Slip1

CREATE TABLE BRANCH(B_ID INT PRIMARY KEY, BR_NAME VARCHAR(20), BR_CITY VARCHAR(20));

CREATE TABLE CUSTOMER(C_NO INT PRIMARY KEY, C_NAME VARCHAR(20), C_ADDR VARCHAR(35), CITY VARCHAR(20));

CREATE TABLE LOAN_APP(L_NO INT PRIMARY KEY, L_AMT_REQ MONEY, L_AMT_APPROVED MONEY, L_DATE DATE);

CREATE TABLE TERNARY(B_ID INT REFERENCES BRANCH(B_ID), C_NO INT REFERENCES CUSTOMER(C_NO), L_NO INT REFERENCES LOAN_APP(L_NO));

--1) To display names of customers for the ‘Pimpri’ branch.

CREATE OR REPLACE VIEW V1 AS

SELECT DISTINCT C_NAME, BR_NAME FROM CUSTOMER A, BRANCH B, TERNARY C WHERE A.C_NO=C.C_NO AND B.B_ID=C.B_ID AND B.BR_NAME='PIMPRI BRANCH';

--2) To display names of customers who have taken loan from the branch in the same city they live.

CREATE OR REPLACE VIEW V2 AS

SELECT DISTINCT C_NAME FROM CUSTOMER A, BRANCH B, TERNARY C WHERE A.CITY=B.BR_CITY AND A.C_NO=C.C_NO AND B.B_ID=C.B_ID;

/*1) Write a trigger which will execute when you update customer number from customer table.

Display message “You can’t change existing customer number”.*/

CREATE OR REPLACE FUNCTION UPD_CNO() RETURNS TRIGGER AS'

BEGIN

 RAISE EXCEPTION ''YOU CANNOT UPDATE EXISTING CUSTOMER NUMBER !!!'';

RETURN 1;

END;

'LANGUAGE 'plpgsql';

CREATE TRIGGER UPD

```
BEFORE UPDATE ON CUSTOMER

FOR EACH ROW

EXECUTE PROCEDURE UPD_CNO();

/*2) Write a stored function to accept branch name as an input parameter and display loan

information of that branch.*/

CREATE OR REPLACE FUNCTION LOAN_INFO(BNAME TEXT) RETURNS INT AS'

DECLARE

 CMD RECORD;

BEGIN

 RAISE NOTICE '' REQUIRED_AMT || APPROVED_AMT   ||     DATE'';

  FOR CMD IN SELECT L_AMT_REQ, L_AMT_APPROVED, L_DATE FROM LOAN_APP A, BRANCH B,
TERNARY C WHERE BNAME=B.BR_NAME AND A.L_NO=C.L_NO AND B.B_ID=C.B_ID

  LOOP

    RAISE NOTICE ''  %     %       %'',CMD.L_AMT_REQ, CMD.L_AMT_APPROVED, CMD.L_DATE;

  END LOOP;

RETURN 1;

END;

'LANGUAGE 'plpgsql';
```

Slip-2

```
-- 1) To display customer details who have applied for a loan of 5,00,000.

CREATE OR REPLACE VIEW V1 AS

SELECT A.C_NAME, A.CITY FROM CUSTOMER A, LOAN_APP B, TERNARY C WHERE A.C_NO=C.C_NO AND
B.L_NO=C.L_NO AND L_AMT_REQ>='500000';


-- 2) To display loan details from the ‘Aundh’ branch.

CREATE OR REPLACE VIEW V2 AS
```

```
SELECT A.* FROM LOAN_APP A, BRANCH B, TERNARY C WHERE A.L_NO=C.L_NO AND B.B_ID=C.B_ID
AND B.BR_NAME='AUNDH BRANCH';

/*1. Write a trigger to validate the loan amount approved. It must be less than or equal to loan

amount required. Display appropriate message.*/

CREATE OR REPLACE FUNCTION VAL_AMT() RETURNS TRIGGER AS'

DECLARE

BEGIN

 IF (NEW.L_AMT_APPROVED > NEW.L_AMT_REQ) THEN

   RAISE EXCEPTION ''VALIDATION FAILED '';

  ELSE

    RAISE INFO ''VALIDATION SUCCESS'';

END IF;

RETURN NEW;

END;

'LANGUAGE 'plpgsql';


CREATE TRIGGER UPD

BEFORE INSERT ON LOAN_APP

FOR EACH ROW

EXECUTE PROCEDURE VAL_AMT();


/*2. Write a stored function to count number of customers of particular branch. (Accept branch

name as an input parameter). Display message for invalid branch name.*/

CREATE OR REPLACE FUNCTION CUST_CNT(BRNAME TEXT) RETURNS INT AS'

DECLARE
```

```
  CNT INT;

BEGIN

 SELECT INTO CNT COUNT(A.C_NO) FROM CUSTOMER A, BRANCH B, TERNARY C WHERE A.C_NO=C.C_NO
AND B.B_ID=C.B_ID AND BRNAME=B.BR_NAME;

  IF CNT=0 THEN

    RAISE NOTICE ''INVALID BRANCH NAME !'';

  END IF;

RETURN CNT;

END;

'LANGUAGE 'plpgsql';
```

