T.E Electronics & Telecommunication Engineering

Database Management

Lab manual

Department
Of
Electronics & Telecommunication Engineering
ARMY INSTITUTE OF TECHNOLOGY, PUNE

ARMY INSTITUTE OF TECHNOLOGY

LAB MANUAL

Name of the subject	Database Management Lab
Department	E&TC
Subject Code	304187
TW/OR/PR	OR
Marks	25
Course (Pattern)	2019
Year	2022-2023
Semester	First

List of Experiments

Sr. No	Title	Pg. No
1.	Study of Open Source Relational Databases : MySQL	
2.	Study of Design and Develop SQL DDL statements	
3.	Design SQL queries for suitable database application using SQL DML statements: Insert, Select, Update, Delete with operators, functions, and set operator.	
4.	Design SQL queries for suitable database application using SQL DML statements: all types of Join, Sub-Query and View.	
5.	Study of Pl SQL Control Structures and Exception Handling	
6.	To Study of all types of Cursor	
7.	Study of PL/SQL Stored Procedure and Stored Function.	
8.	Study all types of Database Trigger	

Additional List of Experiments
(Over and Above SPPU Syllabus)

Sr. No	Title	Pg. No
1.	Implementation of ANN using python –application-classification	
2.		

Pre Requisite for Lab

Companion Courses	
Code	**Subject**
	--
Prerequisite Courses/Lab	
1	Data Structures

Learning Objectives

Sr No	Objectives
1.	To understand fundamental concepts of database from its design to its implementation.
2.	To analyze database requirements and determine the entities involved in the system and with one another.
3.	To manipulate database using SQL Query to create, update and manage Database.
4.	Be familiar with the basic issues of transaction processing and concurrency control.
5.	To learn and understand Parallel Databases and its Architectures.
6.	To learn and understand Distributed Databases and its application

Precautions/Lab Safety Do's and Don'ts/Lab Etiquettes

Sr No	Details
	Do's
1.	Mobile Phones should be Switched OFF in the lab session.
2.	Maintain silence during lab sessions
3.	Carry a separate lab notebook for lab observations
4.	Turn ON the PCs and other instruments only when required
5.	Permission from staff is necessary for using Pen drives/CDs/Printers/Scanners
6.	Before leaving lab, shut down CPU and switch off monitor
	Don'ts
1.	Carry bags in the lab.
2.	Insert pen drive/cds in PCs.
3.	Remove LAN/ Ethernet cable from the machine
4.	Change IP address of the machine
5.	Disable firewall
6.	Install software without staff permission

Evaluation Guidelines

Grade	Poor	Average	Good	Outstanding
Marks	0-3	4-5	6-8	9-10

Criteria	**Grade**		**Marks**
Set-up and Equipment Care	Poor (0-3)	Set up of equipment is not accurate; help is required with several major details.	
	Average (4-5)	Set up equipment is generally workable with several details that need refinement.	
	Good (6-8)	Set up equipment is generally accurate with 1 or 2 small details that need refinement.	
	Outstanding (9-10)	All equipments accurately placed.	
Following Procedure	Poor (0-3)	Lacks appropriate knowledge of the lab procedures. Often requires help from the teacher to even complete basic procedure.	
	Average (4-5)	Demonstrates general knowledge of the lab procedures. Requires help from the teacher with some steps in procedures.	
	Good (6-8)	Demonstrates good knowledge of the lab procedures. Will discuss with peers to solve the problems in procedure.	
	Outstanding (9-10)	Demonstrates very good knowledge of the lab procedures. Gladly help other students to follow procedure.	
Data Collection	Poor (0-3)	Measurements are incomplete, inaccurate and imprecise. Observations are incomplete or not included. Symbols, units and significant figures are not included.	
	Average (4-5)	Measurements are somewhat inaccurate and imprecise. Observations are incomplete There are 3 or more minor errors using symbols, units and significant digits or 2 more errors.	
	Good (6-8)	Measurements are mostly accurate. Observations are generally complete. Work is organized. Only 2 or 3 minor errors using symbols, unit and significant digits.	
	Outstanding (9-10)	Measurements are both accurate and precise. Observations are very thorough and may recognized possible errors in data collection. Work is neat and organized. Include appropriate symbols, unit and significant digits.	
Analyzing and concluding	Poor (0-3)	Provide limited analysis of the data. Demonstrates limited ability to draw conclusions based on the data.	
	Average (4-5)	Provide some analysis of the data. Demonstrates some ability to draw conclusions based on data.	
	Good (6-8)	Provides sufficient analysis of the data. Draws valid conclusions based on the data.	
	Outstanding (9-10)	Provides rich analysis of the data. Draws insightful conclusion based on the data.	

Criteria	Grade		Marks
Safety	Poor (0-3)	Proper safety precautions are consistently missed. Needs to be reminded often during the lab.	
	Average (4-5)	Proper safety precautions are often missed. Needs to be reminded more than one during the lab.	
	Good (6-8)	Proper safety precautions are generally used. Use general reminders of safe practices independently.	
	Outstanding (9-10)	Proper safety precautions are consistently used. Consistently thinks ahead to ensure safety. Will often help other students to conduct lab safety.	
Timely Submission	Poor (0-3)	The write up submission is due more than two weeks.	
	Average (4-5)	The write up is submitted within two weeks.	
	Good (6-8)	The write up is submitted within one week.	
	Outstanding (9-10)	The write up is submitted on time.	
Specifications (Computer Program)	Poor (0-3)	The program is producing incorrect results	
	Average (4-5)	The program is produces correct results but does not display them correctly	
	Good (6-8)	The program is works and produces correct results and display them correctly. It also meets most of the other requirements	
	Outstanding (9-10)	The program works and meets all the requirements	
Readability (Computer Program)	Poor (0-3)	The code is poorly organised and very difficult to understand.	
	Average (4-5)	The code is readable only by someone who knows what it is supposed to be doing	
	Good (6-8)	The code is fairly easy to understand.	
	Outstanding (9-10)	The code is exceptionally well organised and very easy to follow.	
Reusability (Computer Program)	Poor (0-3)	The code is not organised for reusability	
	Average (4-5)	Some parts of the code could be reused in other programs	
	Good (6-8)	Most of the code could be reused in other programs	
	Outstanding (9-10)	The code could be reused as a whole or each routine could be reused in other programs	
Documentation (Computer Program)	Poor (0-3)	The documentation is simply comments embedded in the code and does not help the reader understand the code	
	Average (4-5)	The documentation is simply comments embedded in the code with some simple header comments separating routines	
	Good (6-8)	The documentation consists of embedded comment and some header documentation that	

Criteria	Grade		Marks
		is somewhat useful in understanding the code	
	Outstanding (9-10)	The documentation is well written and clearly explains what the code is accomplishing and how.	
Efficiency (Computer Program)	Poor (0-3)	The code is huge and appears to patched together.	
	Average (4-5)	The code is brute force and unnecessarily long.	
	Good (6-8)	The code is efficient without sacrificing readability and understanding.	
	Outstanding (9-10)	The code is extremely efficient without sacrificing readability and understanding.	

EXPERIMENT NO.1

AIM: Study of Open Source Relational Databases : MySQL

OBJECTIVES: Study of Open Source Relational Databases : MySQL

REQUIREMENTS:

- Windows-10
- mysql-essential-5.1.67-win32.msi
- mysql-gui-tools-5.0-r17-win32.msi
- mysql-workbench-gpl-5.2.44-win32.msi

THEORY:

MySQL is a Relational Management Database System(RDBMS), and ships with no GUI tools to administer MySQL databases or manage data contained within the databases. Users may use the included Command line tools, or use MySQL "front-ends", desktop software and web applications that create and manage MySQL databases, build database structures, back up data, inspect status, and work with data records. The official set of MySQL front-end tools, MySQL workbench is actively developed by Oracle, and is freely available for use.

Client server system has one or more client process and one or more server processes, and a client process can send a query to any one server process. Clients are responsible for user-interface issues,and servers manages data and execute transaction .Thus, a client process could run on a personal computer and send queries to a server to a server running on a mainframe

Commands for installation:

- Sudo apt-get update
- Sudo apt-get install mysql-server
- Mysql_secure_installation
- Sudo mysql –u root –p
- Enter Ubuntu Password and Mysql Password (root)

Steps For creating client server connectivity:

To connecting client and server in a network we have to follows the following steps :

Server side Commands:

1] > sudo -i

It is used to open file my.cnf

2] $ nano /etc/mysql/my.cnf

In this step we have to search #bind_address command and after finding this command comment that command then used following options

ctrl v : It is used to move next page.

ctrl o : It is used to save the changes.

ctrl x : It is used to exit from file

3] $ sudo service mySql restart;

4] $ mySql -u root -p;

It is used for login into mysql. then it is asked for the password. Enter the appropriate password.

5] mySql> grant all privileges on *.* to root@<client_IP> identified by "mysql" with grant option;

Client side Commands:

1] > mySql -h<server_IP> -u root -p;

This commands is used to login into mysql database then enter the correct password.

2] mySql>create user <username>@ '%' ;

This command used for client for cereating the llogin and password.

Example:

mySql>create TEComp@ '%';

where,

% is used for any computer which are in network can access server data.
and if not then insted of % we can also pass the IP address for particular client.

3] mySql> grant all on *.* to TEComp@'%' ;
Exit

4] To connecting to the server used the following command:
$ mySql -h <server_IP> -u TEComp -p;
After that enter the user created password
In this by following server client connectivity steps, we can eassily connect the client and server.

Database Creation commands

SQL> Show Databases;
SQL> create database Bank;
SQL> use Bank;

Components of SQL:

1) DDL (Data Definition Language)

-This SQL syntax is used to create, modify and delete database structures. DDL syntax cannot be applied to the manipulation of business data. DDL is almost always used by the database administrator, a database schema implementer or an application developer. Every DDL command implicitly issues a COMMIT making permanent all changes in the database.

Examples:

- **CREATE:** Create objects in database schema.
- **ALTER:** Alters the structure of objects that exist within the database schema.
- **DROP:** Drops objects that exist within database schema.
- **TRUNCATE:** Removes all records from a table, including all space allocated for the records.
- **COMMENT:**Adds comments ,generally used for proper documentation of a database schema.

2)DML(Data Manipulation Language)

-It is the SQL syntax that allows manipulating data within database tables.

Examples: INSERT, UPDATE, DELETE.

