ASSIGNMENT NO: 1

AIM : Design an E-R diagram for a library management system to handle books, members, and transactions.

Entities:

Book: Attributes include Book_ID, Title, Author, Publisher, Year_Published.

Member: Attributes include Member_ID, Name, Membership_Date, Address.

Transaction: Attributes include Transaction_ID, Issue_Date, Return_Date.

Relationships:

Borrows: Links Members and Books with attributes Issue_Date and Return_Date.

INDEX TERMS: Database Management System, ER Data Model, MySQL

THEORY

Entity-Relationship Data Model :

ER Data Model is based on real world objects and their relationship. In a database, we would be grouping only related data together and storing them under one group name called Entity.

An attribute is a list of all related information of an entity, which has valid value.

Keys are the attributes of the entity, which uniquely identifies the record of the entity.

A relationship defines how two or more entities are interrelated.

Since the ER diagram is the pictorial representation of real world objects, it involves various symbols and notation to draw the diagrams. Let us see one by one below.

Entity: Rectangles are used to represent the entity in the diagram. Name of the Entity is written inside the rectangle.

A strong entity is represented by a simple rectangle as shown above. A weak entity is represented by two rectangles as shown below.

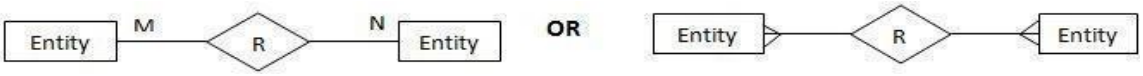
Attribute: An oval shape is used to represent the attributes. Name of the attribute is written inside the oval shape and is connected to its entity by a line. Multivalued attributes are represented by double oval shape; whereas derived attributes are represented by oval shape with dashed lines. A composite attribute is also represented by an oval shape, but these attributes will be connected to its parent attribute forming a tree structure.

Cardinality of Relationship: Different developers use different notation to represent the cardinality of the relationship. Not only for cardinality, but for other objects in ER diagrams will have slightly different notations. But the main difference is noticed in the cardinality. For not to get confused with many, let us see two types of notations for each.

One-to-one relation: - A one-to-one relationship is represented by adding '1' near the entities on the line joining the relation. In another type of notation one dash is added to the relationship line at both ends.

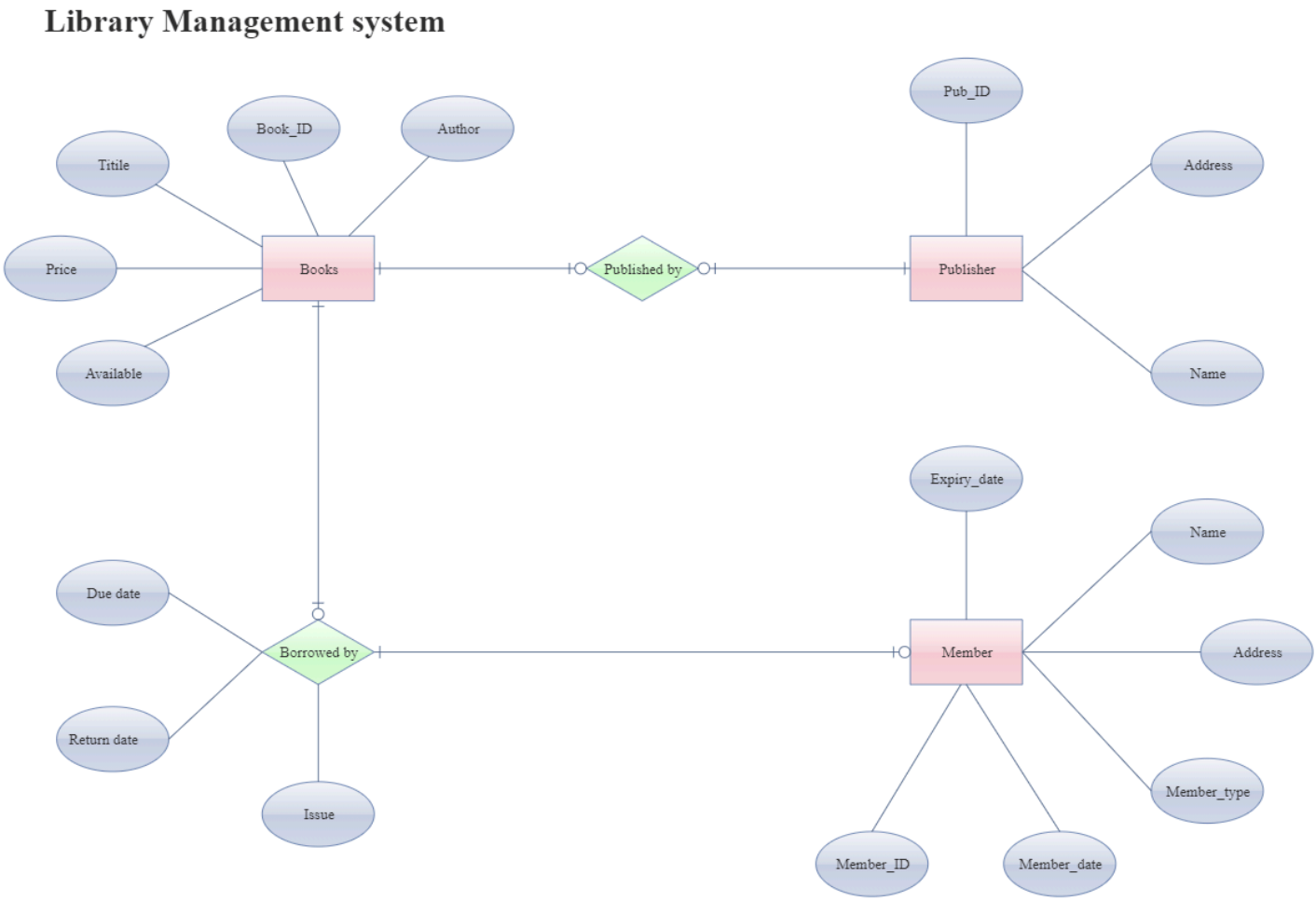
One-to-Many relation: A one-to-many relationship is represented by adding '1' near the entity at the left hand side of relation and 'N' is written near the entity at the right side. Other types of notation will have dash at LHS of relation and three arrow kind of lines at the RHS of relation as shown below.

Many-to-Many relation: A one-to-many relationship is represented by adding 'M' near the entity at left hand side of relation and 'N' is written near the entity at right side. Other type of notation will have three arrow kinds of lines at both sides of relation as shown below.



Participation Constraints: Total participation constraints are shown by double lines and partial participations are shown as single lines.

E-R Diagram for Library management System -



What is Database?

A database is a separate application that stores a collection of data.Each database has one or more distinct APIs for creating, accessing, managing, searching and replicating the data it holds. So nowadays, we use relational database management systems (RDBMS) to store and manage huge volumes of data. This is called relational database because all the data is stored into different tables and Relations are established using primary keys or other keys known as foreign keys

The conversion of ER diagram into Table structure at this would be as below:

MySQL:

MySQL, the most popular Open Source SQL database management system, is developed, distributed, and supported by Oracle Corporation.

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. Since computers are very good at handling large amounts of data, database management systems play a central role in computing, as standalone utilities, or as parts of other applications.

MySQL databases are relational. A relational database stores data in separate tables rather than putting all the data in one big storeroom. The database structures are organized into physical files optimized for speed. The logical model, with objects such as databases, tables, views, rows, and columns, offers a flexible programming environment. 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. The MySQL software uses the GPL.

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. If you dedicate an entire machine to MySQL, you can adjust the settings to take advantage of all the memory, CPU power, and I/O capacity available. MySQL can also scale up to clusters of machines, networked together.

- **MySQL Installation Steps on Windows:**

1) Go to the official MySQL Downloads page.

2) Click on "MySQL Installer for Windows".

3) Choose either the "Web Community" (online) or "Full" (offline) version of the installer. Click on "Download".

- **Run the Installer:**

4) Locate the downloaded installer file and double-click it to run.
5) If prompted by User Account Control, click "Yes" to allow the installer to make changes to your device.

- **Choose Setup Type:**

6) You will be presented with several setup types. Choose one based on your needs:
    a. **Developer Default:** Includes MySQL Server, MySQL Shell, MySQL Workbench, and other tools.
    b. **Server Only:** Only installs MySQL Server.
    c. **Client Only:** Only installs client tools, such as MySQL Workbench.
    d. **Full:** Installs all available MySQL products.
    e. **Custom:** Allows you to choose which components to install.
7) Click "Next" after making your selection.

- **Check Requirements:**

8) The installer will check for any required software. If any dependencies are missing, it will prompt you to install them. Click "Execute" to install any required software.

- **Installation Steps:**

9)Once all requirements are met, click "Next" to proceed.

10)Click "Execute" to begin the installation of the selected MySQL products.

9) Case Study: A. Draw an ER diagram for the following application from the manufacturing industry:

10) Each supplier has a unique name.

11) More than one supplier can be located in the same city.

12) Each part has a unique part number.

13) Each part has a color.

14) A supplier can supply more than one part.

15) A part can be supplied by more than one supplier.

16)

17) Payments: stores payments made by customers based on their accounts.

18) Employees: stores all employee information as well as the organization structure such as who reports to whom.

19) Offices: stores sales office data.

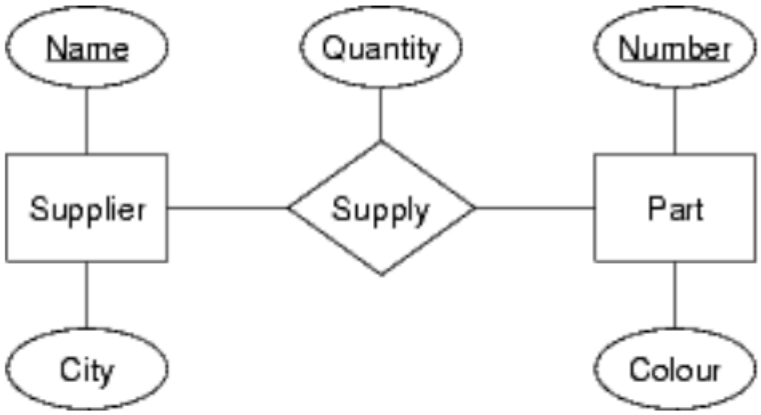


Figure 1: ER-diagram for supplier-part

20)

21)

22)

23) Draw an ER diagram for the following application from the ABC Company:

24) Employees work for many projects and each project has many employees

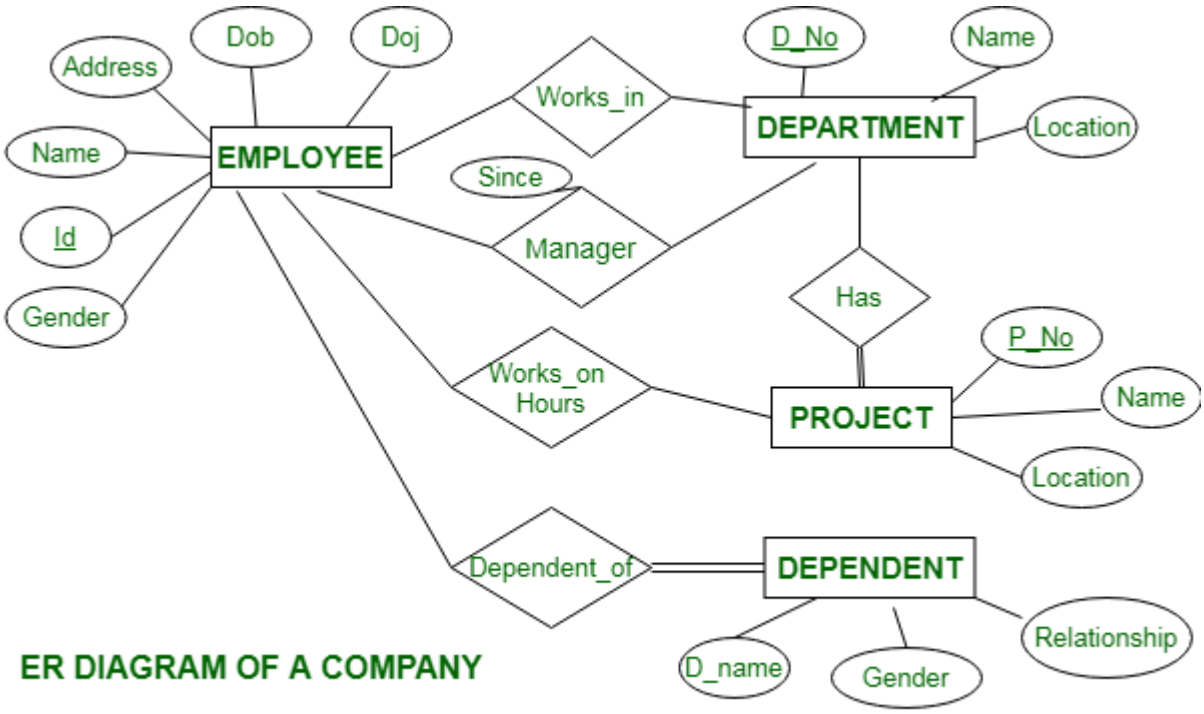
25) Each employee has an unique Emp_No

26) Each employee has a name and name consists of first name, middle name and last name

27) Each project has an unique number and name

28)

29)



ER DIAGRAM OF A COMPANY

30)

31)

32)

33)

34)

35)

36)

37)

38) Draw an ER diagram for the following application from the hospital:

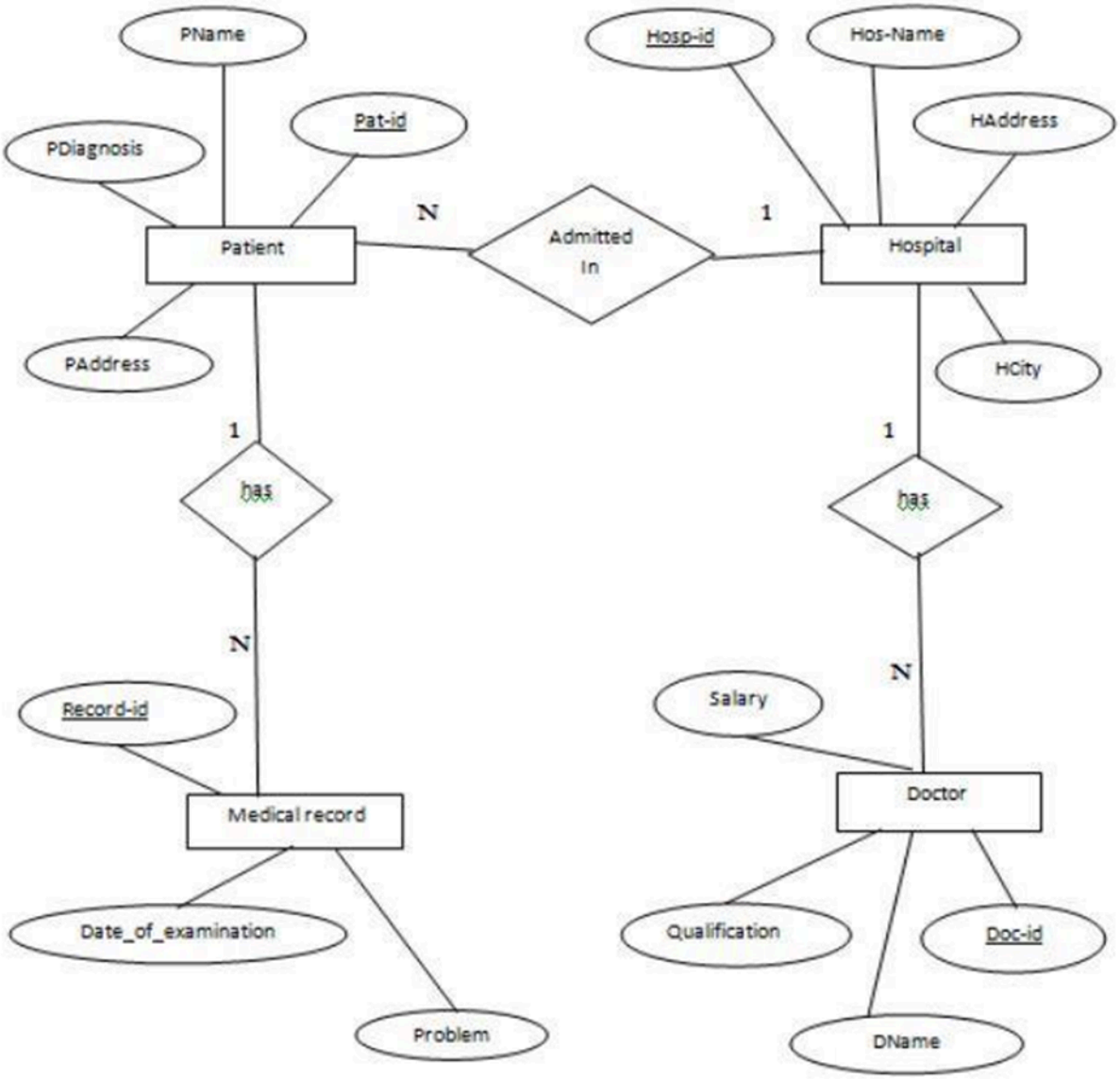
39) A doctor has one or more patients to treat

40) Each doctor has an unique Doctor ID

41) Each patient has a name, phone number, address and date of birth

42) Patient entity is a weak entity

43) Age is a derived attribute



44)

45)

46)

47)

48) Conclusion:

49) Purpose of this assignment is fulfilled by understanding ER Data Model,

50) Relational Databases and MySQL Database Management Systems.

- **Exercise –**

  1) Draw an E-R diagram for an online retail store. The store sells products. Each product belongs to a category, and a product can have multiple variations (like color or size). Customers place orders, and each order can contain multiple products. Each customer can place multiple orders.

  2) Draw an E-R diagram for a real estate agency. The agency lists properties for sale. Each property has an owner, and each owner can own multiple properties. Buyers can express interest in multiple properties. Each property can have multiple interested buyers. Agents facilitate the sale of properties.

  3) Construct an E-R diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.

  4) Explain different types of attributes with the help of notions used to represent it.

FAQs:

1. What are the advantages of DBMS over an additional file system?
2. Explain the terms table and record in a database.

3.Find out databases used for following applications.

A. Twitter :____B. Facebook :_____C. Amazon / Flipkart : _____
D. AADHAR Card: _____

ASSIGNMENT NO: 2

AIM : Design Database schema and implement following DDL commands of SQL with suitable examples

1) Create table

2)Alter table

3) Drop Table

4)Truncate Table

5)Rename Table

Create Database and Tables:

Create a database named SchoolDB.
Within SchoolDB, create two tables: Students and Courses.
Students should have columns for StudentID, StudentName, and Major.
Courses should have columns for CourseID, StudentID, CourseName, and Credits.
Modify Table Structure:

Add a column Email to the Students table.
Change the Credits column in Courses table to allow null values.

INDEX TERMS: DDL,attribute, null value

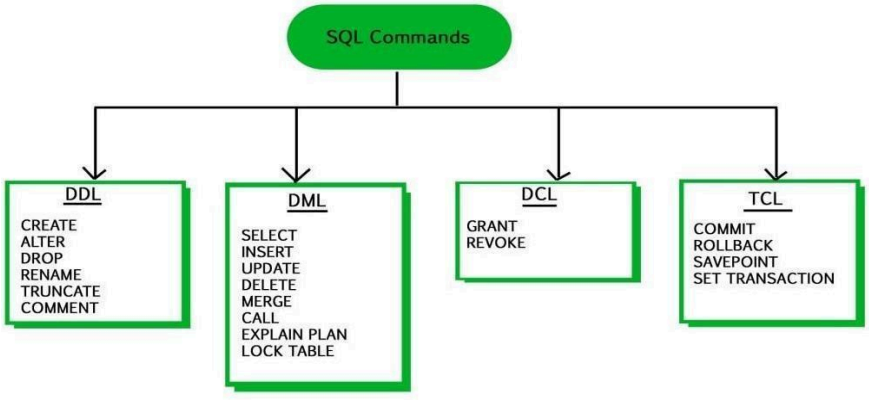
THEORY

Different types of commands in SQL:

A). DDL commands: - To create a database objects

B). DML commands: - To manipulate data of a database objects. C). DQL command: - To retrieve the data from a database.

D). DCL command - To control the data of a database.



- **DDL commands:**

The DDL part of SQL permits database tables to be created or deleted. It also define indexes (keys), specify links between tables, and impose constraints between tables. The schema for each relation. The domain of values associated with each attribute. Integrity constraints And as we will see later, also other information such as The set of indices to be maintained for each relations.

These commands are:

--**CREATE**: to create a new data base schema

– **ALTER**: to change an existing data base schema

--**DROP**: to remove an entire data schema.

–**TRUNCATE** : to remove all rows from table.

–**RENAME** : to rename existing table

1) Create Database

The CREATE DATABASE query is used to create a new database in the database management system. It is also used in MySQL and other relational database management systems (RDBMS) to create databases.

Syntax:

The syntax to use the CREATE DATABASE command is:

CREATE DATABASE database_name;

Example:

```
-- Create the SchoolDB database if it doesn't exist

CREATE DATABASE IF NOT EXISTS SchoolDB;

-- Use the SchoolDB database to work within it

USE SchoolDB;
```

2) Create Table

CREATE TABLE Statement is used to create a new table in a database. Users can define the table structure by specifying the column's name and data type in the CREATE TABLE command.

This statement also allows us to create tables with constraints that define the rules for the table.

Users can create tables in SQL and insert data at the time of table creation.

Syntax-

```
CREATE table table_name
(
Column1 datatype (size),
column2 datatype (size),
.
.
columnN datatype(size)
);
```

Example-

```
-- Create the Students table
CREATE TABLE Students (
    StudentID INT PRIMARY KEY AUTO_INCREMENT,
    StudentName VARCHAR(100) NOT NULL,
    Major VARCHAR(50) NOT NULL,
    Email VARCHAR(100) -- Added later
);


-- Create the Courses table
CREATE TABLE Courses (
    CourseID INT PRIMARY KEY AUTO_INCREMENT,
    StudentID INT,
    CourseName VARCHAR(100) NOT NULL,
    Credits INT, -- Changed later to allow NULL values
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID)
);
```

3) Alter Table –

ALTER is a DDL command which changes or modifies the existing structure of the database, and it also changes the schema of database objects. We can add new attributes or can change the data type/size of the existing attribute.

