**DEPARTMENT OF INFORMATION TECHNOLOGY**

**ENGINEERING LABORATORY MANUAL**

**214456: DATABASE MANAGEMENT SYSTEM LAB**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.No** | **Assignment Name** | | **Page No.** |
| 1 | **Group A: Study of Databases**  Study of MySQL Open source software. Discuss the characteristics like efficiency, scalability, performance and transactional properties | | 7-8 |
| 2 | **Group A: Study of Databases**  Install and configure client and server of MySQL.(Show all commands and necessary steps for installation and configuration) | | 9-10 |
| 3 | **Group A: Study of Databases**  Study of SQLite: What is SQLite? Uses of SQLite. Building and installing SQLite. | | 11-12 |
| 4 | **Group B: MySQL**  Design any database with at least 3 entities and relationships between them.  Draw suitable ER/EER diagram for the system. | | 13-15 |
| 5 | **Group B: MySQL**  Design and implement a database (for assignment no 1) using DDL statements and apply normalization on them |  | 16-21 |
| 6 | **Group B: MySQL**  Create Table with primary key and foreign key constraints. a. Alter table with add n modify b. Drop table |  | 22-25 |
| 7 | **Group B: MySQL**  Perform following SQL queries on the database created assignment 1.   * Implementation of relational operators in SQL * Boolean operators and pattern matching * Arithmetic operations and built in functions * Group functions * Processing Date and Time functions * Complex queries and set operators | in | 26-30 |
| 8 | **Group B: MySQL**  Execute DDL/DML statements which demonstrate the use of views. Try to update the base table using its corresponding view. Also consider restrictions on updatable views and perform view creation from multiple tables. | | 31-33 |

1

|  |  |  |
| --- | --- | --- |
| 9 | **Group C: PL/SQL**  Write and execute PL/SQL stored procedure and function to perform a suitable task on the database. Demonstrate its use. | 34-34 |
| 10 | **Group C: PL/SQL**  Write and execute suitable database triggers .Consider row level and statement level triggers. | 35-36 |
| 11 | **Group C: PL/SQL**  Write a PL/SQL block to implement all types of cursor. | 37-40 |
| 12 | **Group D: Relational Database Design**  Design and case study of any organization (back end only), Project  Proposal and High Level SRS  To prepare for your project, do the following:   1. Form teams of around 3 to 4 people 2. Create a requirements document with the following information;  Give a one/two paragraph description of your goals for topic(s).    List all what all types of users will be accessing your application (e.g., for moodle, the types are teachers, students, teaching assistants, and a few more types).   1. List the various functionalities that your application will support. Explain each in about a paragraph worth of detail. 2. List the hardware and software requirements at the backend and at the front end. 3. Give an estimate of the number of users of each type, the expected load (transactions per day), and the expected database size. | 41-42 |

**Reference Books:**

1.Dr. P. S. Deshpande, SQL and PL/SQL for Oracle 10g Black Book, DreamTech. 2.Ivan

Bayross, SQL, PL/SQL: The Programming Language of Oracle, BPB Publication. 3.Reese G., Yarger R., King T., Williums H, Managing and Using MySQL, Shroff Publishers and Distributors Pvt. Ltd., ISBN: 81 - 7366 - 465 – X, 2nd Edition.

4.Eric Redmond, Jim Wilson, Seven databases in seven weeks, SPD, ISBN: 978-93- 5023918-6.Jay Kreibich, Using SQLite, SPD, ISBN: 978-93-5110-934-1, 1st edition.

**Web Resources:**

1. Udemy 2. Coursera 3.SQL TutorialsPoint

2

# DATABASE

**MANAGEMENT**

**SYSTEM LAB MANUAL**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **DATABASE MANAGEMENT SYSTEM LAB** |  |  |
| **Experiment No: 1** | **Group A: Study of Databases**  Study of MySQL Open source software. Discuss the characteristics like efficiency, scalability, performance and transactional properties | **Page** | **7/8** |

**TITLE:-** Study of MySQL Open source software.

**OBJECTIVE:-** Study of MySQL Open source software. Discuss the characteristics like efficiency, scalability, performance and transactional properties

**THEORY: -**

**Database:**

A database is a collection of information or data which are organized in such a way that it can be easily accessed, managed and retrieved.

**Elements of Database:**

A database table consists of rows and columns which contain data. For example, you have a table that stores profiles of individuals that is, ID, name, address and contact details.

**EMPLOYEE TABLE**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_AGE** | **EMP\_CITY** | **DEPT\_ID** | **DEPT\_NAME** |
| 22 | Denim | 28 | Mumbai | 827 | Sales |
| 33 | Alina | 25 | Delhi | 438 | Marketing |
| 46 | Stephan | 30 | Bangalore | 869 | Finance |
| 52 | Katherine | 36 | Mumbai | 575 | Production |

**MySQL Database Features:**

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses.



MySQL is released under an open-source license.

MySQL is a very powerful program in its own right.

MySQL uses a standard form of the well-known SQL data language.

MySQL works on many operating systems and languages including PHP, PERL, C, C++, JAVA.

MySQL works very quickly and works well even with large data sets.

MySQL is very friendly to PHP, the most appreciated language for web development.

MySQL supports large databases, up to 50 million rows or more in a table.

MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

**1) MySQL Create Database**

MySQL create database is used to create database.

Example:

## create database db1;

**2) MySQL Select/Use Database**

MySQL use database is used to select database. Example:

## use db1;

1. **MySQL Create Query**

MySQL create query is used to create a table, view, procedure and function. Example:

|  |  |
| --- | --- |
| CREATE TABLE customers |  |
| (id int(10), name varchar(50), city varchar(50),PRIMARY KEY (id )); | |

1. **MySQL Alter Query**

MySQL alter query is used to add, modify, delete or drop columns of a table.

Example:

## ALTER TABLE customers ADD age varchar(50);

**5) MySQL Insert Query**

MySQL insert query is used to insert records into table.

Example:

## insert into customers values(101,'rahul','delhi');

**6) MySQL Update Query**

MySQL update query is used to update records of a table.

Example:

## update customers set name='bob', city='london' where id=101;

**7) MySQL Delete Query**

MySQL update query is used to delete records of a table from database.

Example:

## delete from customers where id=101;

**8) MySQL Select Query**

Oracle select query is used to fetch records from database.