3)DCL(Data Control Language)

-It controls access to the database and table data. Occasionally DCL statements are grouped with DML statements.

Examples:

COMMIT,SAVEPOINT,ROLLBACK,SET TRANSACTION

- **The commands used in MySQL are:**

1) CREATE : The CREATE command is used to create a database or create a table in a particular database.

i) FOR CREATING A DATABASE :

Syntax:

create database database_name; //for creating a database

Example:

create database Student_info //Student_info database is created

ii)FOR CREATING A TABLE:

The table creation command requires:
Name of the table
Names of fields
Definitions for each field

Syntax:

CREATE TABLE table_name (column_name1 column_type, column_name2 column_type,.....);

Example:

create table Student(Roll_no tinyint PRIMARY KEY,Fname varchar(20) NOT NULL,Lname varchar(20),Mob_no char(10));

2)ALTER: MySQL**ALTER** command is used to change a name of your table, any table field or if you want to add or delete an existing column in a table.

Syntax:

i)DROP- Used to delete a particular column.

Syntax: mysql> ALTER TABLE table_name DROP i; //i is the row you want to delete.

ii)ADD-Used to add a particular column to an existing table.

Syntax:mysql> ALTER TABLE table_name ADD i int;

iii)CHANGE- Used to change a column's definition, use **MODIFY** or **CHANGE** clause along with ALTER command. After the CHANGE keyword, you name the column you want to change, then specify the new definition, which includes the new name

Syntax:

For example, to change column **c** from CHAR(1) to CHAR(10), do this:

mysql> ALTER TABLE table_name MODIFY c CHAR(10);
mysql> ALTER TABLE testalter_tbl CHANGE i j BIGINT;

3)DELETE:used to delete a record from any MySQL table, then you can use SQL command **DELETE FROM.**

Syntax:

DELETE FROM table_name [WHERE Clause]

INDEX: Indexing is the way of keeping table column data sorted so that searching and locating data consumes less time.Hence indexes essentially improve the speed at which records can be located and retrieved from a table.

Types of Index:

SIMPLE INDEX: An index created on single column data is called Simple index.

COMPOSITE INDEX: An index created on multiple column data is called a composite index.

1)**CREATE INDEX**: A database index is a data structure that improves the speed of operations in a table. Indexes can be created using one or more columns, providing the basis for both rapid random lookups and efficient ordering of access to records.

Syntax:

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

2)**DELETE INDEX**: Used to delete any index.

Syntax:

mysql> ALTER TABLE table_name DROP INDEX (c);

VIEWS: A view is a table whoes rows are not explicitly stored in the database but are

computed as needed from a view definition. To reduce redundant data to the minimum possible, MySQL allows creation of an object called a view .A view is mapped to a SELECT statement. This technique offers a simple, effective way of hiding columns of a table.
stud_info(name,sid,course)

FROM stud S,Enrolled E

1)**CREATE VIEW**: Used to create a view.
Syntax:
mysql> CREATE VIEW database_name.view_name AS SELECT * FROM table_name;
Example:
CREATE VIEW stud_info(name,sid,course)
AS SELECT S.name,S.sid,S.cid
FROM stud S,Enrolled E WHERE S.sid AND E.grade='B'

2)**DELETE VIEW**: DROP view removes one or more views.
Syntax:
mysql>DROP view viewname;
Example:
DROP view stud_info;

CONCLUSION:
Thus, we studied Open Source Relational Databases: MySQL successfully.

REFERENCES:

FREQUENTLY ASKED QUESTIONS

Q No	Questions	BT	CO
1			
2			
3			
4			

EXPERIMENT NO. 2

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

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

Problem Statement:
(Create following tables with constraints, alter table, insert, drop table , rename table, view, index, synonym, sequence/AUTO_INCREMENT)
Account(Acc_no, branch_name,balance)
branch(branch_name,branch_city,assets)
customer(cust_name,cust_street,cust_city)
Depositor(cust_name,acc_no)
Loan(loan_no,branch_name,amount)
Borrower(cust_name,loan_no)

REQUIREMENTS:

- Windows-10
- mysql-essential-5.1.67-win32.msi
- mysql-gui-tools-5.0-r17-win32.msi
- mysql-workbench-gpl-5.2.44-win32.msi

THEORY:
A schema is the collection of multiple database objects,which are known as schema objects.These objects have direct access by their owner schema.Below table lists the schema objects.

- Table - to store data
- View - to project data in a desired format from one or more tables
- Sequence - to generate numeric values
- Index - to improve performance of queries on the tables
- Synonym - alternative name of an object

One of the first steps in creating a database is to create the tables that will store an organization's data.Database design involves identifying system user requirements for various organizational systems such as order entry, inventory management, and accounts receivable. Regardless of database size and complexity, each database is comprised of tables.
Table of Contents
1. DDL 2. DML 3. DCL 4. TCL

DDL
DDL is short name of Data Definition Language, which deals with database schemas and descriptions, of how the data should reside in the database.

- CREATE - to create a database and its objects like (table, index, views, store procedure, function, and triggers)
- ALTER - alters the structure of the existing database
- DROP - delete objects from the database
- TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
- COMMENT - add comments to the data dictionary
- RENAME - rename an object

DML
DML is short name of Data Manipulation Language which deals with data manipulation and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE,

etc., and it is used to store, modify, retrieve, delete and update data in a database.

- SELECT - retrieve data from a database
- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - Delete all records from a database table
- MERGE - UPSERT operation (insert or update)
- CALL - call a PL/SQL or Java subprogram
- EXPLAIN PLAN - interpretation of the data access path
- LOCK TABLE - concurrency Control

DCL

DCL is short name of Data Control Language which includes commands such as GRANT and mostly concerned with rights, permissions and other controls of the database system.

- GRANT - allow users access privileges to the database
- REVOKE - withdraw users access privileges given by using the GRANT command

TCL

TCL is short name of Transaction Control Language which deals with a transaction within a database.

- COMMIT - commits a Transaction
- ROLLBACK - rollback a transaction in case of any error occurs
- SAVEPOINT - to rollback the transaction making points within groups
- SET TRANSACTION - specify characteristics of the transaction

---

SQL Statements For Tables

char(n). Fixed length character string, with user-specified length n.
varchar(n). Variable length character strings, with user-specified maximum length n.
int. Integer (a finite subset of the integers that is machine-dependent).
smallint. Small integer (a machine-dependent subset of the integer domain type).
numeric(p,d). Fixed point number, with user-specified precision of p digits, with n digits to the right of decimal point.
real, double precision. Floating point and double-precision floating point numbers, with machine-dependent precision.
float(n). Floating point number, with user-specified precision of at least n digits.

Constraints

Constraints are the set of rules defined in Oracle tables to ensure data integrity.These rules are enforced placed for each column or set of columns.Whenever the table participates in data action, these rules are validated and raise exception upon violation. The available constraint types are NOT NULL, Primary Key, Unique, Check, and Foreign Key.
The below syntax can be used to impose constraint at the column level.
Syntax:
column [data type] [CONSTRAINT constraint_name] constraint_type
All constraints except NOT NULL, can also be defined at the table level. Composite constraints can only be specified at the table level.

NOT NULL Constraint

A NOT NULL constraint means that a data row must have a value for the column specified as NOT NULL.If a column is specified as NOT NULL,the Oracle RDBMS will not allow rows to be stored to the employee table that violate this constraint.It can only be defined at column level, and not at the table level.
Syntax:
COLUMN [data type] [NOT NULL]

UNIQUE constraint

Sometimes it is necessary to enforce uniqueness for a column value that is not a primary key column.The UNIQUE constraint can be used to enforce this rule and Oracle will reject any rows that violate the unique constraint.Unique constraint ensures that the column values are distinct, without any duplicates.
Syntax:
Column Level:
COLUMN [data type] [CONSTRAINT <name>] [UNIQUE]
Table Level: CONSTRAINT [constraint name] UNIQUE (column name)
Note: Oracle internally creates unique index to prevent duplication in the column values.Indexes would be discussed later in PL/SQL.

```
CREATE TABLE TEST
( ... ,
  NAME VARCHAR2(20)
        CONSTRAINT TEST_NAME_UK UNIQUE,
  ... );
```

In case of composite unique key,it must be defined at table level as below.

```
CREATE TABLE TEST
( ... ,
  NAME VARCHAR2(20),
  STD VARCHAR2(20) ,
     CONSTRAINT TEST_NAME_UK UNIQUE (NAME, STD)
 );
```

Primary Key
Each table must normally contain a column or set of columns that uniquely identifies rows of data that are stored in the table.This column or set of columns is referred to as the primary key.Most tables have a single column as the primary key.Primary key columns are restricted against NULLs and duplicate values.
Points to be noted -

- A table can have only one primary key.
- Multiple columns can be clubbed under a composite primary key.
- Oracle internally creates unique index to prevent duplication in the column values.Indexes would be discussed later in PL/SQL.

