SET A

Q.1 Consider the following entities and their relationships.

Employee (emp\_id, emp\_name, address) Investment (inv\_no, inv\_name, inv\_date, inv\_amount) Relation between Employee and Investment is One to Many.

Constraint: Primary key, inv\_amount should be > 0. Create a RDB in 3NF and write PL/SQL procedure for the following:

postgres=# create database mydb;

CREATE DATABASE

mydb=# create table employee(e\_id int primary key,e\_name varchar not null,address varch

ar);

CREATE TABLE

mydb=# create table investment(inv\_no int primary key,inv\_name varchar not null,inv\_dat

e date not null,inv\_amount decimal not null check(inv\_amount>0),e\_id int references emp

loyee);

CREATE TABLE

mydb=# insert into employee values(1,'prathmesh','pune');

INSERT 0 1

mydb=# insert into employee values(2,'Mr.patil,','pune');

INSERT 0 1

mydb=# insert into employee values(3,'Manu,','chakan');

INSERT 0 1

mydb=# insert into employee values(4,'patel,','nashik');

INSERT 0 1

mydb=# insert into employee values(5,'shete,','mumbai');

INSERT 0 1

mydb=# insert into investment values(101,'mutual fund','7/6/23',75000,1);

INSERT 0 1

mydb=# insert into investment values(102,'mutual fund','5/8/23',60000,2);

INSERT 0 1

mydb=# insert into investment values(103,'LIC','4/9/23',50000,3);

INSERT 0 1

mydb=# insert into investment values(104,'LIC','2/3/23',59000,4);

INSERT 0 1

mydb=# insert into investment values(105,'SIP','8/7/23',80000,5);

INSERT 0 1

mydb=# select \* from employee;

e\_id | e\_name | address

------+-----------+---------

1 | prathmesh | pune

2 | Mr.patil, | pune

3 | Manu, | chakan

4 | patel, | nashik

5 | shete, | mumbai

(5 rows)

mydb=# select \* from investment;

inv\_no | inv\_name | inv\_date | inv\_amount | e\_id

--------+-------------+------------+------------+------

101 | mutual fund | 2023-07-06 | 75000 | 1

102 | mutual fund | 2023-05-08 | 60000 | 2

103 | LIC | 2023-04-09 | 50000 | 3

104 | LIC | 2023-02-03 | 59000 | 4

105 | SIP | 2023-08-07 | 80000 | 5

(5 rows)

**1) Write a stored function which will display details of employees invested amount in Mutual Fund**

mydb=# create or Replace function emp\_mutual()

mydb-# Returns void as $$

mydb$# Declare

mydb$# emp\_record Record;

mydb$# begin

mydb$# for emp\_record in

mydb$# select \* from employee e, investment i where e.e\_id =i.e\_id and i.inv\_name='mut

ual fund'

mydb$# loop

mydb$# Raise notice'Emp Id:%,Name:%,address:%,Investment Amount:%',emp\_record.e\_id,emp

\_record.e\_name,emp\_record.address,emp\_record.inv\_amount;

mydb$# end loop;

mydb$# end;

mydb$# $$ language plpgsql;

CREATE FUNCTION

mydb=# select emp\_mutual();

NOTICE: Emp Id:1,Name:prathmesh,address:pune,Investment Amount:75000

NOTICE: Emp Id:2,Name:Mr.patil,,address:pune,Investment Amount:60000

emp\_mutual

------------

(1 row)

**2) Write a stored function which will display details of investment of employee “Mr.Patil”**

mydb=# create or Replace function emp\_patil()

mydb-# Returns void as $$

mydb$# Declare

mydb$# inv\_record Record;

mydb$# begin

mydb$# for inv\_record in

mydb$# select \* from employee e, investment i where e.e\_id =i.e\_id and e.e\_name='Mr.pa

til,'

mydb$# loop

mydb$# Raise notice'Investment no:%,Investment Name:%,Investment Date:%,Investment Amo

unt:%',inv\_record.inv\_no,inv\_record.inv\_name,inv\_record.inv\_date,inv\_record.inv\_amount;

mydb$# end loop;

mydb$# end;

mydb$# $$ language plpgsql;

CREATE FUNCTION

mydb=# select emp\_patil();

NOTICE: Investment no:102,Investment Name:mutual fund,Investment Date:2023-05-08,Inves

tment Amount:60000

emp\_patil

-----------

(1 row)

**3) Write a stored function which will display date wise investment details.**

mydb=# create or Replace function due\_details()

mydb-# Returns void as $$

mydb$# Declare

mydb$# inv\_record Record;

mydb$# begin

mydb$# for inv\_record in

mydb$# select \* from employee e, investment i where e.e\_id =i.e\_id order by i.inv\_date

mydb$# loop

mydb$# Raise notice'Date:%,Inv No:%,Inv Name:%,Amount:%,Employee Name:%',inv\_record.in

v\_no,inv\_record.inv\_name,inv\_record.inv\_date,inv\_record.inv\_amount,inv\_record.e\_name;

mydb$# end loop;

mydb$# end;

mydb$# $$ language plpgsql;

CREATE FUNCTION

mydb=# select due\_details();

NOTICE: Date:104,Inv No:LIC,Inv Name:2023-02-03,Amount:59000,Employee Name:patel,

NOTICE: Date:103,Inv No:LIC,Inv Name:2023-04-09,Amount:50000,Employee Name:Manu,

NOTICE: Date:102,Inv No:mutual fund,Inv Name:2023-05-08,Amount:60000,Employee Name:Mr.