Syntax-

```
Alter Table name_of_table ADD column_name datatype (size) ;
```

Alter Table name_of_table modify column_name new_datatype(new size);

Example-

Adding Email column to Students table

ALTER TABLE Students

ADD Email VARCHAR(100);

Changing Credits column in Courses table to allow null values

ALTER TABLE Courses

MODIFY COLUMN Credits INT NULL;

4)Truncate Table –

To truncate tables , you use the **TRUNCATE TABLE** command. This command removes all rows from a table efficiently, but it keeps the table structure intact (including column definitions, constraints, etc.).

Syntax:

TRUNCATE TABLE Table_Name;

Example

-- Truncate the Students table

TRUNCATE TABLE Students;

-- Truncate the Courses table

TRUNCATE TABLE Courses;

Explanation:

TRUNCATE TABLE Students: This command removes all rows from the Students table. After truncation, the table will be empty, but its structure (columns, constraints, etc.) remains unchanged.

TRUNCATE TABLE Courses: Similarly, this command removes all rows from the Courses table. It clears out all course records but leaves the table structure intact.

5) Drop Table –

To drop tables, you use the **DROP TABLE** command. Unlike **TRUNCATE TABLE**, which only removes the data while keeping the table structure intact, **DROP TABLE** completely removes both the table and all its data.

Syntax-

Drop table tablename;

Example-

-- Drop the Courses table

DROP TABLE IF EXISTS Courses;

-- Drop the Students table

DROP TABLE IF EXISTS Students;

Explanation:

DROP TABLE Courses: This command removes the Courses table from the database. If the table does not exist (IF EXISTS clause), it prevents an error from occurring.

DROP TABLE Students: Similarly, this command removes the Students table from the database. The IF EXISTS clause ensures that if the table doesn't exist, the command will not cause an error.

6) Rename table

Rename is a DDL command which is used to change the name of the database table.

Syntax -

Rename table Old_Table_Name TO New_Table_Name;

Example-

Rename table Students to stud10;

Conclusion:

Out come of the experiment is understanding of DDL commands and integrity constraints.

Exercise --

1) **Write a SQL statement to create a table named countries including columns** country_id,country_name and region_id and make sure that no duplicate data against column country_id will be allowed at the time of insertion.

2) **Write the SQL DDL command to create a table named Publishers with the following columns:**
PublisherID (Primary Key, Integer, Auto Increment)
Name (VARCHAR, Not Null)
Address (VARCHAR, Not Null)
Phone (VARCHAR)
Write the SQL DDL command to add a unique constraint on the Name column of the Publishers table.
Write the SQL DDL command to add a new column ISBN (VARCHAR, Not Null, Unique) to the Publishers table.

3) Write the SQL DDL command to create a table named Employees with the following columns:

EmployeeID (Primary Key, Integer, Auto Increment)
FirstName (VARCHAR, Not Null)
LastName (VARCHAR, Not Null)
BirthDate (DATE, Not Null)
HireDate (DATE, Not Null)
Salary (DECIMAL, Not Null)

Write the SQL DDL command to add a unique constraint on the Email column of the Members table.

Write the SQL DDL command to add a new column Department (VARCHAR, Not Null) to the Employees table.
Write the SQL DDL command to drop the Salary column from the Employees table.
Write the SQL DDL command to rename the HireDate column in the Employees table to StartDate.

4) **Create Tables as follows by choosing appropriate data type and set the necessary primary and foreign key constraints:**
Customer (Custid, Custname, Addr, phno,panno)
Loan (Loanid, Amount, Interest,Custid)
Account (Accd, Accbal, Custid)
. Add a column CUSDOB in customer table

FAQs:

1) How do I modify an existing table using the ALTER command
2) What are constraints, and how do I add them to a table?
3) What is a composite primary key, and how do I define one?
4) How do I drop a constraint from a table?
5) What is the TRUNCATE command, and how does it differ from DROP?

ASSIGNMENT NO: 3

AIM : Apply following Integrity Constraints on Database.

1) Not Null Constraint

2)Check Constraint

3)Primary Key and Unique Constraint

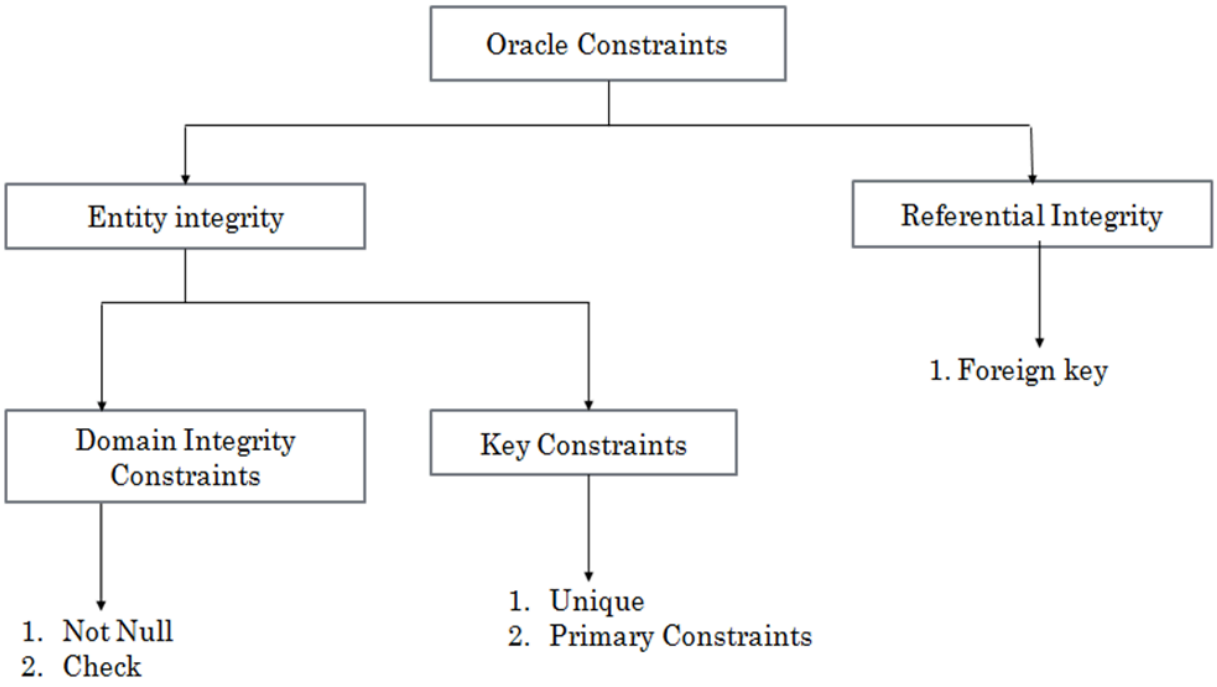
4)Foreign Key Constraint

INDEX TERMS: DDL, Integrity Constraint

THEORY -

Integrity constraints are a set of rules. It is used to maintain the quality of information. o Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected. o Thus, integrity constraint is used to guard against accidental damage to the database.

Types of Integrity Constraint -



1) Domain Integrity constraints: Not null, Check

a) NOT NULL Constraints -

There may be Records in table that do not contain any value for some fields. In Oracle, Null values are stored in such fields. In other words, a NULL value represents an empty field.

.

A NULL value indicates 'not applicable', 'missing', or 'not known'.
A NULL value distinct from zero or other numeric value for numerical data.
A NULL value is also distinct from blank space for character data.
A NULL value will evaluate to null in any expression.
The result of any condition including null value is unknown, and treated as a FALSE.

A column, defined as a NOT NULL, cannot have a NULL value. In other words, such a column becomes a mandatory column and cannot be left empty for any record.

.

A NOT NULL CONSTRAINT DEFINED AT COLUMN LEVEL only

Syntax :

ColumnName datatype (size) NOT NULL

Example:

CREATE TABLE EMPLOYEES (EMPID INTEGER NOT NULL, EName VARCHAR2(10) NOT NULL, DOJ DATE);

b) CHECK constraints

"The CHECK constraint is used to implement conditions on attributes. The record which satisfies the given conditions are only inserted in the table.
The CHECK constraint is bound to a particular column.
Once a CHECK constraint is implemented, any insert or update operation on that table must follow this constraints.
If any operation violates the condition, it will be rejected.

A CHECK CONSTRAINTS DEFINED AT COLUMN LEVEL :

Syntax:
columnName datatype (size) CHECK (CONDITION)

B CHECK CONSTRAINTS DEFINED AT TABLE LEVEL :

Syntax:
CHECK(CONDITION)
Example:

CREATE TABLE Orders (order_id number(10),amount number(10) CHECK (amount > 0));

2) Entity Integrity constraints: Unique, Primary key.

Unique Constraints

· A column must have unique values. This is required to identify all records stored in table uniqly,

A column, defined as a UNIQUE, cannot have duplicate values across all records. In other words, such columns must contain unique values.

A UNIQUE CONSTRAINTS DEFINED AT COLUMN LEVEL :

Syntax:
columnName datatype (size) UNIQUE

A UNIQUE CONSTRAINTS DEFINED AT TABLE LEVEL :

Syntax:
UNIQUE (columnName [, columnName …])

Example:

CREATE TABLE Colleges (college_id number(5) UNIQUE,
college_code VARCHAR(20) UNIQUE, college_name VARCHAR(50)
);

Primary Key Constraints

· A primary key is a set of one or more columns used to identify each record uniquely in a column". A single column primary key is called as simple key, while a multi-column primary key is called a composite key

A column defined as a primary key, cannot have duplicate values across all records and cannot have NULL values.

A PRIMARY KEY CONSTRAINTS DEFINED AT COLUMN LEVEL :

Syntax:
columnName datatype (size) PRIMARY KEY

A PRIMARY KEY CONSTRAINTS DEFINED AT TABLE LEVEL :

Syntax:
PRIMARY KEY (columnName [, columnName …])

3) Referential Integrity constraints: Foreign key, referenced key, on delete cascade

Foreign Key Constraints

A Foreign key is a set of one or more columns whose values are derived from the primary key or unique key of another table.

· The table, in which a foreign key is defined, is called a foreign table, detail table or child table.
The table in which primary key or unique key is referred, is called a primary table, master table or parent table.

The foreign key constraints enforce different restrictions on the detail table and master table.

If bname is defined as a foreign key in Account table referring to bname in branch table, then, there will be following restriction on both of these tables.

1. Restriction on detail table:
- Detail table contains a Foreign key. And, it is related to master table.
- Insert or update operation involving value of Foreign key are not allowed, if corresponding value does not exist in the master table.

2. Restriction on master table:
- Master table contains a primary key or unique key, which is referred by Foreign key in detail.