Syntax:
Column level:
COLUMN [data type] [CONSTRAINT <constraint name> PRIMARY KEY]
Table level:
CONSTRAINT [constraint name] PRIMARY KEY [column (s)]
The following example shows how to use PRIMARY KEY constraint at column level.

```
CREATE TABLE TEST ( ID  NUMBER CONSTRAINT TEST_PK PRIMARY KEY,   ...  );
```

The following example shows how to define composite primary key using PRIMARY KEY constraint at the table level.

```
CREATE TABLE TEST ( ..., CONSTRAINT TEST_PK PRIMARY KEY (ID)  );
```

Foreign Key
When two tables share the parent child relationship based on specific column, the joining column in the child table is known as Foreign Key. This property of corresponding column in the parent table is known as Referential integrity. Foreign Key column values in the child table can either be null or must be the existing values of the parent table. Please note that only primary key columns of the referenced table are eligible to enforce referential integrity.
If a foreign key is defined on the column in child table then Oracle does not allow the parent row to be deleted, if it contains any child rows. However, if ON DELETE CASCADE

option is given at the time of defining foreign key, Oracle deletes all child rows while parent row is being deleted. Similarly, ON DELETE SET NULL indicates that when a row in the parent table is deleted, the foreign key values are set to null.
Syntax:

Column Level:

COLUMN [data type] [CONSTRAINT] [constraint name] [REFERENCES] [table name (column name)]

Table level:

CONSTRAINT [constraint name] [FOREIGN KEY (foreign key column name) REFERENCES] [referenced table name (referenced column name)]

The following example shows how to use FOREIGN KEY constraint at column level.
CREATE TABLE TEST (ccode varchar2(5) CONSTRAINT TEST_FK REFERENCES PARENT_TEST(ccode), …);

Usage of ON DELETE CASCADE clause
CREATE TABLE TEST (ccode varchar2(5) CONSTRAINT TEST_FK REFERENCES PARENT_TEST (ccode) ON DELETE CASCADE, …);

Check constraint
Sometimes the data values stored in a specific column must fall within some acceptable range of values. A CHECK constraint requires that the specified check condition is either true or unknown for each row stored in the table. Check constraint allows to impose a conditional rule on a column, which must be validated before data is inserted into the column. The condition must not contain a sub query or pseudo column CURRVAL NEXTVAL, LEVEL, ROWNUM, or SYSDATE.
Oracle allows a single column to have more than one CHECK constraint. In fact, there is no practical limit to the number of CHECK constraints that can be defined for a column.
Syntax:
Column level:
COLUMN [data type] CONSTRAINT [name] [CHECK (condition)]
Table level:
CONSTRAINT [name] CHECK (condition)
The following example shows how to use CHECK constraint at column level.
CREATE TABLE TEST (…, GRADE char (1) CONSTRAINT TEST_CHK CHECK (upper (GRADE) in ('A','B','C')), …);

The following example shows how to use CHECK constraint at table level.
CREATE TABLE TEST (…, CONSTRAINT TEST_CHK CHECK (stdate < = enddate),);

create database Student;
show databases;
use Student;
drop database Student;

DDL : create, desc, alter, drop, rename,

create table <table_name> (column_name1 dat_type(size) [constraint], column_name2 data_type(size) [constraint],….column_nameN dat_type(size) [constraint]);

create table Student_info(RollNO integer(10) NOT NULL, Name varchar(30), MobNo

integer(10));
desc student_info;
show tables;
drop table student_info;
alter table student_info add(emailid varchar(30));
alter table student_info modify(emailid char(10));
rename Student_info to Stud_data;
create index idx on student_info(rollno);
alter table student_info drop index idx;

index::
show databases;
use student;
show tables;
desc student_info;

Create table using subquery

A table can be created from an existing table in the database using a subquery option.It copies the table structure as well as the data from the table. Data can also be copied based on conditions.The column data type definitions including the explicitly imposed NOT NULL constraints are copied into the new table.
The below CTAS script creates a new table EMP_BACKUP. Employee data of department 20 gets copied into the new table
.

```
CREATE TABLE EMP_BACKUP
AS
SELECT * FROM EMP_TEST
WHERE department_id=20;
```

SQL CREATE INDEX Statement

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

Indexes are used to retrieve data from the database very fast. The users cannot see the indexes, they are just used to speed up searches/queries.

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

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

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

Example:

```
create index roll_no on student_info(rollno);
alter table student_info drop index roll_no;
```

CREATE UNIQUE INDEX Syntax

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

```
CREATE UNIQUE INDEX index_name
```

ON table_name (column1, column2, ...);

Note: The syntax for creating indexes varies among different databases. Therefore: Check the syntax for creating indexes in your database.
CREATE INDEX Example
The SQL statement below creates an index named "idx_lastname" on the "LastName" column in the "Persons" table:
CREATE INDEX idx_lastname
ON Persons (LastName);

If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
DROP INDEX Statement

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

MS Access:
DROP INDEX index_name ON table_name;

SQL Server:
DROP INDEX table_name.index_name;

DB2/Oracle:
DROP INDEX index_name;

MySQL:
ALTER TABLE table_name
DROP INDEX index_name;

SQL Sequence

Sequence is a feature supported by some database systems to produce unique values on demand. Some DBMS like MySQL supports **AUTO_INCREMENT** in place of Sequence. AUTO_INCREMENT is applied on columns, it automatically increments the column value by 1 each time a new record is entered into the table. Sequence is also some what similar to AUTO_INCREMENT but it has some extra features.

create table Student_info(RollNO integer(10) Primary key AUTO_INCREMENT, Name varchar(30), MobNo integer(10));

insert into Student_info (Name, MobNo) values ('Amol', 9049417616);
insert into Student_info (Name, MobNo) values ('Amol', 9049417616);

Creating Sequence

Syntax to create sequences is,
CREATE Sequence sequence-name
start with initial-value
increment by increment-value
maxvalue maximum-value
cycle|nocycle

Initial-value specifies the starting value of the Sequence, increment-value is the value by which sequence will be incremented and maxvalue specifies the maximum value until which sequence will increment itself. Cycle specifies that if the maximum value exceeds

the set limit, sequence will restart its cycle from the beginning. No cycle specifies that if sequence exceeds maxvalue an error will be thrown.

Example to create Sequence
The sequence query is following
CREATE Sequence seq_1
start with 1
increment by 1
maxvalue 999
cycle ;

Example to use Sequence

The class table,

ID	NAME
1	abhi
2	adam
4	alex

The sql query will be,

INSERT into class value(seq_1.nextval,'anu');

Result table will look like,

ID	NAME
1	abhi
2	adam
4	alex
1	anu

Once you use nextval the sequence will increment even if you don't Insert any record into the table.

CREATE SYNONYM:
Examples To define the synonym offices for the table locations in the schema hr, issue the following statement:

CREATE SYNONYM offices
  FOR hr.locations;

To create a PUBLIC synonym for the employees table in the schema hr on the remote database, you could issue the following statement:

CREATE PUBLIC SYNONYM emp_table
  FOR hr.employees@remote.us.oracle.com;
A synonym may have the same name as the underlying object, provided the underlying object is contained in another schema.

Oracle Database Resolution of Synonyms: Example Oracle Database attempts to resolve references to objects at the schema level before resolving them at the PUBLIC synonym level. For example, the schemas oe and sh both contain tables named customers. In the next example, user SYSTEM creates a PUBLIC synonym named customers for oe.customers:

CREATE PUBLIC SYNONYM customers FOR oe.customers;

If the user sh then issues the following statement, then the database returns the count of rows from sh.customers:

SELECT COUNT(*) FROM customers;
To retrieve the count of rows from oe.customers, the user sh must preface customers with the schema name. (The user sh must have select permission on oe.customers as well.)

SELECT COUNT(*) FROM oe.customers;
If the user hr's schema does not contain an object named customers, and if hr has select permission on oe.customers, then hr can access the customers table in oe's schema by using the public synonym customers:

SELECT COUNT(*) FROM customers;

SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.
CREATE VIEW Syntax
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
Note: A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.
SQL CREATE VIEW Examples

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

The view "Current Product List" lists all active products (products that are not discontinued) from the "Products" table. The view is created with the following SQL:
CREATE VIEW [Current Product List] AS
SELECT ProductID, ProductName
FROM Products
WHERE Discontinued = No;

Then, we can query the view as follows:
SELECT * FROM [Current Product List];

Another view in the Northwind sample database selects every product in the "Products" table with a unit price higher than the average unit price:
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, UnitPrice
FROM Products
WHERE UnitPrice > (SELECT AVG(UnitPrice) FROM Products);

We can query the view above as follows:
SELECT * FROM [Products Above Average Price];

Another view in the Northwind database calculates the total sale for each category in 1997. Note that this view selects its data from another view called "Product Sales for 1997":

```
CREATE VIEW [Category Sales For 1997] AS
SELECT DISTINCT CategoryName, Sum(ProductSales) AS CategorySales
FROM [Product Sales for 1997]
GROUP BY CategoryName;
```

We can query the view above as follows:

```
SELECT * FROM [Category Sales For 1997];
```

We can also add a condition to the query. Let's see the total sale only for the category "Beverages":

```
SELECT * FROM [Category Sales For 1997]
WHERE CategoryName = 'Beverages';
```

SQL Updating a View

You can update a view by using the following syntax:

SQL CREATE OR REPLACE VIEW Syntax

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

Now we want to add the "Category" column to the "Current Product List" view. We will update the view with the following SQL:

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

SQL Dropping a View

You can delete a view with the DROP VIEW command.

SQL DROP VIEW Syntax

```
DROP VIEW view_name;
```

Examples:

```
insert into student_info values(3,'amol',2154455,'abc@gmai');
insert into student_info values(1,'vijay',21543255,'asvs@gmai');
select * from student_info;
```

view::

```
create view myview as select rollno,name from student_info;
select * from myview;
create view myview2 as select rollno,name from student_info where rollno>1;
```

(Create following tables with constraints, alter table, insert, drop table , rename table, view, index, synonym, sequence/AUTO_INCREMENT)

Account(Acc_no, branch_name,balance)
branch(branch_name,branch_city,assets)
customer(cust_name,cust_street,cust_city)
Depositor(cust_name,acc_no)
Loan(loan_no,branch_name,amount)

Borrower(cust_name,loan_no)

Solve following queries:

Q1.Create Depositor table with foreign key with on delete cascade constraint on columns cust_name and acc_no.
Q2. Create Borrower table with foreign key with on delete cascade constraint on columns cust_name,loan_no.
Q3. Create Account table with primary key and AUTO_INCREMENT constraint on Acc_no column
Q4. Create Loan table with primary key and AUTO_INCREMENT constraint on loan_no column.
Q5. Create Customer table with primary key constraint on cust_name column.
Q6. Create View on Account table and Loan Table.
Q7. Insert following Data into above tables
Q.8. Create synonym for customer table as cust.
Q.9. Create sequence acc_seq and use in Account table for acc_no column.
Q.10 Insert following data into all above tables.

**********Problem Statements**********

Q1. Q1.Create Depositor table with foreign key with on delete cascade constraint on columns cust_name and acc_no.
Column Level:
SQL> create table depositor (cust_name varchar(20) CONSTRAINT FK_1 REFRENECS customer(cust_name) ON DELETE CASCADE , acc_no integer(10) CONSTRAINT FK_2 REFRENECS account(acc_no) ON DELETE CASCADE);

Q2. Create Borrower table with foreign key with on delete cascade constraint on columns cust_name,loan_no

Table level :
SQL> create table borrower (cust_name varchar(20), loan_no integer(10) , CONSTRAINT FK_1 FOREIGN KEY (cust_name) REFRENECS customer(cust_name) ON DELETE CASCADE, CONSTRAINT FK_2 FOREIGN KEY (loan_no) FK2 REFRENECS loan(loan_no) ON DELETE CASCADE);

Q3. Create Account table with primary key and AUTO_INCREMENT constraint on Acc_no column
SQL> create table account (acc_no integer(10) primary key AUTO_INCREMENT, branch_name varchar(20), balance integer(10));

Q4. Create Loan table with primary key and AUTO_INCREMENT constraint on loan_no column.
SQL> create table loan (loan_no integer(10) primary key AUTO_INCREMENT, branch_name varchar(20), amount integer(10));

Q5. Create Customer table with primary key constraint on cust_name column.
SQL> create table customer (cust_name varchar(20) primary key, cust_street varchar(20), city varchar(20));

Q6. Create View on Account table and Loan Table.

