
Sinhgad Institutes

**SINHGAD TECHNICAL EDUCATION SOCIETY'S**

**NBN SINHGAD TECHNICAL INSTITUTE CAMPUS**

**Ambegaon(BK) Pune**

## Department of Information Technology

**LAB MANUAL**

# Database Management Lab (DBMSL)

**S. E. (2019 Course)**

**Prepared by**

**Mrs.Supriya Kapase**

Sinhgad Institutes

**NBN SINHGAD TECHNICAL INSTITUTE CAMPUS**

Ambegaon(BK) Pune

**DEPARTMENT OF INFORMATION ENGINEERING**

# CERTIFICATE

***This is to certify that Mr. /Miss _____ _____of class _____Div_____Roll No. _____Examination Seat No. _____has completed all the practical work in the subject of _____ satisfactorily as prescribed by Savitribai Phule Pune University in the Academic Year 2023 - 2024.***

| | | |
|---|---|---|
| Prof.S.D.Kapase | Prof.R.M.Samant | Dr. S.P.Patil |
| Staff In-charge | Head of Department | Principal |

# INDEX

| Sr. No. | Name of Experiment | Date | Page No. | Sign & Remark |
|---|---|---|---|---|
| | | | | |
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | |

# Group A-Assignment No-1

**Aim**: Study of MySQL Open source software. Discuss the characteristics like efficiency, scalability, performance and transactional properties
**Objective:** To be aware about Open source software.

**Theory:**

SQL is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc., SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix and SQL Server use SQL as their standard database language.
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.

## What is MySQL?

MySQL, is the Open Source SQL database management system, is developed, distributed, and supported by Oracle Corporation. MySQL is a database management system that allows you to manage relational databases. It is open source software backed by Oracle. It means you can use MySQL without paying a time. Also, if you want, you can change its source code to suit your needs.

- **MySQL is a database management system*:***
  A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or the vast amounts of information in a corporate network. To add, access, and process data stored in a computer database, you need a database management system such as MySQL Server.
- **MySQL software is Open Source:**
  Open Source means that it is possible for anyone to use and modify the software. Anybody can download the MySQL software from the Internet and use it without paying anything. If you wish, you may study the source code and change it to suit your needs. MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc

- **The MySQL Database Server is very fast, reliable, scalable, and easy to use.**

  If that is what you are looking for, you should give it a try. MySQL Server can run comfortably on a desktop or laptop, alongside your other applications, web servers, and so on, requiring little or no attention.

## MySQL Features:

**Relational Database Management System (RDBMS)**

MySQL is a relational database management system. This database language is based on the SQL queries to access and manage the records of the table.

**Easy to use**

MySQL is easy to use. We have to get only the basic knowledge of SQL. We can build and interact with MySQL by using only a few simple SQL statements.

**It is Secure**

MySQL consists of a solid data security layer that protects sensitive data from intruders. Also, passwords are encrypted in MySQL.

**Client/ Server Architecture**

MySQL follows the working of a client/server architecture. There is a database server (MySQL) and arbitrarily many clients (application programs), which communicate with the server; that is, they can query data, save changes, etc.

**It is scalable**

MySQL supports multi-threading that makes it easily scalable. It can handle almost any amount of data, up to as much as 50 million rows or more. The default file size limit is about 4 GB.

**Speed**

MySQL is considered one of the very fast database languages, backed by a large number of the benchmark test.

**High Flexibility**

MySQL supports a large number of embedded applications, which makes MySQL very flexible.

**High Availability:**

Specialized cluster servers offering instant failover.

**High Performance**

MySQL is faster, more reliable, and cheaper because of its unique storage engine architecture. It provides very high-performance results in comparison to other databases without losing an essential functionality of the software. It has fast loading utilities because of the different cache memory.

**Transactional Support:**

Unlimited row level locking and Distributed transactional capability and multi-version transactional support.

**Conclusion:** Studied MySQL Open source software successfully.

# Group A-Assignment No-2

**Aim**:. Install and configure client and server of MySQL.(Show all commands and necessary steps for installation and configuration.

**Objective:** To be aware about MySQL Databases.

**Theory**

The database for the AMC provides data storage to host all the data. The database stores information about MSI files and applications, deployment rules, and deployment rule sets.

## Install MySQL on Ubuntu

- Install the MySQL database server only and select Server Machine as the configuration type.
- Select the option to run MySQL as a service.
-

**Step 1. Update package index**

Execute the following command to update the local packages index with the latest changes made in the repositories:

```
sudo apt update
```

**Step 2. Upgrade packages**

Type the following command to upgrade the system:

```
sudo apt upgrade
```

**Step 3. Configure MySQLPPA**

MySQL provides an APT repository for install MySQL server and tools. You need to add this MySQL repository to your system's package source list.

First, download the repository package using the wget command:

```
wget -c https://repo.mysql.com//mysql-apt-config_0.8.13-1_all.deb
```

Then, install the MySQL repository package using the following dpkg command:

```
sudo dpkg -i mysql-apt-config_0.8.13-1_all.deb
```

**Step 4. Install MySQL**

Execute the following command to start installing MySQL:

```
sudo apt-get install mysql-server
```

It will prompt for the root's password. Enter a secure password and continue.

**Step 5. Secure MySQL server installation**

Execute the following command to adjust security to the MySQL Server:

```
sudo mysql_secure_installation
```

It will prompt you some security options that you should choose in order to secure the MySQL server:

- Remove anonymous users? (Press y|Y for Yes, any other key for No) : y
- Disallow root login remotennnnnnnnnnnnnnnnnn bhhly? (Press y|Y for Yes, any other key for No) : y
- Remove test database and access to it? (Press y|Y for Yes, any other key for No) : y
- Reload privilege tables now? (Press y|Y for Yes, any other key for No) : y

**Step 6. Manage MySQL Server via Systemd**

Typically, MySQL service is automatically started once the package is configured. To check whether the MySQL server is up and running or not, you use this command:

```
sudo systemctl status mysql
```

If you found that the MySQL server does not automatically start, you can use the following command to start it:

```
sudo systemctl status mysql
```

And start it automatically at system startup:

```
sudo systemctl enable mysql
```

**Step 7. Connect to MySQL server**

# To connect to the MySQL Server, use this command:

```
sudo mysql -u root -p
```

It will prompt for the password of the root account. You enter the password and press Enter, the following command will show if the password is valid:

```
mysql>
```

Use the SHOW DATABASES to display all databases in the current server:

```
mysql> show databases;
```

Here is the output:

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

4 rows **in set** (0.05 sec)
Code language: JavaScript (javascript)

**Conclusion:** Here you have learned step by step how to install MySQL 8 on Ubuntu.

# Group A-Assignment No-3

**Aim**: Study of SQLite: What is SQLite? Uses of Sqlite. Building and installing SQLite.
**Objective:** To be aware about SQLite Databases.