- Delete or update operation on records in master table are not allowed, if corresponding records are present in detail table

A FOREIGN KEY CONSTRAINTS DEFINED AT COLUMN LEVEL:

syntax :

columnName datatype (size)
REFERENCES table name (columnName) [ON DELETE CASCADE]

A FOREIGN KEY CONSTRAINTS DEFINED AT TABLE LEVEL:

syntax :

FOREIGN KEY (columnName [, columnName …]) REFERENCES tablename (columnName [, columnName …])

Example:

CREATE TABLE Orders (OrderID number(10), OrderNumber number(10), PersonID number(10), FOREIGN KEY (PersonID) REFERENCES Persons(PersonID));

Conclusion -

Application of SQL integrity constraints is fundamental in designing reliable databases. By enforcing rules on data entry and maintaining relationships between tables, constraints ensure that the database remains consistent, accurate, and reliable.

Exercise Questions -

1)Write a query to insert a new customer into the Customers table. Ensure that the Email is unique and does not duplicate an existing email.

2) Consider the schema: employee(employee-name, street, city)

works(employee-name, company-name, salary)

company(company-name, city)

manages(employee-name, manager-name)

Give an SQL DDL definition for the tables of this database. Identify referential integrity constraints that should hold and include them in the DDL definition.

3)Write an SQL statement to create an Orders table that ensures the CustomerID must exist in the Customers table.

4)Write an SQL statement to create a `Customers` table that enforces unique email addresses.

5) Consider following database schemas -

Customers:

- CustomerID (Primary Key)
- FirstName
- LastName
- Email (Unique)
- PhoneNumber

2. Products:
   - ProductID (Primary Key)
   - ProductName
   - Price
   - StockQuantity (Must be non-negative)
3. Orders:
   - OrderID (Primary Key)
   - OrderDate
   - CustomerID (Foreign Key referencing Customers.CustomerID)
   - TotalAmount
4. **OrderDetails**:

- OrderDetailID (Primary Key)
- OrderID (Foreign Key referencing Orders.OrderID)
- ProductID (Foreign Key referencing Products.ProductID)
- Quantity (Must be greater than 0)
- Price

Questions:

1. **Scenario 1**: A new product is added to the inventory with a negative stock quantity by mistake. Describe the constraint that will prevent this, and write the SQL statement that ensures the stock quantity cannot be negative.
2. **Scenario 2**: An order is placed by a customer, but the customer ID entered does not exist in the Customers table. Describe the constraint that will prevent this, and write the SQL statement that will enforce this rule.
3. **Scenario 3**: A customer tries to create an account using an email that is already in use by another customer. Describe the constraint that will prevent this, and write the SQL statement that ensures email addresses are unique.
4. **Scenario 4**: An order detail is added with a quantity of 0. Describe the constraint that will prevent this, and write the SQL statement to enforce a minimum quantity of 1 for each order detail.

FAQ

1)What are the types of Integrity Constraints?

2)Can a table have multiple Integrity Constraints?

3)How do Integrity Constraints affect database performance?

4)Can Integrity Constraints be disabled?

5)What happens if a data entry violates an Integrity Constraint?

Assignment No.: 4

Aim: Demonstrate implementation of DML commands of SQL with suitable examples

1)Insert

2)Update

3)Delete

4)Select

Software Required: MySQL

Theory: Data Manipulation Language (DML) commands in a Database Management System (DBMS) are a set of commands used to manipulate or interact with the data stored in the database. DML commands are primarily focused on performing operations such as retrieving, inserting, updating, and deleting data within the database.

DML commands in DBMS include:

1. SELECT: The SELECT command is used to retrieve data from one or more tables in the database. It allows you to specify the columns or attributes you want to retrieve, as well as conditions to filter the data based on certain criteria.

SELECT is primarily used for querying and retrieving data.

Select * from tablename where <search_condition>;

2. INSERT: The INSERT command is used to add new records or rows of data into a table in the database. It allows you to specify the values for each column or attribute in the table, and the DBMS will insert the new data accordingly. Insert Command can be used with 4 cases

Syntax: 1) INSERT INTO *table_name* (*column1*, *column2*, *column3*, ...)
VALUES (*value1*, *value2*, *value3*, ...);

2)INSERT INTO table_name VALUES (value1, value2, value3, ...);

3)INSERT INTO *table_name* (*column1*, *column2)* VALUES (*value1*, *value2);*

4) INSERT INTO table_name VALUES (&column1, &column2, &column3, ...)
VALUES (value1, value2);
Give / after inserting first record. It will prompt for next record.

3. UPDATE: The UPDATE command is used to modify existing records or rows in a table. It allows you to specify the changes or updates you want to make to one or more columns or attributes in the table. You can also use conditions to determine which records should be updated.

Syntax:

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

4. DELETE: The DELETE command is used to remove records or rows from a table in the database. It allows you to specify certain conditions to determine which records should be deleted. When executed, the DBMS will remove the specified records from the table.

Syntax:

DELETE FROM *table_name* WHERE *condition*;

SQL commands:

Consider the Student and Courses table created in Assignment 2.

1) **Insert Command -**
Insert into Students(StudentID, StudentName, and Major) values (10,'aarohi', 'DBMS');

2)Update Command -
Update Students set Major= 'Python' where StudentID='101' ;

3) Delete Command –

Delete from Courses where StudentID = '120';

4) Select Command –

Select * from Students;

Select * from Courses where CourseId=’C102’;

Conclusion: These DML commands provide the necessary functionality to manipulate and manage the data within a DBMS. They allow users to interact with the database by retrieving, inserting, updating, and deleting data according to their requirements

.**Exercise :**

I. Design an SQL command to insert a new product record into the "Products" table of the online store database, including details such as product name, price, quantity available, and category.

II. Create a set of DML commands to update the "Employee" table in the HR database, modifying the salary of an employee based on their performance rating and position.

III. Develop an SQL script to delete all inactive user accounts from the "Users" table of the social networking database, where the last login date is older than six months.

IV. Design a series of DML commands to insert a new patient's medical record into the "Patients" table of the hospital database, capturing information like patient ID, name, date of birth, admission date, and medical condition.

V. Create an SQL command to update the "Inventory" table in the retail store database, increasing the quantity of a specific product that has been restocked.

FAQs:

I. What is the purpose of the WHERE clause in SQL DML commands?
II. How can I update multiple columns in a table using UPDATE?
III. Can I insert data into multiple tables at once?
IV. What is the difference between the INSERT and UPDATE commands?

Experiment No. -5

Aim - Implement different types of SQL functions with suitable examples

1) Number function

2)Aggregate Function

3) Conversion Function

4) Date Function

Find the maximum and minimum Credits values of the courses

Software Required - SQL Server 15.0 /16.0

Theory :-

- SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.

1) Number function

1)ABS ()

The absolute value is returned as the output of the numeric expression.

The query returns the absolute value.

```
Select ABS(-22)
```

2)ACOS ()

The arc cosine value is returned as the output of the numeric expression that is specified.

Example

The below code generates the arc cosine value of 0.

Select ACOS (0)

3)ASIN()

Arc sine value is returned as the output of the specific numeric value expression.

Example

The query gives the arc sine value of 0.

4)SQRT ()

The square root of the numeric expression is returned as the output.

Example

It gives 2 for the 4 numeric expressions.

Select SQRT(4)

5)CEILING():

The CEILING () function returns the output after rounding of the decimals, which is the next highest value of the table.

Example:

The query gives 235 for the given 234.25 value.

Select CEILING (234.25)

FLOOR()

The function generates the output after rounding of the decimals, which is equal to or less than the expression value.

Example:

The query gives 231 for the given 231.34 value.

Select FLOOR (231.34)

2)SQL Aggregate Functions –

An aggregate function is a function that performs a calculation on a set of values, and returns a single value.

Aggregate functions are often used with the GROUP BY clause of the SELECT statement. The GROUP BY clause splits the result-set into groups of values and the aggregate function can be used to return a single value for each group.

The most commonly used SQL aggregate functions are:

MIN() - returns the smallest value within the selected column

MAX() - returns the largest value within the selected column

COUNT() - returns the number of rows in a set

SUM() - returns the total sum of a numerical column

AVG() - returns the average value of a numerical column

Aggregate functions ignore null values (except for COUNT()).

1) Count ( ) –

This function returns the number of records(rows) in a table. The Syntax of the count() function is mentioned below.

Syntax:

SELECT COUNT(column_name) FROM table_name WHERE condition;

select COUNT(*) FROM Student;

2) SUM():

This function returns the sum of all values of a column in a table. Here is the syntax for the sum() function.

Syntax:

SELECT SUM(column_name) FROM table_name WHERE condition;

select SUM(Price) FROM sales;

3) AVG()

This function will return the average of all values present in a column. The syntax of the AVG() function is given below.

Syntax:

SELECT AVG(column_name) FROM table_name WHERE condition;

SELECT AVG(Price) FROM sales;

SELECT AVG(Price) FROM sales WHERE Product_name = 'Mobile';

4) MIN():

This function produces the lowest value in a column for a group of rows that satisfy a given criterion. The Syntax of the MIN() function is as follows

Syntax:

SELECT MIN(column_name) FROM table_name WHERE condition;

SELECT MIN(salary) FROM employees;

SELECT MIN(salary) FROM employees WHERE department = 'R&D';

5) MAX( )

The MAX function in SQL is used to return the highest value in a column for a group of

rows that satisfy a given condition in a table. The MAX syntax is as follows:

Syntax:

SELECT MAX(column_name) FROM table_name WHERE condition;

select MAX(salary) FROM employees;

3)SQL Conversion Functions

Explicit Data Type Conversion -

- TO_CHAR: NUMBER and DATE values can be converted explicitly into CHARACTER values
- TO_NUMBER: CHARACTER values can be converted explicitly into NUMBER values
- TO_DATE: CHARACTER values can be converted explicitly into DATE values

1)To_char() -

The TO_CHAR function returns an item of data type VARCHAR2. When applied to items of type NUMBER, several formatting options are available. The syntax is as follows

TO_CHAR(number1, [format])

The number1 parameter is mandatory and must be a value that either is or can be implicitly converted into a number. The optional format parameter may be used to specify numeric formatting information such as width, currency symbol, the position of a decimal point, and group (or thousands) separators, and must be enclosed in single quotation marks.

Example

Query 1

SELECT to_char(00001)||' is a special number' FROM dual;

The TO_NUMBER function returns an item of type NUMBER. Character strings converted into numbers must be suitably formatted so that any nonnumeric components are translated or stripped away with an appropriate format mask. The syntax is as follows:

2) TO_NUMBER(string1, [format])

Only the string1 parameter is mandatory and if no format mask is supplied, it must be a value that can be implicitly converted into a number. The optional format parameter is specified in single quotation marks.

Consider the following two queries

Query 1

SELECT to_number(‘$1,000.55’) FROM dual;

Query 2

SELECT to_number(‘$1,000.55′,’$999,999.99’) FROM dual;

3)To_date( )

The TO_DATE function returns an item of type DATE. Character strings converted to dates may contain all or just a subset of the date-time elements comprising a DATE. When strings with only a subset of the date time elements are converted, Oracle provides default values to construct a complete date. Components of character strings are associated with different date time elements using a format model or mask. The syntax is as follows

TO_DATE(string1, [format])

Only the string1 parameter is mandatory and if no format mask is supplied, string1 must take the form of a value that can be implicitly converted into a date. The optional format parameter is almost always used and is specified in single quotation marks. The format masks are identical to those listed in the TO_CHAR function above.