SQL> create view ac1 AS (select acc_no, balance from account);
SQL> create view ln1 AS (select loan_no, amount from loan);

Q.7. Create synonym for customer table as cust.
SQL> create public synonym cust2 for customer1;
Synonym created.

Q.8. Create sequence acc_seq and use in Account table for acc_no column.
CREATE Sequence seq_1 start with 1 increment by 1 maxvalue 100000 no cycle ;

Q.9 Insert following data into all above tables.

ACC_NO	BRANCH_NAME	BALANCE
1001	Akurdi	15000
1002	Nigdi	11000
1003	Chinchwad	20000
1004	Wakad	10000
1005	Akurdi	14000
1006	Nigdi	17000

********** **Table Structure** **********

create table Account(Acc_no, branch_name,balance) :

SQL> select * from account;

ACC_NO	BRANCH_NAME	BALANCE
1001	Akurdi	15000
1002	Nigdi	11000
1003	Chinchwad	20000
1004	Wakad	10000
1005	Akurdi	14000
1006	Nigdi	17000

6 rows selected.

Create table branch(branch_name,branch_city,assets) :

SQL> select * from branch;

BRANCH_NAME	BRANCH_CITY	ASSETS
Akurdi	Pune	200000
Nigdi	Pimpri_chinchwad	300000
Wakad	Pune	100000
Chinchwad	Pimpri_chinchwad	400000
Sangvi	Pune	230000

create table customer(cust_name,cust_street,cust_city) :

SQL> select * from customer1;

CUST_NAME	CUST_STREET	CUST_CITY
Rutuja	JM road	Pune

Alka	Senapati road	Pune
Samiksha	Savedi road	Pimpri_chinchwad
Trupti	Lakshmi road	Pune
Mahima	Pipeline road	Pimpri_chinchwad
Ayushi	FC road	pune
Priti	Camp road	Pimri_chinchwad

7 rows selected.

Create table Depositor(cust_name,acc_no):

SQL> select * from depositer;

CUST_NAME	ACC_NO
Rutuja	1005
Trupti	1002
Samiksha	1004

Loan(loan_no,branch_name,amount) :
SQL> select * from loan;

LOAN_NO	BRANCH_NAME	AMMOUNT
2001	Akurdi	2000
2002	Nigdi	1200
2003	Akurdi	1400
2004	Wakad	1350
2005	Chinchwad	1490
2006	Akurdi	12300
2007	Akurdi	14000

7 rows selected.

Create table borrower(cust_name,loan_no) :

SQL> select * from borrower;

CUST_NAME	LOAN_NO
Mahima	2005
Trupti	2002
Rutuja	2004
Ayushi	2006
Priti	2007

CONCLUSION: Thus we successfully implemented Table, View, Index, Sequence, Synonym MySQL queries.

REFERENCES:

FREQUENTLY ASKED QUESTIONS

Q No	Questions	BT	CO
1			
2			

Q No	Questions	BT	CO
3			
4			

EXPERIMENT NO. 3

AIM: To Design at least 10 SQL queries for suitable database application using SQL DML statements: Insert, Select, Update, Delete with operators, functions, and set operator.

OBJECTIVES: To Study and Design SQL queries for suitable database application using SQL DML statements: Insert, Select, Update, Delete with operators, functions, and set operator.

Problem Statement:
(Insert, Select, Update, Delete, operators, functions, setoperator, all constraints, view, index, synonym, sequence)
Account(Acc_no, branch_name,balance)
branch(branch_name,branch_city,assets)
customer(cust_name,cust_street,cust_city)
Depositor(cust_name,acc_no)
Loan(loan_no,branch_name,amount)
Borrower(cust_name,loan_no)

Input**:** insert Data into above tables and fire queries on databases;

REQUIREMENTS:

- Windows-10
- mysql-essential-5.1.67-win32.msi
- mysql-gui-tools-5.0-r17-win32.msi
- mysql-workbench-gpl-5.2.44-win32.msi

THEORY:
Set Operators:
The set operations union, intersect, and except operate on relations and correspond to the relational algebra operations È, Ç, -.
Each of the above operations automatically eliminates duplicates; to retain all duplicates use the corresponding multiset versions union all, intersect all and except all.

Suppose a tuple occurs m times in r and n times in s, then, it occurs:
m + n times in r **union all** s
min(m,n) times in r **intersect all** s
max(0, m – n) times in r **except all** s

Aggregate Functions:
These functions operate on the multiset of values of a column of a relation, and return a value

avg: average value
min: minimum value
max: maximum value
sum: sum of values
count: number of values

Solve following queries:
Q1. Find the names of all branches in loan relation.
Q2. Find all loan numbers for loans made at Akurdi Branch with loan amount > 12000.
Q3. Find all customers who have a loan from bank. Find their names,loan_no and loan amount.

Q4. List all customers in alphabetical order who have loan from Akurdi branch.
Q5. Find all customers who have an account or loan or both at bank.
Q6. Find all customers who have both account and loan at bank.
Q7. Find all customer who have account but no loan at the bank.
Q8. Find average account balance at Akurdi branch.
Q9. Find the average account balance at each branch
Q10. Find no. of depositors at each branch.
Q11. Find the branches where average account balance > 12000.
Q12. Find number of tuples in customer relation.
Q13. Calculate total loan amount given by bank.
Q14. Delete all loans with loan amount between 1300 and 1500.
Q15. Delete all tuples at every branch located in Nigdi.
Q.16. Create synonym for customer table as cust.
Q.17. Create sequence roll_seq and use in student table for roll_no column.

Create above tables with appropriate constraints like primary key, foreign key, check constrains, not null etc.

********** **Table Structure** **********

create table Account(Acc_no, branch_name,balance) :

SQL> select * from account;

ACC_NO	BRANCH_NAME	BALANCE
1001	Akurdi	15000
1002	Nigdi	11000
1003	Chinchwad	20000
1004	Wakad	10000
1005	Akurdi	14000
1006	Nigdi	17000

6 rows selected.

Create table branch(branch_name,branch_city,assets) :
SQL> select * from branch;

BRANCH_NAME	BRANCH_CITY	ASSETS
Akurdi	Pune	200000
Nigdi	Pimpri_chinchwad	300000
Wakad	Pune	100000
Chinchwad	Pimpri_chinchwad	400000
Sangvi	Pune	230000

create table customer(cust_name,cust_street,cust_city) :

SQL> select * from customer1;

CUST_NAME	CUST_STREET	CUST_CITY
Rutuja	JM road	Pune
Alka	Senapati road	Pune
Samiksha	Savedi road	Pimpri_chinchwad
Trupti	Lakshmi road	Pune
Mahima	Pipeline road	Pimpri_chinchwad

Ayushi	FC road	pune
Priti	Camp road	Pimri_chinchwad

7 rows selected.

Create table Depositor(cust_name,acc_no):

SQL> select * from depositer;

CUST_NAME	ACC_NO
Rutuja	1005
Trupti	1002
Samiksha	1004

Loan(loan_no,branch_name,amount) :

SQL> select * from loan;

LOAN_NO	BRANCH_NAME	AMMOUNT
2001	Akurdi	2000
2002	Nigdi	1200
2003	Akurdi	1400
2004	Wakad	1350
2005	Chinchwad	1490
2006	Akurdi	12300
2007	Akurdi	14000

7 rows selected.

Create table borrower(cust_name,loan_no) :

SQL> select * from borrower;

CUST_NAME	LOAN_NO
Mahima	2005
Trupti	2002
Rutuja	2004
Ayushi	2006
Priti	2007

**********Problem Statements**********

Q1. Find the names of all branches in loan relation.

SQL>select branch_name from loan;

BRANCH_NAME

Akurdi
Nigdi
Akurdi
Wakad
Chinchwad
Akurdi
Akurdi

7 rows selected.

Q2. Find all loan numbers for loans made at Akurdi Branch with loan amount >12000.

```
SQL> select loan_no from loan where branch_name='Akurdi' and amount>12000;

LOAN_NO
----------
2006
2007
```

Q3. Find all customers who have a loan from bank. Find their names, loan_no and loan amount.

```
SQL> select b.cust_name,b.loan_no,l.amount from borrower b inner join loan l on
b.loan_no=l.loan_no;

CUST_NAME          LOAN_NO          AMOUNT
-------------------- -------------- --------------
Trupti             2002             1200
Rutuja             2004             1350
Mahima             2005             1490
Ayushi             2006             12300
Priti              2007             14000
```

Q4. List all customers in alphabetical order who have loan from Akurdi branch.

```
SQL> select b.cust_name from borrower b inner join loan l on b.loan_no=l.loan_no
where l.branch_name='Akurdi'order by b.cust_name;

CUST_NAME
--------------------
Ayushi
Priti
```

Q5. Find all customers who have an account or loan or both at bank.

```
SQL>select cust_name from depositer union select cust_name from borrower;

CUST_NAME
--------------------
Ayushi
MahimaPriti
Rutuja
Samiksha
Trupti
6 rows selected.
```

Q6. Find all customers who have both account and loan at bank.

```
SQL> select cust_name from depositer intersect select cust_name from borrower;
CUST_NAME
--------------------
Rutuja
Trupti
```

Q7. Find all customer who have account but no loan at the bank.

```
SQL> select cust_name from depositer minus select cust_name from borrower;
```

```
CUST_NAME
--------------------
Samiksha
```

Q8. Find average account balance at Akurdi branch.

```
SQL> select avg(balance) from account where branch_name='Akurdi';

AVG(BALANCE)
------------
14500
```

Q9. Find the average account balance at each branch

```
SQL> select branch_name,avg(balance) from account group by branch_name;
BRANCH_NAME          AVG(BALANCE)
-------------------- --------------------
Chinchwad            20000
Nigdi                14000
Wakad                10000
Akurdi               14500
```

10. Find no. of depositors at each branch.

```
SQL> select branch_name,count(branch_name) from account a inner join depositer d on
a.acc_no=d.acc_no group by branch_name;
BRANCH_NAME          COUNT(BRANCH_NAME)
-------------------- ------------------------
Nigdi                                   1
Wakad                                   1
Akurdi                                  1
```

Q11. Find the branches where average account balance > 12000.

```
SQL> select branch_name from account group by branch_name having
avg(balance)>1200;
BRANCH_NAME
--------------------
Chinchwad
Nigdi
Wakad
Akurdi
```

Q12. Find number of tuples in customer relation.

```
SQL> select count(cust_name) no_of_tuples from customer1;
NO_OF_TUPLES
------------
7
```

Q13. Calculate total loan amount given by bank.

```
SQL> select sum(amount) total_loan_amount from loan;
TOTAL_LOAN_AMOUNT
-----------------
33740
```

Q14. Delete all loans with loan amount between 1300 and 1500.

```
SQL> delete from loan where amount>1300 and amount<1500;
```

LOAN_NO	BRANCH_NAME	AMOUNT
2001	Akurdi	2000
2002	Nigdi	1200
2006	Akurdi	12300
2007	Akurdi	14000

Q15. Delete all tuples at every branch located in Nigdi.
SQL>delete from branch where branch_name='Nigdi';

Q.16. Create synonym for customer table as cust.
SQL> create public synonym cust2 for customer1;
Synonym created.

