoked by Oracle engine automatically whenever a specified event occurs. Trigger is stored into database and invoked repeatedly, when specific condition match.

Triggers in oracle are blocks of PL/SQL code which oracle engine can execute automatically based on some action or event.

These events can be:

- DDL statements (CREATE, ALTER, DROP, TRUNCATE)
- DML statements (INSERT, SELECT, UPDATE, DELETE)
- Database operation like connecting or disconnecting to oracle (LOGON, LOGOFF, SHUTDOWN)

Triggers are automatically and repeatedly called upon by oracle engine on satisfying certain condition.

Triggers can be activated or deactivated depending on the requirements.

If triggers are activated, then they are executed implicitly by oracle engine and if triggers are deactivated then they are executed explicitly by oracle engine.

Uses of Triggers

Here we have mentioned a few use cases where using triggers proves very helpful:

- Maintaining complex constraints which is either impossible or very difficult via normal constraint (like primary, foreign, unique etc) applying technique.
- Recording the changes made on the table.
- Automatically generating primary key values.
- Prevent invalid transactions to occur.
- Granting authorization and providing security to database.

- Enforcing referential integrity.

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

PL/SQL: Parts of a Trigger

Whenever a trigger is created, it contains the following three sequential parts:

- **Triggering Event or Statement:** The statements due to which a trigger occurs is called triggering event or statement. Such statements can be DDL statements, DML statements or any database operation, executing which gives rise to a trigger.
- **Trigger Restriction:** The condition or any limitation applied on the trigger is called trigger restriction. Thus, if such a condition is TRUE then trigger occurs otherwise it does not occur.
- **Trigger Action:** The body containing the executable statements that is to be executed when trigger occurs that is with the execution of Triggering statement and upon evaluation of Trigger restriction as True is called Trigger Action.

PL/SQL: Types of Triggers

The above diagram clearly indicated that Triggers can be classified into three categories:

1. Level Triggers
2. Event Triggers
3. Timing Triggers

which are further divided into different parts.

1. LEVEL TRIGGERS

There are 2 different types of level triggers, they are:

A. **Row Level Triggers:** It fires for every record that got affected with the execution of DML statements like INSERT, UPDATE, DELETE etc. It always use a FOR EACH ROW clause in a triggering statement.

B. **Statement Level Triggers**

It fires once for each statement that is executed.

2. EVENT TRIGGERS

There are 3 different types of event triggers, they are:

i) DDL EVENT TRIGGER

It fires with the execution of every DDL statement (CREATE, ALTER, DROP, TRUNCATE).

ii) DML EVENT TRIGGER

It fires with the execution of every DML statement (INSERT, UPDATE, DELETE).

iii) DATABASE EVENT TRIGGER

It fires with the execution of every database operation which can be LOGON, LOGOFF, SHUTDOWN, SERVERERROR etc.

3. Timing Triggers

There are 2 different types of timing triggers, they are:

i) BEFORE TRIGGER

It fires before executing DML statement.

Triggering statement may or may not executed depending upon the before condition block.

ii) AFTER TRIGGER

It fires after executing DML statement.

Syntax for creating Triggers

Following is the syntax for creating a trigger:

CREATE OR REPLACE TRIGGER <trigger_name>

BEFORE/AFTER/INSTEAD OF

```
    INSERT/DELETE/UPDATE ON <table_name>
REFERENCING (OLD AS O, NEW AS N)
    FOR EACH ROW WHEN (test_condition)
DECLARE
    -- Variable declaration;
BEGIN
    -- Executable statements;
EXCEPTION
    -- Error handling statements;
END <trigger_name>;
END;
```

where,

CREATE OR REPLACE TRIGGER is a keyword used to create a trigger and <trigger_name> is user-defined where a trigger can be given a name.

BEFORE/AFTER/INSTEAD OF specify the timing of the trigger's occurance. INSTEAD OF is used when a view is created.

INSERT/UPDATE/DELETE specify the DML statement.

<table_name> specify the name of the table on which DML statement is to be applied.

REFERENCING is a keyword used to provide reference to old and new values for DML statements.

FOR EACH ROW is the clause used to specify row level tigger.

WHEN is a clause used to specify condition to be applied and is only applicable for row-level trigger.

DECLARE, BEGIN, EXCEPTION, END are the different sections of PL/SQL code block containing variable declaration, executable statements, error handling statements and marking end of PL/SQL block respectively where DECLARE and EXCEPTION part are optional.

Example:

```
CREATE OR REPLACE TRIGGER CheckAge
BEFORE
```

```
INSERT OR UPDATE ON student
FOR EACH ROW
BEGIN
    IF :new.Age>30 THEN
        raise_application_error(-20001, 'Age should not be greater than 30');
    END IF;
END;
```

Output

Trigger created.

Following is the STUDENT table,

ROLLNO	SNAME	AGE	COURSE
11	Anu	20	BSC
12	Asha	21	BCOM
13	Arpit	18	BCA
14	Chetan	20	BCA
15	Nihal	19	BBA

After initializing the trigger CheckAge, whenever we will insert any new values or update the existing values in the above table STUDENT our trigger will check the age before executing INSERT or UPDATE statements and according to the result of triggering restriction or condition it will execute the statement.

Let's take a few examples and try to understand this,

Example 1:

INSERT into STUDENT values(16, 'Saina', 32, 'BCOM');

OUTPUT

```
Age should not be greater than 30
```

Example 2:

INSERT into STUDENT values(17, 'Anna', 22, 'BCOM');

OUTPUT

```
INSERT into STUDENT values(17, 'Anna', 22, 'BCOM');

1 row created
```

Example 3:

UPDATE STUDENT set age=31 where ROLLNO=12;

OUTPUT

```
Age should not be greater than 30
```

Example 4:

UPDATE STUDENT set age=23 where ROLLNO=12;

OUTPUT

```
1 row updated.
```

Conclusion: _____

_____

_____

_____

Programs & Outputs:

Timely Submission (5)	Journal Presentation (5)	Performance (5)	Understanding (5)	Oral (5)	Total with Sign & Date