SAVITRIBAI PHULE PUNE UNIVERSITY

S. Y. B. C. A. (Science)

Semester-III

BCA - 235

Database Management Systems – II
Laboratory Work Book

Name:____

College Name:____

Roll No.:____________________**Division:**______________________

Academic Year:__________________________

From the Chairman's Desk

It gives me a great pleasure to present this workbook prepared by the Board of studies in Computer Applications.

The workbook has been prepared with the objectives of bringing uniformity in implementation of lab assignments across all affiliated colleges, act as a ready reference for both fast and slow learners and facilitate continuous assessment using clearly defined rubrics.

The workbook provides, for each of the assignments, the aims, pre-requisites, related theoretical concepts with suitable examples wherever necessary, guidelines for the faculty/lab administrator, instructions for the students to perform assignments and a set of exercises divided into three sets.

I am thankful to the Chairman of this course and the entire team of editors. I am also thankful to the reviewers and members of BOS, Mr. Rahul Patil and Mr. Arun Gangarde. I thank all members of BOS and everyone who have contributed directly or indirectly for the preparation of the workbook.

Constructive criticism is welcome and to be communicated to the Chairman of the Course and overall coordinator Mr. Rahul Patil. Affiliated colleges are requested to collect feedbacks from the students for the further improvements.

I am thankful to Hon. Vice Chancellor of Savitribai Phule Pune University Prof. Dr. Nitin Karmalkar and the Dean of Faculty of Science and Technology Prof. Dr. M G Chaskar for their support and guidance.

Prof. Dr. S S Sane
Chairman, BOS in Computer Applications
SPPU, Pune

Editors:

Assignment 1	Mr. Deepak Derle	KTHM College, Nashik
Assignment 2	Mrs. Veena Gandhi	Abeda Inamdar Senior College, Pune
Assignment 3	Ms. Shripradа Chaturbhuj Mrs. Veena Gandhi Mr. Deepak Derle	Abeda Inamdar Senior College, Pune Abeda Inamdar Senior College, Pune KTHM College, Nashik.
Assignment 4	Ms. Shripradа Chaturbhuj	Abeda Inamdar Senior College, Pune
Assignment 5	Mrs. Rupali Nankar	Modern College, Ganeshkhind, Pune
Assignment 6	Ms. Suvarna Khairnar	S. N. Arts, D. J. M. Commerce and B.N.S. Science College, Sangamner

Compiled By:
Mrs. Veena Gandhi (Chairman, Database Management Systems-II Laboratory)
Abeda Inamdar Senior College, Pune.

Reviewed By:

1. Prof. Arun Gangarde
   New Arts, Commerce and Science College, Ahmednagar
   BOS,BCA(Science)

2. Prof. Rahul Patil
   KTHM College, Nashik.
   BOS(BCA Science)

Introduction

1. **About the workbook:**
   This workbook is intended to be used by S.Y.B.C.A. (Science) students for the BCA235 – Database Management Systems-II Laboratory Assignments in Semester–III. This workbook is designed by considering all the practical concepts / topics mentioned in syllabus.

2. **The objectives of this workbook are:**
   1) Defining the scope of the course.
   2) To bring the uniformity in the practical conduction and implementation in all colleges affiliated to SPPU.
   3) To have continuous assessment of the course and students.
   4) Providing ready reference for the students during practical implementation.
   5) Provide more options to students so that they can have good practice before facing the examination.
   6) Catering to the demand of slow and fast learners and accordingly providing the practice assignments to them.

3. **How to use this workbook:**
   The workbook is divided into 6 assignments. Each assignment has three SETs. It is mandatory for students to complete SET A, SET B and SET C in given slot.

Instructions to the students

Please read the following instructions carefully and follow them.

1) Students are expected to carry this book every time when they come to the lab for computer science practical.
2) Students should prepare themselves beforehand for the Assignment by reading the relevant material.
3) Instructor will specify which problems to solve in the lab during the allotted slot and student should complete them and get verified by the instructor. However student should spend additional hours in Lab and at home to cover as many problems as possible given in this work book.
4) Students will be assessed for each exercise on a scale from 0 to 5.

Not done	0
Incomplete	1
Late Complete	2
Needs improvement	3
Complete	4
Well Done	5

Guidelines for Instructors

1) Explain the assignment and related concepts in around ten minutes using whiteboard if required or by demonstrating the software.
2) You should evaluate each assignment carried out by a student on a scale of 5 as specified above by ticking appropriate box.
3) The value should also be entered on assignment completion page of the respective Lab course.

Guidelines for Lab administrator

You have to ensure appropriate hardware and software is available to each student in the Lab.

The operating system and software requirements on server side and also client side are as given below:

1) Server and Client Side - (Operating System)Linux (Ubuntu/Red Hat/Fedora) – any distribution
2) Database server – PostgreSQL 7.0 onwards.

Table of Contents

Assignment Completion Sheet

Lab Course: BCA235 – Database Management Systems – II Laboratory			
Sr. No.	**Assignment Name**	**Marks (out of 5)**	**Teachers Sign**
1	Nested query		
2	Views		
3	Stored functions		
4	Errors and Exception handling		
5	Cursors		
6	Triggers		
Total (Out of 30)			
Total (Out of 15)			

Certificate

This is to certify that

Mr. /Ms.______________________________________has successfully completed BCA235 – Database Management Systems-II Laboratory course in year______________and his/her seat no. is_________. He/she has scored Mark____ out of 15.

Instructor *H.O.D. /Coordinator*

Internal Examiner *External Examiner*

Assignment No. 1- Nested Query

Aim: To study execution of nested queries.

Pre-requisite:

- Knowledge of basic SQL.
- DDL, DML commands.
- Aggregate functions.
- Clauses like order by, group by.

Guidelines for Teachers / Instructors:

- Demonstration of creation of tables, use of Data Definition Language, Data Manipulation Language commands, clauses and nested queries.

Instructions for Students:

- Students must read the theory and syntax for creating tables, data types, use of constraints , syntax of clauses before his/her practical slot.
- Solve SET A, B or C assigned by instructor in allocated slots only.

Theory:

Structure of SQL:-

The basic structure of an SQL query consists of three clauses:

- ➢ select
- ➢ from
- ➢ where

The query takes input as the relations listed in the from clause, operates on them as specified in the where and select clauses, and then produces a relation as the output.

PostgreSQL-Data Types

A datatype specifies, what kind of data you want to store in the table field. While creating table, for each column, you have to use a datatype.PostgreSQL supports a wide variety of built-in data types, and it also provides an option to the users to add new data types to PostgreSQL by using the CREATE TYPE command. Table lists the data types officially supported by PostgreSQL. Most data types supported by PostgreSQL are directly derived from SQL standards. There are different categories of data types in PostgreSQL discussed below for your ready reference.

The following table contains PostgreSQL supported data types for your ready reference:

Type	**Data Type**	**Description**
Numeric Types	Smallint	2-byte small-range integer
	integer, int	A signed, fixed precision 4-byte
	Bigint	stores whole numbers, large range 8 byte
	Real	4-byte, single precision, floating-point number
	Serial	4-byte auto incrementing integer
	double precision	8-byte, double precision, floating-point number
	numeric(m,d)	Where m is the total digits and d is the number of digits after the decimal.
Character Types	character(n), char(n)	Fixed n-length character strings.
	character varying(n), varchar(n)	A variable length character string with limit.
	Text	A variable length character string of unlimited length.
Currency Types	Money	currency amount,8 bytes
Boolean type	Boolean	It specifies the state of true or false,1 byte.
Date/Time Type	Date	date (no time of day),4 byte.
	Time	time of day (no date),8 byte
	time with time zone	times of day only, with time zone,12 bytes

Instructions:-

1. Create a relational database in 3NF.
2. Insert sufficient number of records in the table.
3. Read the question carefully and insert data accordingly.
4. Count queries output should be more than 2records.
5. Empty output should not be generated.
6. Instructor should provide the data wherever is necessary to execute the query.

Exercises

SET A **(Number of Slot – 1)**

1) Student-Teacher Database

Consider the following Entities and their Relationships for Student-Teacher database.

Student (s_no int, s_name varchar (20), s_class varchar (10), s_addr varchar (30))

Teacher (t_no int, t_name varchar (20), qualification varchar (15), experience int)

Relationship between Student and Teacher is many to many with descriptive attribute subject.

Constraints: Primary Key,
s_class should not be null.

Solve the following Queries in PostgreSQL:

1. List the names of the teachers who are teaching to a student named "Avinash".
2. List the names of the students to whom'________________' is teaching.
3. List the details of all teachers whose names start with the alphabet 'T'.
4. List the names of teachers teaching subject 'DBMS'.
5. Find the number of teachers having qualification as 'Ph.D.'.
6. Find the number of students living in 'Cidco'.
7. Find the details of maximum experienced teacher.
8. Find the names of student of class 'SYBCA' and living at 'Kothrud'.
9. List name of students exactly contains 6 characters in it.
10. List the names of all teachers with their subjects along with the total number of students they are teaching.

2) Person-Area Database

Consider the following Entities and their Relationships for Person-Area database.

Person (pno integer, pname varchar (20), birthdate date, income money)

Area (aname varchar (20), area_type varchar (5))

An area can have one or more persons living in it, but a person belongs to exactly one area.

Constraints: Primary Key, area_type can be either 'urban' or 'rural'

Solve the following Queries in PostgreSQL:

1. List the names of all persons living in 'Pune' area.
2. List the details of all people whose names start with the alphabet 'D'.
3. Count area wise persons whose income is above______.
4. List the names of all people whose income is between_______and_________.