**Theory:**

SQLite is a small lightweight embedded database used in Application File formats, Database for mobile apps and websites. SQLite has compliance with ACID properties of database. It is faster and has simple to use API. SQLite comes with a standalone command-line interface (CLI) client that can be used to administer SQLite databases.

**SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. The source code for SQLite is in the public domain.**

**SQLite is different from other SQL databases because unlike most other SQL databases, SQLite does not have a separate server process. It reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file.**

**Why SQLite?**

1. SQLite does not require a separate server process or system to operate (serverless).
2. SQLite comes with zero-configuration, which means no setup or administration needed.
3. A complete SQLite database is stored in a single cross-platform disk file.
4. SQLite is very small and light weight, less than 400KiB fully configured or less than 250KiB with optional features omitted.
5. SQLite is self-contained, which means no external dependencies.
6. SQLite transactions are fully ACID-compliant, allowing safe access from multiple processes or threads.
7. SQLite supports most of the query language features found in SQL92 (SQL2) standard.
8. SQLite is written in ANSI-C and provides simple and easy-to-use API.
9. SQLite is available on UNIX (Linux, Mac OS-X, Android, iOS) and Windows (Win32, WinCE, WinRT).

**SQLite Limitations**

There are few unsupported features of SQL92 in SQLite which are listed in the following table.

| Sr.No. | Feature & Description |
|---|---|
| 1 | **RIGHT OUTER JOIN**<br>Only LEFT OUTER JOIN is implemented. |
| 2 | **FULL OUTER JOIN**<br>Only LEFT OUTER JOIN is implemented. |
| 3 | **ALTER TABLE**<br>The RENAME TABLE and ADD COLUMN variants of the ALTER TABLE command are supported. The DROP COLUMN, ALTER COLUMN, ADD CONSTRAINT are not supported. |
| 4 | **Trigger support**<br>FOR EACH ROW triggers are supported but not FOR EACH STATEMENT triggers. |
| 5 | **VIEWs**<br>VIEWs in SQLite are read-only. You may not execute a DELETE, INSERT, or UPDATE statement on a view. |
| 6 | **GRANT and REVOKE**<br>The only access permissions that can be applied are the normal file access permissions of the underlying operating system. |

## SQLite Commands

The standard SQLite commands to interact with relational databases are similar to SQL. They are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into groups based on their operational nature −

### DDL - Data Definition Language

| Sr.No. | Command & Description |
|---|---|
| 1 | **CREATE**<br>Creates a new table, a view of a table, or other object in database. |
| 2 | **ALTER**<br>Modifies an existing database object, such as a table. |

| | |
|---|---|
| 3 | **DROP**<br>Deletes an entire table, a view of a table or other object in the database. |

**DML - Data Manipulation Language**

| Sr.No. | Command & Description |
|---|---|
| 1 | **INSERT**<br>Creates a record |
| 2 | **UPDATE**<br>Modifies records |
| 3 | **DELETE**<br>Deletes records |

**DQL - Data Query Language**

| Sr.No. | Command & Description |
|---|---|
| 1 | **SELECT**<br>Retrieves certain records from one or more tables |

**Installation:**

**Install SQLite on Windows**

- **Step 1** − Go to SQLite download page, and download precompiled binaries from Windows section.
- **Step 2** − Download sqlite-shell-win32-*.zip and sqlite-dll-win32-*.zip zipped files.
- **Step 3** − Create a folder C:\>sqlite and unzip above two zipped files in this folder, which will give you sqlite3.def, sqlite3.dll and sqlite3.exe files.
- **Step 4** − Add C:\>sqlite in your PATH environment variable and finally go to the command prompt and issue sqlite3 command, which should display the following result.

```
C:\>sqlite3
SQLite version 3.7.15.2 2013-01-09 11:53:05
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

**Install SQLite on Linux**

Today, almost all the flavours of Linux OS are being shipped with SQLite. So you just issue the following command to check if you already have SQLite installed on your machine.

```
$sqlite3
SQLite version 3.7.15.2 2013-01-09 11:53:05
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

If you do not see the above result, then it means you do not have SQLite installed on your Linux machine. Following are the following steps to install SQLite −

- **Step 1** − Go to SQLite download page and download sqlite-autoconf-*.tar.gz from source code section.
- **Step 2** − Run the following command −

```
$tar xvfz sqlite-autoconf-3071502.tar.gz
$cd sqlite-autoconf-3071502
$./configure --prefix=/usr/local
$make
$make install
```

The above command will end with SQLite installation on your Linux machine.