Q.17. Create sequence roll_seq and use in student table for roll_no column.

CONCLUSION: Thus we successfully implemented MySQL queries.

REFERENCES:

FREQUENTLY ASKED QUESTIONS:

Sr. No.	Question	BT level	CO
1			
2			
3			

EXPERIMENT NO. 4

AIM: Design SQL queries for suitable database application

OBJECTIVES: Design SQL queries for suitable database application using SQL DML statements: all types of Join, Sub-Query and View.

1. Create following Tables
cust_mstr(cust_no,fname,lname)
add_dets(code_no,add1,add2,state,city,pincode)
Retrieve the address of customer Fname as 'xyz' and Lname as 'pqr'

2. Create following Tables
cust_mstr(custno,fname,lname)
acc_fd_cust_dets(codeno,acc_fd_no)
fd_dets(fd_sr_no,amt)
List the customer holding fixed deposit of amount more than 5000

3. Create following Tables
emp_mstr(e_mpno,f_name,l_name,m_name,dept,desg,branch_no)
branch_mstr(name,b_no)
List the employee details along with branch names to which they belong

4. Create following Tables
emp_mstr(emp_no,f_name,l_name,m_name,dept)
cntc_dets(code_no,cntc_type,cntc_data)
List the employee details along with contact details using left outer join & right join

5. Create following Tables
cust_mstr(cust_no,fname,lname)
add_dets(code_no,pincode)
List the customer who do not have bank branches in their vicinity.

6. a) Create View on borrower table by selecting any two columns and perform insert update delete operations
b) Create view on borrower and depositor table by selecting any one column from each table
perform insert update delete operations
c) create updateable view on borrower table by selecting any two columns and perform insert, update and delete operations.

REQUIREMENTS:

- Windows-10
- mysql-essential-5.1.67-win32.msi
- mysql-gui-tools-5.0-r17-win32.msi
- mysql-workbench-gpl-5.2.44-win32.msi

SOLUTIONS:
1. Create following Tables
cust_mstr(cust_no,fname,lname)
add_dets(code_no,add1,add2,state,city,pincode)

Retrieve the address of customer Fname as 'Rutuja' and Lname as 'Deshmane'

SQL> select * from cust_mstr;

```
CUSTNO     FNAME       LNAME
---------- ------------------- --------------------
C101       Rutuja              Deshmane
C102       Trupti              Bargaje
C103        Samiksha   Dharmadhikari
C104       Mahima      Khandelwal
```

SQL> select add1,add2 from add_dets where code_no in(select custno from cust_mstr where fname='Rutuja' and lname='Deshmane');

```
ADD1          ADD2
------------------- --------------------
venu nagar    dange chowk
```

2. Create following Tables
cust_mstr(custno,fname,lname)
acc_fd_cust_dets(codeno,acc_fd_no)
fd_dets(fd_sr_no,amt)

List the customer holding fixed deposit of amount more than 5000

SQL> select fname,lname from cust_mstr where custno in(select codeno from acc_fd_cust_dets where acc_fd_no in(select fd_sr_no from fd_dets where amt>5000));

```
FNAME       LNAME
------------------- --------------------
Rutuja              Deshmane
Samiksha    Dharmadhikari
```

3. Create following Tables
emp_mstr(e_mpno,f_name,l_name,m_name,dept,desg,branch_no)
branch_mstr(name,b_no)

List the employee details along with branch names to which they belong

SQL> select emp_no,fname,lname,mname,dept,desg,branch_no,b.name from emp_mstr e inner join branch_tb b on e.branch_no=b.b_no;

```
EMP_NO FNAME      LNAME           MNAME      DEPT DESG       BRANCH_NO
       NAME
-------- ---------- --------------- ---------- ------------ ---------- ----------- -------------------------
1011    Samarth    Deshmane        Suryakant  sports       trainer    2011
       Akurdi
1012   Alka        Choudhari       Rohitash   comp  tester 2012       nigdi
1013   Shriyash    Shingare        Santosh    comp  coder  2013       chinchwad
```

4. Create following Tables
emp_mstr(emp_no,f_name,l_name,m_name,dept)
cntc_dets(code_no,cntc_type,cntc_data)

List the employee details along with contact details using left outer join & right join

SQL> select emp_no,fname,lname,mname,dept,c.code_no,c.cntc_type,c.cntc_data from emp_mstr e left outer join cntc_dets c on e.emp_no=c.code_no;

EMP_NO	FNAME	LNAME	MNAME	DEPT	CODE_NO	CNTC_TYPE	CNTC_DATA
1011	Samarth	Deshmane	Suryakant	sports	1011	phno	9689349523
1012	Alka	Choudhari	Rohitash	comp	1012	email	rutu@gmail.com
1013	Shriyash	Shingare	Santosh	comp			

SQL> select emp_no,fname,lname,mname,dept,c.code_no,c.cntc_type,c.cntc_data from emp_mstr e right outer join cntc_dets c on e.emp_no=c.code_no;

EMP_NO	FNAME	LNAME	MNAME	DEPT	CODE_NO	CNTC_TYPE	CNTC_DATA
1011	Samarth	Deshmane	Suryakant	sports	1011	phno	9689349523
1012	Alka	Choudhari	Rohitash	comp	1012	email	rutu@gmail.com
1014						email	shrink@gmail.com

5. Create following Tables
cust_mstr(cust_no,fname,lname)
add_dets(code_no,pincode)

List the customer who do not have bank branches in their vicinity.

SQL> select * from cust_mstr where cust_no in (select code_no from add_dets where code_no like 'C%' and pincode not in (select pincode from add_dets where code_no like 'B%'));

CUST_NO	FNAME	LNAME
C102	Trupti	Bargaje

6. A) Create View on borrower table by selecting any two columns and perform insert update delete operations

SQL> select * from borrower;

ACC_NO	NAME	AMOUNT
101	Aish	10000
102	Adi	10000
103	Swati	45216

SQL> create view b1 as select name, amount from borrower;
View created.

SQL> select * from b1;

NAME	AMOUNT
Aish	10000
Adi	10000
Swati	45216

SQL> update b1 set amount=7845 where name='swati';
1 row updated.

SQL> select * from borrower;

ACC_NO	NAME	AMOUNT
101	Aish	10000
102	Adi	10000
103	Swati	7845

SQL> delete from b1 where name='swati';
1 row deleted.

SQL> select * from borrower;

ACC_NO	NAME	AMOUNT
101	Aish	10000

B) Create view on borrower and depositor table by selecting any one column from each table
perform insert update delete operations

SQL> select * from borrower;

ACC_NO	NAME	AMOUNT
101	Aish	10000
102	Adi	10000

SQL> select * from depositor;

DACC_NO	DNAME	DAMOUNT
102	Adi	45789
104	Sneha	7895
103	Swati	79854

SQL> create view b3 as select amount loan, damount deposit from borrower, depositor;
View created.

SQL> select * from b3;

LOAN	DEPOSIT
10000	45789
10000	7895
10000	79854

C) create updateable view on borrower table by selecting any two columns and perform insert,
Update and delete operations.

SQL> create table borrower(acc_no number(10) primary key,name varchar(10),amount number(10));

```
Table created.

SQL> insert into borrower values(&acc,'&name',&amount);
Enter value for acc: 101
Enter value for name: Aish
Enter value for amount: 10000
old 1: insert into borrower values(&acc,'&name',&amount)
new 1: insert into borrower values(101,'aish',10000)
1 row created.

SQL> /
Enter value for acc: 102
Enter value for name: Adi
Enter value for amount: 4500
old 1: insert into borrower values(&acc,'&name',&amount)
new 1: insert into borrower values(102,'adi',4500)
1 row created.

SQL> /
Enter value for acc: 103
Enter value for name: Swati
Enter value for amount: 45216.
old 1: insert into borrower values(&acc,'&name',&amount)
new 1: insert into borrower values(103,'swati',45216.)
1 row created.

SQL> create view bview as select acc_no, amount from borrower;
View created.

SQL> insert into bview values(&acc,&amount);
Enter value for acc: 104
Enter value for amount: 58901
old 1: insert into bview values(&acc,&amount)
new 1: insert into bview values(104,58901)
1 row created.


SQL> select * from borrower;

ACC_NO     NAME       AMOUNT
---------- ---------- ----------
101        aish       10000
102        adi        4500
103        swati      45216
104                   58901


SQL> update bview set amount=45000 where acc_no=104;
1 row updated.

SQL> select * from borrower;

ACC_NO     NAME       AMOUNT
---------- ---------- ----------
```

```
101          aish          10000
102          adi           4500
103          swati         45216
104                        45000

SQL> select * from bview;

ACC_NO       AMOUNT
----------    ----------
101          10000
102          4500
103          45216
104          45000

SQL> delete from bview where acc_no=104;
1 row deleted.

SQL> select * from bview;

ACC_NO       AMOUNT
----------    ----------
101          10000
102          4500
103          45216
```

CONCLUSION: Thus we successfully implemented MySQL queries.