Consider the following five queries

Query 1

SELECT to_date(’25-DEC-2010′) FROM dual;

Query 2

SELECT to_date(’25-DEC’) FROM dual;

Sample Example

Employees Table

id	name	department	salary
1	Alice	HR	55000
2	Bob	IT	60000
3	Charlie	IT	70000
4	David	HR	50000
5	Eve	Finance	80000
6	Frank	IT	65000
7	Grace	Finance	75000
8	Heidi	HR	52000
9	Ivan	Marketing	56000

1)Total Number of Employees:

```
SELECT COUNT(*) AS total_employees FROM employees;
```

2)Total Salary Expenditure:

```
SELECT SUM(salary) AS total_salary FROM employees;
```

3) Average Salary in the Company:

```
SELECT AVG(salary) AS average_salary FROM employees;
```

4) Minimum and Maximum Salary:

```
SELECT MIN(salary) AS minimum_salary, MAX(salary) AS maximum_salary
FROM employees;
```

5) Average Salary by Department:

```
SELECT department, AVG(salary) AS average_salary FROM employees GROUP BY department;
```

6) Departments with Total Salary Greater than $120,000:

```
SELECT department, SUM(salary) AS total_salary FROM employees GROUP BY department
HAVING SUM(salary) > 120000;
```

7)**Number of Unique Departments:**

```
SELECT COUNT(DISTINCT department) AS unique_departments FROM employees;
```

8)**Average Salary of Employees Earning More than $60,000:**

SELECT AVG(salary) AS average_salary FROM employees WHERE salary > 60000;

9)**Total Number of Employees in Each Department:**

SELECT department, COUNT(*) AS num_employees FROM employees GROUP BY department;

10) **Departments with More than 2 Employees:**

SELECT department, COUNT(*) AS num_employees FROM employees GROUP BY department HAVING COUNT(*) > 2;

Conclusion –

Outcome of the experiment is understanding of different types of SQL functions.

Exercise –

1) Write a query to calculate the following from the `employees` table:

- The total number of employees.
- The average salary.
- The highest salary.
- The lowest salary.
- The sum of all salaries.

Using the employees table, write a query to find the average salary and total salary for each department.

2) Using the students and grades tables, write a query to find the highest average grade achieved by any student**.** (Assume suitable data)

3) **Sales Data Analysis**

Scenario: You are analyzing sales data for an e-commerce platform.

- **Task**: Calculate the total revenue and round it to two decimal places.
- **Table:** sales
- Columns: sale_id (INT), amount (DECIMAL), sale_date (DATE)

4)Employee Salary Adjustments

Scenario: The company is giving a 10% raise to all employees, but the new salary should be rounded up to the nearest whole number.

- Task: Calculate the new salaries.
- Table: employees
- Columns: employee_id (INT), current_salary (DECIMAL)

5) Scenario-Based SQL Date Functions Exercise Questions

1. Customer Registrations

Scenario: You need to calculate the number of days since each customer registered.

- **Task**: Calculate the days since registration for each customer.
- **Table**: customers
- Columns: customer_id (INT), registration_date (DATE)

FAQ –

1)How can I calculate the difference between two dates in SQL?
2)How can I get the current date and time in SQL?
3)What is the difference between COUNT(*) and COUNT(column_name)
4)What is the difference between CEIL() and FLOOR() functions?
5)What is the MOD() function and how is it used?

Assignment No.: 6

Aim: Study and Implement SQL Clauses.

1) Group By & having clause

2)Order by clause

3) Indexing

Software required: MySQL

Theory: In a database management system (DBMS), the GROUP BY clause and the HAVING clause are used in conjunction with the SELECT statement to perform advanced data analysis and filtering on groups of rows. Here's a brief introduction to each clause:

GROUP BY Clause:

The GROUP BY clause is used to group rows in a result set based on one or more columns. It is commonly used in combination with aggregate functions like COUNT, SUM, AVG, MAX, or MIN to perform calculations on groups of data. The result of a GROUP BY query is a set of rows where each row represents a unique combination of values in the specified column(s) and the aggregate function(s) are applied to the corresponding groups.

For example, if you have a "Sales" table with columns like "Product", "Category", and "Revenue", you can use the GROUP BY clause to calculate the total revenue per product category. The query might look like this:

```
SELECT Category, SUM(Revenue) AS TotalRevenue  FROM Sales
GROUP BY Category;
```

HAVING Clause:

The HAVING clause is used to filter the results of a GROUP BY query based on specified conditions. It allows you to apply filtering criteria to the grouped data after the aggregation has taken place. This clause operates similarly to the WHERE clause, but

while the WHERE clause filters individual rows, the HAVING clause filters groups of rows.

Continuing with the previous example, if you want to retrieve only the product categories with total revenue greater than a certain value, you can use the HAVING clause. For instance:

```
SELECT Category, SUM(Revenue) AS TotalRevenue
FROM Sales
GROUP BY Category
HAVING SUM(Revenue) > 100000;
```

This query will return the product categories with a total revenue greater than 100,000.

SQL Commands:

A.

Syntax:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

Example:

```
SELECT COUNT(Capacity), Hosp_Name FROM Hospital GROUP BY Capacity;
```

B.

Syntax:

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

. **Conclusion:** In summary, the GROUP BY clause helps to group rows based on specific columns, while the HAVING clause allows you to filter the grouped data based on aggregate conditions. Together, they provide powerful tools for data analysis and summarization in DBMS

Exercise :

I. Write an SQL query to analyze sales data by grouping it by product category and calculate the total revenue for each category. Display only the categories with total revenue greater than $10,000.

II. Develop an SQL query to retrieve the department names and the count of employees in each department. Display only the departments that have more than 50 employees.

III. Create an SQL query to group the products by their suppliers and calculate the average stock quantity for each supplier. Display only the suppliers whose average stock quantity exceeds 500 units.

IV. Write an SQL query to group customers by their age range (e.g., 18-25, 26-35, etc.) and calculate the count of customers in each age range. Display only the age ranges with more than 100 customers.

V. Develop an SQL query to analyze exam scores by grouping them by the subject and calculate the average score for each subject. Display only the subjects with an average score above 80.

FAQs: (Answer all FAQs using suitable examples)

I. What is the difference between the WHERE clause and the HAVING clause?

II. Can I use the GROUP BY clause without any aggregate functions?

III. Can I use the HAVING clause without the GROUP BY clause?

IV. Can I include columns in the SELECT statement that are not part of the GROUP BY clause?

V. Can I use multiple aggregate functions in the HAVING clause?

Experiment No. -7

Aim -

Implement SQL Sub queries, nested queries , views with suitable examples

Software Required - SQL Server 15.0 /16.0

Theory :-

- SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.

Nested Sub Queries-

Nested subqueries are subqueries that are used inside another subquery in a SQL statement. They are used to perform more complex operations on data by using the results of one query as the input to another query.

Nested subqueries can be used to perform a wide range of operations, including filtering,sorting, and aggregating data. They are a powerful tool for working with complex data sets and can help to simplify SQL code by breaking down complex operations into smaller, more manageable parts.

Example-

1) Nested Subquery in SELECT Clause

Find the employees whose salary is above the average salary of their department.

```
SELECT employee_id, employee_name, salary FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees AS e
 WHERE e.department_id = employees.department_id);
```

2)Nested Subquery in WHERE Clause

Retrieve the names of customers who have placed an order.

```
SELECT customer_name FROM customers
WHERE customer_id IN (SELECT customer_id
      FROM orders);
```

3) Nested Subquery in UPDATE Statement

Increase the salary of employees who earn below the average salary of their department by 10%.

```
UPDATE employees SET salary = salary * 1.10
WHERE salary < (SELECT AVG(salary)
        FROM employees AS e
        WHERE e.department_id = employees.department_id);
```

4) Nested Subquery in DELETE Statement

Delete orders that were placed by inactive customers.

```
DELETE FROM orders WHERE customer_id IN (SELECT customer_id
  FROM customers  WHERE status = 'inactive');
```

Conclusion –

Outcome of the experiment is understanding of SQL nested sub queries .

FAQ –

1)What are the types of nested subqueries?
2) How do nested subqueries differ from joins?
3) Are there performance considerations with nested subqueries?
4) Can nested subqueries be used with aggregate functions?
5) What is a correlated subquery?

Exercise –

Consider Employee table with following attributes-

(emp_id,_name, last_name, department_id,salary)

And Department table with attributes (department_id,department_name)

1)Find the names of employees who work in the same department as 'Jane Smith'.

2)List the employees whose salary is above the average salary in their department.

3) Retrieve the departments which have at least one employee with a salary greater than $75,000.

4) Find the highest paid employee in each department.

5) Get the list of employees who earn more than the average salary across all departments.

Experiment No 8

Aim :- Join Operations

Title :- Execute the queries for implementation of Inner, Outer , Natural and Cross Join with suitable examples

Theory :-

1) **SQL Joins** :-

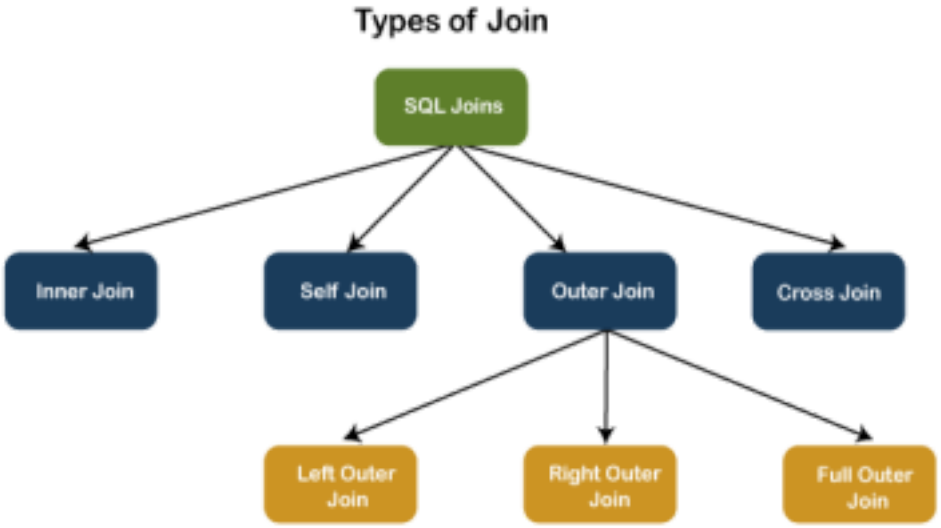
A SQL Join statement combines data or rows from two or more tables based on a common field between them. The join keyword merges two or more tables and creates a temporary image of the merged table. Then according to the conditions provided, it extracts the required data from the image table, and once data is fetched, the temporary image of the merged tables is dumped.