patil,

NOTICE: Date:101,Inv No:mutual fund,Inv Name:2023-07-06,Amount:75000,Employee Name:pra

thmesh

NOTICE: Date:105,Inv No:SIP,Inv Name:2023-08-07,Amount:80000,Employee Name:shete,

due\_details

-------------

(1 row)

**Q.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).**

**CREATE TABLE TRAIN(train\_no int primary key, 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);**

**CREATE TABLE PASSENGER(passenger\_id int primary key, passenger\_name varchar(20), address varchar(30),**

**age int ,gender char);**

**CREATE TABLE TICKET( train\_no int references TRAIN, passenger\_id int references PASSENGER, ticket\_no int ,**

**bogie\_no int, no\_of\_berths int ,tdate date , ticket\_amt decimal(7,2),status char check(status in('W','C')));**

**insert into TRAIN values(11,'vande bharat','08:00:00','09:00:00','mumbai','vashi',8,9);**

**insert into TRAIN values(12,'Bharat express','10:00:00','14:00:00','kokan','goa',7,10);**

**insert into PASSENGER values(101,'rutuja','pune',23,'f');**

**insert into PASSENGER values(102,'tejal','mumbai',25,'f');**

**insert into TICKET values(11,102,1234,4,5,'2024-10-09',500.00,'W');**

**insert into TICKET values(12,101,5678,9,8,'2023-11-19',900.00,'C');**

**select \* from TRAIN;**

**train\_no | train\_name | depart\_time | arrival\_time | source\_stn | dest\_stn | no\_of\_res\_bogies | bogie\_capacity**

**----------+----------------+-------------+--------------+------------+----------+------------------+----------------**

**11 | vande bharat | 08:00:00 | 09:00:00 | mumbai | vashi | 8 | 9**

**12 | Bharat express | 10:00:00 | 14:00:00 | kokan | goa | 7 | 10**

**(2 rows)**

**CREATE TABLE**

**select \* from PASSENGER;**

**passenger\_id | passenger\_name | address | age | gender**

**--------------+----------------+---------+-----+--------**

**101 | rutuja | pune | 23 | f**

**102 | tejal | mumbai | 25 | f**

**(2 rows)**

**CREATE TABLE**

**select \* from TICKET;**

**train\_no | passenger\_id | ticket\_no | bogie\_no | no\_of\_berths | tdate | ticket\_amt | status**

**----------+--------------+-----------+----------+--------------+------------+------------+--------**

**11 | 102 | 1234 | 4 | 5 | 2024-10-09 | 500.00 | W**

**12 | 101 | 5678 | 9 | 8 | 2023-11-19 | 900.00 | C**

**(2 rows)**

**CREATE TABLE**

**1. Write a stored function to print the details of train wise confirmed bookings on date “2024-11-06” (Accept date as input parameter).Raise an error in case of invalid date.**

**CREATE OR REPLACE FUNCTION confirmed\_bookings(input\_date DATE) RETURNS VOID AS $$**

**DECLARE**

**booking\_record RECORD;**

**BEGIN**

**IF input\_date IS NULL OR input\_date < CURRENT\_DATE THEN**

**RAISE EXCEPTION 'Invalid date: %', input\_date;**

**END IF;**

**FOR booking\_record IN**

**SELECT \*FROM TRAIN t,Ticket ti where t.train\_no = ti.train\_no and**

**ti.tdate = input\_date AND ti.status = 'C'**

**LOOP**

**RAISE NOTICE 'Train No: %, Train Name: %, Ticket No: %, No of Berths: %, Ticket Amount: %',**

**booking\_record.train\_no, booking\_record.train\_name, booking\_record.ticket\_no,**

**booking\_record.no\_of\_berths, booking\_record.ticket\_amt;**

**END LOOP;**

**END;**

**$$ LANGUAGE plpgsql;**

1. **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.**

**CREATE OR REPLACE FUNCTION ticket\_details(input\_name VARCHAR, input\_date DATE) RETURNS VOID AS $$**

**DECLARE**

**ticket\_record RECORD;**

**passenger\_count INT;**

**BEGIN**

**SELECT COUNT(\*) INTO passenger\_count**

**FROM PASSENGER**

**WHERE passenger\_name = input\_name;**

**IF passenger\_count = 0 THEN**

**RAISE EXCEPTION 'Invalid passenger name: %', input\_name;**

**END IF;**

**FOR ticket\_record IN**

**SELECT ti.train\_no, t.train\_name, ti.ticket\_no, ti.no\_of\_berths, ti.ticket\_amt**

**FROM TICKET ti**

**INNER JOIN PASSENGER p ON ti.passenger\_id = p.passenger\_id**

**INNER JOIN TRAIN t ON ti.train\_no = t.train\_no**

**WHERE p.passenger\_name = input\_name AND ti.tdate = input\_date**

**LOOP**

**RAISE NOTICE 'Train No: %, Train Name: %, Ticket No: %, No of Berths: %, Ticket Amount: %',**

**ticket\_record.train\_no, ticket\_record.train\_name,**

**ticket\_record.ticket\_no, ticket\_record.no\_of\_berths,**

**ticket\_record.ticket\_amt;**

**END LOOP;**

**END;**

**$$ LANGUAGE plpgsql;**

**CREATE FUNCTION**

**SELECT ticket\_details('rutuja', '2023-11-19')**

**ticket\_details**

**----------------**

**(1 row)**

**psql:commands.sql:89: NOTICE: Train No: 12, Train Name: Bharat express, Ticket No: 5678, No of Berths: 8, Ticket Amount: 900.00**