Example:

## SELECT \* from customers;

**9) MySQL Truncate Table Query**

MySQL update query is used to truncate or remove records of a table. Example:

## truncate table customers;

**10) MySQL Drop Query**

MySQL drop query is used to drop a table, view or database. Example:

## drop table customers;

|  |  |  |  |
| --- | --- | --- | --- |
|  | **DATABASE MANAGEMENT SYSTEM LAB** |  |  |
| **Experiment No: 2** | **Group A: Study of Databases**  Install and configure of MySQL. Show all commands and necessary steps for installation and configuration | **Page** | **9/10** |

**TITLE:** Install and configure of MySQL.

**OBJECTIVE:-** Install and configure of MySQL. Show all commands and necessary steps for installation and configuration

**THEORY: -**

[MySQL i](https://www.mysql.com/)s an open-source database management system, commonly installed as part of the popular [LAMP (](https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-ubuntu-18-04)Linux, Apache, MySQL, PHP/Python/Perl) stack. It uses a relational database and SQL (Structured Query Language) to manage its data.

**INSTALL MYSQL ON UBUNTU**

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 remotely? (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>**

**mysql> show databases;** The output is:

|  |  |
| --- | --- |
| + +  | Database |  + +  | information\_schema | | |
| | mysql |  | performance\_schema |  | sys |  + + | |
| 4 rows **in set** (0.05 sec) |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **DATABASE MANAGEMENT SYSTEM LAB** |  |  |
| **Experiment No: 3** | **Group A: Study of Databases**  Study of SQLite: What is SQLite? Uses of SQLite. Building and installing SQLite | **Page** | **11/12** |

**TITLE**: Study of SQLite:

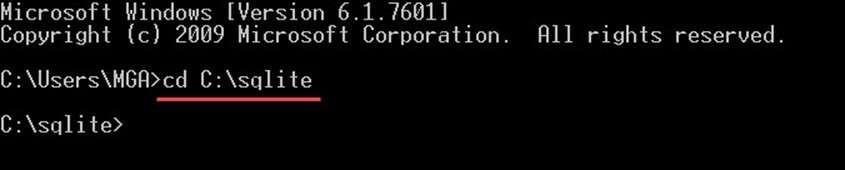
**OBJECTIVE: -** Study of SQLite: What is SQLite? Uses of SQLite. Building and installing SQLite.

**THEORY: -**

[MySQL i](https://www.mysql.com/)s an open-source database management system, commonly installed as part of the popular [LAMP (](https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-ubuntu-18-04)Linux, Apache, MySQL, PHP/Python/Perl) stack. It uses a relational database and SQL (Structured Query Language) to manage its data.

**SQLite Database**

* SQLite is a self-contained, high-reliability, embedded, full-featured, public domain, SQL database engine. SQLite is the most used database engine in the world
* SQLite databases are very lightweight. Unlike other database systems, there is no configuration, installation required to start working on SQLite database.
* What you need is the SQLite library which is less than 500KB size.



## SQLITE CREATE TABLE

To create a table, you should use the **"CREATE TABLE"** command as following:

|  |  |
| --- | --- |
| C:\SQLITE>CREATE TABLE guru99 ( Id Int, Name Varchar); |  |

DROP TABLE To drop a table, use the **"DROP TABLE"** command followed by the table name as following:

|  |  |
| --- | --- |
| C:\SQLITE>DROP TABLE guru99; |  |

## ALTER TABLE

You can use **"ALTER TABLE"** command to rename a table as following:

|  |  |
| --- | --- |
| C:\SQLITE>ALTER TABLE guru99 RENAME TO guru100; |  |

## SQLITE ADD COLUMNS- USING ALTER TABLE COMMAND

You can also use the **"ALTER TABLE"** command to add columns:

|  |  |
| --- | --- |
| C:\SQLITE>ALTER TABLE guru100 ADD COLUMN Age INT; |  |

|  |  |  |
| --- | --- | --- |
| SQLITE INSERT VALUE INTO A TABLE |  | |
| C:\SQLITE>INSERT INTO guru100 VALUES(1, 'Mike', 25); | |  |

SQLITE DISPLAY ALL VALUES

|  |  |
| --- | --- |
| C:\SQLITE>SELECT \* FROM Students; |  |

## **INSTALL SQLITE ON WINDOWS**

* **Step 1** − Go to SQLite for Windows
* **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 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**

 **Step 1**-Download sqlite-autoconf-\*.tar.gz from source code section of SQLite Site  **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.

**$sqlite3**

SQLite version 3.7.15.2 2013-01-09 11:53:05

Enter ".help" for instructions

Enter SQL statements terminated with a ";" sqlite>

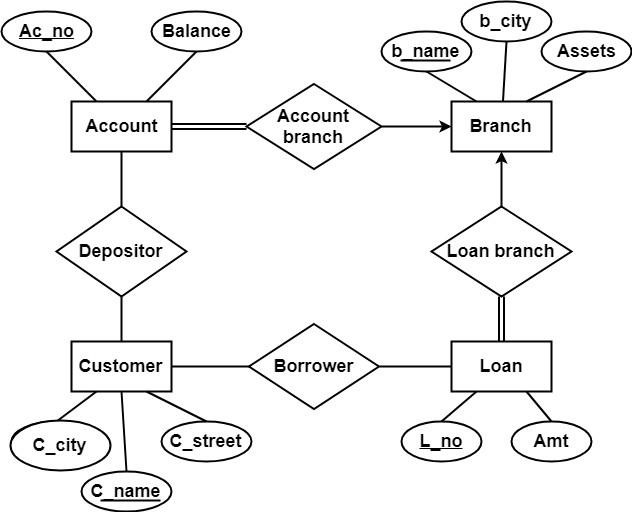
|  |  |  |  |
| --- | --- | --- | --- |
|  | **DATABASE MANAGEMENT SYSTEM LAB** |  |  |
| **Experiment No: 4** | **Group B: MySQL**  Design any database with at least 3 entities and relationships between them. Draw suitable ER/EER diagram for the system. | **Page** | **13/15** |

**TITLE**: Draw suitable ER/ EER diagram for the system

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

**Entity Relationship(ER) Diagrams:** ER Diagram stands for Entity Relationship Diagram, also known as ERD. It displays the relationship of entity sets stored in a database. ER Diagrams contain different symbols that use rectangles to represent entities, ovals to define attributes and diamond shapes to represent relationships.