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

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

Types of JOINS in SQL

SQL mainly supports **four types of JOINS**, and each join type defines how two tables are related in a query. The following are types of join supports in SQL Server:



a) **INNER JOIN**

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

INNER JOIN Syntax

The following syntax illustrates the use of INNER JOIN in SQL Server:

```
SELECT columns

FROM table1

INNER JOIN table2 ON condition1

INNER JOIN table3 ON condition2
```

b) OUTER JOIN

OUTER JOIN in SQL Server returns all records from both tables that satisfy the join condition. In other words, this join will not return only the matching record but also return all unmatched rows from one or both tables.

- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

c) SELF JOIN

A table is joined to itself using the SELF JOIN. It means that each table row is combined with itself and with every other table row. The SELF JOIN can be thought of as a JOIN of two copies of the same tables. We can do this with the help of table name aliases to assign a specific name to each table's instance. The table aliases enable us to use the table's temporary name that we are going to use in the query.

SELF JOIN Syntax

```
SELECT T1.col_name, T2.col_name...

FROM table1 T1, table1 T2

WHERE join_condition;
```

d) NATURAL JOIN

In SQL, a natural join is a type of join that combines tables based on columns that have the same name and data types in both tables. When performing a natural join, SQL automatically matches columns with the same names between the two tables and returns rows that have equal values in these columns.

NATURAL JOIN Syntax

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

e) CROSS JOIN

CROSS JOIN in SQL Server combines all of the possibilities of two or more tables and returns a result that includes every row from all contributing tables. It's also known as CARTESIAN JOIN because it produces the Cartesian product of all linked tables.

CROSS JOIN Syntax

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

Sample Examples -

1) SELECT Students.StudentID, Students.StudentName, Courses.CourseId, Courses.CourseName FROM Students LEFT JOIN Courses ON Students.StudentID = Courses.StudentID;

2) SELECT Students.StudentID, Students.StudentName, Courses.CourseName FROM Students FULL OUTER JOIN Courses ON Students.StudentID = Courses.StudentID;

4) SELECT Students.StudentID, Students.StudentName, Courses.CourseID, Courses.CourseName FROM Students CROSS JOIN Courses;

Conclusion:-

SQL joins allow you to combine data from multiple tables in various ways (INNER, LEFT OUTER , RIGHT OUTER , FULL, NATURAL JOIN, CROSS) to suit different relational data needs and queries.

Exercise:-

Consider following tables

employees(employee_id, name, department_id)

departments(department_id, department_name)

1) Write a query to get the names of all employees and their corresponding department names.
2) Write a query to get all employees and their corresponding department names. Include employees who do not belong to any department. (Hint -Left Join)
3) Write a query to get all departments and their corresponding employees. Include departments that do not have any employees. (Hint - Right Join)
4) Write a query to get all employees and their corresponding department names, including those employees who do not belong to any department and those departments that do not have any employees.
5) Write a query to find the total number of employees in each department.

6) Write a query to find the names of employees who work in the department with the highest number of employees.

FAQ –

1) What are the different types of SQL JOINs?
2) What is the difference between INNER JOIN and OUTER JOIN?
3) Can we use more than one JOIN in a single query?
4) What are common performance considerations when using JOINs?
5) What is a SELF JOIN?
6) What is NATURAL JOIN?
7) What is the difference between NATURAL JOIN and SELF JOIN?

Experiment No 9

Aim :-Write a code for PL/SQL stored procedures and functions to perform a suitable operation on the database

Software Required - Oracle Server, SQLite

Theory :-

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:

1. 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.
2. 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.
3. 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.

Syntax for creating procedure:

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

- Below are the characteristics of Procedure subprogram unit in PL/SQL:

1) Procedures are standalone blocks of a program that can be stored in the database.

2) Call to these PLSQL procedures can be made by referring to their name, to execute the PL/SQL statements.

3) It is mainly used to execute a process in PL/SQL.

4) It can have nested blocks, or it can be defined and nested inside the other blocks or packages.

5) It contains the declaration part (optional), execution part, exception handling part (optional).

6) The values can be passed into the Oracle procedure or fetched from the procedure through parameters.

7) These parameters should be included in the calling statement.

8) A Procedure in SQL can have a RETURN statement to return the control to the calling block, but it cannot return any values through the RETURN statement.

9) Procedures cannot be called directly from SELECT statements. They can be called from another block or through the EXEC keyword.

Example1: Creating Procedure and calling it using EXEC

In this example, we are going to create an Oracle procedure that takes the name as input and prints the welcome message as output. We are going to use the EXEC command to call the procedure.

```
CREATE OR REPLACE PROCEDURE welcome_msg (p_name IN VARCHAR2)

IS

BEGIN

dbms_output.put_line (‘Welcome '|| p_name);

END;

/

EXEC welcome_msg (‘XYZ’);
```

- **PL/SQL Function** :-

Functions is a standalone PL/SQL subprogram. Like PL/SQL procedure, functions have a unique name by which it can be referred. These are stored as PL/SQL database objects.

Below are some of the characteristics of functions.

1)Functions are a standalone block that is mainly used for calculation purposes.

2)Function uses the RETURN keyword to return the value, and the datatype of this is defined at the time of creation.

3)It can have nested blocks, or it can be defined and nested inside the other blocks or packages.

4)It contains the declaration part (optional), execution part, exception handling part (optional).

5) The values can be passed into the function or fetched from the procedure through the parameters.

6)These parameters should be included in the calling statement. A PLSQL function can also return the value through OUT parameters other than using RETURN.

Syntax

```
CREATE OR REPLACE FUNCTION
<procedure_name>
(
<parameterl IN/OUT <datatype>
)
RETURN <datatype>
[ IS | AS ]
<declaration_part>
BEGIN
<execution part>
EXCEPTION
<exception handling part>
END;
```

A) Write a function to square the number taken from user

1) using pl/sql procedure

```
declare
x integer;
begin
x:=&x;
dbms_output.put_line('square of number: '||(x*x));
end;
```

output

```
Enter value for x: 4
old   4: x:=&x;
new   4: x:=4;
square of number: 16
```

2) using In and Out mode.

```
DECLARE
   a number;
PROCEDURE squareNum(x IN OUT number) IS
BEGIN
  x := x * x;
END;
BEGIN
   a:= 2;
   squareNum(a);
   dbms_output.put_line(' Square of (2): ' | |  a);
END;
```

3)using inbuilt SQRT function

```
DECLARE

  Test_Number number := 25;


BEGIN

  dbms_output.put_line(SQRT(Test_Number number));

  END;
```

B)Write a procedure to display the records from the Manufacturing industry / Hospital/ Company table.

1) Write a program in PL/SQL to display table based detailed information for the employee of ID 149 from the employees/company table.

Note :- Assume suitable attributes for employees table.

```
DECLARE
   z_employee employees%ROWTYPE;
BEGIN
   SELECT *
   INTO   z_employee          -- INTO clause always notifies only single row can be fetch
   FROM   employees
   WHERE  employee_id = 149;

   dbms_output.Put_line('Employee Details :   ID:'
                    ||z_employee.employee_id
                    ||'  Name: '
                    ||z_employee.first_name
                    ||' '
                    ||z_employee.last_name
                    ||'  Salary:  '
                    ||z_employee.salary);
END;
  /
```

2) Create a PL/SQL procedure that takes a department as input from a user & prints the number of employees working in that department in the same variable?

```
Create or Replace Procedure deptCnt(pX In Out emp.DeptNo%Type) Is

Begin

Select Count(*) Into pX

From Emp

Where DeptNo = pX;

Exception

When No_Data_Found Then

DBMS_Output.Put_Line('Incorrect Dept No.');

End;

/

Declare

x number;

Begin

x := 10;

deptCnt(x);

DBMS_Output.Put_Line('Count: ' || x);

End;

/
```

Conclusion :-

PL/SQL stored procedures and functions demonstrated enhanced efficiency and maintainability in database operations, validating their practical benefits for complex data processing tasks.

Exercise –

1)Write a PL/SQL program using WHILE loop for calculating the average of the numbers entered by the user. Stop the entry of numbers whenever the user enters the number 0.

2) Write a PL/SQL code to find whether a given string is palindrome or not.

3) Scenario 3: Customer Orders

Question:-
In a customer order system, there is a table of orders with columns order_id, customer_id, order_date, and total_amount.
-Write a PL/SQL stored procedure named get_customer_orders that accepts a customer_id and a date range (start_date and end_date), and returns a cursor with all orders placed by the customer within the given date range. Use this cursor to display the orders in the calling environment.

4)Scenario based:- Employee Management System
Question:
-Your company has an employee management system with a table named employees containing columns such as employee_id, first_name, last_name, salary, and department_id.
-Write a PL/SQL stored procedure named update_salary that takes an employee_id and a percentage increase, then updates the employee's salary by the given percentage.
-Additionally, ensure the procedure checks if the employee_id exists, and if not, it should raise a custom exception.

5)Scenario 2: Inventory Control
Question:
-You are tasked with managing the inventory for an online store. There is a table inventory with columns product_id, product_name, quantity, and price.
-Write a PL/SQL function named calculate_inventory_value that returns the total value of the inventory (sum of quantity * price for all products).
-This function should be called within a PL/SQL block to display the total inventory value.

FAQ:-

1)Write a PL/SQL program to display the employee IDs, names, job titles, hire dates, and salaries of all employees.

2) Write a PL/SQL program to display the names of all countries.

3) What is the Difference Between Pl SQL and SQL?

4) What is an Alias in SQL Statements?

5) What is a Dual Table?

6) What is Invalid_number, Value_error?

7)Explain Different Methods to Trace the Pl/SQL Code?

Experiment No. -10

Aim: Write PL/SQL blocks for demonstrating triggers and cursors

Software Required -MySQL

Theory :-

Cursor:

Cursors In MySQL, a cursor allows row-by-row processing of the result sets. A cursor is used for the result set and returned from a query. By using a cursor, you can iterate, or by step through the results of a query and perform certain operations on each row. The cursor allows you to iterate through the result set and then perform the additional processing only on the rows that require it.

A cursor contains the data in a loop. Cursors may be different from SQL commands that operate on all the rows returned by a query at one time.

There are some steps we have to follow, given below :

Declare a cursor

Open a cursor statement

Fetch the cursor

Close the cursor

1 . **Declaration of Cursor :** To declare a cursor you must use the DECLARE statement. With the help of the variables, conditions and handlers we need to declare a cursor before we can use it. First of all we will give the cursor a name, this is how we will refer to it later in the procedure. We can have more than one cursor in a single procedure so it's necessary to give it a name that will in some way tell us what it's doing. We then need to specify the select statement we want to associate with the cursor. The SQL statement can be any valid SQL statement and it is possible to use a dynamic where clause using variables or parameters as we have seen previously.

Syntax : DECLARE cursor_name CURSOR FOR select_statement;

2 **. Open a cursor statement** : To open a cursor we must use the open statement.If we want to fetch rows from it you must open the cursor.

Syntax : OPEN cursor_name;

3 . **Cursor fetch statement :** When we have to retrieve the next row from the cursor and move the cursor to the next row then you need to fetch the cursor.

Syntax : FETCH cursor_name INTO var_name;

If any row exists, then the above statement fetches the next row and cursor pointer moves ahead to the next row.