5. List the names of all people whose birthday falls in the month of August.
6. List names of persons whose income is same.
7. Display area wise maximum income of person.
8. Update the income of all people living in rural area by 10%.
9. Delete the record of person which has income below__________________.
10. Display person details in descending order of their names.

3) Book-Author Database

Consider the following Entities and their Relationships for Book-Author database.

Book(b_no int, b_name varchar (20), pub_name varchar (10), b_price float)
Author (a_no int, a_name varchar (20), qualification varchar (15), address varchar (15))

Relationship between Book and Author is many to many.

Constraints: Primary Key,pub_name should not be null.

Solve the following Queries in PostgreSQL:

1. List details of all books written by '_______________'
2. Count the number of books published by 'Nirali Publication'.
3. List book details for which book price is between 400.00 and 600.00.
4. List all author details sorted by their name in ascending order.
5. Change the publisher name from 'Niyati Publications' to 'Jagruti Publications'.
6. List the details of all books whose names start with the alphabet 'S'.
7. List author wise details of books.
8. Display details of authors who have written atleast 2 books.
9. List the details of all books written by author living in 'Nashik'.
10. Display details of authors who have written maximum number of books.

SET B **(Number of Slot – 1)**

1) Movie-Actor Database

Consider the following Entities and their Relationships for Movie-Actor database.
Movie (m_name varchar (25), release_year integer, budget money)
Actor (a_name varchar (20), role char(20), charges money, a_address varchar (20))
Producer (producer_id integer, p_name char (30), p_address varchar (20))

Each actor has acted in one or more movies. Each producer has produced many movies and each movie can be produced by more than one producers. Each movie has one or more actors acting in it, in different roles.

Constraints: Primary Key,role and p_name should not be null.

Solve the following Queries in PostgreSQL:

1. List the names of the actors and their movie names.
2. List the names of movies whose producer is 'Mr.Subhash Ghai'
3. Display details of the movies with the minimum budget.
4. List the names of movies released after year 2015.
5. Display count and total budget of all movies released in year2017.
6. List the names of actors who have acted in minimum number of movies.
7. List the names of movies produced by more than one producer.
8. List the names of actors who played the role of 'Villan'.
9. List the names of actors who have acted in at least one movie, in which 'Mr. Ritesh Deshmukh' has acted.
10. Display total number of actors acted in movie 'Sholey'.

2) Bank database

Consider the following Entities and their Relationships for Bank database.

Branch (<u>bid</u> integer, brname char (30), brcity char (10))

Customer (<u>cno</u> integer, cname char (20), caddr char (35), city char(20))

Loan_application (<u>lno</u> integer, l_amt_require money, l_amt_approved money, l_date date)

The relationships are as follows:

Branch, Customer, Loan_application are related with ternary relationship.

Ternary (bid integer, cno integer, lno integer).

Solve the following queries in PostgreSQL

1. Find out the total loan amount approved at 'Nagar' Branch.
2. Find the names of customers for the 'Karve Nagar' branch.
3. List the names of customers who have taken loan from the branch in the same city they live.
4. List the names of customers who have received loan less than their requirement.
5. List the names of the customer along with the branch names who have applied for loan in the month of ________.
6. Find the names of customers who required loan with amount > 100000.
7. Find the maximum loan amount approved.
8. Count number of customers from 'ShivajiNagar' branch.
9. List names of customers requesting for loan amount in between 200000 and 400000.
10. List branch wise name of customers.

<u>SET C</u> **(Number of Slot – 1)**

1) **Warehouse database**.

Consider the following Entities and their Relationships for Warehouse database.

Cities (city char(20),state char(20))

Warehouses (<u>wid</u> integer, wname char(30),location char(20))

Stores (<u>sid</u> integer , store_name char(20), location_city char(20))

Items (<u>itemno</u> integer, description text, weight decimal(5,2), cost decimal(5,2))

Customer (cno integer, cname char(50),addr varchar(50), cu_city char(20))

Orders (ono int,odate date)

The Relationship is as follows

Cities-warehouses 1 to m

Warehouses-stores 1 to m

Customer– orders 1 to m

Items– orders m - m relationship with descriptive attribute ordered_quantity

Stores-items m - m relationip with descriptive attribute quantity.

Solve the following queries in PostgreSQL.

1. Find the different warehouses in "Nagar".
2. Find the item that has minimum weight.
3. Delete the orders placed by "Mr.Patil."
4. Find the details of items ordered by a customer "Mr.Baviskar".
5. Find a Warehouse which has maximum stores.
6. Find an item which is ordered for minimum number of times.
7. Find the details orders given by each customer.

2) Bus Transport System

Consider the following database of Bus transport system. Many buses run on one route. Drivers are allotted to the buses shift-wise.

Following are the tables:

BUS(bus_no int , capacity int , depot_name varchar(20))
ROUTE (route_no int, source char(20), destination char(20),no_of_stations int)
DRIVER (driver_no int , driver_name char(20), license_no int, address char(20), d_age int , salary float)

The relationships are as follows:

BUS_ROUTE: M-1
BUS_DRIVER: M-M with descriptive attributes Date_of_duty_allotted and Shift – it can be 1 (Morning) 0r 2 (Evening).

Constraints:

1) License_no is unique.
2) Bus capacity is not null.

Solve the following queries in PostgreSQL.

1. Find out the drivers working in shift 2.
2. Find out the route details on which buses of capacity 20 runs.
3. Increase the salary of all drivers by 5% if driver's age> 40.
4. Find out the driver's name working on both shifts.

5. Find out the name of the driver having maximum salary.
6. Delete the record of bus having capacity < 10
7. Print the names & license nos. of drivers working on both shifts.

0: Not Done [ ]	1: Incomplete [ ]	2:Late Complete[ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of the instructor:______________________Date:______________________

Assignment No. 2 – Views

Aim: To study creating, execution and dropping of views.

Pre-requisite: Knowledge of simple SQL and Nested Queries must.

Guidelines for Teachers / Instructors:

- Demonstration of creation of views, execution and dropping of views on database is expected.

Instructions for Students:

- Students must read the theory and syntax for creating and dropping views before his/her practical slot.
- Solve SET A,B or C assigned by instructor in allocated slots only.

Concept

1. Views are imaginary tables, they are not real tables. A view can contain all rows of a table or selected rows from one or more tables. We can use views to restrict table access so that the users see only specific rows and columns of the tables. When we create a view, we basically create a query and assign it a name, so it is basically used to wrap complex query. A normal view does not store any data so we cannot able to execute a DELETE, INSERT, or UPDATE statement on a view.

Creating View:

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

Syntax for creating view:

```
CREATE [or REPLACE] [TEMP|TEMPORARY] VIEW view_name AS
 SELECT column1, column2….. FROM table_name
WHERE [condition];
```

Parameters Used:

view_name: The name of a view to be created.

1. **TEMPORARY or TEMP:** If the optional TEMP or TEMPORARY keyword is present, the view will be created in the temporary space. Temporary views are automatically dropped at the end of the current session.
2. **column_name:** Name of the columns should be displayed in view.
3. **table_name:** Name of the table from which data should be fetched.
4. **Condition:** According to the condition data fetched from the table.

Example:

- **List the details of all employees having designation as 'Manager'.**

```
CREATE VIEW employee_view AS
SELECT *
FROM employee
WHERE designation = 'Manager';
```

This will create a view containing the columns that are in the employee table at the time of view creation. Now we can query employee_view in similar way as we query an actual table to see the data.

- ***To see the data from view:***

Syntax: select * from view_name;

For Example: select * from employee_view;

The above query displays the record of only those employees whose designation is 'Manager' from view.

- ***DroppingView:***

To delete a view, simply use the Drop view statement with the view name.

Syntax: DROP VIEW view_name;

For Example : DROP VIEW employee_view;

Exercises

SET A **(Number of Slots – 1)**

1) Project-Employee Database

Consider the following Entities and their Relationships for Project-Employee database.

Project (pno integer, pname char (30), ptype char (20), duration integer)

Employee (eno integer, ename char (20), qualification char (15), joining_date date)

Relationship between Project and Employee is many to many with descriptive attribute start_date date, no_of_hours_worked integer.

Constraints: Primary Key, duration should be greater than zero,
pname should not be null.

Create view for the following:

1. Display all employees working on "ERP" Project.
2. Display the project details and start date of the project, sort it by start date of

the project where duration of project is more than 6 months.
3. Display employee details having qualification MCA.
4. Display employee and project names where employees worked more than 300 hours.

2) Person-Area Database

Consider the following Entities and their Relationships for Person-Area database.
Person (pno integer, pname varchar (20), birthdate date, income money)
Area (aname varchar (20), area_type varchar (5))

An area can have one or more persons living in it, but a person belongs to exactly one area.

Constraints: Primary Key,
area_type can be either 'urban' or 'rural'.

Create view for the following:
1. Display all person names which contain word like "Tupe" in name living in Hadapsar area.
2. Count all persons living in "rural" area having income more than 10,000.
3. Display area wise count of people having age more than 60.
4. Display area and person name having maximum income, sorted by area name.

SET B **(Number of Slots – 1)**

1) Bus Transport Database

Consider the following Entities and their Relationships for Bus Transport database.
Bus (bus_no int ,b_capacity int , depot_name varchar(20))
Route (route_no int, source char (20), destination char (20), no_of_stations int)
Driver (driver_no int ,driver_name char(20), license_no int, address char(20), d_age int , salary float)

Relationship between Bus and Route is many to one and relationship between Bus and Driver is many to many with descriptive attributes date_of_duty_allotted and shift.