**Install SQLite on Mac OS X**

Though the latest version of Mac OS X comes pre-installed with SQLite but if you do not have installation available then just follow these following steps −

- **Step 1** − Go to SQLite download page, and download sqlite-autoconf-*.tar.gz from source code section.
- **Step 2** − Run the following command −

```
$tar xvfz sqlite-autoconf-3071502.tar.gz
$cd sqlite-autoconf-3071502
$./configure --prefix=/usr/local
$make
$make install
```

The above procedure will end with SQLite installation on your Mac OS X machine. Which you can verify by issuing the following command −

```
$sqlite3
SQLite version 3.7.15.2 2013-01-09 11:53:05
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

Finally, you have SQLite command prompt where you can issue SQLite commands.

**Conclusion:** Studied SQLite database successfully.

# Group B-Assignment No-1

**Aim**: Design any database with at least 3 entities and relationships between them. Draw suitable ER/EER diagram for the system.

**Objectives:**

1. To design a database system.
2. To draw an ER/EER diagram for the system.

**Theory:**

**Introduction to SQL:**

The Structured Query Language (SQL) comprises one of the fundamental building blocks of modern database architecture. SQL defines the methods used to create and manipulate relational databases on all major platforms.

SQL comes in many flavors. Oracle databases utilize their proprietary PL/SQL. Microsoft SQL Server makes use of Transact-SQL. However, all of these variations are based upon the industry standard ANSI SQL.

SQL commands can be divided into two main sublanguages.

1. Data Definition Language
2. Data Manipulation Language

**1. DATA DEFINITION LANGUAGE (DDL)**

It contains the commands used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project.

**a) Create table command :**

**Syntax**

```
CREATE TABLE table_name
(
column_name1 data_type(size),
column_name2 data_type(size),
.....
)
```

**Example 1**

This example demonstrates how you can create a table named "Person", with four columns. The column names will be "LastName", "FirstName", "Address", and "Age":