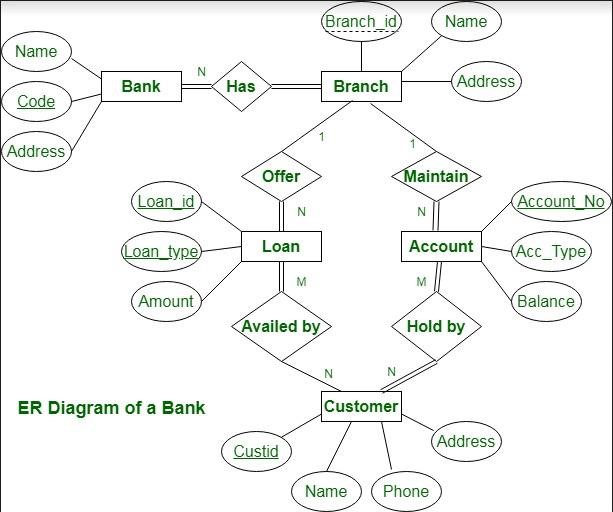
**EXAMPLE-1: ER DIAGRAM OF BANK MANAGEMENT SYSTEM**



**EXAMPLE-2: MORE DETAILED ER DIAGRAM OF BANK MANAGEMENT SYSTEM**

ER diagram of Bank has the following description:

* Bank have Customer.
* Banks are identified by a name, code, address of main office.
* Banks have branches.
* Branches are identified by a branch\_no., branch\_name, address.
* Customers are identified by name, cust-id, phone number, address.
* Customer can have one or more accounts.
* Accounts are identified by acc\_no., acc\_type, balance.
* Customer can avail loans.
* Loans are identified by loan\_id, loan\_type and amount.
* Account and loans are related to bank’s branch.



**Entities** and their **Attributes** are:

* **Bank Entity :** Attributes of Bank Entity are Bank Name, Code and Address. Code is Primary Key for Bank Entity.

* **Customer Entity :** Attributes of Customer Entity are Customer\_id, Name, Phone Number and Address. Customer\_id is Primary Key for Customer Entity.

* **Branch Entity :** Attributes of Branch Entity are Branch\_id, Name and Address. Branch\_id is Primary Key for Branch Entity.

* **Account Entity :** Attributes of Account Entity are Account\_number, Account\_Type and Balance. Account\_number is Primary Key for Account Entity.

* **Loan Entity :** Attributes of Loan Entity are Loan\_id, Loan\_Type and Amount. Loan\_id is Primary Key for Loan Entity.

[**Relationships a**](https://www.geeksforgeeks.org/attributes-to-relationships-in-er-model/)**re :**

* **Bank has Branches => 1 : N**

One Bank can have many Branches but one Branch can not belong to many Banks, so the relationship between Bank and Branch is one to many relationship.

* **Branch maintain Accounts => 1 : N**

One Branch can have many Accounts but one Account can not belong to many Branches, so the relationship between Branch and Account is one to many relationship.

* **Branch offer Loans => 1 : N**

One Branch can have many Loans but one Loan can not belong to many Branches, so the relationship between Branch and Loan is one to many relationship.

* **Account held by Customers => M : N**

One Customer can have more than one Accounts and also One Account can be held by one or more Customers, so the relationship between Account and Customers is many to many relationship.

* **Loan availed by Customer => M : N**

(Assume loan can be jointly held by many Customers). One Customer can have more than one Loans and also One Loan can be availed by one or more Customers, so the relationship between Loan and Customers is many to many relationship.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **DATABASE MANAGEMENT SYSTEM LAB** | |  |
| **Experiment No: 5** | **Group B: MySQL:** Database using DDL statements and apply normalization on them. | **Page** | **16/21** |

**TITLE**: Database using DDL statements

**OBJECTIVE: -** Database using DDL statements and apply normalization on them.

**THEORY:**

**Data Definition Language (DDL) Statements:**

DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database. The list of DDL queries include: i)Create ii)Insert iii)Alter iv)Delete v) Truncate vi)Drop vii)Rename viii)Commit ix)Grant x)Revoke

## **i) CREATE TABLE**

SQL> CREATE TABLE CUSTOMERS(ID INT NOT NULL, NAME VARCHAR (20) NOT

NULL, AGE INT NOT NULL, ADDRESS CHAR (25) ,SALARY DECIMAL (18, 2), PRIMARY KEY (ID));

ID is Primary Key. We can verify if your table has been created successfully by using the **DESC** command as follows –

|  |
| --- |
| SQL> DESC CUSTOMERS;  + + ++ + + +  | Field | Type | Null | Key | Default | Extra |  + + + + + + +  | ID | int(11) | NO | PRI | | |  | NAME | varchar(20) | NO | | | |  | AGE | int(11) | NO | | | |  | ADDRESS | char(25) | YES | | NULL | |  | SALARY | decimal(18,2) | YES | | NULL | |  + + + + + + +  5 rows in set (0.00 sec) |

## **ii) SQL INSERT INTO Statement**

The following statements would create six records in the CUSTOMERS table.

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (6, 'Komal', 22, 'MP', 4500.00 );

All the above statements will produce records in the CUSTOMERS table as shown below.

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

### **iii)** SQL - ALTER TABLE Command

Following is the example to DROP SALARY column from the existing table.

ALTER TABLE CUSTOMERS DROP SALARY;

The complete COLUMN SALARY with it’s values will be deleted

ALTER TABLE CUSTOMERS ADD RESULT char(10);;

The column RESULT will be added in the table

### **iv)** SQL - DELETE Query

Consider the CUSTOMERS table having the following records –

|  |
| --- |
| ++ + + + +  | ID | NAME | AGE | ADDRESS | SALARY |  ++ + + + +  | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |  | 2 | Khilan | 25 | Delhi | 1500.00 |  | 3 | kaushik | 23 | Kota | 2000.00 |  | 4 | Chaitali | 25 | Mumbai | 6500.00 |  | 5 | Hardik | 27 | Bhopal | 8500.00 |  | 6 | Komal | 22 | MP | 4500.00 |  | 7 | Muffy | 24 | Indore | 10000.00 |  ++ + + + + |

SQL> DELETE FROM CUSTOMERS

WHERE ID = 6;