**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.**

**CREATE OR REPLACE FUNCTION deatils\_train(input\_name VARCHAR) RETURNS VOID AS $$**

**DECLARE**

**ticket\_record RECORD;**

**train\_count INT;**

**BEGIN**

**SELECT COUNT(\*) INTO train\_count FROM TRAIN WHERE train\_name = input\_name;**

**IF train\_count = 0 THEN**

**RAISE EXCEPTION 'Invalid train name: %', input\_name;**

**END IF;**

**FOR ticket\_record IN**

**SELECT \*FROM TICKET ti,TRAIN t where ti.train\_no = t.train\_no and t.train\_name = input\_name**

**LOOP**

**RAISE NOTICE 'Train No: %, Train Name: %, Ticket No: %, No of Berths: %, Ticket Amount: %,**

**Status:%',ticket\_record.train\_no, ticket\_record.train\_name, ticket\_record.ticket\_no,**

**ticket\_record.no\_of\_berths, ticket\_record.ticket\_amt, ticket\_record.status;**

**END LOOP;**

**END;**

**$$ LANGUAGE plpgsql;**

**CREATE FUNCTION**

**select deatils\_train('vande bharat')**

**CREATE FUNCTION**

**deatils\_train**

**---------------**

**(1 row)**

**psql:commands.sql:114: NOTICE: Train No: 11, Train Name: vande bharat, Ticket No: 1234, No of Berths: 5, Ticket Amount: 500.00,**

**Status:W**

**SET B**

**Q.3 Consider the following entities and their relationships.**

**Library(Lno, Lname, Location, Librarian, no\_of\_books) Book(Bid, Bname, Author\_Name, Price, publication) Relation between Library and Book is one to many.**

**Constraint: Primary key, Price should not be null. Create a RDB in 3NF and write PL/SQL stored function for the following:**

**select \* from library;**

**l\_id | l\_name | location | librarian | no\_of\_books**

**------+--------+----------+-----------+-------------**

**1 | lib A | pune | Mayuri | 20**

**2 | lib B | nashik | Manu | 30**

**3 | lib C | moshi | Parth | 90**

**(3 rows)**

**mydb=# select \* from book;**

**b\_id | b\_name | author\_name | price | publication | l\_id**

**------+----------+-------------+-------+-------------+------**

**11 | sambhaji | xyz | 250 | nirali | 1**

**12 | mrugjal | abc | 300 | nvneet | 2**

**13 | success | mno | 200 | target | 3**

**(3 rows)**

1. **Write a stored function which will accept publication name from user and display total price of books of that publication.**

mydb=# create or replace function total\_price(pubn

ame varchar) returns void as $$

mydb$# declare

mydb$# total float;

mydb$# begin

mydb$# select sum(price) into total from book whe

re publication=pubname;

mydb$# raise notice 'total price for % is %',pubn

ame,total;

mydb$# end;

mydb$# $$ language 'plpgsql';

CREATE FUNCTION

mydb=# select total\_price('nirali');

NOTICE: total price for nirali is 250

total\_price

-------------

1. row)

**2) Write a stored function which will display to calculate the total number of books in each library.**

CREATE OR REPLACE FUNCTION t\_books(lib\_no I

NTEGER)

mydb-# RETURNS INTEGER AS $$

mydb$# DECLARE

mydb$# total\_books INT;

mydb$# BEGIN

mydb$# SELECT COUNT(\*) INTO total\_books

mydb$# FROM Book

mydb$# WHERE l\_id = lib\_no;

mydb$# RETURN total\_books;

mydb$# END;

mydb$# $$ LANGUAGE plpgsql;

CREATE FUNCTION

mydb=# select t\_books(2);

t\_books

---------

1

1. row)

**3) Write a stored function to find books by a specific author**

create or replace function author\_book(anam

e varchar) returns void as $$

mydb$# declare

mydb$# book\_name varchar;

mydb$# begin

mydb$# select b.b\_name into book\_name from book b

where author\_name=aname;

mydb$# raise notice 'book name:%',book\_name;

mydb$# end;

mydb$# $$ language plpgsql;

CREATE FUNCTION

mydb=# select author\_book('mno');

NOTICE: book name:success

author\_book

-------------

(1 row)

**Q.4. 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 TABLE Branch (

br\_id INTEGER PRIMARY KEY,

br\_name CHAR(30) NOT NULL,

br\_city CHAR(10) NOT NULL

);

CREATE TABLE Customer (

cno INTEGER PRIMARY KEY,

c\_name CHAR(20) NOT NULL,

caddr CHAR(35),

city CHAR(20) NOT NULL

);

CREATE TABLE Loan\_applicationn (

lno INTEGER PRIMARY KEY,

l\_amt\_required float NOT NULL CHECK (l\_amt\_required > 0), -- Enforce l\_amt\_required > 0

l\_amt\_approved float,

l\_date DATE NOT NULL

);

CREATE TABLE Ternary (

br\_id INT references Branch,

cno INT references Customer,

lno INT references Loan\_applicationn

);

insert into Branch values(101,'manchar','pune');

insert into Branch values(102,'vashi','mumbai');

insert into Customer values(11,'rutuja','khed','pune');

insert into Customer values(12,'omkar','sangamner','Ahemdnagar');

insert into Loan\_applicationn values(111,200000,100000,'2024-11-09');

insert into Loan\_applicationn values(222,300000,200000,'2024-08-12');

insert into Ternary values(101,11,222);

insert into Ternary values(102,12,111);

select \* from Branch;

br\_id | br\_name | br\_city

-------+--------------------------------+------------

101 | manchar | pune

102 | vashi | mumbai

(2 rows)