Constraints: Primary Key,license_no is unique, b_capacity should not be null,
shift can be 1 (Morning) or 2(Evening).

Create view for the following:
1. Display details of all the drivers having age more than 40.
2. List details of bus no. 10 along with details of all drivers who have driven that bus in Evening shift.
3. Display driver names along with shift of Kothrud bus depot having duty allocated between 1st June 2020 to 1st July 2020.
4. Display all bus numbers running on route "Deccan" to "Katraj".

2) **Bank Database**

Consider the following Entities and their Relationships for Bank database.

Branch (br_id integer, br_name char (30), br_city char (10))

Customer (cno integer, c_name char (20), caddr char (35), city char (20))

Loan_application(lno integer, l_amt_required money, l_amt_approved money, l_date date)

Relationship between Branch, Customer and Loan_application is Ternary.

Ternary (br_id integer, cno integer, lno integer)

Constraints: Primary Key,
l_amt_required should be greater than zero.

Create view for the following:

1. Display the details of all customers who have received loan amount less than their requirement.
2. Display sum of loan amount approved branch wise from 1st June 2019 to 1st June 2020.
3. Count branch wise all customers who required loan amount more than 30 lakhs.
4. Display all customer names branch wise who requested loan amount less than one lakh.

SET C **(Number of Slots– 1)**

1) **Business-Trips Database**

Consider the following Entities and their Relationships for Business-Trips database.

Salesman (sno integer, s_name varchar (30), start_year integer)

Trip (tno integer, from_citychar (20), to_citychar (20), departure_date date, return_date date)

Dept(deptno varchar (10), dept_name char (20))

Expense (eid integer, amount money)

Relationship between Dept and Salesman is one to many, Salesman and Trip is one to many, and Trip and Expense is one to one.

Constraints: Primary Key,
s_name should not be null.

Create view for the following:

1. Find the total expenses incurred by the salesman 'Mr.Pawar'.
2. List the names of departments that have salesmen, who have done minimum number of trips.
3. Count the number of trips from Pune to Mumbai of salesman Mr. Kale in month of June.

2) **Warehouse Database**

Consider the following Entities and their Relationships for Warehouse database.

Cities (city char (20), state char (20))

Warehouses (wid integer, wname char (30), location char (20))
Stores (sid integer, store_name char (20), location_city char (20))
Items (itemno integer, description text, weight decimal (5, 2), cost decimal (5, 2))
Customer (cno integer, cname char (50), addr varchar (50), c_city char (20))
Orders (ono int, odate date)

Relationship between Cities-Warehouses is one to many, Warehouses-Stores is one to many, Customer-Orders is one to many,Items-Orders is many to many with descriptive attribute ordered_quantity, Stores-Items is many to many with descriptive attribute quantity.

Constraints: Primary Key,wname should not be null.

Create view for the following:

1. Display all the stores of a Warehouse named ‘Spares’ located at ‘Pune’.
2. List the details of all customers who have placed orders on the date ‘12-11-2015’.
3. Display city wise count of all warehouses.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2:Late Complete[]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of the instructor:______________________Date:______________________

Assignment No. 3 Stored Functions

Aim: To study syntax, creation and deletion of stored function

Pre-requisite: Knowledge of simple SQL and nested queries.

Guidelines for Teacher / Instructor:

- Demonstration of creation of stored function in a file, how to save it, how to execute it and how to drop it on databases is expected.

Instruction for students:

- Students must read the theory and syntax for creating and dropping functions before their practical slot.
- Solve SET A, SET B and SET C assigned by instructor in allotted time slot only.

Theory: Introduction to Stored Functions or Procedures

Concept:

Stored Functions are user-defined functions which need to be called explicitly for its execution. PostgreSQL functions are also called as Stored Procedures that allows you to carry out operations that would normally take several queries.

❖ **Syntax for creating a stored function:**

```
CREATE [OR REPLACE] FUNCTION function_name (arguments) RETURNS
return_datatype AS'
DECLARE
        Variable_Declarations;
        [...]
BEGIN
        <function_body>
        [...]
RETURN {variable_name | value};
END;
'LANGUAGE 'plpgsql';
```

Where,

1. **function_name:** Specifies the name of the function.
2. **Arguments:** In PL/pgSQL functions can accept arguments/parameters of different data-types. Function arguments allow a user to pass information to a function while calling a function. Each function parameter has assigned identifier that begins with dollar ($) sign and labeled with the parameter number. Identifier $1 is used for first parameter, Identifier $2 is used for second parameter and so on.

PL/pgSQL allows us to create Aliases. With the help of aliases, it is possible to assign more than just one name to a variable. Aliases should be declared within the DECLARE section of a block.

Syntax:

```
variable_name ALIAS FOR $1;
```

E.g.

```
Project_name ALIAS FOR $1;
```

3. **[OR REPLACE]:** This option allows modification of an existing function.
4. **AS:** The AS keyword is used for creating a standalone function.
5. **function_body:** Contains the executable part or the program logic.
6. **Return:** The function must contain a return statement.
7. **PL/pgSQL** is the name of the language that the function is implemented in. For backward compatibility, the name can be enclosed by single quotes.
8. **Attributes:** PL/pgSQL provides two attributes to declare variables. Use attributes to assign a variable either the type of a database object, with the %TYPE attribute, or the row structure of a table with the %ROWTYPE attribute.

   a) The %TYPE attribute: The %TYPE is used to declare a variable with the type of a referenced database object. It is used to declare variables based on definitions of columns in a table. This attribute is used to declare the variable whose type will be same as that of a previously defined variable or a column in a table.

   Syntax: variable_name table_name.column_name%TYPE;

   E.g.dob student.date_of_birth%type;

   b) The %ROWTYPE attribute: It is used to declare a record variable with same structure as the rows in a table that are specified during declaration. It will have structure exactly similar to the table's row.

   Syntax: variable_name table_name%ROWTYPE;

   E.g.student_record student%rowtype;

❖ **Calling Function:** We can either call a function directly using SELECT or use it during assignment of a variable**.** Function can be called by using below command on terminal.

Syntax:

```
select function_name(arguments);
```

❖ **Drop Function:**DROP FUNCTION deletes the definition of an existing function.

Syntax:

```
DROP FUNCTION function_name (argtype[, ...]) [CASCADE | RESTRICT]
```

Where,

1. **function_name:** The name of an existing function.
2. **argtype:** The data type(s) of the function's arguments if any.
3. **CASCADE:** Automatically drop objects that depend on the function (such as triggers).
4. **RESTRICT:** Refuse to drop the function if any objects depend on it. This is the default.

Control Structures

Like the most programming languages, PL/pgSQL also provides ways for controlling flow of program execution by using conditional statements and loops.

1. Conditional Statements:

A conditional statement specifies an action (or set of actions) that should be executed depending upon the result of logical condition specified within the statement.

❖ **IF…. THEN Statement:**

It is a statement or block of statements which is executed if given condition evaluates to true. Otherwise control is passed to next statement after END IF.

Syntax:

```
IF condition THEN
        Statements;
END IF;
```

Example:

```
CREATE OR REPLACE FUNCTION if_demo () RETURNS integer AS'
DECLARE
        num1 integer = 34;
        num2 integer = 42;
BEGIN
        IF num1 > num2 THEN
                RAISE NOTICE '' num1 is greater than num2'';
        END IF;
        IF num1 < num2 THEN
                RAISE NOTICE '' num1 is less than num2'';
        END IF;
        IF num1 = num2 THEN
                RAISE NOTICE '' num1 is equal to num2'';
        END IF;
return null;
END;
' LANGUAGE 'plpgsql';
```

```
postgres=# \i c:/data/demo.sql;
CREATE FUNCTION
postgres=# select if_demo();
NOTICE: num1 is less than num2
```

```
 if_demo
---------

(1 row)
```

❖ **IF…. THEN…. ELSE Statement:**

This statement allows you to execute a block of statements if a condition evaluates to true, otherwise a block of statements in else part will be executed.

Syntax:

```
IF condition THEN
        Statements
ELSE
        Statements
END IF;
```

Example:

```
CREATE OR REPLACE function if_else_demo(int) RETURNS void as'
DECLARE
        num alias for $1;
BEGIN
        IF (num % 2 = 0) then
             raise notice ''% is Even'',num;
        ELSE
                raise notice ''% is Odd'',num;
        END IF;
END;
'language 'plpgsql';
```

```
postgres=# select if_else_demo(99);
NOTICE: 99 is Odd
 if_else_demo
--------------

(1 row)
```

❖ **IF…. THEN…. ELSIF…. THEN…ELSE Statement:**

IF-THEN-ELSIF provides a convenient method of checking several alternatives in turn. The IF conditions are tested successively until the first true is found, if none is true then statements in ELSE part are executed.

Syntax:

```
IF condition THEN
        Statements;
ELSIF condition THEN
        Statements;
```

```
ELSIF condition THEN
        Statements;
ELSE
        Statements;
END IF;
```

Example:

```
CREATE or replace FUNCTION elsif_demo () RETURNS void AS'
DECLARE
     x integer := 72;
     y integer := 72;
BEGIN
     IF x > y THEN
             RAISE NOTICE "% is greater than %",x, y;
     ELSIF x < y THEN
             RAISE NOTICE "% is less than %",x, y;
     ELSE
             RAISE NOTICE "% is equal to %",x, y;
     END IF;
END;
' LANGUAGE 'plpgsql';
```

```
postgres=# select elsif_demo();
NOTICE: 72 is equal to 72
elsif_demo
------------

(1 row)
```

❖ **CASE Statement:**

The simple form of CASE provides conditional execution based on equality of operands. The search-expression is evaluated (once) and successively compared to each expression in the WHEN clauses. If a match is found, then the corresponding statements are executed, and then control passes to the next statement after END CASE. If no match is found, the ELSE statements are executed; but if ELSE is not present, then a CASE_NOT_FOUND exception is raised.