Now, the CUSTOMERS table would have the following records.

|  |
| --- |
| ++ + + + +  | ID | NAME | AGE | ADDRESS | SALARY |  ++ + + + +  | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |  | 2 | Khilan | 25 | Delhi | 1500.00 |  | 3 | kaushik | 23 | Kota | 2000.00 |  | 4 | Chaitali | 25 | Mumbai | 6500.00 |  | 5 | Hardik | 27 | Bhopal | 8500.00 |  | 7 | Muffy | 24 | Indore | 10000.00 |  ++ + + + + |

**v)** SQL- TRUNCATE TABLE Command

All tuples will be deleted. Consider a CUSTOMERS table having the following records −

+++ + ++

| ID | NAME | AGE | ADDRESS | SALARY |

+++ + ++

|  |  |  |
| --- | --- | --- |
| | 1 | Ramesh | 32 | Ahmedabad | 2000.00 | | | + |
| | 2 | Khilan | 25 | Delhi  | 3 | kaushik | 23 | Kota  | 4 | Chaitali | 25 | Mumbai  | 5 | Hardik | 27 | Bhopal  | 6 | Komal | 22 | MP  | 7 | Muffy | 24 | Indore  +++ + | | 1500.00 |  | 2000.00 |  | 6500.00 |  | 8500.00 |  | 4500.00 |  | 10000.00 |  + |

SQL > TRUNCATE TABLE CUSTOMERS;

Now, the CUSTOMERS table is truncated and the output from SELECT statement will be as shown in the code block below –

SQL> SELECT \* FROM CUSTOMERS;

Empty set (0.00 sec)

### **vi)** SQL- DROPTable

|  |
| --- |
| SQL> DESC CUSTOMERS;  + + + + + + +  | Field | Type | Null | Key | Default | Extra |  + + + + + + +  | ID | int(11) | NO | PRI | | |  | NAME | varchar(20) | NO | | | |  | AGE | int(11) | NO | | | |  | ADDRESS | char(25) | YES | | NULL | |  | SALARY | decimal(18,2) | YES | | NULL | |  + + + + + + +  5 rows in set (0.00 sec) |

SQL> DROP TABLE CUSTOMERS;

Query OK, 0 rows affected (0.01 sec)

Table will be physically deleted

SQL> DESC CUSTOMERS;

ERROR 1146 (42S02): Table 'TEST.CUSTOMERS' doesn't exist

**vii)** SQL RENAME TABLE

**SQL RENAME TABLE** syntax is used to change the name of a table. Sometimes, we choose non- meaningful name for the table. So it is required to be changed. Let's see the syntax to rename a table from the database.

**ALTER TABLE** table\_name RENAME **TO** new\_table\_name;

### **viii)** SQL Commit Command

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

The syntax for the COMMIT command is as follows.

#### COMMIT;

Consider the CUSTOMERS table having the following records –

|  |
| --- |
| ++ + + + +  | ID | NAME | AGE | ADDRESS | SALARY |  ++++++  | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |  | 2 | Khilan | 25 | Delhi | 1500.00 |  | 3 | kaushik | 23 | Kota | 2000.00 |  | 4 | Chaitali | 25 | Mumbai | 6500.00 |  | 5 | Hardik | 27 | Bhopal | 8500.00 |  | 6 | Komal | 22 | MP | 4500.00 |  | 7 | Muffy | 24 | Indore | 10000.00 |  ++ + + + + |

Following is an example which would delete those records from the table which have age = 25 and then COMMIT the changes in the database.

SQL> DELETE FROM CUSTOMERS WHERE AGE = 25; SQL> COMMIT;

Thus, two rows from the table would be deleted and the SELECT statement will give result:

+++ + ++

| ID | NAME | AGE | ADDRESS | SALARY |

+++ + ++

|  |  |
| --- | --- |
| | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |  | 3 | kaushik | 23 | Kota | 2000.00 |  | 5 | Hardik | 27 | Bhopal | 8500.00 |  | 6 | Komal | 22 | MP | 4500.00 |  | 7 | Muffy | 24 | Indore | 10000.00 |  +++ + + | + |

### IX) SQL Grant Command

testdb=# CREATE USER manisha WITH PASSWORD 'password'; CREATE ROLE

The message CREATE ROLE indicates that the USER "manisha" is created. Consider the table [COMPANY h](https://www.tutorialspoint.com/postgresql/company.sql)aving records as follows –

testdb# select \* from COMPANY;

|  |
| --- |
| id | name | age | address | salary  ~~+~~ ++ +   1. | Paul | 32 | California| 20000 2. | Allen | 25 | Texas | 15000 3. | Teddy | 23 | Norway | 20000 4. | Mark | 25 | Rich-Mond | 65000 5. | David | 27 | Texas | 85000 6. | Kim | 22 | South-Hall| 45000 7. | James | 24 | Houston | 10000   (7 rows) |

Next, let us grant all privileges on a table COMPANY to the user "manisha" as follows –

testdb=# GRANT ALL ON COMPANY TO manisha; GRANT

### X) SQL Revoke Command

The message GRANT indicates that all privileges are assigned to the USER. Next, let us revoke the privileges from the USER "manisha" as follows –

testdb=# REVOKE ALL ON COMPANY FROM manisha; REVOKE

The message REVOKE indicates that all privileges are revoked from the USER. You can even delete the user as follows –

testdb=# DROP USER manisha;

DROP ROLE

The message DROP ROLE indicates USER ‘Manisha’ is

|  |  |  |  |
| --- | --- | --- | --- |
|  | **DATABASE MANAGEMENT SYSTEM LAB** | |  |
| **Experiment No: 5** | **Group B: MySQL:** Create Table with primary key and foreign key constraints. a. Alter table with add n modify b. Drop table | **Page** | **22/25** |

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

**OBJECTIVE: -** Create Table with primary key and foreign key constraints. a. Alter table with add n modify b. Drop table

**THEORY:**

**CREATE TABLE WITH PRIMARY KEY:**

SQL> CREATE TABLE CUSTOMERS(

ID INT NOT NULL,NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL, ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2), PRIMARY KEY (ID));

ID is Primary Key

|  |
| --- |
| SQL> DESC CUSTOMERS;  + + + + + + +  | Field | Type | Null | Key | Default | Extra | |
| + + + + + + +  | ID | int(11) | NO | PRI | | |  | NAME | varchar(20) | NO | | | |  | AGE | int(11) | NO | | | |  | ADDRESS | char(25) | YES | | NULL | |  | SALARY | decimal(18,2) | YES | | NULL | |  + + + + + + +  5 rows in set (0.00 sec) |

## **SQL INSERT INTO Statement**

**Example:**

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );

|  |  |  |
| --- | --- | --- |
|  | **PRAVARA RURAL ENGINEERING COLLEGE, LONI** |  |
| INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)  VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );    INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)  VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );    INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)  VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );    INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)  VALUES (6, 'Komal', 22, 'MP', 4500.00 ); | | |

Above statements give records in the CUSTOMERS table as shown below.