CREATE TABLE

select \* from Customer;

cno | c\_name | caddr | city

-----+----------------------+-------------------------------------+----------------------

11 | rutuja | khed | pune

12 | omkar | sangamner | Ahemdnagar

(2 rows)

CREATE TABLE

select \* from Loan\_applicationn;

lno | l\_amt\_required | l\_amt\_approved | l\_date

-----+----------------+----------------+------------

111 | 200000 | 100000 | 2024-11-09

222 | 300000 | 200000 | 2024-08-12

(2 rows)

CREATE TABLE

select \* from Ternary;

br\_id | cno | lno

-------+-----+-----

101 | 11 | 222

102 | 12 | 111

(2 rows)

CREATE TABLE

**1).Write a stored function to accept branch name and display customer details whose loan amount required is more than loan approved.**

CREATE OR REPLACE FUNCTION pending\_loans(branch\_name VARCHAR)

RETURNS VOID AS $$

DECLARE

customer\_record RECORD;

BEGIN

FOR customer\_record IN

SELECT c.cno, c.c\_name, c.caddr, c.city, l.l\_amt\_required, l.l\_amt\_approved

FROM Branch b

JOIN Ternary t ON b.br\_id = t.br\_id

JOIN Customer c ON t.cno = c.cno

JOIN Loan\_applicationn l ON t.lno = l.lno

WHERE b.br\_name = branch\_name AND l.l\_amt\_required > l.l\_amt\_approved

LOOP

RAISE NOTICE 'Customer No: %, Name: %, Address: %, City: %, Loan Required: %, Loan Approved: %',

customer\_record.cno, customer\_record.c\_name, customer\_record.caddr,

customer\_record.city, customer\_record.l\_amt\_required, customer\_record.l\_amt\_approved;

END LOOP;

IF NOT FOUND THEN

RAISE NOTICE 'No customers found with pending loans in branch %', branch\_name;

END IF;

END;

$$ LANGUAGE plpgsql;

CREATE FUNCTION

SELECT pending\_loans('manchar')

pending\_loans

---------------

(1 row)

psql:commands.sql:90: NOTICE: Customer No: 11, Name: rutuja

**2).Write a stored function to accept branch name and display customername, loan number, loan amount approved on or after 01/06/2019.**

CREATE OR REPLACE FUNCTION loandate(branchName CHAR(30)) RETURNS VOID AS $$

declare

rec record;

BEGIN

FOR rec IN

SELECT c.c\_name, t.lno, l.l\_amt\_approved

FROM Customer c, Ternary t, Branch b, Loan\_applicationn l where c.cno = t.cno

and b.br\_id = t.br\_id

and l.lno = t.lno

and b.br\_name = branchName

and l.l\_date >= '2019-06-01'

LOOP

RAISE NOTICE 'Customer: %, Loan No: %, Approved Amount: %', rec.c\_name, rec.lno,rec.l\_amt\_approved;

END LOOP;

END;

$$ language plpgsql;

SELECT loandate('manchar')

CREATE FUNCTION

loandate

----------

(1 row)

psql:commands.sql:107: NOTICE: Customer: rutuja, Loan No: 222, Approved Amount: 200000

**3).Write a stored function to display total loan amount approved by all branches after date 30/05/2019.**

CREATE FUNCTION totloan()

RETURNS VOID

AS $$

DECLARE

ttotalApproved int;

BEGIN

SELECT SUM(l.l\_amt\_approved)

INTO totalApproved

FROM Loan\_applicationn l

WHERE l.l\_date > '2023-10-09';

RAISE NOTICE 'Total loan amount approved after May 30, 2019: %',totalApproved;

END;

$$ language plpgsql;

SELECT totloan()

CREATE FUNCTION

totloan

---------

(1 row)

psql:commands.sql:119: NOTICE: Total loan amount approved after May 30, 2019: 300000

**4).Write a stored function to display customer details who have applied for loan more than one branches.**

CREATE OR REPLACE FUNCTION multiple() RETURNS void AS $$

declare

rec record;

BEGIN

FOR rec IN

SELECT c.c\_name, c.caddr, c.city

FROM Customer c,Ternary t where c.cno = t.cno

GROUP BY c.cno, c.c\_name, c.caddr, c.city

HAVING COUNT(DISTINCT t.br\_id) > 101

LOOP

RAISE NOTICE 'Customer: %, Address: %, City: %', rec.c\_name, rec.caddr, rec.city;

END LOOP;

END;

$$ language plpgsql;

SELECT multiple()

CREATE FUNCTION

multiple

----------

(1 row)

**EXCEPTION HANDLING**

**Q.1 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.**

CREATE TABLE Movies (

m\_name VARCHAR(25) NOT NULL primary key,

release\_year INTEGER NOT NULL,

budget float NOT NULL

);

CREATE TABLE Actor (

a\_name CHAR(30) NOT NULL primary key,

role CHAR(30) NOT NULL,

charges float NOT NULL,

a\_address VARCHAR(30)

);

CREATE TABLE Producer (

producer\_id INTEGER PRIMARY KEY,

name CHAR(30) NOT NULL,

p\_address VARCHAR(30)

);

CREATE TABLE Movie\_Actor (

m\_name VARCHAR(25) REFERENCES Movies,

a\_name CHAR(30) NOT NULL REFERENCES Actor

);