Syntax:

```
CASE search-expression
WHEN expression [, expression [...]] THEN
       Statements
[WHEN expression [, expression [...]] THEN
       Statements
... ]
[ELSE
       Statements]
END CASE;
```

Example:

```
CREATE FUNCTION get_price(film_id1 integer) RETURNS varchar AS'
DECLARE
    rate1 float;
    price VARCHAR;
BEGIN
    SELECT INTO rate1 rate
    FROM film
    WHERE filmid = film_id1;
CASE rate1
    WHEN 0.99 THEN
            price = "AVERAGE";
    WHEN 2.99 THEN
            price = "NORMAL";
    WHEN 4.99 THEN
            price = "HIGH";
    ELSE
            price = "UNSPECIFIED";
END CASE;
RETURN price;
END;
'LANGUAGE 'plpgsql';
```

2. Loop Statements:

Loop condition is used to perform some task repeatedly for fixed number of times.

a) **Simple Loop:** Simple loop is an unconditional loop which starts with keyword LOOP and executes the statements within its body until terminated by an EXIT or RETURN statement.

Syntax:

```
LOOP
    Statement; [...]
EXIT [label] [WHEN condition];
END LOOP;
```

Example:

```
create or replace function simple_loop() returns void as'
declare
    i int:= 0;
begin
    loop
            i:= i+1;
                    raise notice "Level : %",i;
            exit when i=5;
    end loop;
end;
'language 'plpgsql';
```

```
postgres=# select simple_loop();
NOTICE: Level : 1
NOTICE: Level : 2
NOTICE: Level : 3
NOTICE: Level : 4
NOTICE: Level : 5
 simple_loop
-------------

(1 row)
```

b) **<u>While Loop</u>:** The While statement repeats a sequence of statements till the condition evaluates to true. It is also called as entry controlled loop because the condition is checked before each entry to the loop body.

Syntax:

```
[<<Label>>]
WHILE condition LOOP
        Statements;
[...] END LOOP;
```

Example:

```
create or replace function Disp_Sum(int) returns int as'
declare
        n1 alias for $1;
        sum int= 0;
        cnt int= 1;
begin
        while (cnt <=n1) loop
                sum:= sum+cnt;
                cnt:= cnt+1;
        end loop;
raise notice "Sum of first % numbers is %",n1,sum;
return sum;
end;
'language 'plpgsql';
```

```
postgres=# select disp_sum(6);
NOTICE: Sum of first 6 numbers is 21
disp_sum
----------
     21
(1 row)
```

c) **<u>For Loop</u>:**

4. FOR loop iterates over a range of integer values. The variable name is automatically created with integer data type and exists only inside the loop. The two expressions giving the lower and upper bound of the range are evaluated once when entering the loop. If the BY clause isn't specified the iteration step is 1, otherwise it's the value specified in the BY clause. If REVERSE is specified then the step value is subtracted after each iteration.

Syntax:

```
[<<Label>>]
FOR name IN [REVERSE] expression.. Expression [ BY expression ] LOOP
        Statements
END LOOP [label];
```

Example 1:

```
CREATE FUNCTION Example_for_loop () RETURNS integer AS '
DECLARE
        counter integer;
BEGIN
        FOR counter IN 1..6 BY 2 LOOP
                RAISE NOTICE "Counter: %", counter;
        END LOOP;
return counter;
END;
' LANGUAGE 'plpgsql';

postgres=# select example_for_loop();
NOTICE: Counter: 1
NOTICE: Counter: 3
NOTICE: Counter: 5
 example_for_loop
------------------

(1 row)
```

Example 2:

```
CREATE OR REPLACE FUNCTION Example_for_loop () RETURNS integer AS '
DECLARE
   counter integer;
BEGIN
   FOR counter IN reverse 21..12 BY 2 LOOP
           RAISE NOTICE "Counter: %", counter;
   END LOOP;
return counter;
END;
' LANGUAGE 'plpgsql';
```

```
postgres=# select example_for_loop();
NOTICE: Counter: 21
NOTICE:  Counter: 19
NOTICE:  Counter: 17
NOTICE:  Counter: 15
NOTICE: Counter: 13
 example_for_loop
------------------

(1 row)
```

Looping Through Query Results by Using for Loop:
Using a different type of FOR loop, you can iterate through the results of a query and manipulate that data accordingly.
Syntax:

```
[<<Label>>]
FOR record _variable IN query LOOP
        Statements;
END LOOP;
```

Example:

```
CREATE or replace FUNCTION extract_title (integer) RETURNS text AS'
DECLARE
     sub_id1 ALIAS FOR $1;
     text_output TEXT;
     row_data book%ROWTYPE;
BEGIN
     FOR row_data IN SELECT * FROM book
             WHERE sub_id = sub_id1 LOOP
             text_output := row_data.title;
     END LOOP;
RETURN text_output;
END;
' LANGUAGE 'plpgsql';
```

```
postgres=# select extract_title(22);
 extract_title
---------------
 RDBMS
(1 row)
```

3. Exit Statement:
These statements are used to exit from a loop.

a) EXIT: This statement is used to exit from loop without condition. Condition needs to be specified separately.

Syntax:
```
exit;
```
Example:
```
CREATE OR REPLACE FUNCTION exit_demo() RETURNS integer AS '
BEGIN
        for i in 1..10 loop
        if i = 4 then EXIT;
                else
                        raise notice''Input is %'',i;
                end if;
        end loop;
return null;
END;
' LANGUAGE 'plpgsql';
```

```
postgres=# select exit_demo();
NOTICE:  Input is 1
NOTICE:  Input is 2
NOTICE: Input is 3
exit_demo
-----------

(1 row)
```

b) exit-when: This statement is used to exit from loop by specifying condition within exit statement.

Syntax:
```
EXIT WHEN condition;
```
Example:
```
CREATE OR REPLACE FUNCTION exit_demo() RETURNS integer AS '
BEGIN
     for i in 1..10 loop
        Exit When i=5;
             raise notice''Input is %'',i;
     end loop;
return null;
END;
' LANGUAGE 'plpgsql';
```

```
postgres=# select exit_demo();
NOTICE: Input is 1
```

```
NOTICE: Input is 2
NOTICE: Input is 3
NOTICE: Input is 4
exit_demo
-----------

(1 row)
```

<u>Example 1: Simple Function.</u>

Consider the following Relational Database:

Student (<u>sno</u>, s_name, s_class, address)

Teacher (<u>tno</u>, t_name, qualification, experience)

Stud_teach(sno, tno, subject)

- **Write a function to count the number of the teachers who are teaching to a student named " ". (Accept student name as input parameter). Display appropriate message.**

```
create or replace function stud_det(varchar) returns int as'
declare
        s_name alias for $1;
        cnt int;
begin
        select into cnt count(teacher.tno) from student, teacher, stud_teach where
        student.sno=stud_teach.sno and teacher.tno=stud_teach.tno and
        sname=s_name;
        if cnt=0 then
                raise notice ''% Student not present'',s_name;
        else
                return cnt;
        end if;
end;
'language 'plpgsql';
```

```
postgres=# select stud_det('Gauri');
 stud_det
----------
        4
(1 row)
```

<u>Example 2: Looping Through Query Results by Using for Loop.</u>

- **Accept teacher name as input and print the names of student to whom that teacher teaches.**

```
create or replace function teach_det(varchar) returns int as'
declare
        t_name alias for $1;
        rec record;
```

```
begin
        raise notice "Teacher Name || Name of Student";
        for rec in select tname, sname from student, teacher, stud_teach where
        student.sno=stud_teach.sno and teacher.tno=stud_teach.tno and
        tname=t_name loop
                raise notice "% %",rec.tname, rec.sname;
        end loop;
return null;
end;
'language 'plpgsql';
```

```
postgres=# select teach_det('Shripradaʼ);
NOTICE: Teacher Name || Name of Student
NOTICE:    Shripradа              Faisal
NOTICE:    Shriprada              Umar
NOTICE:    Shriprada              Zeba
 teach_det
-----------

(1 row)
```

Exercise

SET A **(Number of Slots – 2)**

1) Project-Employee Database

Consider the following Entities and their Relationships for Project-Employee database.
Project (pno integer, pname char (30), ptype char (20), duration integer)
Employee (eno integer, ename char (20), qualification char (15), joining_date date)

Relationship between Project and Employee is many to many with descriptive attribute start_date date, no_of_hours_worked integer.

Constraints: Primary Key,
duration should be greater than zero,
pname should not be null.

1. Write a stored function to find the number of employees whose joining date is before ‘01/01/2007’.
2. Write a stored function to accept eno as input parameter and count number of projects on which that employee is working.
3. Write a stored function to accept project name and display employee details who worked more than 2000 hours.
4. Write a stored function to display all projects started after date “01/01/2019”.

2) **Person-Area Database**

Consider the following Entities and their Relationships for Person-Area database.

Person (pno integer, pname varchar (20), birthdate date, income money)
Area (aname varchar (20), area_type varchar (5))

An area can have one or more persons living in it, but a person belongs to exactly one area.

Constraints: Primary Key,area_type can be either 'urban' or 'rural'.

1. Write a stored function to print total number of persons of a particular area. Accept area name as input parameter.
2. Write a stored function to update the income of all persons living in urban area by 20%.
3. Write a stored function to accept area_type and display person's details area wise.
4. Write a stored function to accept area name and display all persons having age more than 60.