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

## **SQL - SELECT Query**

SQL> SELECT \* FROM CUSTOMERS;

This would produce the result as shown below.

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

## **SQL - ALTER TABLE Command**

Consider the CUSTOMERS table having the following records −

|  |
| --- |
| ++ + + + +  | ID | NAME | AGE | ADDRESS | SALARY |  + + + + + +  | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |  | 2 | Khilan | 25 | Delhi | 1500.00 |  | 3 | kaushik | 23 | Kota | 2000.00 |  | 4 | Chaitali | 25 | Mumbai | 6500.00 |  | 5 | Hardik | 27 | Bhopal | 8500.00 |  | 6 | Komal | 22 | MP | 4500.00 |  | 7 | Muffy | 24 | Indore | 10000.00 |  ++ + + + + |

Following is the example to DROP Salary column from the existing table.

ALTER TABLE CUSTOMERS DROP SALARY;

The complete COLUMN SALARY with it’s values will be deleted

ALTER TABLE CUSTOMERS ADD RESULT char(10);

The column RESULT will be added in the table

## **SQL-ALTER TABLE Command withADD**

ALTER TABLE Customers

ADD Email varchar(255);

+

+

| BeginnersBook

|

| AbcTemp

|

| Customers

|

| Student

|

| Faculty

|

| MyTest

|

| Demo

|

+

+

+

|

| Database

SQL>

SHOW

DATABASES;

+

SQL>

DROP

DATABASE

Employee

SQL

**DROP**

**DATABASE**

**PRAVARA RURAL ENGINEERING COLLEGE, LONI**

## **SQL FOREIGN KEY on CREATE TABLE**

A foreign key is a key used to link two tables together. This is sometimes also called as a referencing key.

A Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.

**The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table.**

If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).

Example:

Consider the structure of the following two tables.

**CUSTOMERS table**

CREATE TABLE CUSTOMERS(ID INT NOT NULL,NAME VARCHAR (20) NOT NULL, AGE INT NOT NULL, ADDRESS CHAR (25) ,SALARY DECIMAL (18, 2), PRIMARY KEY (ID));

**ORDERS table**

CREATE TABLE ORDERS (ID INT NOT NULL,DATE DATETIME,

CUSTOMER\_ID INT references CUSTOMERS(ID),AMOUNT double, PRIMARY KEY (ID));

**CUSTOMER\_ID** of ORDERS table matches with **ID** of CUSTOMERS table. This is set by **references** keyword i.e. **referencing key** i.e. **foreign key.**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **DATABASE MANAGEMENT SYSTEM LAB** | | |
| **Experiment No: 7** | **Group B: MySQL:** Perform Relational, Boolean, Arithmetic,  Set operators, Group Functions, Date, time, complex queries SQL queries | **Pages** | **26/29** |

**TITLE:** Relational, Boolean, Arithmetic, Set operators, Group and complex queries

**OBJECTIVE:**

Perform following SQL queries on the database:

* Implementation of relational operators in SQL
* Boolean operators and pattern matching
* Arithmetic operations and built in functions
* Group functions
* Processing Date and Time functions
* Complex queries and set operators

**THEORY:**

1. **SQL ARITHMETIC OPERATORS:**

+ (Addition), - (Subtraction), \* (Multiplication), / (Division), % (Modulus)

1. **SQL COMPARISON OPERATORS:**

=, !=, <>, >, <, >=, <=, !<, !>

1. **SQL LOGICAL OPERATORS:**

ALL, ANY, BETWEEN, EXISTS, IN, NOT, IS NULL, UNIQUE **iv)SQL: RELATIONAL OPERATORS: AND, OR**

Consider the CUSTOMERS table having the following records −

|  |
| --- |
| ++ + + + +  | ID | NAME | AGE | ADDRESS | SALARY |  ++ + + + +  | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |  | 2 | Khilan | 25 | Delhi | 1500.00 |  | 3 | kaushik | 23 | Kota | 2000.00 |  | 4 | Chaitali | 25 | Mumbai | 6500.00 |  | 5 | Hardik | 27 | Bhopal | 8500.00 |  | 6 | Komal | 22 | MP | 4500.00 |  | 7 | Muffy | 24 | Indore | 10000.00 |  ++ + + + + |

**The AND Operator**

SQL> SELECT ID, NAME, SALARY

|  |  |  |
| --- | --- | --- |
|  | **PRAVARA RURAL ENGINEERING COLLEGE, LONI** |  |
| FROM CUSTOMERS  WHERE SALARY > 2000 AND age < 25; | | |

This would produce the following result −

|  |
| --- |
| ++ + +  | ID | NAME | SALARY |  ++ + +  | 6 | Komal | 4500.00 |  | 7 | Muffy | 10000.00 |  ++ + + |

**The OR Operator**

SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE SALARY > 2000 OR age < 25;

This would produce the following result −

|  |
| --- |
| ++ + +  | ID | NAME | SALARY |  ++ + +  | 3 | kaushik | 2000.00 |  | 4 | Chaitali | 6500.00 |  | 5 | Hardik | 8500.00 |  | 6 | Komal | 4500.00 |  | 7 | Muffy | 10000.00 |  ++ + + |

**The ARITHMETIC Operators**

SQL> SELECT \* FROM CUSTOMERS

WHERE SALARY = SALARY + 1000;

|  |
| --- |
| ++ + + + +  | ID | NAME | AGE | ADDRESS | SALARY |  ++ + + + +  | 1 | Ramesh | 32 | Ahmedabad | 3000.00 |  | 2 | Khilan | 25 | Delhi | 2500.00 |  | 3 | kaushik | 23 | Kota | 3000.00 |  | 4 | Chaitali | 25 | Mumbai | 7500.00 |  | 5 | Hardik | 27 | Bhopal | 9500.00 |  | 6 | Komal | 22 | MP | 5500.00 |  | 7 | Muffy | 24 | Indore | 11000.00 |  ++ + + + + |