REFERENCES:

FREQUENTLY ASKED QUESTIONS

Q No	Questions	BT	CO
1			
2			
3			
4			

EXPERIMENT NO. 5

AIM: Study of Pl SQL Control Structures and Exception Handling.

OBJECTIVES:

Input: Student roll no and attendance is input to Procedure**.**

REQUIREMENTS:

- Windows-10
- mysql-essential-5.1.67-win32.msi
- mysql-gui-tools-5.0-r17-win32.msi
- mysql-workbench-gpl-5.2.44-win32.msi

Theory:

The PL/SQL programming language was developed by Oracle Corporation in the late 1980s as procedural extension language for SQL and the Oracle relational database. Following are certain notable facts about PL/SQL −
PL/SQL is a completely portable, high-performance transaction-processing language.
PL/SQL provides a built-in, interpreted and OS independent programming environment.
PL/SQL can also directly be called from the command-line SQL*Plus interface.
Direct call can also be made from external programming language calls to database.
PL/SQL's general syntax is based on that of ADA and Pascal programming language.
Apart from Oracle, PL/SQL is available in TimesTen in-memory database and IBM DB2.

Features of PL/SQL

- PL/SQL has the following features −
- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data types.
- It offers a variety of programming structures.
- It supports structured programming through functions and procedures.
- It supports object-oriented programming.
- It supports the development of web applications and server pages.

Advantages of PL/SQL

- PL/SQL has the following advantages −
- SQL is the standard database language and PL/SQL is strongly integrated with SQL. PL/SQL supports both static and dynamic SQL. Static SQL supports DML operations and transaction control from PL/SQL block. In Dynamic SQL, SQL allows embedding DDL statements in PL/SQL blocks.
- PL/SQL allows sending an entire block of statements to the database at one time. This reduces network traffic and provides high performance for the applications.
- PL/SQL gives high productivity to programmers as it can query, transform, and update data in a database.
- PL/SQL saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object-oriented data types.
- Applications written in PL/SQL are fully portable.
- PL/SQL provides high security level.
- PL/SQL provides access to predefined SQL packages.
- PL/SQL provides support for Object-Oriented Programming.
- PL/SQL provides support for developing Web Applications and Server Pages.

DECLARE :- if you want to decalre a variable in plsql program then it takes place in declare section

BEGIN:- is used to start the working of program and end is used to terminate the begin. Delimiter is used to run (/)
SET SERVEROUTPUT ON ; is run before every time when you compiled a program in a session.
SET ECHO ON : is optional
DBMS_OUTPUT.PUT_LINE command for e.g. if sal=10 and you want to print it Then it looks like dbms_output.put_line('the salary is ' ||sal);

IF STATEMEN
Common syntax
IF condition THEN
statement 1;
ELSE
statement 2;
END IF;
INTO command: is used to catch a value in variable from table under some while condition
Only one value must be returned For e.g. in the above example if there are two people who's name is john then it shows error

Exception Handling:
An exception is an error condition during a program execution. PL/SQL supports programmers to catch such conditions using EXCEPTION block in the program and an appropriate action is taken against the error condition. There are two types of exceptions –

- System-defined exceptions
- User-defined exceptions

Syntax for Exception Handling
The general syntax for exception handling is as follows. Here you can list down as many exceptions as you can handle. The default exception will be handled using *WHEN others THEN* –

```
DECLARE
  <declarations section>
BEGIN
    <executable command(s)>
EXCEPTION
   <exception handling goes here >
  WHEN exception1 THEN
    exception1-handling-statements
  WHEN exception2  THEN
    exception2-handling-statements
  WHEN exception3 THEN
    exception3-handling-statements
  ........
  WHEN others THEN
    exception3-handling-statements
END;
```

Problem Statement:
Use of Control structure and Exception handling is mandatory. Write a PL/SQL block of code for the following requirements: Schema:
1. Borrower(Rollin, Name, DateofIssue, NameofBook, Status)
2. Fine(Roll_no,Date,Amt)

a) Accept roll_no & name of book from user.
b) Check the number of days (from date of issue), if days are between 15 to 30 then

fine amount will be Rs 5per day.
c) If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.
d) After submitting the book, status will change from I to R.
e) If condition of fine is true, then details will be stored into fine table.

Solution in Mysql:
Steps are:
1) Create borrower and fine table with primary and foreign keys
2) insert records in borrower table
3) create procedure to insert entries in fine table with exception handling
4) call procedure to calculate fine and display fine table.

mysql> create table borrower(rollin int primary key,name varchar(20),dateofissue date,nameofbook varchar(20),status varchar(20));
Query OK, 0 rows affected (0.30 sec)

```
mysql> desc borrower;
+-------------+-------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| rollin | int(11) | NO | PRI | NULL | |
| name | varchar(20) | YES | | NULL | |
| dateofissue | date | YES | | NULL | |
| nameofbook | varchar(20) | YES | | NULL | |
| status | varchar(20) | YES | | NULL | |
+-------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> create table fine(rollno int,foreign key(rollno) references borrower(rollin),returndate date,amount int);
Query OK, 0 rows affected (0.38 sec)

mysql> desc fine;
+------------+---------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+------------+---------+------+-----+---------+-------+
| roll_no | int(11) | YES | MUL | NULL | |
| returndate | date | YES | | NULL | |
| amnt | int(11) | YES | | NULL | |
+------------+---------+------+-----+---------+-------+
3 rows in set (0.02 sec)

mysql> insert into borrower values(1,'abc','2017-08-01','SEPM','PEN')$
Query OK, 1 row affected (0.16 sec)

mysql> insert into borrower values(2,'xyz','2017-07-01','DBMS','PEN')$
Query OK, 1 row affected (0.08 sec)

mysql> insert into borrower values(3,'pqr','2017-08-15','DBMS','PEN')$
Query OK, 1 row affected (0.03 sec)


mysql> delimiter $
mysql> create procedure calc_fine_lib6(in roll int)
```

```
begin
declare fine1 int;
declare noofdays int;
declare issuedate date;
declare exit handler for SQLEXCEPTION select'create table definition';
select dateofissue into issuedate from borrower where rollin=roll;
select datediff(curdate(),issuedate) into noofdays;
if noofdays>15 and noofdays<=30 then
set fine1=noofdays*5;
insert into fine values(roll,curdate(),fine1);
elseif noofdays>30 then
set fine1=((noofdays-30)*50) + 15*5;
insert into fine values(roll,curdate(),fine1);
else
insert into fine values(roll,curdate(),0);
end if;
update borrower set status='return' where rollin=roll;
end $

mysql> call calc_fine_lib6(1)$
Query OK, 0 rows affected (0.09 sec)
mysql> call calc_fine_lib6(2)$
Query OK, 0 rows affected (0.09 sec)
mysql> call calc_fine_lib6(3)$
Query OK, 0 rows affected (0.09 sec)

mysql> select * from fine;
-> $
+---------+------------+------+
| roll_no | returndate | amnt |
+---------+------------+------+
| 1 | 2017-08-22 | 105 |
| 2 | 2017-08-22 | 780 |
| 3 | 2017-08-22 | 0 |
+---------+------------+------+
3 rows in set (0.00 sec)

mysql>drop table fine$
Query OK, 0 rows affected (0.21 sec)

mysql> call calc_fine_lib6(1)$
+-------------------------+
| create table definition |
+-------------------------+
| create table definition |
+-------------------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> create table fine(rollno int,foreign key(rollno) references borrower(rollin),returndate
date,amount int)$
Query OK, 0 rows affected (0.34 sec)

mysql> call calc_fine_lib6(1)$
```

Query OK, 0 rows affected (0.09 sec)

```
mysql> select * from fine$
+--------+------------+--------+
| rollno | returndate | amount |
+--------+------------+--------+
| 1 | 2017-08-22 | 105 |
+--------+------------+--------+
1 row in set (0.00 sec)
```

Problem Statement: Consider table Stud(Roll, Att,Status)

Write a PL/SQL block for following requirement and handle the exceptions. Roll no. of student will be entered by user. Attendance of roll no. entered by user will be checked in Stud table. If attendance is less than 75% then display the message “Term not granted” and set the status in stud table as “D”. Otherwise display message “Term granted” and set the status in stud table as “ND”

Solution in Oracle:

```
SQL> create table stud1(roll_no number(5),attendance number(5),status varchar(7));
Table created.
SQL> select * from stud1;
ROLL_NO     ATTENDANCE       STATUS
----------      ----------              -------
101           80
102           65
103           92
104           55
105           68

SQL> set serveroutput on;
SQL>
declare
roll number(10);
att number(10);
 begin
 roll:=&roll;
 select attendance into att from stud1 where
roll_no=roll;  if att<75 then
 dbms_output.put_line(roll||'is detained');
 update stud1 set status='D' where roll_no=roll;
 else
 dbms_output.put_line(roll||'is not detained');
 update stud1 set status='ND' where roll_no=roll;
 end if;
exception
when no_data_found then
dbms_output.put_line(roll||'not found');
end;
/
Enter value for roll: 102
old  5: roll:=&roll;
new  5: roll:=102;
102is detained
PL/SQL procedure successfully completed.
```

```
SQL> /
Enter value for roll: 101
old  5: roll:=&roll;
new  5: roll:=101;
101is not detained
PL/SQL procedure successfully completed.
SQL> /
Enter value for roll: 103
old  5: roll:=&roll;
new  5: roll:=103;
103is not detained
PL/SQL procedure successfully completed.
SQL> /
Enter value for roll: 104
old  5: roll:=&roll;
new  5: roll:=104;
104is detained
PL/SQL procedure successfully completed.
SQL> /
Enter value for roll: 105
old  5: roll:=&roll;
new  5: roll:=105;
105is detained
PL/SQL procedure successfully completed.
SQL> select * from stud1;
```

ROLL_NO	ATTENDANCE	STATUS
101	80	ND
102	65	D
103	92	ND
104	55	D

CONCLUSION:

Thus we successfully implemented procedures.

EXPERIMENT NO. 6

AIM: To Study of all types of Cursor (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor)

Input: New roll calls and old roll calls

REQUIREMENTS:

- Windows-10
- mysql-essential-5.1.67-win32.msi
- mysql-gui-tools-5.0-r17-win32.msi
- mysql-workbench-gpl-5.2.44-win32.msi

THEORY:

Oracle creates a memory area, known as the **context area,** for processing an SQL statement, which contains all the information needed for processing the statement; for example, the number of rows processed, etc. 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**.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. 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.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

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

%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.