SET B **(Number of Slots – 3)**

1) **Bus Transport Database**
Consider the following Entities and their Relationships for Bus Transport database.

Bus (bus_no int ,b_capacity int , depot_name varchar(20))
Route (route_no int, source char (20), destination char (20), no_of_stations int)
Driver (driver_no int ,driver_name char(20), license_no int, address char(20), d_age int , salary float)

Relationship between Bus and Route is many to one and relationship between Bus and Driver is many to many with descriptive attributes date_of_duty_allotted and shift.

Constraints: Primary Key, license_no is unique, b_capacity should not be null, shift can be 1 (Morning) or 2(Evening).

1. Write a stored function to accept route no and display bus information running on that route.
2. Write a stored function to accept shift and depot name and display driver details who having duty allocated after '01/07/2020'.
3. Write a stored function to accept source name and display count of buses running from source place.
4. Write a stored function to accept depot name and display driver details having age more than 50.

2) **Bank Database**
Consider the following Entities and their Relationships for Bank database.
Branch (br_id integer, br_name char (30), br_city char (10))

Customer (cno integer, c_name char (20), caddr char (35), city char (20))

Loan_application(lno integer, l_amt_required money, l_amt_approved money, l_date date)

Relationship between Branch, Customer and Loan_application is Ternary.
Ternary (br_id integer, cno integer, lno integer)
Constraints: Primary Key,
l_amt_required should be greater than zero.

1. Write a stored function to accept branch name and display customer details whose loan amount required is more than loan approved.
2. Write a stored function to accept branch name and display customer name, loan number, loan amount approved on or after 01/06/2019.
3. Write a stored function to display total loan amount approved by all branches after date 30/05/2019.
4. Write a stored function to display customer details who have applied for loan more than one branches.

SET C **(Number of Slot – 2)**

1) Business trip database

Consider Business trip database that keeps track of the business trips of salesman in an office.

Following are the tables:
Salesman (sno integer, s_name char (30), start_year integer, dept_no varchar(10))
Trip(tno integer, from_city char (20), to_citychar (20),departure_date date, return_date date)
Dept(dept_no varchar (10), dept_name char(20))
Expense(eid integer, amount money)

Relationships:
Dept-Salesman: 1 to M
Salesman-Trip: 1 to M
Trip-Expense: 1 to 1

Execute the following stored functions.
a) Write a stored function to find a business trip having maximum expenses.
b)Write a stored function to count the total number of business trips from ‘Pune’ to ‘Mumbai’.

2) Railway Reservation Database

Consider a Railway reservation system for passengers. The bogie capacity of all the bogies of a train is same.
TRAIN (train_no int, train_name varchar(20), depart_time time , arrival_time time, source_stn varchar (20),dest_stn varchar (20), no_of_res_bogies int ,bogie_capacity int)
PASSENGER (passenger_id int, passenger_name varchar(20), address varchar(30), age int ,gender char)
Relationships:

Train _Passenger: M-M relationship named ticket with descriptive attributes as follows:
TICKET(train_no int, passenger_id int, ticket_no int ,bogie_no int, no_of_berths int ,tdate date , ticket_amt decimal(7,2),status char)

Constraints:
The status of a berth can be 'W' (waiting) or 'C' (confirmed).

Execute the following stored functions.

1. Write a stored function to calculate the ticket amount paid by all the passengers on 12/12/2019 for all the trains.

2. Write a stored function to update the status of the ticket from ‘waiting’ to ‘confirm’ for passenger named “Mr.Mohite”.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2:Late Complete[ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of the instructor:______________________Date:______________________

Assignment No. 4 Error and Exception Handling

Aim: To study how to handle different errors and exceptions that arises while using database.

Pre-requisite: Knowledge of how to create stored functions.

Guidelines for Teacher / Instructor:

- Demonstration of creation of raise statement on databases is expected.

Instruction for students:

- Students must read the theory and syntax for handling error before their practical slot.
- Solve SET A, SET B and SET C assigned by instructor in allotted time slot only.

Theory: Error and Exception Handling

Concept:

Any error occurring in a PL/pgSQL function aborts execution of the function. Errors can be trapped and recovered by using a Begin block with an Exception clause.

Syntax:

```
Declare
        Declarations
Begin
        Statements
        Exception
When condition then
        Handler statements
End;
```

If no errors occur, this form of block simply executes all the statements, and then control passes to the next statement. But if error occurs within the statements, further processing of the statements is stopped and control passes to the exception list.

Raise Statement: It is used to raise errors, report messages and exceptions during a PL/pgSQL function's execution.
This statement can raise built in exceptions, such as division_by_zero, not found etc.

Syntax:
RAISE level "format string" [, expression [, ...]];

Where,

- **level:** The level option specifies the severity of an error.
  Level can be:
  1. DEBUG: DEBUG level statements send the specified text as a message to the PostgreSQL log and the client program if the client is running in debug mode.

2. NOTICE: This level statement sends the specified text as a message to the client program.
3. EXCEPTION: This statement sends the specified text as error message. It causes the current transaction to be aborted.

- **format:** The format is a string that specifies the message. The format uses percentage (%) placeholders that will be substituted by the next arguments. The number of placeholders must match the number of arguments; otherwise PostgreSQL will report the following error message
  example 1: ERROR: too many parameters specified for RAISE.
  example 2: ERROR: control reached end of function without RETURN.

<u>**Example**</u>**:**

❖ **In the example given below, the first raise statement gives a debug level message and sends specified text to PostgreSQL log. The second statements send a notice to the user. The third raise statement displays an error and throws an exception, which causes the function to end.**

```
CREATE FUNCTION raise_test(int, int) RETURNS void
AS' DECLARE
        x1 alias for $1;
        x2 alias for $2;
        div INTEGER;
BEGIN
        RAISE DEBUG "The raise_test() function begins from here.";
        if x2 != 0 then
                div = x1 / x2;
                raise notice "Division of % and % is %",x1,x2,div;
        else
                RAISE EXCEPTION "Transaction aborted due to division by zero
exception.";
        end if;
END;
'LANGUAGE 'plpgsql';
```

```
postgres=# select raise_test(24,3);
NOTICE: Division of 24 and 3 is 8
raise_test
------------

(1 row)
postgres=# select raise_test(18,0);
ERROR: Transaction aborted due to division by zero exception.
CONTEXT: PL/pgSQL function raise_test(integer,integer) line 12 at RAISE
```

❖ **Consider the database**
PROJECT (<u>pno</u>, p_name, ptype, duration)
EMPLOYEE(<u>eno</u>, e_name, qualification, joindate)
PROJ_EMP (pno, eno, start_date, no_of_hours_worked)

Write a stored function to accept project name as input and print the names of employees working on that project. Raise an exception for an invalid project name.

```
create or replace function chk_emp(varchar) returns int as'
declare
      rec record;
      pname1 alias for $1;
      ecount int;
begin
      for rec in select pname from project loop
              if(rec.pname<>pname1)then
                      continue;
              else
                      select into ecount count(proj_emp.eno)
                      from employee,project, proj_emp where
                      project.pno=proj_emp.pno and
                      employee.eno=proj_emp.eno and
                      pname=pname1;
              end if;
      end loop;
      if ecount>0 then
              raise notice''Employee count is : %'',ecount;
      else
              raise notice''Invalid Project Name'';
      end if;
return null;
end;
'language 'plpgsql';
```

```
postgres=# select chk_emp('system');
NOTICE: Employee count is : 4
chk_emp
---------

(1 row)


postgres=# select chk_emp('web design');
NOTICE: Invalid Project Name
chk_emp
---------

(1 row)
```

SET A **(Number of Slot – 1)**

1. Person-Area Database

Person (pno integer, pname varchar (20), birthdate date, income money)

Area (aname varchar (20), area-type varchar (5))

An area can have one or more persons living in it, but a person belongs to exactly one area. The attribute 'area_type' can have values either 'urban' or 'rural'.

1. Write a stored function to print total number of person of a particular area. (Accept area_name as input parameter). Display appropriate message for invalid area name.
2. Write a stored function to print sum of income of person living in "______" area type. (Accept area type as input parameter). Display appropriate message for invalid area type.
3. Write a stored function to display details of person along with area name whose birthday falls in the month of "_________". (Accept month as input parameter). Display error message for invalid month name.

2. Student Teacher Database

Student (sno integer, s_name char(30), s_class char(10), s_addr Char(50))
Teacher (tno integer, t_name char (20), qualification char (15),experience integer)

The relationship is as follows:

Student-Teacher: M-M with descriptive attribute Subject.

1. Write a stored function to count the number of the teachers teaching to a student named "________". (Accept student name as input parameter). Raise an exception if student name does not exist.
2. Write a stored function to count the number of the students who are studying subject named "________" (Accept subject name as input parameter). Display error message if subject name is not valid.
3. Write a stored function to display teacher details who have qualification as "________" (Accept teacher's qualification as input parameter). Raise an exception for invalid qualification.

SET B **(Number of Slot – 1)**

1) Bus Driver Database

BUS (bus_no int , capacity int , depot_name varchar(20))
ROUTE (route_no int, source char(20), destination char(20),no_of_stations int)
DRIVER (driver_no int , driver_name char(20), license_no int, address char(20), d_age int , salary float)

The relationships are as follows:
BUS_ROUTE: M-1
BUS_DRIVER: M-M with descriptive attributes Date of duty allotted and Shift – it can be 1 (Morning) or 2 (Evening).