4 . **Cursor close statement :** By this statement closed the open cursor. Syntax: CLOSE_name;

By this statement we can close the previously opened cursor. If it is not closed explicitly then a cursor is closed at the end of the compound statement in which that was declared.

Example

```
1)  CREATE TABLE cust(
  C_ID INT PRIMARY KEY AUTO_INCREMENT,
  c_name VARCHAR(50),
  c_address VARCHAR(200)
);


CREATE TABLE backupdata(
  C_ID INT,
  c_name VARCHAR(50),
  c_address VARCHAR(200)
);


INSERT INTO cust(c_name, c_address) VALUES('Test', '132, Vatsa Colony'),
  ('Admin', '133, Vatsa Colony'),
  ('Vatsa', '134, Vatsa Colony'),
  ('Onkar', '135, Vatsa Colony'),
```

```
  ('Rohit', '136, Vatsa Colony'),
  ('Simran', '137, Vatsa Colony'),
  ('Jashmin', '138, Vatsa Colony'),
  ('Anamika', '139, Vatsa Colony'),
  ('Radhika', '140, Vatsa Colony');

SELECT * FROM cust;
SELECT * FROM backupdata;

delimiter //
CREATE PROCEDURE firstCurs()
BEGIN
DECLARE d INT DEFAULT 0;
DECLARE c_id INT;
DECLARE c_name, c_address VARCHAR(20);

DECLARE Get_cur CURSOR FOR SELECT * FROM cust;
DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'
SET d = 1;
DECLARE CONTINUE HANDLER FOR SQLSTATE '23000'
SET d = 1;

OPEN Get_cur;
lbl: LOOP
```

```
IF d = 1 THEN
LEAVE lbl;
END IF;
IF NOT d = 1 THEN

FETCH Get_cur INTO c_id, c_name, c_address;
INSERT INTO backupdata VALUES(c_id, c_name, c_address);
END IF;
END LOOP;

CLOSE Get_cur;
END;
//

CALL mysql.firstCurs();

delete from backupdata;
```

2) Delimiter $$

```
Create procedure p1(in_customer_id int)
begin
declare v_id int;
declare v_name varchar(20);
declare v_finished integer
```

```
default 0;
declare c1 cursor for select sid,sname from students where sid=in_customer_id;
declare continue handler for NOT FOUND set v_finished=1;
 open c1;
std:LOOP
fetch c1 into v_id,v_name;
if v_finished=1 then leave std;
end if;
select concat(v_id,v_name);
end LOOP std;
close c1;
 end;
```

Difference between Implicit and Explicit Cursors :

Implicit Cursors	Explicit Cursors
Implicit cursors are automatically created when select statements are executed.	Explicit cursors needs to be defined explicitly by the user by providing a name.
They are capable of fetching a single row at a time.	Explicit cursors can fetch multiple rows.
Closes automatically after execution.	Need to close after execution.
They are more vulnerable to errors such as	They are less vulnerable to

Data errors, etc.	
Provides less programmatic control to the users	User/Programmer has the entire control.
Implicit cursors are less efficient.	Comparative to Implicit cursors, explicit cursors are more efficient.
Implicit Cursors are defined as: BEGIN SELECT attr_name from table_name where CONDITION; END	Explicit cursors are defined as: DECLARE CURSOR cur_name IS SELECT attr_name from table_name where CONDITION; BEGIN

Trigger:

A trigger is a named MySQL object that activates when an event occurs in a table. Triggers are a particular type of stored procedure associated with a specific table.

Triggers allow access to values from the table for comparison purposes using NEW and OLD. The availability of the modifiers depends on the trigger event you use:

Trigger Event	OLD	NEW
INSERT	No	Yes
UPDATE	Yes	Yes
DELETE	Yes	No

Checking or modifying a value when trying to insert data makes the NEW.<column name> modifier available. This is because a table is updated with new content. In contrast, an OLD.<column name> value does not exist for an insert statement because there is no information exists in its place beforehand.

When updating a table row, both modifiers are available. There is OLD.<colum name> data which we want to update to NEW.<column name> data.

Finally, when removing a row of data, the OLD.<column name> modifier accesses the removed value. The NEW.<column name> does not exist because nothing is replacing the old value upon removal.

Create Triggers
Use the CREATE TRIGGER statement syntax to create a new trigger:

CREATE TRIGGER <trigger name> <trigger time > <trigger event>

ON <table name>

FOR EACH ROW
<trigger body>;
The best practice is to name the trigger with the following information:

<trigger time>_<table name>_<trigger event>
For example, if a trigger fires before insert on a table named employee, the best convention is to call the trigger:

before_employee_insert
Alternatively, a common practice is to use the following format:

<table name>_<first letter of trigger time><first letter of trigger name>
The before insert trigger name for the table employee looks like this:

employee_bi
The trigger executes at a specific time of an event on a table defined by <table name> for each row affected by the function.

Delete Triggers
To delete a trigger, use the DROP TRIGGER statement:

DROP TRIGGER <trigger name>;

Alternatively, use:

DROP TRIGGER IF EXISTS <trigger name>;
The error message does not display because there is no trigger, so no warning prints.

Example-

Create Example Database
Create a database for the trigger example codes with the following structure:

1. Create a table called *person* with *name* and *age* for columns.

CREATE TABLE person (name varchar(45), age int);
Insert sample data into the table:

INSERT INTO person VALUES ('Matthew', 25), ('Mark', 20);
Select the table to see the result:

SELECT * FROM person;

2. Create a table called *average_age* with a column called *average*:

CREATE TABLE average_age (average double);
Insert the average age value into the table:

INSERT INTO average_age SELECT AVG(age) FROM person;
Select the table to see the result:

SELECT * FROM average_age;

3. Create a table called *person_archive* with *name*, *age*, and *time* columns:

CREATE TABLE person_archive (

name varchar(45),

age int,
time timestamp DEFAULT NOW());

Note: The function NOW() records the current time.

Create a BEFORE INSERT Trigger
To create a BEFORE INSERT trigger, use:

CREATE TRIGGER <trigger name> BEFORE INSERT

ON <table name>

FOR EACH ROW
<trigger body>;

The BEFORE INSERT trigger gives control over data modification before committing into a database table. Capitalizing names for consistency, checking the length of an input, or catching faulty inputs with BEFORE INSERT triggers further provides value limitations before entering new data.

BEFORE INSERT Trigger Example

Create a BEFORE INSERT trigger to check the age value before inserting data into the *person* table:

delimiter //

CREATE TRIGGER person_bi BEFORE INSERT

```
ON person

FOR EACH ROW

IF NEW.age < 18 THEN

SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'Person must be older than 18.';

END IF; //
delimiter ;
```

Inserting data activates the trigger and checks the value of *age* before committing the information:

```
INSERT INTO person VALUES ('John', 14);
```

The console displays the descriptive error message. The data does not insert into the table because of the failed trigger check.

Create an AFTER INSERT Trigger
Create an AFTER INSERT trigger with:

```
CREATE TRIGGER <trigger name> AFTER INSERT

ON <table name>

FOR EACH ROW
<trigger body>;
```

The AFTER INSERT trigger is useful when the entered row generates a value needed to update another table.

AFTER INSERT Trigger Example

Inserting a new row into the *person* table does not automatically update the average in the *average_age* table. Create an AFTER INSERT trigger on the *person* table to update the *average_age* table after insert:

```
delimiter //

CREATE TRIGGER person_ai AFTER INSERT

ON person

FOR EACH ROW

UPDATE average_age SET average = (SELECT AVG(age) FROM person); //
delimiter ;
```

Inserting a new row into the *person* table activates the trigger:

INSERT INTO person VALUES ('John', 19);

The data successfully commits to the *person* table and updates the *average_age* table with the correct average value.

Create a BEFORE UPDATE Trigger
Make a BEFORE UPDATE trigger with:

CREATE TRIGGER <trigger name> BEFORE UPDATE

ON <table name>

FOR EACH ROW
<trigger body>;
The BEFORE UPDATE triggers go together with the BEFORE INSERT triggers. If any restrictions exist before inserting data, the limits should be there before updating as well.

BEFORE UPDATE Trigger Example

If there is an age restriction for the *person* table before inserting data, the age restriction should also exist before updating information. Without the BEFORE UPDATE trigger, the age check trigger is easy to avoid. Nothing restricts editing to a faulty value.

Add a BEFORE UPDATE trigger to the *person* table with the same body as the BEFORE INSERT trigger:

delimiter //

CREATE TRIGGER person_bu BEFORE UPDATE

ON person

FOR EACH ROW

IF NEW.age < 18 THEN

SIGNAL SQLSTATE '50002' SET MESSAGE_TEXT = 'Person must be older than 18.';

END IF; //
delimiter ;

Updating an existing value activates the trigger check:

UPDATE person SET age = 17 WHERE name = 'John';

Updating the *age* to a value less than 18 displays the error message, and the information does not update.

Create an AFTER UPDATE Trigger
Use the following code block to create an AFTER UPDATE trigger:

CREATE TRIGGER <trigger name> AFTER UPDATE

ON <table name>

FOR EACH ROW
<trigger body>;
The AFTER UPDATE trigger helps keep track of committed changes to data. Most often, any changes after inserting information also happen after updating data.

AFTER UPDATE Trigger Example

Any successful updates to the *age* data in the table *person* should also update the intermediate average value calculated in the *average_age* table.

Create an AFTER UPDATE trigger to update the *average_age* table after updating a row in the *person* table:

delimiter //

CREATE TRIGGER person_au AFTER UPDATE

ON person

FOR EACH ROW

UPDATE average_age SET average = (SELECT AVG(age) FROM person); //
delimiter ;

Updating existing data changes the value in the *person* table:

UPDATE person SET age = 21 WHERE name = 'John';

Updating the table *person* also updates the average in the *average_age* table.

Create a BEFORE DELETE Trigger
To create a BEFORE DELETE trigger, use:

CREATE TRIGGER <trigger name> BEFORE DELETE

ON <table name>

FOR EACH ROW
<trigger body>;
The BEFORE DELETE trigger is essential for security reasons. If a parent table has any children attached, the trigger helps block deletion and prevents orphaned tables. The trigger also allows archiving data before deletion.

BEFORE DELETE Trigger Example

Archive deleted data by creating a BEFORE DELETE trigger on the table *person* and insert the values into the *person_archive* table:

```
delimiter //

CREATE TRIGGER person_bd BEFORE DELETE

ON person

FOR EACH ROW

INSERT INTO person_archive (name, age)

VALUES (OLD.name, OLD.age); //
delimiter ;
```

Deleting data from the table *person* archives the data into the *person_archive* table before deleting:

```
DELETE FROM person WHERE name = 'John';
```

Inserting the value back into the *person* table keeps the log of the deleted data in the *person_archive* table:

```
INSERT INTO person VALUES ('John', 21);
```

The BEFORE DELETE trigger is useful for logging any table change attempts.

Conclusion :-

PL/SQL triggers and cursors effectively managed data integrity and facilitated precise control over database operations, confirming their pivotal role in automation and data manipulation tasks.