```
CREATE TABLE Person
( LastName varchar,
FirstName varchar,
Address varchar,
Age int )
```

This example demonstrates how you can specify a maximum length for some columns:

**Example 2**

**CREATE TABLE Person**
**(**
**LastName varchar(30),**
**FirstName varchar,**
**Address varchar,**
**Age int(3)**
**)**

**b) Insert Command:**

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

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

Syntax 1:

**INSERT INTO table_name (column1, column2, column3,…)**
**VALUES (value1, value2, value3, …)**

Example:
INSERT INTO Person (LastName, FirstName, Address, Age)
VALUES ('Sharma','Aryan','Delhi',25);

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

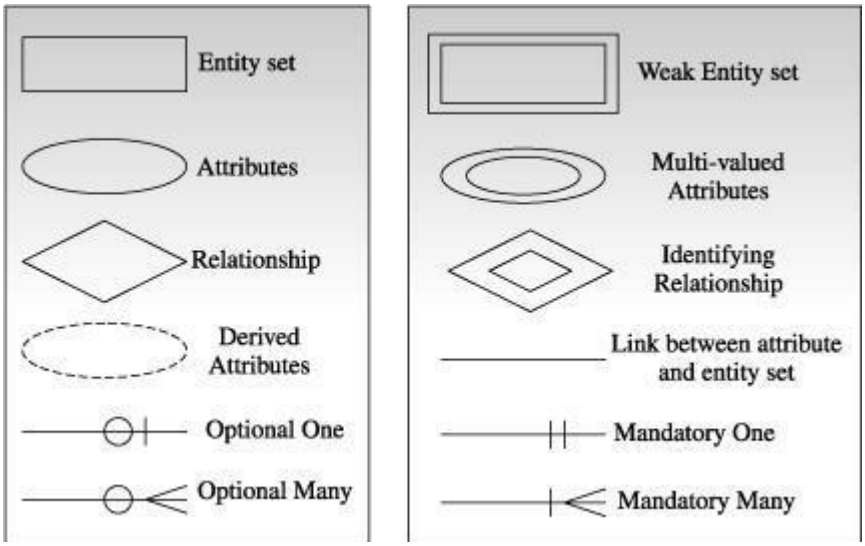
Syntax 2:
INSERT INTO table_name VALUES (value1, value2, value3, ...);
Example:
Insert INTO Person VALUES ('Sharma','Aryan','Delhi',25);

## Entity Relationship (ER) and Extended Entity Relationship (EER) Diagram

### Entity Relationship (ER) Diagram:

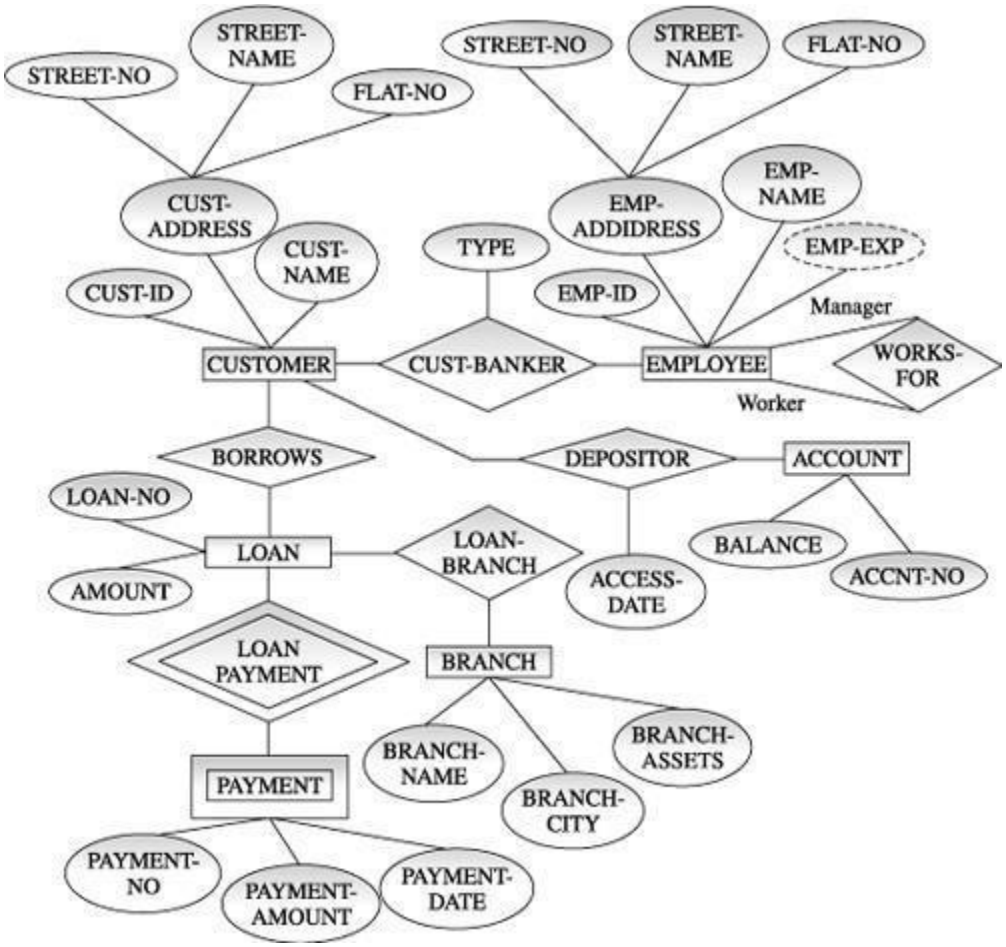
An entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is a component of data. In other words, ER diagrams illustrate the logical structure of databases. ER-Diagram is a visual representation of data that describes how data is related to each other.

### Basic Building Blocks of ER Diagram




**Sample ER Diagram**

Complete E-R diagram of banking organization database

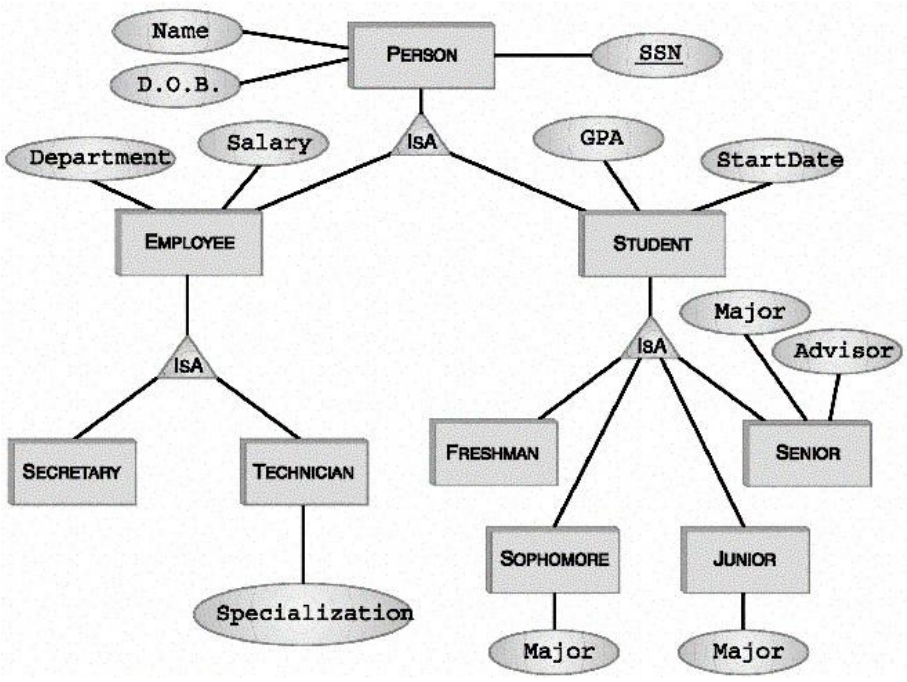


**Extended Entity Relationship**

The **enhanced entity–relationship** (**EER**) model (or **extended entity–relationship** model) in computer science is a high-level or conceptual data model incorporating extensions to the original entity–relationship (ER) model, used in the design of databases.

The Extended Entity-Relationship Model is a more complex and high-level model that extends an E-R diagram to include more types of abstraction, and to more clearly express constraints. All of the concepts contained within an E-R diagram are included in the EE-R model, along with additional concepts that cover more semantic information. These additional concepts include generalization/specialization, union, inheritance, and subclass/super class.

**Sample EER Diagram**




**Conclusion:** Performed implementation on database system design and draw an ER/EER diagram for the system successfully.

# Group B Assignment No.-2

**Aim:** Design and implement a database (for assignment no 1) using DDL statements and apply normalization on them.

**Objective:** Understand the Data Definition Language and commands to design Database.

**Theory:**

**DATA DEFINITION LANGUAGE (DDL)**: The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project.

Let's take a look at the structure and usage of four basic DDL commands:

1.CREATE 2. ALTER 3. DROP 4. RENAME

1. **CREATE:**

**CREATE TABLE**: This is used to create a new relation and thecorresponding

Syntax: CREATE TABLE relation_name (field_1 data_type(Size),field_2 data_type(Size), .. . );

Example: SQL>CREATE TABLE Student (sno NUMBER(3),sname CHAR(10),class CHAR(5));

**CREATE TABLE..AS SELECT**....: This is used to create the structure of anew relation from the structure of an existing relation.

Syntax: CREATE TABLE (relation_name_1, field_1,field_2,.....field_n) AS SELECT

field_1,field_2, ..... field_n FROM relation_name_2;

Example: SQL>CREATE TABLE std(rno,sname) AS SELECT sno,sname FROM student;

2. **ALTER :**

**ALTER TABLE ...ADD**...: This is used to add some extra fields into existing relation.

Syntax: ALTER TABLE relation_name ADD(new field_1 data_type(size), new field_2 data_type(size),..);

Example : SQL>ALTER TABLE std ADD(Address CHAR(10));

**ALTER TABLE...MODIFY...:** This is used to change the width as well as data type of fields of existing relations.

Syntax: ALTER TABLE relation_name MODIFY (field_1 newdata_type(Size), field_2 newdata_type(Size),... field_newdata_type(Size));

Example:SQL>ALTER TABLE student MODIFY(sname VARCHAR(10),class VARCHAR(5));

3. **DROP TABLE:** This is used to delete the structure of a relation. It permanently deletes the records in the table.

   Syntax: DROP TABLE relation_name;

   Example: SQL>DROP TABLE std;

4. **RENAME**: It is used to modify the name of the existing database object.

   4.1Syntax: RENAME TABLE old_relation_name TO new_relation_name;

   Example: SQL>RENAME TABLE std TO std1;

5. **TRUNCATE:** This command will remove the data permanently. But structure will not be removed.

   Syntax: TRUNCATE TABLE
   Example TRUNCATE TABLE student;

**Difference between Truncate & Delete:-**

- By using truncate command data will be removed permanently & will not get back where as by using delete command data will be removed temporally & get back by using roll back command.
- By using delete command data will be removed based on the condition where as by using truncate command there is no condition.
- Truncate is a DDL command & delete is a DML command.

**Conclusion :** Database is designed and all the DDL commands executed successfully.

# Group B Assignment No.-3

**Aim**: Create Table with primary key and foreign key constraints.

**Objective:** Understand the primary key and foreign key constraints.

**Theory:**

**What is a Primary Key?**

**Primary Key** is a column or group of columns in a table that uniquely identify every row in that table. The Primary Key can't be a duplicate meaning the same value can't appear more than once in the table. A table cannot have more than one primary key.

**Rules for defining Primary key:**

- Two rows can't have the same primary key value
- It must for every row to have a primary key value.
- The primary key field cannot be null.
- The value in a primary key column can never be modified or updated if any foreign key refers to that primary key.

**Example:**

1. In the following example, <code>ID</code> is a Primary Key.