Constraints:

License_no is unique. 2. Bus capacity is not null

1. Write a stored function to accept the bus_no and date and print its allotted drivers. Raise an exception in case of invalid bus number.
2. Write a stored function to display the all Dates on which a driver has driven any bus. (Accept driver name as input parameter).Raise an exception in case of invalid driver name.
3. Write a stored function to display the details of the buses that run on route_no = "______". (accept route_no as input parameter). Raise an error in case of invalid driver name.

2.Railway Reservation System Database

TRAIN: (train_no int, train_name varchar(20), depart_time time , arrival_time time, source_stn varchar (20),dest_stn varchar (20), no_of_res_bogies int ,bogie_capacity int)

PASSENGER : (passenger_id int, passenger_name varchar(20), address varchar(30), age int ,gender char)

Relationships:

Train _Passenger: M-M relationship named ticket with descriptive attributes as follows

TICKET: (train_no int, passenger_id int, ticket_no int ,bogie_no int, no_of_berths int ,tdate date , ticket_amt decimal(7,2),status char)

Constraints: The status of a berth can be 'W' (waiting) or 'C' (confirmed).

1. Write a stored function to print the details of train wise confirmed bookings on date "____________" (Accept date as input parameter).Raise an error in case of invalid date.
2. Write a stored function to accept date and passenger name and display no of berths reserved and ticket amount paid by him. Raise exception if passenger name is invalid.
3. Write a stored function to display the ticket details of a train. (Accept train name as input parameter).Raise an exception in case of invalid train name.

SET C **(Number of Slot – 1)**

1. Bank Database

Branch (bid integer, br_name char (30), br_city char (10))

Customer (cno integer, cname char (20), caddr char (35), city char(20))

Loan_application (lno integer, l_amt_require money, l_amt_approved Money, l_date date)

The relationships are as follows:

Branch, customer, loan_application are related with ternary relationship.

Ternary (bid integer, cno integer, lno integer).

1. Write a stored function to accept customer name as input and display the loan details of that customer. (Accept customer name as input parameter). Raise an exception for an invalid customer name.
2. Write a stored function to display details of customers of particular branch. (Accept branch name as input parameter) Display appropriate error message if branch name is invalid.
3. Write a stored function to display numbers of loan approved after a particular date. (Accept loan date as input parameter) Display appropriate message if loan date does not exist.

2. Movie-Actor Database

Movies (m_name varchar (25), release_year integer, budget money)
Actor (a_name char (30), role char (30), charges money, a_address varchar(30))
Producer (producer_id integer, name char (30), p_address varchar (30))

The relationships are as follows:

Each actor has acted in one or more movies. Each producer has produced many movies and each movie can be produced by more than one producers. Each movie has one or more actors acting in it, in different roles.

1. Write a stored function to accept movie name as input and display the details of actors for that movie and sort it by their charges in descending order. (Accept movie name as input parameter). Raise an exception for an invalid movie name.
2. Write a stored function to accept actor / actress name as input and display the names of movies in which that actor has acted in. (Accept actor name as input parameter). Raise an exception for an invalid actor name.
3. Write a stored function to accept producer name as input and display the count of movies he/she has produced. (Accept producer name as input parameter). Raise an exception for an invalid producer name.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete []
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor:_____________________________ Date:____________________

Assignment No. 5 - Cursors

Aim: Learn how to create and execute cursors.

Pre-requisite: Knowledge of stored functions, control statements, SQL and Nested Queries must.

Guidelines for Teachers / Instructors:

- Demonstration of creation and execution of cursors on database is expected.

Instructions for Students:

- Students must read the theory and syntax for creating cursors with and without parameter before his/her practical slot.

- Solve SET A, B or C assigned by instructor in allocated slots only.

Theory:Introduction to Cursor

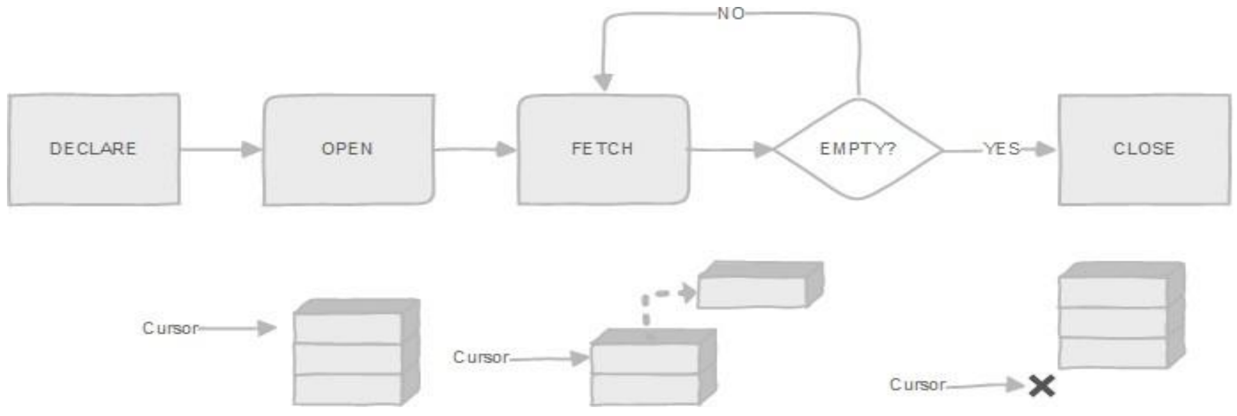
Concept:

In PL/pgSQL, a cursor allows to encapsulate the query rather than executing a whole query at once. After encapsulating query it is possible to read few rows from result set. The main reason for doing this is to avoid memory overrun when the result contains a large number of rows. (However, PL/pgSQL users do not normally need to worry about that, since FOR loops automatically use a cursor internally to avoid memory problems.)

A PL/pgSQL cursor allows user to access or retrieve multiple rows of data from a table, so that user can process different operations on each individual row at a time.

We use cursors when we want to divide a large result set into parts and process each part individually. If we process it at once, we may have a memory overflow error.

The following diagram illustrates how to use a cursor in PostgreSQL:



- ***Steps for cursor:***
  1. First, declare a cursor.
  2. Next, open the cursor.

3. Then, fetch rows from the result set into a target.
4. After that, check if there are more rows left to fetch. If yes, go to step 3, otherwise go to step 5.
5. Finally, close the cursor.

We will examine each step in more detail in the following sections.

1. ***Declaration:***
We can declare a cursor variable by using two ways:

a) **Bound Cursor Variable:**
To create a bound cursor variable, use the following cursor declaration.

Syntax:
```
DECLARE
cursor_name CURSOR [(arguments)]
```

For Example:

i) **declare**
```
declare
movie_cursor cursor for
            select *
            from movie;
```
Here, the movie_cursor is a cursor that encapsulates all rows in the movie table.

j) **declare**
```
declare
movie_cursor1 cursor(year integer) for
              select *
              from movie
             where rel_year = year;
```

Here, the movie_cursor1 is a cursor that encapsulates movie with a particular release year in the movie table.

b) **Unbound Cursor Variable:**
To create an unbound cursor variable, PL/pgSQL provides a special type called REFCURSOR i.e. Declare the cursor variable of type REFCURSOR.

Syntax:
```
DECLARE
cursor_name REFCURSOR;
```

For Example: declare my_cursor **refcursor** ;

2. ***Opening Cursor:***
Cursors must be opened before they can be used to fetch rows.PL/pgSQL has three forms of the OPEN statement.

a) **Opening unbound cursors :**

We open an unbound cursor using the following syntax:

i) Because the unbound cursor variable is not bounded to any query when we declared it, we have to specify the query when we open it.

Syntax : **OPEN** unbound_cur_variable **FOR** query;

For Example: open my_cursor **for**
select *
from city
where country = p_country;

ii) PostgreSQL allows you to open a cursor and bound it to a dynamic query.

Syntax :**OPEN** unbound_cur_variable **FOR EXECUTE** query_string
USING expression [,…]];

For Example : query_str = 'select * from city order by $1;
Open cur_city **for execute** query_str **using** sort_field;

In the above example, we build a dynamic query that sorts rows based on asort_field parameter and open the cursor that executes the dynamic query.

b) **Opening a Bound Cursor**

Because a bound cursor already bounds to a query when we declared it, so when we open it, we just need to pass the arguments to the query if necessary.

Syntax: **OPEN** bound_cursor_name [(argument values)];
For Example: i)**open** movie_cursor;
ii) **open** movie_cursor1 (year :=2020);

3. Fetching Rows:

After opening a cursor, we can manipulate it using FETCH, MOVE, UPDATE, DELETE statement.

I. Fetching the next row:

The FETCH statement gets the next row from the cursor and assigns it a target_variable which could be a record, a row variable, or a comma-separated list of variables. If no more rows found, the target_variable is set to NULL(s).

Syntax: fetch[direction{from | in}] cur_variable **into** target_variable;

The *direction* clause can be any of the following variants:

NEXT, PRIOR, FIRST, LAST, ABSOLUTE*count*, RELATIVE*count*, FORWARD,

or BACKWARD.
By default, a cursor gets the NEXT row if we don't specify the direction explicitly.

For Example: i)**fetch** movie_cursor **into** row_movie;
ii)**fetch last** from row_movie **into** title, rel_year;

II.Move :
If you want to move the cursor only without retrieving any row, you use the MOVE statement.

Syntax: move[direction { from | in }] cursor_variable

The *direction* clause can be any of the variants NEXT, PRIOR, FIRST, LAST, ABSOLUTE*count*, RELATIVE*count*, ALL, FORWARD [*count*| ALL], or BACKWARD [*count* | ALL].