Exercise:

Consider the Following schema

Emp (eno, ename, designation, salary, dno)

Dept (dno dname,dhod)

1. Increment the salary of all 'comp' dept employees with 10 %
2. Display the ename and designation if salary is above 35000 of dno 101
3. Create the trigger on emp Table: The deleted record from the emp table should be insert in Dummy Table

Consider the Following schema

Boats(Bid, Name, Bcolor)

Sailors(Sid,Sname, Srating)

Reserves (Bid, Sid, Date of Reservation)

1. Create the trigger on Sailors Table: The Rating of the Sailor should get incremented by 1 once the sailor reserves a boat.
2. Create the Cursor which will Insert the Sid, Sname, Bid who reserved red color Boat in Red_Boats Table;

Consider the Following schema

Books (Sid, Bid, BName, BPrice)
Transactions (Sid,Bid, Date_Issue,Date_Return, Status)
Return_books (Sid,Bid, Fine_amout)

1. Create a trigger on Books Table such that insertion of Books details to insert a record in Transaction table (Sid and Bid values should be Same, others values can be Assumable)
2. Display the Book Names Issued to Sid 'XXX' using Cursor
3. Create a trigger on the Books Table so that BName will be stored in uppercase.
4. Update the Date_Return of Sid 'xxx' . Then Create the Trigger to Update the Status of Book to 'Return'
5. Create a cursor which will calculate the Fine_Amount and insert the 'Return' Books in Return_books table.
   Conditions:
   If No of Days Between Date_Issue and Date_Return > 15 Days, Fine_amount is : 10 Rs Per Day
   If No of Days Between Date_Issue and Date_Return > 16 Days and < 30 Days, Fine_amount is : 20 Rs Per Day
   If No of Days Between Date_Issue and Date_Return > 30 Days, Fine_amount is : 30 Rs Per Day

FAQ:-

1. What is a cursor?
2. What are the types of cursor?
3. What is the use of a parameterized cursor?
4. What is the use of the cursor variable?
5. What is a normal cursor?
6. What are Explicit cursor attributes?

Experiment No. -11

Aim - Study & Implementation of Database Backup & Recovery commands :-
Rollback, Commit, Savepoint.

Software Required :- MYSQL

Theory :- Understanding and implementing database backup and recovery commands, such as Rollback, Commit, and Savepoint, is crucial for maintaining data integrity and consistency in any database management system. Here's a detailed explanation and implementation guide for each command.

1. Rollback

The Rollback command is used to undo changes made in the current transaction. It can revert the database to its previous state since the last commit or savepoint.

Syntax: ROLLBACK;

Example:

Consider a scenario where you are updating a user's account balance. If an error occurs during the update, you can use Rollback to undo the changes.

```
BEGIN TRANSACTION;

UPDATE accounts SET balance = balance - 500 WHERE user_id = 1;

-- Assume something goes wrong here

ROLLBACK; -- This will undo the update
```

2. Commit

The **Commit** command is used to save all changes made during the current transaction. Once a commit is executed, the changes are permanent and cannot be undone using Rollback.

Syntax:

COMMIT;

Example:

Continuing from the previous example, if the update is successful, you can use Commit to save the changes.

```
BEGIN TRANSACTION;
UPDATE accounts SET balance = balance - 500 WHERE user_id = 1;
 -- Assume everything goes right here
COMMIT;
 -- This will save the update
```

3. Savepoint

The **Savepoint** command is used to set a point within a transaction to which you can later roll back. This allows partial rollbacks within a transaction.

Syntax:

```
SAVEPOINT savepoint_name;
```

Example:

Using Savepoint, you can mark a point in your transaction and roll back to it if needed.

```
BEGIN TRANSACTION;

SAVEPOINT sp1;

UPDATE accounts SET balance = balance - 500 WHERE user_id = 1;

SAVEPOINT sp2;

UPDATE accounts SET balance = balance + 300 WHERE user_id = 2;

-- Assume something goes wrong after sp2

ROLLBACK TO sp2; -- This will undo the update to user_id = 2, but keep the update to user_id = 1

COMMIT; -- This will save the changes made before sp2
```

Another Example

Creating a Sample Table:

```
CREATE TABLE accounts (
   user_id INT PRIMARY KEY,
   balance DECIMAL(10, 2)
);

INSERT INTO accounts (user_id, balance) VALUES (1, 1000.00), (2, 2000.00);
```

Using Rollback, Commit, and Savepoint:

```
START TRANSACTION;

UPDATE accounts SET balance = balance - 500.00 WHERE user_id = 1;
SAVEPOINT sp1;

UPDATE accounts SET balance = balance + 300.00 WHERE user_id = 2;
SAVEPOINT sp2;

-- Simulate an error
-- SELECT * FROM nonexistent_table;

ROLLBACK TO sp2; -- Undo changes after sp2
COMMIT; -- Commit all changes before sp2
```

Exercise Questions

Question 1: Basic Transaction Management

1. **Create a table**:
    - Create a table named `employees` with columns `id`, `name`, and `salary`.
    - Insert 3 records into the `employees` table.
2. **Transaction with Rollback**:
    - Start a transaction.
    - Update the salary of one employee.
    - Rollback the transaction.
    - Verify that the salary change has not been applied.

Question 2: Using Commit

1. **Create a table**:
    - Create a table named `products` with columns `product_id`, `product_name`, and `price`.
    - Insert 3 records into the `products` table.
2. **Transaction with Commit**:
    - Start a transaction.
    - Update the price of one product.
    - Commit the transaction.
    - Verify that the price change has been applied.

Question 3: Savepoint Usage

1. **Create a table**:
    - Create a table named `orders` with columns `order_id`, `customer_name`, and `order_amount`.
    - Insert 3 records into the `orders` table.
2. **Transaction with Savepoint**:
    - Start a transaction.
    - Update the order amount for two different orders.
    - Set a savepoint after the first update.
    - Simulate an error by attempting to select from a nonexistent table.
    - Rollback to the savepoint.
    - Commit the transaction.
    - Verify that only the first update has been applied.

Scenario Based Questions

1. You are working on a banking application where you need to transfer funds between two accounts. Describe how you would use BEGIN, COMMIT, and ROLLBACK commands to ensure the transaction is handled correctly in case of an error during the transfer.

2. While updating multiple records in a customer database, an error occurs after updating a few records. Explain how you would use SAVEPOINT and ROLLBACK commands to revert only the changes made after a specific point.

3. In an e-commerce system, you need to process an order by updating the inventory and recording the sale. Describe how you would ensure data integrity using transactions in case one of the updates fails.

4. During a bulk insert operation, you encounter an issue after inserting half of the records. How would you use transaction commands to handle this situation and maintain the consistency of your data?
5. You are maintaining a payroll system, and you need to ensure that salary updates for employees are either fully applied or fully reverted in case of any errors. Explain how you would achieve this using BEGIN TRANSACTION, COMMIT, and ROLLBACK.

6. While performing a complex series of updates and inserts in a sales database, you realize an error after several successful operations. How would you utilize SAVEPOINT to minimize data loss and rollback only the erroneous part of the transaction?
7. In a student management system, you need to enroll a student in multiple courses. If enrolling in any course fails, all previous enrollments should be reverted. Describe how you would use transaction control commands to implement this.

8. During a multi-step data processing task, you set multiple savepoints. Describe how you would use these savepoints to rollback to a specific point if an error occurs, without losing all previous successful operations.

9. You are tasked with implementing a feature that applies multiple discount updates to a product catalog. If an error occurs during any discount update, all previous updates should be undone. Explain how you would use transaction management commands to handle this.

10. In a financial application, you need to ensure that a series of debit and credit operations are either all committed or all rolled back. How would you structure your transaction commands to handle this requirement effectively?

Oral Questions:

1. What is the primary purpose of the Rollback command in a database transaction?
2. Can you explain what happens when the Commit command is executed in a transaction?
3. How does a Savepoint differ from a Rollback?
4. Describe a scenario where you might use a Savepoint in a transaction.
5. What would happen if you issue a Rollback command after a Commit command has been executed?
6. Can you provide an example of using the Rollback command to handle an error in a transaction?
7. Why is it important to use transactions (with Commit and Rollback) in database operations?
8. What is the significance of the Commit command in the context of concurrent database transactions?
9. How would you explain the concept of a transaction to someone new to databases?
10. What are the potential consequences of not using transactions properly in a database application?

Experiment No. -12

Aim - Building a Library Management System with Python and SQL Database Connectivity

Software Required – MYSQL

Theory :-

1. **Set up a MySQL Database**:
    - Install MySQL server and create a sample database.
    - Create tables and populate them with sample data.
2. **Install Python Libraries**:
    - Install mysql-connector-python for connecting to the MySQL database.
    - Install tkinter for creating a user interface.
3. **Create a Python Script**:
    - Connect to the MySQL database.
    - Create a simple user interface using tkinter.
    - Execute SQL queries based on user input and display the results.

Let's go through each step in detail.
Here we are using my sql as backend database because of it is opensource,free and portable and widely used.
Anyone of mysql-connectoror MySQLdb can be used for database programming.
To create a MySQL user interface in Python using MySQL Connector, you typically need to use a GUI framework like Tkinter along with MySQL Connector/Python for database connectivity. Here's a basic example of how you can create a simple interface to connect to MySQL and perform some operations:

Requirements

Make sure you have installed the following packages:

- MySQL Connector/Python (mysql-connector-python)
- Tkinter (Python's standard GUI package)

You can install mysql-connector-python using pip:
bash
Copy code
pip install mysql-connector-python

Example Code

Key Features:

1. Book Management:

- Add new books to the library database with details like title, author, ISBN, genre, and quantity.
- Update existing book information (e.g., title, author, quantity) if needed.
- Remove books from the library database when they are no longer available.

2. Borrower Management:
   - Add new borrowers to the library system, including information like name, contact details, and membership ID.
   - Update borrower information when required (e.g., contact details).
   - Remove borrowers from the system if necessary.
3. Book Borrowing and Returning:
   - Allow borrowers to borrow books by linking the borrower's membership ID to the book details.
   - Record the due date for each borrowed book and handle overdue books.
   - Implement a mechanism for borrowers to return books, updating the database accordingly.
4. Book Search and Availability:
   - Provide a search feature that enables users to find books by title, author, or genre.
   - Show the availability status (number of copies available) for each book in the search results.
5. Fine Calculation:
   - Calculate and manage fines for late book returns based on pre-defined rules.
   - Update fines automatically when books are returned after the due date.
6. Database Connectivity:
   - Establish a connection between the Python application and the MySQL database.
   - Create appropriate database tables to store books, borrowers, transactions, and fines data.
   - Implement CRUD (Create, Read, Update, Delete) operations for seamless data management.

Technologies Used:

Python: The core programming language used for the development of the Library Management System.

MySQL Database: The database management system used to store and manage library-related data.

MySQL Connector: A Python library to facilitate connectivity between the Python application and the MySQL database.

Conclusion:-

The experiment successfully demonstrates how to integrate MySQL database functionality into a Python application using tkinter for the user interface and MySQL Connector/Python for database connectivity.

FAQ :-

1) How to connect to a MySQL database using Python?
2) What are the prerequisites for connecting to MySQL using Python?
3) What are the security considerations when connecting to MySQL
4) How can I close the MySQL connection properly?