Slip-3

```
-- 1) To display the names of customers who required loan > 2,00,000

CREATE OR REPLACE VIEW V1 AS

SELECT DISTINCT A.C_NAME, A.CITY FROM CUSTOMER A, LOAN_APP B, TERNARY C WHERE
A.C_NO=C.C_NO AND B.L_NO=C.L_NO AND L_AMT_REQ>'200000';


-- 2) To display branch wise name of customers

CREATE OR REPLACE VIEW V1 AS

SELECT A.C_NAME, B.BR_NAME FROM CUSTOMER A, BRANCH B, TERNARY C WHERE A.C_NO=C.C_NO
AND B.B_ID=C.B_ID GROUP BY A.C_NAME,B.BR_NAME ORDER BY B.BR_NAME;

/* 1. Write a trigger before inserting record of customer in customer table. If the customer

number is less than or equal to zero then display the appropriate error message.*/

CREATE OR REPLACE FUNCTION UPD_CNO() RETURNS TRIGGER AS'

BEGIN

 IF(NEW.C_NO < 0 OR NEW.C_NO = 0) THEN

   RAISE EXCEPTION''CUSTOMER NUMBER MUST BE GREATER THAN 0'';
 END IF;

RETURN NEW;

END;

'LANGUAGE 'plpgsql';


CREATE TRIGGER AMD

BEFORE INSERT ON CUSTOMER

FOR EACH ROW

EXECUTE PROCEDURE UPD_CNO();


/* 2. Write a cursor to display customer details along with their approved loan amount*/

CREATE OR REPLACE FUNCTION CUSTOMER_DETAILS() RETURNS INT AS'

DECLARE

CURS CURSOR FOR SELECT DISTINCT A.C_NAME,A.CITY,B.L_AMT_APPROVED FROM CUSTOMER A,
LOAN_APP B, TERNARY C

WHERE A.C_NO=C.C_NO AND B.L_NO=C.L_NO GROUP BY A.C_NAME,A.CITY, B.L_AMT_APPROVED;

CNM TEXT;

CITY1 TEXT;

AMT_APPROVED MONEY;

BEGIN

RAISE NOTICE''CUSTOMER NAME || CUSTOMER CITY || AMOUNT APPROVED'';

OPEN CURS;

LOOP

 FETCH CURS INTO CNM,CITY1,AMT_APPROVED;

 EXIT WHEN NOT FOUND;
 RAISE NOTICE''  %       %           %'',CNM,CITY1,AMT_APPROVED;

END LOOP;

CLOSE CURS;

RETURN 1;

END;

'LANGUAGE 'plpgsql';
```

Slip4

```
CREATE TABLE ROUTE(ROUTE_NO INT PRIMARY KEY, SRC VARCHAR(15), DEST VARCHAR(15),
NO_OF_STATION INT);


CREATE TABLE BUS(BUS_NO INT PRIMARY KEY, CAPACITY INT NOT NULL, DEPOT_NAME CHAR(15),
ROUTE_NO INT REFERENCES ROUTE(ROUTE_NO));


CREATE TABLE DRIVER(D_NO INT PRIMARY KEY, D_NAME VARCHAR(10), LIC_NO INT UNIQUE, ADDR
VARCHAR(10), AGE INT, SALARY FLOAT);


CREATE TABLE BUS_DRIVER(BUS_NO INT REFERENCES BUS(BUS_NO), D_NO INT REFERENCES
DRIVER(D_NO), DATE_OF_DUTY DATE, SHIFT INT CHECK(SHIFT IN(1,2)));


-- 1. To display driver details working in Morning shift.

CREATE OR REPLACE VIEW V1 AS

SELECT A.D_NAME, A.LIC_NO, A.ADDR,A.AGE, A.SALARY FROM DRIVER A, BUS_DRIVER B WHERE
A.D_NO=B.D_NO AND B.SHIFT=1;


-- 2. To display driver details having salary > 20,000.

CREATE OR REPLACE VIEW V2 AS

SELECT * FROM DRIVER WHERE SALARY>'10000';
```

```
/* 1. Write a trigger before inserting the driver record in driver table, if the age is not between 18

and 35, then display error message ‘Invalid input’.*/

CREATE OR REPLACE FUNCTION INS_AGE() RETURNS TRIGGER AS'

BEGIN

 IF (NEW.AGE < 18 OR NEW.AGE>35) THEN

   RAISE EXCEPTION ''INVALID INPUT FOR AGE !'';

 END IF;

RETURN NEW;

END;

'LANGUAGE 'plpgsql';


CREATE TRIGGER AGE_INS

BEFORE INSERT ON DRIVER

FOR EACH ROW

EXECUTE PROCEDURE INS_AGE();


/* 2. Write a stored function to display details of buses running on route_no = ‘ ’. (Accept

route_no as an input parameter.) */

CREATE OR REPLACE FUNCTION BUS_DETAILS(RT_NO INT) RETURNS INT AS'

DECLARE

 CMD RECORD;

BEGIN

 RAISE NOTICE ''BUS NO. || BUS CAPACITY || DEPOT NAME'';

 FOR CMD IN SELECT BUS_NO, CAPACITY, DEPOT_NAME FROM BUS WHERE RT_NO=ROUTE_NO

  LOOP
  RAISE NOTICE''  %         %         %'',CMD.BUS_NO, CMD.CAPACITY,CMD.DEPOT_NAME;

 END LOOP;

RETURN 1;

END;

'LANGUAGE 'plpgsql';
```