1. **SQL- GROUPBY** Employees

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **employee\_number** | **last\_name** | **first\_name** | **salary** | **dept\_id** |
| 1001 | Smith | John | 62000 | 500 |
| 1002 | Anderson | Jane | 57500 | 500 |
| 1003 | Everest | Brad | 71000 | 501 |
| 1004 | Horvath | Jack | 42000 | 501 |

|  |  |
| --- | --- |
|  | SELECT dept\_id, SUM(salary) AS total\_salaries  FROM employees  GROUP BY dept\_id; |

There will be 2 records selected. These are the results that you should see:

|  |  |  |
| --- | --- | --- |
| **dept\_id** | **total\_salaries** | **dept\_id** |
| 500 | 119500 | 500 |
| 501 | 113000 | 501 |

1. **SQL- LIKE CLAUSE FOR PATTERN MATCHING**

Let us take a real example, consider the CUSTOMERS table

|  |
| --- |
| ++ + + + +  | ID | NAME | AGE | ADDRESS | SALARY |  ++ + + + +  | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |  | 2 | Khilan | 25 | Delhi | 1500.00 |  | 3 | kaushik | 23 | Kota | 2000.00 |  | 4 | Chaitali | 25 | Mumbai | 6500.00 |  | 5 | Hardik | 27 | Bhopal | 8500.00 |  | 6 | Komal | 22 | MP | 4500.00 |  | 7 | Muffy | 24 | Indore | 10000.00 |  ++ + + + + |

Display all from the CUSTOMERS table, where the SALARY starts with 200.

SQL> SELECT \* FROM CUSTOMERS WHERE SALARY LIKE '200%';

This would produce the following result −

|  |
| --- |
| ++ + + + +  | ID | NAME | AGE | ADDRESS | SALARY |  ++ + + + +  | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |  | 3 | kaushik | 23 | Kota | 2000.00 |  ++ + + + + |

**Examples:**

The following table has a few examples showing the WHERE part having different LIKE clause with '%' and '\_' operators

|  |  |
| --- | --- |
| **Sr.No.** | **Statement & Description** |
| 1 | **WHERE SALARY LIKE '200%'**  Finds any values that start with 200. |
| 2 | **WHERE SALARY LIKE '%200%'**  Finds any values that have 200 in any position. |
| 3 | **WHERE SALARY LIKE '\_00%'**  Finds any values that have 00 in the second and third positions. |
| 4 | **WHERE SALARY LIKE '2\_%\_%'**  Finds any values that start with 2 and are at least 3 characters in length. |
| 5 | **WHERE SALARY LIKE '%2'**  Finds any values that end with 2. |
| 6 | **WHERE SALARY LIKE '\_2%3'**  Finds any values that have a 2 in the second position and end with a 3. |
| 7 | **WHERE SALARY LIKE '2 3'**  Finds any values in a five-digit number that start with 2 and end with 3. |

1. **SQL SET OPERATIONS**

**SQL- UNIONS CLAUSE**

|  |
| --- |
| SQL>select \* from hostel1  UNION  Select \* from hostel2’ |

**SQL INTERSECT Operation**

|  |
| --- |
| SQL>select \* from hostel1  INTERSECT  Select \* from hostel2’ |

**SQL MiNUS Operation**

|  |
| --- |
| SQL>select \* from hostel1  MINUS  Select \* from hostel2’ |

## **VIII)** SQL NESTED QUERIES OR COMPLEX QUERIES

Consider the CUSTOMERS table having the following records –

|  |
| --- |
| ++ + + + +  | ID | NAME | AGE | ADDRESS | SALARY |  + + + + + +  | 1 | Ramesh | 35 | Ahmedabad | 2000.00 |  | 2 | Khilan | 25 | Delhi | 1500.00 |  | 3 | kaushik | 23 | Kota | 2000.00 |  | 4 | Chaitali | 25 | Mumbai | 6500.00 |  | 5 | Hardik | 27 | Bhopal | 8500.00 |  | 6 | Komal | 22 | MP | 4500.00 |  | 7 | Muffy | 24 | Indore | 10000.00 |  ++ + + + + |

Now, let us check the following subquery with a SELECT statement.

SQL> SELECT \*

FROM CUSTOMERS

WHERE ID IN (SELECT ID FROM CUSTOMERS WHERE SALARY > 4500) ;

This would produce the following result.

|  |
| --- |
| ++ + + + +  | ID | NAME | AGE | ADDRESS | SALARY |  ++ + + + +  | 4 | Chaitali | 25 | Mumbai | 6500.00 |  | 5 | Hardik | 27 | Bhopal | 8500.00 |  | 7 | Muffy | 24 | Indore | 10000.00 |  ++ + + + + |

**IX) SQL DATE AND TIME**

* DATE - format YYYY-MM-DD
* DATETIME - format: YYYY-MM-DD HH:MI:SS
* TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
* YEAR - format YYYY or YY

**EXAMPLE :**

SELECT DATE("2017-06-15 09:34:21");

Result:

2017-06-15

SELECT \* FROM Orders WHERE OrderDate='2008-11-11'

|  |  |  |  |
| --- | --- | --- | --- |
|  | **DATABASE MANAGEMENT SYSTEM LAB** |  | |
| **Experiment No: 8** | **Group B: MySQL:** Execute DDL/DML statements which demonstrate the use of views. | **Pages** | **31/33** |

**TITLE:** Execute DDL/DML statements for Views

**OBJECTIVE:**

Execute DDL/DML statements which demonstrate the use of views.

**THEORY:**

**VIEWS IN DBMS**

Views in SQL are kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition.

**CREATING VIEWS**

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

Example

Consider the CUSTOMERS table having the following records –

|  |
| --- |
| ++ + + + +  | ID | NAME | AGE | ADDRESS | SALARY |  + + + + + +  | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |  | 2 | Khilan | 25 | Delhi | 1500.00 |  | 3 | kaushik | 23 | Kota | 2000.00 |  | 4 | Chaitali | 25 | Mumbai | 6500.00 |  | 5 | Hardik | 27 | Bhopal | 8500.00 |  | 6 | Komal | 22 | MP | 4500.00 |  | 7 | Muffy | 24 | Indore | 10000.00 |  ++ + + + + |

Following is an example to create a view from the CUSTOMERS table. This view would be used to have customer name and age from the CUSTOMERS table.