CREATE TABLE Movie\_Producer (

m\_name VARCHAR(25) REFERENCES Movies,

producer\_id INTEGER REFERENCES Producer

);

insert into Movies values('sholey',1978,1000000);

insert into Movies values('sahoo',2016,3000000);

insert into Movies values('2.0',2023,5000000);

insert into Actor values('amitabh','lead',200000,'mumbai');

iinsert into Actor values('prbhas','lead',500000,'chennai');

insert into Actor values('akshay','support',300000,'pune');

insert into Producer values(1,'sunil','pune');

insert into Producer values(2,'omkar','mumbai');

insert into Producer values(3,'suresh','goa');

insert into Movie\_Actor values('sholey','amitabh');

insert into Movie\_Actor values('2.0','prbhas');

insert into Movie\_Actor values('sahoo','akshay');

insert into Movie\_Producer values('sholey',1);

insert into Movie\_Producer values('sahoo',3);

insert into Movie\_Producer values('2.0',2);

select \* from Movies;

m\_name | release\_year | budget

--------+--------------+---------

sholey | 1978 | 1000000

sahoo | 2016 | 3000000

2.0 | 2023 | 5000000

(3 rows)

CREATE TABLE

select \* from Actor;

a\_name | role | charges | a\_address

--------------------------------+--------------------------------+---------+-----------

amitabh | lead | 200000 | mumbai

prbhas | lead | 500000 | chennai

akshay | support | 300000 | pune

(3 rows)

CREATE TABLE

select \* from Producer;

producer\_id | name | p\_address

--------------+--------------------------------+-----------

1 | sunil | pune

2 | omkar | mumbai

3 | suresh | goa

(3 rows)

CREATE TABLE

select \* from Movie\_Actor;

m\_name | a\_name

--------+--------------------------------

sholey | amitabh

2.0 | prbhas

sahoo | akshay

(3 rows)

CREATE TABLE

**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.**

CREATE OR REPLACE FUNCTION actordet(movie\_name VARCHAR)

RETURNS VOID AS $$

DECLARE

movie\_count INT;

rec record;

BEGIN

SELECT COUNT(\*) INTO movie\_count FROM Movies WHERE m\_name = movie\_name;

IF movie\_count = 0 THEN

RAISE EXCEPTION 'Error: Movie "

%" does not exist.', movie\_name;

END IF;

FOR rec IN

SELECT a.a\_name, a.role, a.charges

FROM Actor a, Movie\_Actor ma where a.a\_name = ma.a\_name

and ma.m\_name = movie\_name

ORDER BY a.charges DESC

LOOP

RAISE NOTICE 'Actor: %, Role: %

, Charges: %', rec.a\_name, rec.role, rec.charges;

END LOOP;

END; -------

END;

$$ LANGUAGE PLPGSQL;

CREATE FUNCTION

SELECT actordet('sholey')

actordet

----------

(1 row)

psql:commands.sql:91: NOTICE: Actor: amitabh , Role: lead

, Charges: 200000

**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.**

CREATE OR REPLACE FUNCTION getmovie(actor\_name VARCHAR) RETURNS VOID AS $$

DECLARE

actor\_count INT;

rec record;

BEGIN

SELECT COUNT(\*) INTO actor\_count FROM Actor WHERE a\_name = actor\_name;

IF actor\_count = 0 THEN

RAISE EXCEPTION 'Error: Actor "

%" does not exist.', actor\_name;

END IF;

FOR rec IN

SELECT m.m\_name

FROM Movies m, Movie\_Actor ma where m.m\_name = ma.m\_name

and ma.a\_name = actor\_name

LOOP

RAISE NOTICE 'Movie: %', rec.m\_name;

END LOOP;

END;

$$ LANGUAGE plpgsql;

CREATE FUNCTION

SELECT getmovie('amitabh')

getmovie

----------

(1 row)

psql:commands.sql:116: NOTICE: Movie: sholey

**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.**

CREATE OR REPLACE FUNCTION prodcount(producer\_name VARCHAR)

RETURNS VOID AS $$

DECLARE

producer\_count INT;

movie\_count INT;

BEGIN

SELECT COUNT(\*) INTO producer\_count

FROM Producer WHERE name = producer\_name;

IF producer\_count = 0 THEN

RAISE EXCEPTION 'Error: Producer "%" does not exist.', producer\_name;

END IF;

SELECT COUNT(m.m\_name) INTO movie\_count

FROM Movies m, Movie\_Producer mp, Producer p

where m.m\_name = mp.m\_name

and mp.producer\_id = p.producer\_id

and p.name = producer\_name;

RAISE NOTICE 'Producer "%" has prod

uced % movies.', producer\_name, movie\_count;

END;

$$ LANGUAGE plpgsql;

CREATE FUNCTION

SELECT prodcount('omkar')

prodcount

-----------

(1 row)

psql:commands.sql:142: NOTICE: Producer "omkar" has prod

uced 1 movies.