Slip5

```
-- 1. To display details of Bus_no 102 along with details of all drivers who have driven that bus.

CREATE OR REPLACE VIEW V1 AS

SELECT A., B. FROM BUS A, DRIVER B, BUS_DRIVER C WHERE A.BUS_NO='102' AND
A.BUS_NO=C.BUS_NO AND B.D_NO=C.D_NO;


-- 2. To display the route details on which buses of capacity 30 runs.

CREATE OR REPLACE VIEW V2 AS

SELECT A.* FROM ROUTE A, BUS B WHERE A.ROUTE_NO=B.ROUTE_NO AND B.CAPACITY='35';

/*1. Write a trigger before inserting the driver record in driver table, if the salary is less than or

equal to zero, then return the error message ‘Invalid Salary’.*/

CREATE OR REPLACE FUNCTION CHK_SAL() RETURNS TRIGGER AS'

BEGIN

 IF(NEW.SALARY < 0 OR NEW.SALARY=0) THEN

  RAISE EXCEPTION''INVALID SALARY !'';

END IF;

RETURN NEW;

END;

'LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER CHK

BEFORE INSERT ON DRIVER

FOR EACH ROW

EXECUTE PROCEDURE CHK_SAL();



/* 2. Write a function using cursor to display all the dates on which a driver has driven a bus

(Accept the driver name as an input parameter)*/

CREATE OR REPLACE FUNCTION DATE_DETAILS(DNM TEXT) RETURNS INT AS'

DECLARE

 CU1 CURSOR FOR SELECT DATE_OF_DUTY FROM BUS_DRIVER A, DRIVER B WHERE DNM=B.D_NAME
AND B.D_NO=A.D_NO;

 DUTY_DATE DATE;

BEGIN

 RAISE NOTICE ''DATE OF DUTY'';

 OPEN CU1;

  LOOP

  FETCH CU1 INTO DUTY_DATE;

  EXIT WHEN NOT FOUND;

    RAISE NOTICE'' %'',DUTY_DATE;

  END LOOP;

 CLOSE CU1;

RETURN 1;

END;

'LANGUAGE 'plpgsql';
```

Slip6

```
-- 1. To display driver names working in both shifts.

CREATE OR REPLACE VIEW V1 AS

SELECT D_NAME FROM DRIVER A, BUS_DRIVER B WHERE B.SHIFT=1 AND A.D_NO=B.D_NO INTERSECT

SELECT D_NAME FROM DRIVER A, BUS_DRIVER B WHERE B.SHIFT=2 AND A.D_NO=B.D_NO;


-- 2. To display route details on which Bus_no 101 is running.

CREATE OR REPLACE VIEW V2 AS

SELECT A.* FROM ROUTE A, BUS B WHERE B.BUS_NO='101' AND A.ROUTE_NO=B.ROUTE_NO;

/* 1. Write a trigger after deleting the bus record which has capacity < 20. Display the

appropriate message.*/

CREATE OR REPLACE FUNCTION DEL_REC() RETURNS TRIGGER AS'

BEGIN

 IF(OLD.CAPACITY < 25) THEN

  RAISE NOTICE''BUS RECORD IS BEING DELETED....'';

 ELSE

   RETURN OLD;

 END IF;

RETURN NEW;

END;

'LANGUAGE 'plpgsql';


CREATE TRIGGER REC

AFTER DELETE ON BUS

FOR EACH ROW
```

```
EXECUTE PROCEDURE DEL_REC();


/*2. Write a cursor to display details of buses running on route_no = 1.*/

CREATE OR REPLACE FUNCTION BUS_DET() RETURNS INT AS'

DECLARE

 CU1 CURSOR FOR SELECT * FROM BUS WHERE ROUTE_NO=1;

 BNO INT;

 CAP INT;

 DEPO_NM VARCHAR;

 RT_NO INT;

BEGIN

 RAISE NOTICE''BUS NO || CAPACITY || DEPOT NAME || ROUTE NO'';

 OPEN CU1;

  LOOP

   FETCH CU1 INTO BNO,CAP,DEPO_NM,RT_NO;

   EXIT WHEN NOT FOUND;

   RAISE NOTICE'' %       %        %       %'',BNO,CAP,DEPO_NM,RT_NO;

  END LOOP;

CLOSE CU1;

RETURN 1;

END;

'LANGUAGE 'plpgsql';
```