```
CREATE TABLE student(
      ID NOT NULL,
      LastName varchar(255) NOT NULL,
      FirstName varchar(255),
      Age  int,
      PRIMARY KEY (ID)
   );
```

2. To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE student (
       ID int NOT NULL,
       LastName varchar(255) NOT NULL,
```

```
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_student PRIMARY KEY (ID,LastName)
);
```

3. To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

```
ALTER TABLE student
ADD CONSTRAINT PK_student PRIMARY KEY (ID,LastName);
```

4. To drop a PRIMARY KEY constraint, use the following SQL:

```
ALTER TABLE student
DROP CONSTRAINT PK_student;
```

**What is FOREIGN KEY ?**

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

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

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

To drop a FOREIGN KEY constraint, use the following SQL:

```
ALTER TABLE Orders
DROP FOREIGN KEY FK_PersonOrder;
```

**Conclusion :** Primary key and foreign key for the tables implemented successfully.

# Group B Assignment No.-4

**Aim:** Perform following SQL queries on the database created in assignment

1. Implementation of relational operators in SQL
2. Boolean operators and pattern matching
3. Arithmetic operations and built in functions
4. Group functions
5. Processing Date and Time functions
6. Complex queries and set operators

**Objective:** Understand the sql operations, functions and complex queries.

**Theory:**

The SQL Relational operators AND, OR and NOT 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;
```

**OR Syntax**

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

**NOT Syntax**

```
SELECT column1, column2,                                    ...
FROM table_name
WHERE NOT condition;
```

**Arithmatic OPERATORS**

SELECT 30 + 20;
SELECT 30 - 20;
SELECT 30 * 20;
SELECT 30 /20;
SELECT 30 %20;

**ORDER BY Claus**

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

**ORDER BY Syntax**

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

**MIN() and MAX()**

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;
```

MAX() Syntax

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

**SQL COUNT(), AVG() and SUM()**

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

**COUNT() Syntax**

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

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

**AVG() Syntax**

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

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

**SUM() Syntax**

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

**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

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

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

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

DATE & TIME FUNCTION

SQL> select sysdate from dual;

SYSDATE

---

07-APR-10

SQL> select round(sysdate)from dual;

ROUND(SYS

---

07-APR-10

SQL> select add_months(sysdate,3)from dual;

ADD_MONTH

---

07-JUL-10

SQL> select last_day(sysdate)from dual;

LAST_DAY(

---

30-APR-10

SQL> select sysdate+20 from dual;

SYSDATE+2

--------- 27-APR-10

SQL> select next_day(sysdate,'tuesday')from dual;

NEXT_DAY(

---

13- APR-10

**HAVING Clause**

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

HAVING Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

**UNION Operator**

The UNION operator is used to combine the result-set of two or more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

UNION Syntax

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

**Conclusion :** SQL Operators, functions, queries , commands executed successfully.

# Group B Assignment No.-5

**Aim:** Execute DDL/DML statements which demonstrate the use of views. Update the base table using its corresponding view. Also consider restrictions on updatable views and perform view creation from multiple tables.

**Objective:** Understand the concept of view and perform various operations on view

**Theory:**

**What is View?**

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.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

CREATE VIEW Syntax

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

SQL CREATE VIEW Examples

If you have the Northwind database you can see that it has several views installed by default.

The view "Current Product List" lists all active products (products that are not discontinued) from the "Products" table. The view is created with the following SQL:

```
CREATE VIEW [Current Product List] AS
SELECT ProductID, ProductName
FROM Products
WHERE Discontinued = No;
```

Then, we can query the view as follows:

```
SELECT * FROM [Current Product List];
```

### MySQL Create View with JOIN

CREATE VIEW command can be used along with a JOIN statement.

**Example :**
**Sample table : category**
**Sample table : purchase**

```
CREATE VIEW view_purchase
AS SELECT a.cate_id,a.cate_descrip, b.invoice_no,
b.invoice_dt,b.book_name
FROM category a,purchase b
WHERE a.cate_id=b.cate_id;
```

The above MySQL statement will create a view 'view_purchase' along with a JOIN statement.

The JOIN statement here retrieves cate_id, cate_descrip from category table and invoice_no, invoice_dt and book_name from purchase table if cate_id of category table and that of purchase are same.

### MySQL Create View with LIKE

CREATE VIEW command can be used with LIKE operator.

**Example :**
**Sample table : author**
**Code :**

```
1.  CREATE VIEW view_author
2.  AS SELECT *
3. FROM author
4. WHERE aut_name
5.  NOT LIKE 'T%' AND aut_name NOT LIKE 'W%';
```

The above MySQL statement will create a view 'view_author' taking all the records of author table, if (A)name of the author (aut_name) does not start with 'T' and (B) name of the author (aut_name) does not start with 'W'.

### MySQL Create View using Subquery

CREATE VIEW command can be used with subqueries.

**Example :**

**Sample table : purchase**
**Sample table : book_mast**
**Code :**

```
1. CREATE VIEW view_purchase
2. AS SELECT invoice_no,book_name,cate_id
3. FROM purchase
4. WHERE cate_id=  (SELECT cate_id FROM book_mast WHERE no_page=201)
5.
```

Create table Employee(ID,First_Name,Last_Name,Start_Date,End_Date,Salary,City).