For Example : i) **move** movie_cursor1;
ii) **move last** from movie_cursor;
iii)**move relative** -1 from movie_cursor;
iv)**move forward** 3 from movie_cursor;

III. Deleting or updating the row
Once a cursor is positioned, we can delete or update row identifying by the cursor using DELETE WHERE CURRENT OF or UPDATE WHERE CURRENT OF statement as follows:

Syntax: update table_name
set column_name = value,…
where current of cursor_variable;

delete from table_name
where current of cursor_variable;

For Example: update movie
set rel_year =p_year
where current of movie_cursor;

4. **Closing cursor :**
To close an opening cursor, we use **CLOSE** statement as follows:

Syntax : **close** cursor_variable;
For Example : close movie_cursor;

The CLOSE statement releases resources or frees up cursor variable to allow it to be opened again using OPEN statement.

Examples for Practice:

1) **Using Parameterized Cursor :**

   Consider the following Relational Database:
   Doctor (d_no, d_name, d_city)
   Hospital (h_no, h_name, h_city)
   DH(d_no, h_no**)**

- **Display Hospital wise doctor details.**

```
 create or replace function Doc_cursor() returns
void as'
declare
c1 cursor for select * from Hospital;
c2 cursor(hno Hospital.h_no%type) for
select h_name,DH.d_no,d_name
 from Hospital,Doctor, DH
 where Hospital.h_no=DH.h_no
 and Doctor.d_no=DH.d_no
 and DH.h_no=hno;
 rec1 Hospital %rowtype;
 rec2 record;
  begin
      open c1;
           raise notice "Hospital Name Doctor No Doctor Name ";
           loop
                fetch c1 into rec1;
                exit when not found;
                open c2(rec1.h_no);
                   loop
                        fetch c2 into rec2;
                        exit when not found;
                        raise notice "% %%",rec2.h_name,rec2.d_no,rec2.d_name;
                  endloop;
                close c2;
            end loop;
      close c1;
end;'language 'plpgsql';
```

```
pgsql=# select Doc_cursor ();
```

2) **Using Multiple Cursors:**

   Consider the following Relational Database:
   Route (rno, source, destination, no_of_stations)
   Bus (bno, capacity, depot_name, rno)

- **Display the details of the buses that run on route_no=1 and route_no=2 using multiple cursors.**

```
create or replace function Disp_route() returns void as' declare
c1 cursor for select *
                  from bus
                  where rno=1;
c2 cursor for select *
                  from bus
                 where rno=2;
rec1 record;
rec2 record;
begin                    open c1;
                              raise notice "Details of Buses run on route 1 are:";
                              raise notice "Bus_No Capacity Depot Name";
                        loop
                             fetch c1 into rec1;
                             exit when not found;
                              raise notice " % % % ",rec1.bno,rec1.capacity,rec1.depot_name;
                       end loop;
                       open c2;
                                raise notice "Details of Buses run on route 2 are:";
                               raise notice "Bus No Capacity Depot Name";
                               loop
                                     fetch c2 into rec2;
                                     exit when not found;
                                     raise notice "% % % ",rec2.bno,rec2.capacity,rec2.depot_name;
                              end loop;
                        closec2;
                  close c1;
              end;'language 'plpgsql';

pgsql=# select Disp_route();
```

3) **Consider the relation**
   Employee (<u>eno</u>, ename, deptno,salary).

- **Write a cursor to print the details of the employee along with commission earned for each employee. Commission is 20% of salary for employees of dept no = 5; its 50% of salary for employees of deptno=8; its 30% of salary for employees of deptno=10.**

```
create function cursor_demo( ) returns integer as'
declare
emp_recEmployee%rowtype
C1 cursor for Select *
                     from employee;
Comm Number (6,2);
begin
     open C1;
           loop
```

```
                    fetch C1 into emp_rec;
                     if emp_rec.deptno = 5 then
                        Comm:=emp_rec.salary * 0.2;
                        else if emp_rec.deptno = 8 then
                            Comm :=emp_rec.salary * 0.5;
                            else If emp_rec.deptno = 10 then
                                  Comm :=emp_rec.salary * 0.3;
                             end if;
                         end if;
                    endif;
                    raise notice "emp_rec.ename||emp_rec.deptno||emp_rec.salary||comm. ";
                    exit when not found;
                end loop;
            close C1;
        end;'language 'plpgsql';
```

Exercises

SET A (**Number of Slots – 2**)

1) **Student-Teacher Database**

Consider the following Entities and their Relationships for Student-Teacher database.

Student (s_no integer, s_name char (20), address char (25), class char (10))
Teacher (t_no integer,t_namechar (10), qualification char (10),experience integer)

Relationship between Student and Teacher is many to many with descriptive attribute subject and marks_scored.

Constraints: Primary Key,s_name,t_name should not be null,marks_scored> 0

a) Write a cursor which will accept s_no from the user and display s_name and t_name who taught 'RDBMS' subject.
b) Write a cursor to accept the class from user and display the student names,subject and marks of that class.
c) Write a stored function using cursor to find the details of maximum experienced teacher.

2) **Movie - Actor Database:**

Movie (m_name char (25), release_year integer, budget money)
Actor (a_name varchar (20), role char (20), charges money, a_address varchar (20))
Producer (producer_id integer, p_name char (30), p_address varchar (20))

Each actor has acted in one or more movies. Each producer has produced many movies and each movie can be produced by more than one producers. Each movie has one or more actors acting in it, in different roles.

Constraints: Primary Key,
role should be 'Main','Supportive','Villan','Comedy'
p_name should not be null.
budget,charges > 0

a) Write a cursor to pass a_name as a parameter to a function and return total number of movies in which given actor is acting.
b) Write a stored function using cursor to display role wise the names of actors.
c) Write a cursor to display producer name that produces more than 2 movies in which 'Amitabh' is acted.

SETB **(Number of Slots – 2)**

1) **Using Business-Trip Database:**

Consider the following Entities and their Relationships for Business-Trips database.
Salesman (sno integer, s_namevarchar (30), start_year integer)
Trip (tno integer, from_citychar (20), to_citychar (20), departure_date date, return_date date)
Dept(dept_no varchar (10), dept_name char (20))
Expense (eid integer, amount money)

Relationship between Dept and Salesman is one to many, Salesman and Trip is one to many, and Trip and Expense is one to one.

Constraints: Primary Key,
s_name,dept_name should not be null.
amount > 0

a) Write a stored function with cursor, which accepts dept_no as input and prints the names of all salesmen working in that department.
b) Write a cursor to display trip details which has maximum expenses.
c) Write a cursor to accept departure_date as parameter and display the count of trips for given departure_date.

2) **Warehouse Database**

Consider the following Entities and their Relationships for Warehouse database.
Cities (city char (20), state char (20))
Warehouses (wid integer, wname char (30), location char (20))
Stores (sid integer, store_name char (20), location_city char (20))
Items (itemno integer, description text, weight decimal (5, 2), cost decimal (5, 2))
Customer (cno integer, cname char (50), addrvarchar (50), c_city char (20))
Orders (ono int, odate date)

Relationship between:

Cities-Warehouses is 1 - M
Warehouses-Stores is 1- M
Customer-Orders is 1- M
Items-Orders is M – M with descriptive attribute ordered_quantity,
Stores-Items isM – M with descriptive attribute quantity.

Constraints: Primary Key,
wname should not be null.

a) Write a cursor to accept a city from the user and list all warehouses in that city.
b) Write a cursor to find the name of customer who orderd items between Rs. 50000 to Rs. 100000
c) Write a cursor to list all find out the cost of items which are stored at "Pune" city.

<u>SET C</u> **(Number of Slots – 2)**

3) Bank Database

Consider the following Entities and their Relationships for Bank database.

Branch (<u>br_id</u> integer, br_name char (30), br_city char (10))
Customer (<u>cno</u> integer, c_name char (20), caddr char (35), city char (20))
Loan_application(<u>lno</u> integer, l_amt_required money, l_amt_approved money, l_date date)

Relationship between Branch, Customer and Loan_application is Ternary.

Ternary (br_id integer, cno integer, lno integer)
Constraints: Primary Key,
l_amt_required should be greater than zero.

a) Write a cursor to accept br_name from user and display the name of customers who have approved loan amount greater than 50000 from the given branch.
b) Write a stored function using cursor to count the number of customers of particular branch. (Accept branch name as input parameter).
c) Write a cursor to increase the loan approved amount for all loans by 35%.

4) Railway Reservation Database

Consider the following Entities and their Relationships Railway -Reservation Database:
Train (<u>tno</u> int, tname varchar (20), depart_time time, arrival_time time, source_stn char (10), dest_stn char (10), no_of_res_bogies int ,bogie_capacity int)
Passenger (<u>passenger_id</u> int, passenger_name varchar (20), address varchar (30), age int, gender char)

Relationship between Train and Passenger is many to many with descriptive attribute ticket.

Ticket (train_no int, passenger_id int, ticket_no int,bogie_no int, no_of_berths int, tdate date, ticket_amt decimal (7,2),status char)

Constraints: Primary Key,
Status of a berth can be 'W' (waiting) or 'C' (confirmed).

a) Write a stored function using cursor to accept date and passenger name and display no. of berths reserved and ticket amount paid by him/her.
b) Write a stored function using cursors to find the confirmed bookings of all the trains on a particular date (Accept date from user).
c) Write a stored function using cursors to accept a date and find the total number of berths

reserved for all the trains on given date.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2:LateComplete[]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor:____________________________ **Date:**___________________

Assignment No. 6 - Triggers

Aim: To study creating, execution and dropping of triggers.

Pre-requisite: Knowledge of simple SQL, Stored functions and Errors and Exception handling must.