%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.

%ISOPEN

Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.

%ROWCOUNT

Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Explicit Cursors
Explicit cursors are programmer-defined cursors for gaining more control over the **context area**. An explicit cursor should be defined in the declaration section of the PL/SQL Block. 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. For example −
```
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;
```

To handle a result set inside a stored procedure, you use a cursor. A cursor allows you to iterate a set of rows returned by a query and process each row individually.
MySQL cursor is read-only, non-scrollable and asensitive.
- **Read-only**: you cannot update data in the underlying table through the cursor.
- **Non-scrollable**: you can only fetch rows in the order determined by the SELECT statement. You cannot fetch rows in the reversed order. In addition, you cannot skip rows or jump to a specific row in the result set.
- **Asensitive**: there are two kinds of cursors: asensitive cursor and insensitive cursor. An asensitive cursor points to the actual data, whereas an insensitive cursor uses a temporary copy of the data. An asensitive cursor performs faster than an insensitive cursor because it does not have to make a temporary copy of data. However, any change that made to the data from other connections will affect the data that is being used by an asensitive cursor, therefore, it is safer if you do not update the data that is being used by an asensitive cursor. MySQL cursor is asensitive.

Problem Statement :
Cursors: (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor)
Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N_RollCall with the data available in the table

O_RollCall. If the data in the first table already exist in the second table then that data should be skipped.

Solution in MySQL:

Steps are:

1) Create new_roll call and old_roll call tables
2) Insert records in both tables with few records duplication
3) create procedure and use cursor to merge above two tables to finalize roll list without duplication.

EXPERIMENT code:

```
mysql>create table new_roll(roll int,name varchar(10));
Query OK, 0 rows affected (0.29 sec)
mysql> create table old_roll(roll int,name varchar(10));
Query OK, 0 rows affected (0.28 sec)
mysql> insert into new_roll values(2,'b')$
Query OK, 1 row affected (0.05 sec)
mysql> insert into old_roll values(4,'d')$
Query OK, 1 row affected (0.05 sec)
mysql> insert into old_roll values(3,'bcd')$
Query OK, 1 row affected (0.04 sec)
mysql> insert into old_roll values(1,'bc')$
Query OK, 1 row affected (0.04 sec)
mysql> insert into old_roll values(5,'bch')$
Query OK, 1 row affected (0.04 sec)
mysql> insert into new_roll values(5,'bch')$
Query OK, 1 row affected (0.05 sec)
mysql> insert into new_roll values(1,'bc')$
Query OK, 1 row affected (0.04 sec)
mysql> select * from new_roll$
+------+------+
| roll | name |
+------+------+
| 2 | b |
| 4 | d |
| 5 | bch |
| 1 | bc |
+------+------+
4 rows in set (0.00 sec)

mysql> select * from old_roll$
+------+------+
| roll | name |
+------+------+
| 2 | b |
| 4 | d |
| 3 | bcd |
| 1 | bc |
| 5 | bch |
+------+------+
5 rows in set (0.00 sec)

delimiter $
create procedure roll_list()
begin
declare oldrollnumber int;
declare oldname varchar(10);
```

```
declare newrollnumber int;
declare newname varchar(10);
declare done int default false;
declare c1 cursor for select roll,name from old_roll;
declare c2 cursor for select roll,name from new_roll;
declare continue handler for not found set done=true;
open c1;
loop1:loop
fetch c1 into oldrollnumber,oldname;
if done then
leave loop1;
end if;
open c2;
loop2:loop
fetch c2 into newrollnumber,newname;
if done then
insert into new_roll values(oldrollnumber,oldname);
set done=false;
close c2;
leave loop2;
end if;
if oldrollnumber=newrollnumber then
leave loop2;
end if;
end loop;
end loop;
close c1;
end $
```

```
mysql> call roll_list()$
Query OK, 1 row affected (0.04 sec)

mysql> select * from new_roll$
+------+------+
| roll | name |
+------+------+
| 2 | b |
| 4 | d |
| 5 | bch |
| 1 | bc |
| 3 | bcd |
+------+------+
5 rows in set (0.01 sec)
```

Problem Statement 2: The bank manager has decided to activate all those accounts which were previously marked as inactive for performing no transaction in last 365 days. Write a PL/SQ block (using implicit cursor) to update the status of account, display an approximate message based on the no. of rows affected by the update. (Use of %FOUND, %NOTFOUND, %ROWCOUNT)

Solution in Oracle:

```
Declare
Rows_affe number(10);
Begin
```

```
update bankcursor set status='active'where
status='inactive'; Rows_affe:=(SQL%rowcount);
dbms_output.put_line(Rows_affe||' rows are
affected...');
END;
```

Solution :

SQL> **create table bankcursor(acc_no number(10),status varchar(10));**
Table created.

SQL> select * from bankcursor;

ACC_NO	STATUS
101	active
102	inactive
103	inactive
104	active
105	inactive

```
SQL>
Declare
Rows_affe number(10);
Begin
update bankcursor set status='active'where status='inactive';
Rows_affe:=(SQL%rowcount);
dbms_output.put_line(Rows_affe||' rows are
affected...');  END;
3 rows are affected...
PL/SQL procedure successfully completed.
```

SQL> select * from bankcursor;

ACC_NO	STATUS
101	active
102	active
103	active
104	active
105	active

Problem Statement 3: Organization has decided to increase the salary of employees by 10% of existing salary, who are having salary less than average salary of organization, Whenever such salary updates takes place, a record for the same is maintained in the increment_salary table.

EMP (E_no , Salary)
increment_salary(E_no ,
Salary) code:

Solution in Oracle:

```
Declare
Cursor crsr_sal is select e_no,salary from emp2 where salary<(select avg(salary) from
emp2);
me_no emp2.e_no%type;
```

```
msalary emp2.salary%type;

Begin
open crsr_sal;
if crsr_sal%isopen then
loop
fetch crsr_sal into me_no,msalary;
exit when crsr_sal%notfound;
if crsr_sal%found then
update emp2 set salary=salary+(salary*0.1) where
e_no=me_no; select salary into msalary from emp2 where
e_no=me_no; insert into increament_t
values(me_no,msalary); end if;
end loop;
end if;
end;

SQL> create table emp2(e_no number(10),salary number(10));
Table created.

SQL> select * from emp2;
          E_NO       SALARY
         ----------  ----------
101                     1000
102                     2000
103                     113
104                     40

SQL> create table increament_t(eno number(10),sal number(10));
Table created.

SQL>
Declare
Cursor crsr_sal is select e_no,salary from emp2 where salary<(select
avg(salary) from emp2);
me_no emp2.e_no%type;
msalary emp2.salary%type;

Begin
open crsr_sal;
if crsr_sal%isopen then
loop
fetch crsr_sal into me_no,msalary;
exit when crsr_sal%notfound;
if crsr_sal%found then
update emp2 set salary=salary+(salary*0.1) where
e_no=me_no; 14 select salary into msalary from emp2 where
e_no=me_no;
insert into increament_t values(me_no,msalary);
end if;
end loop;
end if;
end;

PL/SQL procedure successfully completed.
```

```
SQL> select * from emp2;
          E_NO       SALARY
         ----------  ----------
101                    1100
102                    2000
103                    113
104                    4000
SQL> select * from increament_t;

          ENO          SAL
         ----------   ----------
                      1100
             103      113
```

CONCLUSION: Thus we successfully implemented procedures.

REFERENCES:

FREQUENTLY ASKED QUESTIONS:

Sr. No.	Question	BT level	CO
1			
2			
3			

EXPERIMENT NO. 7

AIM: To Study of PL/SQL Stored Procedure and Stored Function.
Input: Students details and marks

REQUIREMENTS:

- Windows-10
- mysql-essential-5.1.67-win32.msi
- mysql-gui-tools-5.0-r17-win32.msi
- mysql-workbench-gpl-5.2.44-win32.msi

THEORY:

Procedure: A **subprogram** is a program unit/module that performs a particular task. These subprograms are combined to form larger programs. This is basically called the 'Modular design'. A subprogram can be invoked by another subprogram or program which is called the **calling program**.
A subprogram can be created −
At the schema level
Inside a package
Inside a PL/SQL block
At the schema level, subprogram is a **standalone subprogram**. It is created with the CREATE PROCEDURE or the CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement.
A subprogram created inside a package is a **packaged subprogram**. It is stored in the database and can be deleted only when the package is deleted with the DROP PACKAGE statement. We will discuss packages in the chapter '**PL/SQL - Packages**'.
PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of subprograms −
Functions − These subprograms return a single value; mainly used to compute and return a value.
Procedures − These subprograms do not return a value directly; mainly used to perform an action.
This chapter is going to cover important aspects of a **PL/SQL procedure**. We will discuss **PL/SQL function** in the next chapter.

Creating a Function:
A standalone function is created using the **CREATE FUNCTION** statement. The simplified syntax for the **CREATE OR REPLACE PROCEDURE** statement is as follows −

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

Where,
function-name specifies the name of the function.
[OR REPLACE] option allows the modification of an existing function.
The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
The function must contain a **return** statement.
The *RETURN* clause specifies the data type you are going to return from the function.

function-body contains the executable part.
The AS keyword is used instead of the IS keyword for creating a standalone function.

Calling a Function
While creating a function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. When a program calls a function, the program control is transferred to the called function.
A called function performs the defined task and when its return statement is executed or when the **last end statement** is reached, it returns the program control back to the main program.
To call a function, you simply need to pass the required parameters along with the function name and if the function returns a value, then you can store the returned value.