1. Create a simple view to display First_Name,Last_Name from employee.
2. Create a view to display First_Name,Last_Name of those employee whose salary is greater than 2000 from employee table.
3. Create a view to display first_name starting with “S” and last_name end with “t”.

**Conclusion:** Implemented Views and performed operation on view

# Group C Assignment No.-1

**Aim:** Write and execute PL/SQL stored procedure and function to perform a suitable task on the database. Demonstrate its use.

**Objective:** 1) To understand the differences between procedure and function

2) To understand commands related to procedure and function

**Theory:**
A subprogram is a program unit/module that performs a particular task. These subprograms are combined to form larger programs. This is basically called the 'Modular design'. A subprogram can be invoked by another subprogram or program which is called the calling program.
A subprogram can be created:

- At schema level
- Inside a package
- Inside a MYSQL block

Parts of a MYSQL Subprog ram

Each MYSQL subprogram has a name, and may have a parameter list. Like anonymous PL/SQL blocks and, the named blocks a subprograms will also have following three parts:
1. Declarative Part
2. Executable part
3. Exception-handling

What is procedure? How to create it?

Procedures: these subprogram rams do not return a value directly, mainly used to perform an action.

Creating a Procedure
A procedure is created with the CREATE OR REPLACE PROCEDURE statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]

BEGIN
< procedure_body >
END ;
```

Where,
*procedure-name* specifies the name of the procedure.

[OR REPLACE] option allows modifying an existing procedure.

The optional parameter list contains name, mode and types of the parameters. IN represents, that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.

*Procedure-body* contains the executable part.
The IS keyword is used for creating a standalone procedure.
T he following example creates a simple procedure that displays the string 'Hello World!' on the screen when executed.

```
Delimiter //
CREATE OR REPLACE PROCEDURE greeting
Select concat('Hello World!');/
```

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

```
Query ok
```

(2) How to execute procedure?

Executing a Standalone Procedure

- Calling the name of the procedure from a PL/SQL block

```
Call greeting( )
```

```
Output:
Hello world
```

Deleting a Standalone Procedure
A standalone procedure is deleted with the DROP PROCEDURE statement. Syntax for deleting a procedure is:

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

So you can drop *greetings* procedure by using the following statement:

```
DROP PROCEDURE greetings;
```

Parameter modes in PL/SQL subprograms:
1. IN:
An IN parameter lets you pass a value to the subprogram.
It is a read-only parameter.

It is the default mode of parameter passing.
Parameters are passed by reference.

2. OUT:
An OUT parameter returns a value to the calling program.
The actual parameter must be variable and it is passed by value.

3. IN-OUT:
An IN OUT parameter passes an initial value to a subprogram and returns an updated value to the caller.
Actual parameter is passed by value.
IN & OUT Mode Example 1
T his program finds the minimum of two values, here procedure takes two numbers using IN mode and returns their minimum using OUT parameters.

```
delimiter $
create procedure addp()
begin
declare a,b,c int;
set a=2;
set b=3;
set c=a+b;
select concat('value',c);
end;
$
```

```
delimiter ;
call addp();
Result: value 5
```

```
mysql> delimiter //
mysql> create procedure difference (in a int,in b int, out c int)
    -> begin
    -> if a>b then
    -> set c=1;
    -> else if a=b then
    -> set c=2;
    -> else
    -> set c=3;
    -> end if;
```

```
mysql> call difference(5,9,@x);
    -> select @x;
    -> //
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> create table student
    -> ( sid int(5) not null,
    -> student_name varchar(9),
    -> DOB date,
    -> primary key(sid));
Query OK, 0 rows affected (0.06 sec)

mysql> insert into student values(5,'Harry',20130412);
Query OK, 1 row affected (0.03 sec)

mysql> insert into student values(6,'Jhon',20100215);
Query OK, 1 row affected (0.03 sec)

mysql> insert into student values(7,'Mary',20140516);
Query OK, 1 row affected (0.03 sec)

mysql> insert into student values(8,'Kay',20131116);
Query OK, 1 row affected (0.01 sec)

mysql> select * from student;
+___+__________+________+
| sid | student_name | DOB        |
+___+__________+________+
|   5 | Harry       | 2013-04-12 |
|   6 | Jhon        | 2010-02-15 |
|   7 | Mary        | 2014-05-16 |
|   8 | Kay         | 2013-11-16 |
+___+..........+..........+
```

**Q] Write a Procedure to display SID & Student.**

```
mysql> delimiter //
mysql> create procedure myprocedure()
    -> select sid,student_name from student
    -> //
Query OK, 0 rows affected (0.55 sec)

mysql> call myprocedure()//
+___+..........+
| sid | student_name |
+___+..........+
|   5 | Harry       |
|   6 | Jhon        |
|   7 | Mary        |
|   8 | Kay         |
```

Q] Write a procedure which gets the name of the student when the student id is passed ?

```
mysql> create procedure stud(IN id INT(5),OUT name varchar(9))
    -> begin
    -> select student_name into name
    -> from student
    -> where sid=id;
    -> end//
Query OK, 0 rows affected (0.01 sec)

mysql> call stud(5,@x)//
Query OK, 0 rows affected (0.00 sec)

mysql> select @x//
+_____+
| @x    |
+_____+
| Harry |
+_____+
1 row in set (0.00 sec)

mysql> call stud(7,@x)//
Query OK, 0 rows affected (0.00 sec)

mysql> select @x//
+____+
| @x |
+------+
| Mary |
+------+
1 row in set (0.00 sec)

mysql> call stud(5,@x);
    -> select @x;
    -> //
Query OK, 0 rows affected (0.00 sec)

+_____+
| @x    |
+_____+
| Harry |
+_____+
```

Q] Write a procedure cleanup() to delete all the students records from student table.