SQL > CREATE VIEW CUSTOMERS\_VIEW AS

SELECT name, age

FROM CUSTOMERS;

Now, you can query CUSTOMERS\_VIEW in a similar way as you query an actual table. Following is an example for the same.

SQL > SELECT \* FROM CUSTOMERS\_VIEW;

This would produce the following result.

+++

| name | age |

+++

| Ramesh | 32 |

| Khilan | 25 |

| kaushik | 23 | | Chaitali | 25 |

|  |  |
| --- | --- |
| | Hardik | | 27 | |
| | Komal | | 22 | |
| | Muffy | | 24 | |
| + | ++ |

**UPDATING A VIEW**

A view can be updated under certain conditions which are given below:

* The SELECT clause may not contain the keyword DISTINCT.
* The SELECT clause may not contain summary functions.
* The SELECT clause may not contain set functions.  The SELECT clause may not contain set operators.
* The SELECT clause may not contain an ORDER BY clause.
* The FROM clause may not contain multiple tables.
* The WHERE clause may not contain subqueries.
* The query may not contain GROUP BY or HAVING.
* Calculated columns may not be updated.
* All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So, if a view satisfies all the above-mentioned rules then you can update that view. The following code block has an example to update the age of Ramesh.

SQL > UPDATE CUSTOMERS\_VIEW

SET AGE = 35 WHERE name = 'Ramesh';

This will produce the following result.

+++ + ++

| ID | NAME | AGE | ADDRESS | SALARY |

+++ + ++

|  |  |  |
| --- | --- | --- |
| | 1 | Ramesh | 35 | Ahmedabad | 2000.00 | | | + |
| | 2 | Khilan | 25 | Delhi  | 3 | kaushik | 23 | Kota  | 4 | Chaitali | 25 | Mumbai  | 5 | Hardik | 27 | Bhopal  | 6 | Komal | 22 | MP  | 7 | Muffy | 24 | Indore  +++ + | | 1500.00 |  | 2000.00 |  | 6500.00 |  | 8500.00 |  | 4500.00 |  | 10000.00 |  + |

**INSERTING ROWS INTO A VIEW**

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command. Here, we cannot insert rows in the CUSTOMERS\_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in a similar way as you insert them in a table.

**DELETING ROWS INTO A VIEW**

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

Following is an example to delete a record having AGE = 22.

SQL > DELETE FROM CUSTOMERS\_VIEW WHERE age = 22;

It deletes a row from the base table CUSTOMERS and the same would reflect in the view itself.

+++ + ++

| ID | NAME | AGE | ADDRESS | SALARY |

+++ + ++

|  |  |
| --- | --- |
| | 1 | Ramesh | 35 | Ahmedabad | 2000.00 |  | 2 | Khilan | 25 | Delhi | 1500.00 |  | 3 | kaushik | 23 | Kota | 2000.00 |  | 4 | Chaitali | 25 | Mumbai | 6500.00 |  | 5 | Hardik | 27 | Bhopal | 8500.00 |  | 7 | Muffy | 24 | Indore | 10000.00 |  +++ + + | + |

**DROPPING VIEWS**

Following is an example to drop the CUSTOMERS\_VIEW from the CUSTOMERS table.

DROP VIEW CUSTOMERS\_VIEW;

|  |  |  |  |
| --- | --- | --- | --- |
|  | **DATABASE MANAGEMENT SYSTEM LAB** |  | |
| **Experiment No: 9** | **Group C: PL/SQL:** Write and execute PL/SQL stored procedure and function to perform a suitable task on the database. Demonstrate its use. | **Pages** | **34/34** |

**TITLE:** Write and execute PL/SQL stored procedure and function

**OBJECTIVE:**

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

**THEORY:**

**Function:**

Function, in computer programming language context, a set of instructions which takes some input and performs certain tasks. In PL/SQL, a function returns a value.

**Procedure:**

Procedure, as well, is a set of instructions which takes input and performs certain task. In SQL, procedure does not return a value.

**Example:**

DECLARE

a number; b number; c number;

PROCEDURE findMin(x IN number, y IN number, z OUT number) IS

BEGIN

IF x < y THEN z:= x;

ELSE

z:= y;

END IF;

END;

BEGIN

a:= 23; b:= 45; findMin(a, b, c);

dbms\_output.put\_line(' Minimum of (23, 45) : ' || c);

END;

/

**Output:** Minimum of (23, 45) :23

|  |  |  |  |
| --- | --- | --- | --- |
|  | **DATABASE MANAGEMENT SYSTEM LAB** |  | |
| **Experiment No: 10** | **Group C: PL/SQL:**Write and execute suitable database triggers .Consider row level and statement level triggers. | **Pages** | **35/36** |

**TITLE:** Write and execute suitable database triggers

**OBJECTIVE:**

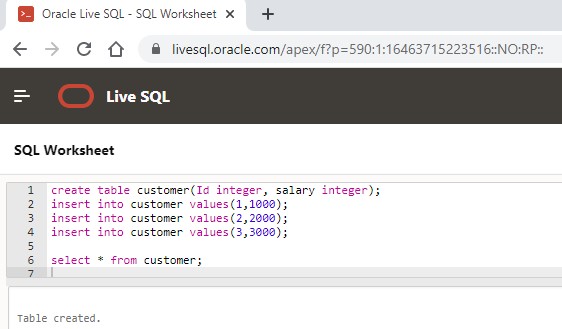
Write and execute suitable database triggers .Consider row level and statement level triggers.

**THEORY:**

**Trigger:** A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated. Trigger is **EVENT- CONDITION-ACTION** Model

**Example:**

To create-Insert-Select Queries using Live Oracle SQL



Similarly Trigger program is executed using LIVE ORACLE SQL:

**Example:**

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

CREATE OR REPLACE TRIGGER display\_salary\_changes

BEFORE DELETE OR INSERT OR UPDATE ON customers

FOR EACH ROW

WHEN (NEW.ID > 0)

DECLARE

sal\_diff number;

BEGIN

sal\_diff := :NEW.salary - :OLD.salary; dbms\_output.put\_line('Old salary: ' || :OLD.salary); dbms\_output.put\_line('New salary: ' || :NEW.salary); dbms\_output.put\_line('Salary difference: ' || sal\_diff);

END;

/

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

Trigger created.