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

Problem Statement 1: PL/SQL Stored Procedure and Stored Function.
Write a Stored Procedure namely proc_Grade for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and900 category is first class, if marks 899 and 825 category is Higher Second Class
Write a PL/SQL block for using procedure created with above requirement.
Stud_Marks(name, total_marks) Result(Roll,Name, Class)

Solution in MySQL:
Steps:
1) create stud_marks and result table with primary and foreign keys
2)insert values in stud_marks
3)write and execute PL/SQL procedure for inserting grades in result table
EXPERIMENT is as follows:

```
mysql> desc stud_marks;
+-------------+-------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| name | varchar(20) | NO | PRI | NULL | |
| total_marks | int(11) | YES | | NULL | |
+-------------+-------------+------+-----+---------+-------+
2 rows in set (0.01 sec)

mysql> desc result;
+-------+-------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| roll | int(11) | YES | | NULL | |
| class | varchar(10) | YES | | NULL | |
| name | varchar(20) | YES | MUL | NULL | |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> insert into stud_marks values('abhijit',1020)$
```

```
Query OK, 1 row affected (0.15 sec)
mysql> insert into stud_marks values('anand',979)$
Query OK, 1 row affected (0.04 sec)
mysql> insert into stud_marks values('vijay',864)$
Query OK, 1 row affected (0.04 sec)
mysql> insert into stud_marks values('vikas',755)$
Query OK, 1 row affected (0.03 sec)

Create procedure proc_grade()
begin
declare done int default false;
declare roll int;
declare totmarks int;
declare class varchar(10);
declare name1 varchar(20);
declare c1 cursor for select name,total_marks from stud_marks;
declare continue handler for not found set done=true;
open c1;
set roll=1;
readloop:loop
fetch c1 into name1,totmarks;
if done then
leave readloop;
end if;
if totmarks<=1500 and totmarks>=990 then
insert into result values(roll,'dist',name1);
elseif totmarks<=989 and totmarks>=900 then
insert into result values(roll,'first',name1);
elseif totmarks<=899 and totmarks>=825 then
insert into result values(roll,'HSC',name1);
else
insert into result values(roll,'poor',name1);
end if;
set roll=roll+1;
end loop;
end $
mysql> call proc_grade()$
Query OK, 0 rows affected (0.38 sec)
mysql> select * from result$
+------+-------+---------+
| roll | class | name |
+------+-------+---------+
| 1 | dist | abhijit |
| 2 | first | anand |
| 3 | HSC | vijay |
| 4 | poor | vikas |
+------+-------+---------+
4 rows in set (0.00 sec)
```

Problem Statement 2. Write a PL/SQL stored Procedure for following requirements and call the procedure in appropriate PL/SQL block.

1. Borrower(Rollin, Name, DateofIssue, NameofBook, Status)
2. Fine(Roll_no,Date,Amt)

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

Solution :

```
SQL>create or replace function cal_fine(diffdate number) return number is
begin
if diffdate<15 then
return 0;
elsif diffdate<30 then
return (5*(diffdate-15));
else
return (50*(diffdate-30)+5*(15));end if;
end;
/
```

```
SQL> Declare
troll_no varchar(5);
tdays number(5);
tdate date;
diffdate number(5);
begin
troll_no := '&troll_no';
select to_date(sysdate,'DD-MM-YY')"Now" into tdate from dual;
select ((select to_date(sysdate,'DD-MM-YY')"Now" from dual)-dateofissue) into diffdate
from Borrower
where roll_no=troll_no;
insert into Fine values(troll_no,tdate,cal_fine(diffdate));
update borrower set status = 'R' where roll_no=troll_no;
End;
/

create function cal_fss(diffdate number) return number is
begin
if diffdate<15 then
return 0;
elsif diffdate<30 then
return (5*(diffdate-15));
else
return (50*(diffdate-30)+5*(15));
end if;
end ;

create table borrower(rollno number primary key, name varchar2(20), dateofissue date,
nameofbook varchar2(20), status varchar2(20));
create table fine(rollno number, foreign key(rollno) references borrower(rollno), returndate
date, amount number);

insert into borrower values(1,'abc',date '2021-06-01','SEPM','I');
insert into borrower values(2,'xyz',date '2021-05-01','OOP','I');
```

```
insert into borrower values(3,'pqr',date '2021-06-15','DBMS','I');
insert into borrower values(4,'def',date '2021-06-30','DSA','I');
insert into borrower values(5,'lmn',date '2021-07-05','ADS','I');


create procedure calc_fine_lib3(roll number) is
troll_no number(5);
tdays number(5);
tdate date;
diffdate number(5);
begin
troll_no := roll;
select to_date(sysdate,'DD-MM-YY')"Now" into tdate from dual;
select ((select to_date(sysdate,'DD-MM-YY')"Now" from dual)-dateofissue) into diffdate from
Borrower where rollno=troll_no;
insert into Fine values(troll_no,tdate,cal_fss(diffdate));
update borrower set status = 'R' where rollno=troll_no;
End;
```

CONCLUSION: Thus we have successfully implemented PL/SQL stored procedures.

REFERENCES:

FREQUENTLY ASKED QUESTIONS:

Sr. No.	Question	BT level	CO
1			
2			
3			

EXPERIMENT NO. 8

AIM: Database Triggers

OBJECTIBVES To study all types of Database Trigger (All Types: Row level and Statement level triggers, Before and After Triggers).

Input: Student library books information

REQUIREMENTS:

- Windows-10
- mysql-essential-5.1.67-win32.msi
- mysql-gui-tools-5.0-r17-win32.msi
- mysql-workbench-gpl-5.2.44-win32.msi

THEORY:

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events −

A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)

A **database definition (DDL)** statement (CREATE, ALTER, or DROP).

A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

- Triggers can be written for the following purposes −
- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating Triggers

The syntax for creating a trigger is −

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

Where,

CREATE [OR REPLACE] TRIGGER trigger_name − Creates or replaces an existing trigger with the *trigger_name*.

{BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The

INSTEAD OF clause is used for creating trigger on a view.
{INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
[OF col_name] − This specifies the column name that will be updated.
[ON table_name] − This specifies the name of the table associated with the trigger.
[REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
[FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

Problem Statement:
Database Trigger (All Types: Row level and Statement level triggers, Before and After Triggers).
Write a database trigger on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library_Audit table.

Solution in MySQL:
Steps are:
1) Create lib_audit and lib_audit2 tables
2) Insert records in lib_audit
3) create trigger for before update and before delete on lib_audit.

```
//Trigger for delete on lib_audit
mysql> create table lib_audit(bookid int,bookname varchar(20),price int)$
Query OK, 0 rows affected (0.58 sec)

mysql> create table lib_audit2(bookid int,bookname varchar(20),price int)$
Query OK, 0 rows affected (0.36 sec)

mysql> Create trigger before_delete_lib_audit before delete on lib_audit for each row
begin
insert into lib_audit2 values(old.bookid,old.bookname,old.price);
end$
Query OK, 0 rows affected (0.13 sec)

mysql> insert into lib_audit values(1,'ab',100)$
Query OK, 1 row affected (0.05 sec)
mysql> insert into lib_audit values(2,'cd',10)$
Query OK, 1 row affected (0.05 sec)
mysql> insert into lib_audit values(3,'dg',101)$
Query OK, 1 row affected (0.05 sec)

mysql> select * from lib_audit$
+--------+----------+-------+
| bookid | bookname | price |
+--------+----------+-------+
| 1 | ab | 100 |
| 2 | cd | 10 |
| 3 | dg | 101 |
+--------+----------+-------+
3 rows in set (0.00 sec)
```

```
mysql> select * from lib_audit2$
Empty set (0.00 sec)

mysql> delete from lib_audit where bookid=1$
Query OK, 1 row affected (0.14 sec)

mysql> select * from lib_audit$
+--------+----------+-------+
| bookid | bookname | price |
+--------+----------+-------+
| 2 | cd | 10 |
| 3 | dg | 101 |
+--------+----------+-------+
2 rows in set (0.00 sec)

mysql> select * from lib_audit2$
+--------+----------+-------+
| bookid | bookname | price |
+--------+----------+-------+
| 1 | ab | 100 |
+--------+----------+-------+
1 row in set (0.00 sec)

mysql> delete from lib_audit where bookid=3$
Query OK, 1 row affected (0.04 sec)

mysql> select * from lib_audit2$
+--------+----------+-------+
| bookid | bookname | price |
+--------+----------+-------+
| 1 | ab | 100 |
| 3 | dg | 101 |
+--------+----------+-------+
2 rows in set (0.00 sec)

//Trigger for update on lib_audit
mysql> Create trigger before_update_lib_audit before update on lib_audit for each row
begin
insert into lib_audit2 values(old.bookid,old.bookname,old.price);
end$

mysql> update lib_audit set bookname='xy' where bookid=2$
Query OK, 1 row affected (0.07 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> select * from lib_audit$
+--------+----------+-------+
| bookid | bookname | price |
+--------+----------+-------+
| 2 | xy | 10 |
+--------+----------+-------+
1 row in set (0.00 sec)
mysql> select * from lib_audit2$
+--------+----------+-------+
| bookid | bookname | price |
```

```
+--------+----------+-------+
| 1 | ab | 100 |
| 3 | dg | 101 |
| 2 | cd | 10 |
+--------+----------+-------+
3 rows in set (0.00 sec)
```

Problem Statement : Write a update, delete trigger on client mstr table. The System should keep track of the records that ARE BEING updated or deleted. The old value of updated or deleted records should be added in audit trade table. (separate implementation using both row and statement triggers).

Solution in Oracle:

Row trigger:

```
SQL> create or replace trigger t1 after update or delete on client_master 2
for each row
declare
op varchar(10);
begin
if updating then
op:='update';
end if;
if deleting then
op:='Delete';   end if;
into stat values(:old.id,op);
insert into audit_trade values(:old.id,:old.cname);
dbms_output.put_line('Details updated to stat and audit_trade table');
end;
/ Trigger created.
```

Statement Trigger:

```
SQL> create or replace trigger t1 after update or delete on client_master 2
for each row
declare
op varchar(10);
begin
if updating then
op:='update';
end if;
if deleting then
op:='Delete';   end if;
into stat values('',op);
insert into audit_trade values(:old.id,:old.cname);
dbms_output.put_line('Details updated to stat and audit_trade table');   end;
/ Trigger created.
```

Conclusion: Thus we have successfully implemented trigger.

GUIDELINES FOR STUDENTS:

Every experiment must be included in the file in the following format

1. Title, Aim, Theory, Algorithm/Procedure, Observations/Input-Output, Conclusion, Source Code
2. Diagram must be drawn on blank sheet.
3. Submit the write-up in the next practical session.

Marking Criteria

a) Experiment completion (Timely)
b) Lab file (neatness and regularity)
c) Viva (from time to time)
d) Mock Oral Exam
e) Exam(End term) : Viva

ASSESSMENT METHODOLOGY:

1. Attendance – 02 marks
2. Understanding – 03 marks
3. Timely Submission- 02 marks
4. Documentation & Presentation - 03 marks