**Q.2 Consider the following entities and their relationships. Wholesaler (w\_no, w\_name, address, city)**

**Product (product\_no, product\_name, rate)**

**Relation between Wholesaler and Product is Many to Many with quantity as descriptive attribute.**

**Constraint: Primary key, rate should be > 0.**

CREATE TABLE Wholesaler (

w\_no int PRIMARY KEY,

w\_name VARCHAR(50) NOT NULL,

address VARCHAR(100),

city VARCHAR(50));

CREATE TABLE Product (

product\_no int PRIMARY KEY,

product\_name VARCHAR(50) NOT NULL,

rate DECIMAL(10, 2) CHECK (rate > 0));

CREATE TABLE Wholesaler\_Product (

w\_no INTEGER references Wholesaler,

product\_no INTEGER references Product,

Quantity int);

insert into Wholesaler values(1,'pavan','moshi','pune');

insert into Wholesaler values(2,'ram','vashoi','mumbai');

insert into Product values(11,'pencil',70);

insert into Product values(12,'pen',50);

insert into Wholesaler\_Product values(1,12,100);

insert into Wholesaler\_Product values(2,11,200);

select \* from Wholesaler;

w\_no | w\_name | address | city

------+--------+---------+--------

1 | pavan | moshi | pune

2 | ram | vashoi | mumbai

(2 rows)

CREATE TABLE

select \* from Product;

product\_no | product\_name | rate

------------+--------------+-------

11 | pencil | 70.00

12 | pen | 50.00

(2 rows)

CREATE TABLE

select \* from Wholesaler\_Product;

w\_no | product\_no | quantity

------+------------+----------

1 | 12 | 100

2 | 11 | 200

(2 rows)

CREATE TABLE

**1. Write a stored function to accept quantity from user. Quantity must be within range 50-200. If user enters the quantity out of range, then raise an user defined exception “quantity\_out\_of \_range” otherwise enter the record in table.**

CREATE OR REPLACE FUNCTION insertprod(w\_no INTEGER, product\_no INTEGER, Quantity INTEGER)

RETURNS VOID AS $$

BEGIN

IF Quantity < 50 OR Quantity > 200 THEN

RAISE EXCEPTION 'Error: Quantity % is out of the allowed range (50-200).', Quantity;

END IF;

INSERT INTO Wholesaler\_Product VALUES (w\_no, product\_no, Quantity);

RAISE NOTICE 'Record inserted successfully';

END;

$$ LANGUAGE plpgsql;

select insertprod(1,12,100)

CREATE FUNCTION

insertprod

------------

(1 row)

psql:commands.sql:50: NOTICE: Record inserted successfully

**2. Write a stored function to accept rate from user. If user enters rate less than or equal to zero then raise an user defined exception “Invalid\_Rate\_Value” otherwise display message “Correct Input”**

CREATE OR REPLACE FUNCTION ValidateRate(rate int)RETURNS VOID AS $$

BEGIN

IF rate <= 0 THEN

RAISE EXCEPTION 'Error: Invalidrate value %. Rate must be greater than zero.', rate;

END IF;

RAISE NOTICE 'Correct Input: Rate %', rate;

END;

$$ LANGUAGE plpgsql;

CREATE FUNCTION

select ValidateRate(50)

validaterate

--------------

(1 row)

psql:commands.sql:91: NOTICE: Correct Input: Rate 50

**3. Write a stored function to accept product name as parameter. If entered product name is not valid then raise an user defined exception”Invalid\_Product\_Name” otherwise display product details of specified product.**

CREATE OR REPLACE FUNCTION GetProductDetails(pname VARCHAR)RETURNS VOID AS $$

DECLARE

rec RECORD;

BEGIN

SELECT \* INTO rec FROM Product WHERE product\_name = pname;

IF rec IS NULL THEN

RAISE EXCEPTION 'Error: Invalidproduct name %.', pname;

END IF;

RAISE NOTICE 'Product No: %, Product Name: %, Rate: %', rec.product\_no, rec.product\_name, rec.rate;

END;

$$ LANGUAGE plpgsql;

SELECT GetProductDetails('pen')

CREATE FUNCTION

getproductdetails

-------------------

(1 row)

psql:commands.sql:106: NOTICE: Product No: 12, Product Name: pen, Rate: 50.00

**SET C**

**1. Consider the following entities and their relationships. Student (rollno, sname, class, timetable, mobileno) Lab (LabNo, LabName, capacity, equipment) Relation between Student and Lab is Many to One. Constraint: Primary Key, capacity should not be null.**

db=# create table lab( lab\_no int primary key, lab\_name varchar(25),capacity int,equipment text);

CREATE TABLE

db=# create table student(roll\_no int primary key,s\_name varchar(20),class varchar(25),timetable time, mb\_no varchar(20),lab\_no int references lab);

CREATE TABLE

db=# create table student(roll\_no int primary key,s\_name varchar(20),class varchar(25),timetable time, mb\_no varchar(20),lab\_no int references lab);

CREATE TABLE

db=# insert into lab values(1,'java',25,'pc');

INSERT 0 1

db=# insert into lab values(2,'os',54,'cpu');

INSERT 0 1

db=# insert into lab values(2,'fod',40,'laptop');

ERROR: duplicate key value violates unique constraint "lab\_pkey"

DETAIL: Key (lab\_no)=(2) already exists.

db=# insert into lab values(3,'fod',40,'laptop');

INSERT 0 1

db=# insert into student values(11,'manu','msc','8:10:00');

INSERT 0 1

db=# insert into student values(11,'manu','msc','8:10:00','9977886453',1);

INSERT 0 1

db=# insert into student values(12,'mayuri','mca','9:30:00','8977886453',2);

INSERT 0 1

db=# insert into student values(13,'kajal','mba','10:00:00','7977886453',3);

INSERT 0 1

db=# select \* from lab;

lab\_no | lab\_name | capacity | equipment

--------+----------+----------+-----------

1 | java | 25 | pc

2 | os | 54 | cpu

3 | fod | 40 | laptop

(3 rows)

db=# select \* from student;

roll\_no | s\_name | class | timetable | mb\_no | lab\_no

---------+--------+-------+-----------+------------+--------

11 | manu | msc | 08:10:00 | 9977886453 | 1

12 | mayuri | mca | 09:30:00 | 8977886453 | 2

13 | kajal | mba | 10:00:00 | 7977886453 | 3

(3 rows)