```
mysql> create procedure cleanup()
    -> delete from student;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> call cleanup()//
Query OK, 4 rows affected (0.03 sec)

mysql> select * from student;//
Empty set (0.00 sec)
```

---

2.*FUNCTIONS*

---

**Functions**: these subprograms return a single value, mainly used to compute and return a value.
**Creating a Function:**

A standalone function is created using the CREATE FUNCT ION statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows:

```
CREATE FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
BEGIN
< function_body >
RETURN variable
END [function_name];
```

Where,

*function-name* specifies the name of the function.
[OR REPLACE] option allows modifying an existing function.
The optional parameter list contains name, mode and types of the parameters. IN represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.
T he 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.

**Example:**

Following example illustrates creating and calling a standalone function. The function returns the total number of CUSTOMERS in the customers table. We will use the CUSTOMERS table.

```
delimiter &
mysql> create function hello(s char(20))
   -> returns char(50)
   -> return concat('hello,s,!');
   -> &
```

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

```
Query OK, 0 rows affected (0.01 sec)
```

**Calling a Function**

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

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

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

```
->select hello('world');
```

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

```
-> Hell  world
```

```
mysql> delimiter *
mysql> create function add1(a int, b int) returns int
   -> return (a+b);
   -> select add1(10,20);
   -> *
Query OK, 0 rows affected (0.00 sec)

+-------------+
| add1(10,20) |
+-------------+
```

```
|        30 |
+__________+
1 row in set (0.02 sec)
```

**Example:**

The following is one more example which demonstrates Declaring , Defining , and Invoking a Simple MYSQL Function that computes and returns the maximum of two values.

```
mysql> delimiter //
mysql> CREATE FUNCTION grt(a INT,b INT,c INT) RETURNS INT
   -> BEGIN
   -> if a>b AND a>c then
   -> RETURN a;
   -> end if;
   -> if b>c AND b>a then
   -> RETURN b;
   -> end if;
   -> RETURN c;
   -> end;
   -> //
Query OK, 0 rows affected (0.12 sec)

mysql> select grt(23,78,98);
   -> //
+___________+
| grt(23,78,98) |
+---------------+
|          98 |
+---------------+
1 row in set (0.05 sec)

mysql> select grt(23,98,72);
   -> //
+___________+
| grt(23,98,72) |
+---------------+
|          98 |
+---------------+
1 row in set (0.01 sec)

mysql> select grt(45,2,3); //
+-------------+
| grt(45,2,3) |
+_____________+
|          45 |
```

```
+_________+
1 row in set (0.00 sec)

mysql> delimiter //
mysql> CREATE FUNCTION odd_even(a INT) RETURNS varchar(20)
    -> BEGIN
    -> if a%2=0 then
    -> RETURN 'even';
    -> end if;
    -> RETURN 'odd';
    -> end;
    -> //
Query OK, 0 rows affected (0.06 sec)

mysql> select odd_even(54);
    -> //
+-------------+
| odd_even(54) |
+_____________+
| even         |
+_____________+
1 row in set (0.03 sec)

mysql> select odd_even(51); //
+_____________+
| odd_even(51) |
+_____________+
| odd          |
+-------------+
1 row in set (0.00 sec)
```

**Conclusion:** Performed implementation of procedures and functions in MYSQL successfully.

# /Group C Assignment No.-2

**Aim:** Write and execute suitable database triggers . Consider row level and statement level triggers.

**Objectives:** To understand the concept of database MYSQL Trigger

**Theory:**

1) Introduction to MYSQL Trigger

***What is a Trigger?***

A trigger is a MYSQL block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed.

2) **Types of triggers**

### I) Types of Triggers

There are two types of triggers based on the which level it is triggered.

**1) Row level trigger** - An event is triggered for each row upated, inserted or deleted.
**2) Statement level trigger** - An event is triggered for each sql statement executed.

### Trigger Execution Hierarchy

The following hierarchy is followed when a trigger is fired.

**1)** BEFORE statement trigger fires first.
**2)** Next BEFORE row level trigger fires, once for each row affected.
**3)** Then AFTER row level trigger fires once for each affected row. This events will alternates between BEFORE and AFTER row level triggers.
**4)** Finally the AFTER statement level trigger fires.

### Syntax of Triggers

The Syntax for creating a trigger is:

*CREATE TRIGGER trigger_name*
*{BEFORE | AFTER | }*
*{INSERT [OR] | UPDATE [OR] | DELETE}*
*[OF col_name]*
*ON table_name*
*[REFERENCING OLD AS o NEW AS n]*

*[FOR EACH ROW]*
*WHEN (condition)*
*BEGIN*
*--- sql statements*
*END;*

- *CREATE TRIGGER trigger_name* - This clause creates a trigger with the given name or overwrites an existing trigger with the same name.
- *{BEFORE | AFTER | INSTEAD OF }* - This clause indicates at what time should the trigger get fired. i.e for example: before or after updating a table. INSTEAD OF is used to create a trigger on a view. before and after cannot be used to create a trigger on a view.
- *{INSERT [OR] | UPDATE [OR] | DELETE}* - This clause determines the triggering event. More than one triggering events can be used together separated by OR keyword. The trigger gets fired at all the specified triggering event.
- *[OF col_name]* - This clause is used with update triggers. This clause is used when you want to trigger an event only when a specific column is updated.
- *CREATE [OR REPLACE ] TRIGGER trigger_name* - This clause creates a trigger with the given name or overwrites an existing trigger with the same name.
- *[ON table_name]* - This clause identifies the name of the table or view to which the trigger is associated.
- *[REFERENCING OLD AS o NEW AS n]* - This clause is used to reference the old and new values of the data being changed. By default, you reference the values as :old.column_name or :new.column_name. The reference names can also be changed from old (or new) to any other user-defined name. You cannot reference old values when inserting a record, or new values when deleting a record, because they do not exist.
- *[FOR EACH ROW]* - This clause is used to determine whether a trigger must fire when each row gets affected ( i.e. a Row Level Trigger) or just once when the entire sql statement is executed(i.e.statement level Trigger).
- *WHEN (condition)* - This clause is valid only for row level triggers. The trigger is fired only for rows that satisfy the condition specified

**Trigger Examples**

**Example 1)**

This example is based on the following two tables:

```
CREATE  TABLE  T4 ( a  INTEGER ,  b  CHAR(10));

CREATE  TABLE  T5 ( c  CHAR(10) ,  d  INTEGER);
```

-- create a trigger that may insert a tuple into T5 when a tuple is inserted into T4. inserts the reverse tuple into T5:

1) Create trigger as follows:

```
CREATE TRIGGER trig1 AFTER INSERT ON T4

 FOR EACH ROW BEGIN

INSERT INTO t5 SET c = NEW.b,d = NEW.a;

    END;
```

2) Insert values in T4.

3) Check the values in T5.

---

**Example2)**

1)The price of a product changes constantly. It is important to maintain the history of the prices of the products. Create a trigger to update the 'product_price_history' table when the price of the product is updated in the 'product' table.

Create the 'product' table and 'product_price_history' table

```
CREATE TABLE product_price_history
(product_id number(5),
product_name varchar2(32),
supplier_name varchar2(32),
unit_price number(7,2) );

CREATE TABLE product
```

```
(product_id number(5),
product_name varchar2(32),
supplier_name varchar2(32),
unit_price number(7,2) );
drop trigger if exists price_history_trigger;

CREATE TRIGGER price_history_trigger

BEFORE UPDATE on product1

FOR EACH ROW BEGIN

INSERT INTO product_price_history

set product_id=old.product_id,

product_name=old.product_name,

supplier_name=old.supplier_name,

unit_price=old.unit_price;

END
```

**3)** Lets update the price of a product.

```
UPDATE PRODUCT SET unit_price = 800 WHERE product_id = 100
```

Once the above update query is executed, the trigger fires and updates the 'product_price_history' table.

---

Example 3

```
create table account(accno int,amount int)
```

Create a trigger on account table before update in new inserted amount is less than “0” then set amount “0” else if amount is greater than 100 then set amount 100

```
CREATE TRIGGER upd_check BEFORE UPDATE ON account

FOR EACH ROW

BEGIN

 IF NEW.amount < 0 THEN

SET NEW.amount = 0;
```

```
ELSEIF NEW.amount > 100 THEN

  SET NEW.amount = 100;

   END IF;

END
```

```
update account set amount= -12 where accno=101
```

Deleting a trigger
DROP TRIGGER

**Name**
DROP TRIGGER -- Removes a trigger definition from a database.

**Synopsis**
DROP TRIGGER *name* ON *table*

**Parameters**
*name*

The name of the trigger you wish to remove.

*table*
The name of the table the trigger is on.

**Results**

**Conclusion:** Studied and implemented MYSQL Trigger

**FAQs:-**

1) What is trigger and cursor in PL/SQL?
2) What are the types of trigger and cursor?
3) How to delete a trigger?
4) Why we write a cerate or replace in PL/SQL Block?
5) What is row level and statement level trigger?

# Group C Assignment No.-3

**Aim:** Write a PL/SQL block to implement all types of cursor.

**Objective:** 1] to understand the basic concept of cursors used in PL/SQL

**Theory:**

1] Cursor its use:

- A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor.
- 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.

2] Types of cursors:

- **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.

Whenever a DML statement (INSERT , UPDAT E and DELET E) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to beinserted. For UPDAT E and

DELET E operations, the cursor identifies the rows that would be affected.

| Attribute | Desc ription |
|---|---|
| %FOUND | Returns T RUE if an INSERT , UPDAT E, or DELET E statement affected one or more rows or a SELECT INT O statement returned one or more rows. Otherwise, it returns FALSE. |
| %NOT FOUND | T he logical opposite of %FOUND. It returns T RUE if an INSERT , UPDAT E, or DELET E statement affected no rows, or a SELECT INT O statement returned no rows. Otherwise, it returns FALSE. |
| %ISOPEN | Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement. |
| %ROWCOUNT | Returns the number of rows affected by an INSERT , UPDAT E, or DELET E statement, or returned by a SELECT INT O statement. |

- **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.
T he syntax for creating an explicit cursor is :
CURSOR cursor_name IS select_statement;

Working with an explicit cursor involves four steps:
Declaring the cursor for initializing in the memory
Opening the cursor for allocating memory
Fetching the cursor for retrieving data
Closing the cursor to release allocated memory

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;
```

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

```
OPEN c_customers;
```

Fetching the Cursor
Fetching the cursor involves accessing one row at a time. For example we will fetch rows from the aboveopened
cursor as follows:

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

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

```
CLOSE c_customers;
```

**Cursor Example**

**Example 1**

Create table emp_tbl as follows

Emp_tbl(first_name,last_name,salary);

Write a procedure with cursor to display employees first name and last name whose salary greater than 1000;

```
drop procedure if exists pcursor;

create procedure  pcursor()

begin

DECLARE fn varchar(30);

declare ln varchar(30);

DECLARE cur1 CURSOR FOR SELECT first_name,last_name from emp_tbl where salary>1000;

OPEN cur1;

read_loop: LOOP

   FETCH cur1 INTO fn,ln;

select concat(fn,' ',ln) as name;

end loop;

CLOSE  cur1;

END
```

**Example 2**

```
create table t1(id int,data int);( id char(16),data int)

create table t2(i int);

create table t3(i1 int,12 int);     //t3 table blank (i1 char(16),i2 int)

CREATE PROCEDURE curdemo()
BEGIN
```

```
DECLARE done INT DEFAULT FALSE;
DECLARE a CHAR(16);
DECLARE b, c INT;
DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
OPEN cur1;
OPEN cur2;
read_loop: LOOP
   FETCH cur1 INTO a, b;
   FETCH cur2 INTO c;
  IF b < c THEN
 INSERT INTO test.t3 VALUES (a,b);
   ELSE
     INSERT INTO test.t3 VALUES (a,c);
  END IF;
 END  LOOP;
CLOSE  cur1;
CLOSE  cur2;
END;
```

**FAQs:**

1) What is cursor? State its type.
2) Explain difference between implicit cursor and explicit cursors.

**Conclusion:** Thoroughly understood the basic concept of cursors used in PL/SQL.