UPDATE customers

SET salary = salary + 500

WHERE id = 2;

When a record is updated in the CUSTOMERS table, the above create trigger, **display\_salary\_changes** will be fired and it will display the following result :

Old salary: 1500

New salary: 2000

Salary difference: 500

|  |  |  |  |
| --- | --- | --- | --- |
|  | **DATABASE MANAGEMENT SYSTEM LAB** |  | |
| **Experiment No: 11** | **Group C: PL/SQL:** Write a PL/SQL block to implement all types of cursor. | **Pages** | **37/40** |

**TITLE:** Write a PL/SQL block to implement all types of cursor. To study implicit and explicit cursor

**OBJECTIVE:**

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

**THEORY:**

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**.

There are two types of cursors −  Implicit cursors

 Explicit 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.

In PL/SQL, you can refer to the most recent implicit cursor as the **SQL cursor**, which always has attributes such as **%FOUND, %ISOPEN, %NOTFOUND**, and **%ROWCOUNT**.

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

**Example:**

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

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

DECLARE

total\_rows number(2);

BEGIN

UPDATE customers

SET salary = salary + 500;

IF sql%notfound THEN dbms\_output.put\_line('no customers selected');

ELSIF sql%found THEN total\_rows := sql%rowcount;

dbms\_output.put\_line( total\_rows || ' customers selected '); END IF;

END;

/

6 customers selected PL/SQL procedure successfully completed.

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

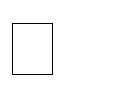
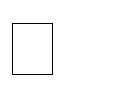
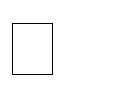
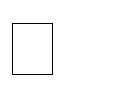
**EXPLICIT CURSORS:**

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

CURSOR cursor\_name IS select\_statement;

Working with an explicit cursor includes the following steps −

Declaring the cursor for initializing the memory



Opening the cursor for allocating the memory

Fetching the cursor for retrieving the data

Closing the cursor to release the allocated memory

**Declaring the Cursor**

Declaring the cursor defines the cursor with a name and the associated SELECT statement.

CURSOR c\_customers IS SELECT id, name, address FROM customers;

**Opening the Cursor**

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

OPEN c\_customers;

**Fetching the Cursor**

Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows –

FETCH c\_customers INTO c\_id, c\_name, c\_addr;

**Closing the Cursor**

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

CLOSE c\_customers;

**Example**

|  |
| --- |
| DECLARE  c\_id customers.id%type; c\_name customers.name%type; c\_addr customers.address%type;  CURSOR c\_customers is  SELECT id, name, address FROM customers;  BEGIN  OPEN c\_customers;  LOOP  FETCH c\_customers into c\_id, c\_name, c\_addr; EXIT WHEN c\_customers%notfound;  dbms\_output.put\_line(c\_id || ' ' || c\_name || ' ' || c\_addr);  END LOOP;  CLOSE c\_customers;  END;  / |

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

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | **DATABASE MANAGEMENT SYSTEM LAB** |  | |
| **Experiment No: 12** | **Group D:** Relational Database Design: To Design and case study of any organization (back end only), Project Proposal and High Level SRS. | **Pages** | **41/42** |

**TITLE:** To Design and case study of any organization (back end only), Project Proposal and High Level SRS.

**OBJECTIVE:** Design and case study of any organization (back end only), Project Proposal and High Level SRS. To prepare for your project, do the following:

1. Form teams of around 3 to 4 people
2. Create a requirements document with the following information:
   * Give a one or two paragraph description of your goals for the topic(s).
   * List all what all types of users will be accessing your application (e.g., for moodle, the types are teachers, students, teaching assistants, and a few more types).
3. List the various functionalities that your application will support. Explain each in about a paragraph worth of detail.
4. List the hardware and software requirements at the backend and at the front end.
5. Give an estimate of the number of users of each type, the expected load (transactions per day), and the expected database size.

**THEORY:**

**Software requirements specification (SRS):**

A SRS is a description of a software system to be developed. It lays out functional and non- functional requirements, and may include a set of use cases that describe user interactions that the software must provide.

**Guidelines to prepare SRS:**

Following points should be considered to form a structure of good SRS:

1. Introduction
   1. Purpose of this document
   2. Scope of this document
   3. Overview
2. General description
3. Functional Requirements
4. Interface Requirements
5. Performance Requirements
6. Design Constraints
7. Non-Functional Attributes
8. Preliminary Schedule and Budget
9. Appendices

The details are explained below:

1. **Introduction :**

**(i) Purpose of this Document:**

At first, main aim of why this document is necessary and what’s purpose of document is explained and described. **(ii) Scope of this document:**

In this, overall working and main objective of document and what value it will provide to customer and description of development cost, time required is described.

**(iii) Overview:**

In this, description of product is explained. It’s simply summary or overall review of product.

1. **General description:**

This includes objective of user, user characteristic, features and benefits, about why its importance is mentioned. It also describes features of user community.

1. **Functional Requirements:**

This includes possible outcome of software system including effects due to operation of program is fully explained. All functional requirements which may include calculations, data processing, etc. are placed in a ranked order.

1. **Interface Requirements:**

In this, software interfaces which mean how software program communicates with each other or users either in form of any language, code, or message are fully described and explained. Examples can be shared memory, data streams, etc.

1. **Performance Requirements:**

In this, how a software system performs desired functions under specific condition is explained. It also explains required time, required memory, maximum error rate, etc.

1. **Design Constraints:**

In this, constraints which simply means limitation or restriction are specified and explained for design team. E.g use of a particular algorithm, hardware and software limitations, etc.

1. **Non-Functional Attributes:**

In this, non-functional attributes are explained. E.g Security, Portability, Reliability, Reusability, Application compatibility, Data integrity, Scalability capacity, etc.

1. **Preliminary Schedule and Budget:**

In this, initial version and budget of project plan are explained which include overall time duration required and overall cost required for development of project.

1. **Appendices:**

It includes references from where information is gathered, definitions of some specific terms, acronyms, abbreviations, etc. are given and explained.