**1) Write a function to accept lab number from user as parameter. ” if user enters invalid lab number then raise an user defined exception “Invalid\_Lab\_No” otherwise display the student details of the same lab**

db=# CREATE OR REPLACE FUNCTION stud(lno INTEGER) RETURNS VOID AS $$

db$# DECLARE

db$# student\_count INTEGER;

db$# rec record;

db$# BEGIN

db$# SELECT COUNT(\*)

db$# INTO student\_count FROM Student WHERE lab\_no = lno;

db$# IF student\_count = 0 THEN RAISE EXCEPTION 'Error: InvalidLab No %', lno;

db$# END IF;

db$# FOR rec IN

db$# SELECT \* FROM Student WHERE lab\_no = lno

db$# LOOP

db$# RAISE NOTICE 'Student Roll No:%, Name: %, Class: %, Mobile No: %',rec.roll\_no, rec.s\_name, rec.class,

db$# rec.mb\_no;

db$# END LOOP;

db$# END;

db$# $$ LANGUAGE plpgsql;

CREATE FUNCTION

db=# select stud(1);

NOTICE: Student Roll No:11, Name: manu, Class: msc, Mobile No: 9977886453

stud

------

(1 row)

db=# CREATE OR REPLACE FUNCTION stud(lno INTEGER) RETURNS VOID AS $$

db$# DECLARE

db$# student\_count INTEGER;

db$# rec record;

db$# BEGIN

db$# SELECT COUNT(\*)

db$# INTO student\_count FROM Student WHERE lab\_no = lno;

db$# IF student\_count = 0 THEN RAISE EXCEPTION 'Error: InvalidLab No %', lno;

db$# END IF;

db$# FOR rec IN

db$# SELECT \* FROM Student WHERE lab\_no = lno

db$# LOOP

db$# RAISE NOTICE 'Student Roll No:%, Name: %, Class: %, Mobile No: %',rec.roll\_no, rec.s\_name, rec.class,

db$# rec.mb\_no;

db$# END LOOP;

db$# END;

db$# $$ LANGUAGE plpgsql;

CREATE FUNCTION

db=# select stud(5);

ERROR: Error: InvalidLab No 5

CONTEXT: PL/pgSQL function stud(integer) line 8 at RAISE

**2) Write a function which accept a mobile number from user. If mobile no less than or more than 10 digits then raise an user defined exception “Invalid\_Mobile No” otherwise display the “Correct input…! .**

db=# CREATE OR REPLACE FUNCTION mobileno(mobile\_no VARCHAR) RETURNS VOID AS $$

db$# BEGIN

db$# IF LENGTH(mobile\_no) != 10 THEN

db$# RAISE EXCEPTION 'Error: Invalid Mobile No %. It must be exactly 10 digits.', mobile\_no;

db$# END IF;

db$#

db$#

db$# RAISE NOTICE 'Correct input...!';

db$# END;

db$#

db$# $$ LANGUAGE plpgsql;

CREATE FUNCTION

db=# select mobileno('9977886453');

NOTICE: Correct input...!

mobileno

----------

(1 row)

db=# select mobileno('997788656');

ERROR: Error: Invalid Mobile No 997788656. It must be exactly 10 digits.

CONTEXT: PL/pgSQL function mobileno(character varying) line 4 at RAISE

db=# create or replace function labb\_details(lab\_no integer,lab\_name varchar(25),capacity integer,equipment integer)

db-# Returns void as $$

db$# DECLARE

db$# Invalid\_capacity\_range exception;

db$# begin

db$# if capacity>40 then

db$# Raise Invalid\_capacity\_Range using message= 'capacity must be 40 or less';

db$# end if;

db$#

db$# INSERT INTO Lab VALUES (lab\_no, lab\_name, capacity, equipment);

db$#

db$# RAISE NOTICE 'Lab details inserted successfully';

db$# END;

db$# $$ LANGUAGE plpgsql;

ERROR: type "exception" does not exist

LINE 4: Invalid\_capacity\_range exception;

^

**3) Write a PL/SQL block which accepts lab detail from user. If capacity is more than 40 then raise an user defined exception “Invalid\_Capacity\_Range” otherwise insert the record in the table.**

create or replace function labb\_details(lab\_no integer,lab\_name varchar(25),capacity integer,equipment text)

db-# Returns void as $$

db$#

db$# begin

db$# if capacity>40 then

db$# Raise Exception 'Error:Invalid\_capacity\_Range using message= capacity must be 40 or less';

db$# end if;

db$#

db$# INSERT INTO Lab VALUES (lab\_no,lab\_name, capacity, equipment);

db$# --notify that the lab details where successfully inserted

db$# RAISE NOTICE 'Lab details inserted successfully';

db$# END;

db$# $$ LANGUAGE plpgsql;

CREATE FUNCTION

db=#

select labb\_details(5,'phy',50,'capliper');

ERROR: Error:Invalid\_capacity\_Range using message= capacity must be 40 or less

CONTEXT: PL/pgSQL function labb\_details(integer,character varying,integer,text) line 5 at RAISE

db=#

select labb\_details(5,'phy',30,'cpu');

NOTICE: Lab details inserted successfully

labb\_details

--------------

(1 row)