Guidelines for Teachers / Instructors:

- Demonstration of creation of triggers, execution and dropping of triggers on database is expected.

Instructions for Students:

- Students must read the theory and syntax for creating and dropping triggers before his/her practical slot.
- Solve SET A, B or C assigned by instructor in allocated slots only.

Concept

- PL/pgSQL can be used to define trigger procedures. PostgreSQL **Triggers** are database callback functions, which are automatically invoked when a specified database event (insert, delete, and update) occurs.
- A trigger procedure is created with the **CREATE FUNCTION** command, declaring it as a function with no arguments and a return type of trigger.
- A trigger that is marked FOR EACH ROW is called once for every row that the operation modifies. In contrast, a trigger that is marked FOR EACH STATEMENT only executes once for any given operation, regardless of how many rows it modifies.

Syntax of Creating Trigger:

```
CREATETRIGGER trigger_name
{BEFORE |AFTER} {event_ name} ON table_name
FOR EACH {ROW | STATEMENT}
EXECUTE PROCEDURE function_name (arguments);
```

Parameters Used:

1. **trigger_name:** User defined name of the trigger.
2. **before/after:** Determines whether the function is called before or after an event.
3. **event_name:** Database events can be Insert, Update and Delete.
4. **table_name:** The name of the table the trigger is for.
5. **for each row /for each statement:** specifies whether the trigger procedure should be fired once for every row affected by the trigger event or just once per SQL statement. If neither is specified, **for each** statement is the default.
6. **function_name:** Name of the function to execute.

Syntax of Drop Trigger:
DROP TRIGGER trigger_name ON table_name;
Parameters Used:

1. **trigger_name:** The name of the trigger to remove.
2. **table_name:** The name of the table for which trigger is defined.

Variables used in Trigger: When a PL/pgSQL function is called as a trigger, several special variables are created automatically in the top-level block. They are:

Variable Name	Description
NEW	Data type RECORD; variable holding the new database row for INSERT/UPDATE operations in row-level triggers. This variable is NULL in statement-level triggers and for DELETE operations.
OLD	Data type RECORD; variable holding the old database row for UPDATE/DELETE operations in row-level triggers. This variable is NULL in statement-level triggers and for INSERT operations
TG_NAME	Data type name; variable that contains the name of the trigger actually fired.
TG_WHEN	Data type text; a string of BEFORE, AFTER, or INSTEAD OF, depending on the trigger's definition.
TG_LEVEL	Data type text; a string of either ROW or STATEMENT depending on the trigger's definition.
TG_OP	Data type text; a string of INSERT, UPDATE, DELETE, or TRUNCATE telling for which operation the trigger was fired.
TG_TABLE_NAME	Data type name; the name of the table that caused the trigger invocation.
TG_TABLE_SCHEMA	Data type name; the name of the schema of the table that caused the trigger invocation.
TG_NARGS	Data type integer; the number of arguments given to the trigger procedure in the CREATE TRIGGER statement.
TG_ARGV []	Data type array of text; the arguments from the CREATE TRIGGER statement.

Example 1:
Consider the following Relational Database:
Department (dno, dname)
Employee (eno, ename, sal, dno)

Delete Employee record from Employee table and display message to user 'Employee record being deleted'.

Function:
```
create or replace function Emp_Del() returns trigger as'
declare
begin
raise notice "Employee record being deleted";
return old;
end;'
language 'plpgsql';
```

Trigger:
```
create trigger trig_del
before delete on Employee
for each row
execute procedure Emp_Del();
```

After creating function and trigger type delete query on terminal.

```
Lab=# delete from Employee;
```

The trigger gets fired and executes procedure/function Emp_Del() and text message displayed to the user.

Example 2:
Consider the following Relational Database:
Student (rollno, s_name , class)
Subject (scode , subject_name)
Stud_Sub(rollno, scode, marks)

The below example ensures that if student marks entered less than 0 or greater than 100, trigger gets fired.

Create Function:
```
create or replace function stud_marks() returns trigger as'
declare
begin
if(NEW.marks_scored < 0 or NEW.marks_scored >100) then
raise exception "Student marks should not be less than 0 or greater than 100";
end if;
return new;
end;'
language 'plpgsql';
```

Create Trigger:
```
create trigger trig_stud_marks
before insert on stud_sub
for each row
execute procedure stud_marks();
```

After creating function and trigger type insert query on terminal.

```
Lab=# insert into stud_sub values(1,301,101);
Lab=# insert into stud_sub values(1,301,-1);
```

The trigger gets fired and executes procedure/function stud_marks() and error message displayed to the user if student marks entered less than 0 or greater than 100.

Exercises

SET A **(Number of Slots – 2)**

1) Student-Teacher Database
Consider the following Entities and their Relationships for Student-Teacher database.
Student (s_no int, s_name varchar (20), s_class varchar (10), s_addr varchar (30))
Teacher (t_no int, t_name varchar (20), qualification varchar (15), experience int)

Relationship between Student and Teacher is many to many with descriptive attribute subject.

Constraints: Primary Key,
s_class should not be null.

Create trigger for the following:

1. Write a trigger before insert the record of Student. If the sno is less than or equal to zero give the message "Invalid Number".
2. Write a trigger before update a student's s_class from student table. Display appropriate message.
3. Write a trigger before inserting into a teacher table to check experience. Experience should be minimum 2 year. Display appropriate message.

1) Project-Employee Database Consider the following Entities and their Relationships for Project-Employee database.
Project (pno integer, pname char (30), ptype char (20), duration integer)
Employee (eno integer, ename char (20), qualification char (15), joining_date date)

Relationship between Project and Employee is many to many with descriptive attribute start_date date, no_of_hours_worked integer.
Constraints: Primary Key,
pname should not be null.

Create trigger for the following:

1. Write a trigger before inserting into an employee table to check current date should be always

greater than joining date. Display appropriate message.

2. Write a trigger before inserting into a project table to check duration should be always greater than zero. Display appropriate message.
3. Write a trigger before deleting an employee record from employee table. Raise a notice and display the message “Employee record is being deleted”.

SET B **(Number of Slots – 2)**

1) Railway Reservation Database

Consider the following Entities and their Relationships for Railway Reservation database.

Train (tno int, tname varchar (20), depart_time time, arrival_time time, source_stn char (10), dest_stn char (10), no_of_res_bogies int ,bogie_capacity int)

Passenger (passenger_id int, passenger_name varchar (20), address varchar (30), age int, gender char)

Relationship between Train and Passenger is many to many with descriptive attribute ticket.

Ticket (train_no int, passenger_id int, ticket_no int,bogie_no int, no_of_berths int, tdate date, ticket_amt decimal (7,2),status char)

Constraints: Primary Key,

Status of a berth can be 'W' (waiting) or 'C' (confirmed)

Create trigger for the following:

1. Write a trigger to restrict the bogie capacity of any train to 30.
2. Write a trigger after insert on passenger to display message “Age above 5 will be charged full fare” if age of passenger is more than 5.
3. Write a trigger to restrict no. of berths of ticket booking to 20.

2) Bus Transport Database

Consider the following Entities and their Relationships for Bus Transport database.

Bus (bus_no int , b_capacity int , depot_name varchar(20))

Route (route_no int, source char (20), destination char (20), no_of_stations int)

Driver (driver_no int ,driver_name char(20), license_no int, address char(20), d_age int , salary float)

Relationship between Bus and Route is many to one and relationship between Bus and Driver is many to many with descriptive attributes date_of_duty_allotted and shift.

Constraints: Primary Key

license_no is unique, b_capacity should not be null,
shift can be 1 (Morning) or 2(Evening).

Create trigger for the following:

1. Write a trigger after insert or update the record of driver if the age is between 18 and 50 give the message “valid entry” otherwise give appropriate message.
2. Write a trigger after delete the record of bus having capacity < 10. Display the appropriate message.
3. Write a trigger which will prevent deleting drivers living in________.

<u>SET C</u> **(Number of Slots – 2)**

1) Student Competition Database

Consider the following Entities and their Relationships for Student-Competition database.

Student (<u>sreg_no</u> int ,s_name varchar(20), s_class char(10))

Competition (<u>c_no</u> int ,c_name varchar(20), c_type char(10))

Relationship between Student and Competition is many to many with descriptive attributes rank and year.

Constraints: Primary Key,
c_type should not be null,
c_type can be ‘sport’ or ‘academic**’.**

Create trigger for the following:

1. Write a trigger that restricts insertion of rank value greater than 3. (Raise user defined exception and give appropriate massage)
2. Write a trigger on relationship table .If the year entered is greater than current year, it should display message “Year is Invalid”.
3. Write a trigger on relationship table .If the year entered is greater than current year, it should be changed current year.

2) Bank Database

Consider the following Entities and their Relationships for Bank database.

Branch (<u>br_id</u> integer, br_name char (30), br_city char (10))

Customer (<u>cno</u> integer, c_name char (20), caddr char (35), city char (20))

Loan_application (lno integer, l_amt_required money, l_amt_approved money, l_date date)

Relationship between Branch, Customer and Loan_application is Ternary.

Ternary (br_id integer, cno integer, lno integer)

Constraints: Primary Key,
l_amt_required should be greater than zero.

Create trigger for the following:

1. Write a trigger which will execute when you update customer number from customer. Display message “You can’t change existing customer number”.

2. Write a trigger to validate the loan amount approved. It must be less than the loan amount required.
3. Write a trigger before insert record of customer. If the customer number is less than or equal to zero and customer name is null then give the appropriate message.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well done [ ]

Signature of the instructor:______________________Date:______________________