**Q.4**

**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**

create table ROUTE(route\_no int primary key, source char(20), destination char(20),no\_of\_stations int);

create table BUS(bus\_no int primary key , capacity int not null, depot\_name varchar(20),route\_no int references ROUTE);

create table DRIVER(driver\_no int primary key , driver\_name char(20), license\_no int , address char(20), d\_age int , salary float);

create table BUS\_DRIVER(bus\_no int references BUS,driver\_no int references DRIVER, date\_of\_duty date, shift char check(shift in('M','E')));

insert into ROUTE values(101,'khed','bhosari',5);

insert into ROUTE values(102,'swarget','pune',10);

insert into BUS values(111,60,'aalandi',101);

insert into BUS values(222,70,'shivajinagar',102);

insert into DRIVER values(1010,'ram',1234,'pune',34,56000);

insert into DRIVER values(1020,'sham',5678,'moshi',40,60000);

insert into BUS\_DRIVER values(111,1010,'2024-06-05','M');

insert into BUS\_DRIVER values(222,1020,'2024-11-05','E');

select \* from ROUTE;

route\_no | source | destination | no\_of\_stations

----------+----------------------+----------------------+----------------

101 | khed | bhosari | 5

102 | swarget | pune | 10

(2 rows)

CREATE TABLE

select \* from BUS;

bus\_no | capacity | depot\_name | route\_no

--------+----------+--------------+----------

111 | 60 | aalandi | 101

222 | 70 | shivajinagar | 102

(2 rows)

CREATE TABLE

select \* from DRIVER;

driver\_no | driver\_name | license\_no | address | d\_age | salary

-----------+----------------------+------------+----------------------+-------+--------

1010 | ram | 1234 | pune | 34 | 56000

1020 | sham | 5678 | moshi | 40 | 60000

(2 rows)

CREATE TABLE

select \* from BUS\_DRIVER;

bus\_no | driver\_no | date\_of\_duty | shift

--------+-----------+--------------+-------

111 | 1010 | 2024-06-05 | M

222 | 1020 | 2024-11-05 | E

(2 rows)

CREATE TABLE

**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.**

CREATE OR REPLACE FUNCTION GetDriversForBus(bno INT, ddate DATE) RETURNS VOID AS $$

DECLARE

driver\_count INT;

rec record;

BEGIN

SELECT COUNT(\*) INTO driver\_count FROM BUS WHERE bus\_no = bno;

IF driver\_count = 0 THEN

RAISE EXCEPTION 'Error: Invalid Bus Number %', bno;

END IF;

RAISE NOTICE 'Drivers allotted for Bus No % on %:', bno, ddate;

FOR rec IN

SELECT d.driver\_name

FROM DRIVER d,BUS\_DRIVER bd

where d.driver\_no = bd.driver\_no

and bd.bus\_no = bno AND bd.date\_of\_duty = ddate

LOOP

RAISE NOTICE 'Driver: %',rec.driver\_name;

END LOOP;

END;

$$ LANGUAGE plpgsql;

CREATE FUNCTION

SELECT GetDriversForBus(111, '2024-11-05')

getdriversforbus

------------------

(1 row)

psql:commands.sql:53: NOTICE: Drivers allotted for Bus No 111 on 2024-11-05:

**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.**

CREATE OR REPLACE FUNCTION datedriver(dname CHAR(20))

RETURNS VOID AS $$

DECLARE

driver\_count INT;

rec record;

BEGIN

SELECT COUNT(\*)

INTO driver\_count FROM DRIVER WHERE driver\_name = dname;

IF driver\_count = 0 THEN

RAISE EXCEPTION 'Error: Invalid Driver Name %', dname;

END IF;

RAISE NOTICE 'Dates on which Driver % has driven:', dname;

FOR rec IN

SELECT DISTINCT bd.date\_of\_duty

FROM BUS\_DRIVER bd, DRIVER d

where bd.driver\_no = d.driver\_no

and d.driver\_name = dname

LOOP

RAISE NOTICE 'Date: %', rec.date\_of\_duty;

END LOOP;

END;

$$ LANGUAGE plpgsql;

CREATE FUNCTION

SELECT datedriver('ram')

datedriver

------------

(1 row)

psql:commands.sql:115: NOTICE: Dates on which Driver ram has driven:

psql:commands.sql:115: NOTICE: Date: 2024-06-05

**3.Write a stored function to display the details of the buses that run on route\_no = 5. (accept route\_no as input parameter). Raise an error in case of invalid driver name.**

CREATE OR REPLACE FUNCTION GetBusByRoute(rno INT)

RETURNS VOID AS $$

DECLARE

route\_count INT;

rec record;

BEGIN

SELECT COUNT(\*) INTO route\_count FROM ROUTE WHERE route\_no = rno;

IF route\_count = 0 THEN

RAISE EXCEPTION 'Error: Invalid Route Number %', rno;

END IF;

RAISE NOTICE 'Buses running on Route No %:', rno;

FOR rec IN

SELECT b.bus\_no, b.capacity, b.depot\_name

FROM BUS b, ROUTE r

where b.bus\_no = b.bus\_no

and r.route\_no = r.rno

LOOP

RAISE NOTICE 'Bus No: %, Capacity: %, Depot: %',

rec.bus\_no, rec.capacity, rec.depot\_name;

END LOOP;

END;

$$ LANGUAGE plpgsql;

CREATE FUNCTION

SELECT GetBusByRoute(1);

ERROR:Error:Invalid Route Number 1

CONTEXT: PL/pgSQL function GetBusByRoute(integer)

Line 9 at RAISE