Slip7

```
CREATE TABLE TRAIN(T_NO INT PRIMARY KEY, T_NAME VARCHAR(20), DEPT_TIME TIME, ARRIVAL_TIME
TIME, SRC_STN VARCHAR(20), DEST_STN VARCHAR(20), RES_BOGIES INT, BOGIE_CAPACITY INT);

CREATE TABLE PASSENGER(P_ID INT PRIMARY KEY, P_NAME VARCHAR(10), ADDR VARCHAR(10), AGE INT,
```

```sql
GENDER CHAR(10));

CREATE TABLE TRAIN_PASS_TICKET(T_NO INT REFERENCES TRAIN(T_NO), P_ID INT REFERENCES
PASSENGER(P_ID), TICKET_NO INT, BOGIE_NO INT, NO_OF_BERTHS INT, T_DATE DATE, TICKET_AMT
DECIMAL(5, 2), TICKET_STATUS CHAR(2) CHECK (TICKET_STATUS IN('W','C')));


-- 1. To display names of ‘Shatabdi Express’ passengers whose ticket status is waiting on 02-03-2022

CREATE OR REPLACE VIEW V1 AS

SELECT A.P_NAME FROM PASSENGER A, TRAIN B, TRAIN_PASS_TICKET C WHERE C.TICKET_STATUS='W'
AND C.T_DATE='03/02/2022' AND B.T_NAME='SHATABDI EXPRESS' AND A.P_ID=C.P_ID AND
B.T_NO=C.T_NO;


-- 2. To display first three bookings for 'Rajdhani Express' on 04-05-2021.

CREATE OR REPLACE VIEW V2 AS

SELECT P_NAME FROM PASSENGER A, TRAIN B, TRAIN_PASS_TICKET C WHERE B.T_NAME='RAJDHANI
EXPRESS' AND C.T_DATE='11/06/2022' AND A.P_ID=C.P_ID AND B.T_NO=C.T_NO LIMIT 3;

-- 1. Write a trigger to restrict the bogie capacity of any train to 25.

CREATE OR REPLACE FUNCTION RES_TR() RETURNS TRIGGER AS'

BEGIN

 IF(NEW.BOGIE_CAPACITY > 25) THEN

   RAISE WARNING''BOGIE CAPACITY MUST BE 25 SEATS ONLY'';

 ELSE

   RETURN NEW;

 END IF;

RETURN OLD;

END;

'LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER RES_BOGIE

BEFORE INSERT OR UPDATE ON TRAIN

FOR EACH ROW

EXECUTE PROCEDURE RES_TR();
```

-- 2. Write a function using cursor to display train wise confirmed bookings on 19-04-2022.

```
CREATE OR REPLACE FUNCTION CONFIRMED_BOOKINGS() RETURNS INT AS'

DECLARE

CUR1 CURSOR FOR SELECT A.TICKET_NO, A.TICKET_STATUS, B.P_NAME FROM TRAIN_PASS_TICKET A,
PASSENGER B WHERE B.P_ID=A.P_ID AND A.TICKET_STATUS=''C'' AND A.T_DATE=''11/06/2022'';

TNO INT;

TS CHAR;

PNM VARCHAR;

BEGIN

RAISE INFO ''TICKET NO || TICKET STATUS || PASSENGER NAME'';

 OPEN CUR1;

  LOOP

   FETCH CUR1 INTO TNO,TS,PNM;

   EXIT WHEN NOT FOUND;

   RAISE INFO''  %           %          %'',TNO,TS,PNM;

  END LOOP;

CLOSE CUR1;

RETURN 1;

END;
```

'LANGUAGE 'plpgsql';

Slip8

-- 1. To display names of ‘Shatabdi Express’ passengers whose ticket status is confirmed on 02-03-2022.

CREATE OR REPLACE VIEW V1 AS

SELECT A.P_NAME FROM PASSENGER A, TRAIN B, TRAIN_PASS_TICKET C WHERE C.TICKET_STATUS='C' AND C.T_DATE='03/02/2022' AND B.T_NAME='SHATABDI EXPRESS' AND A.P_ID=C.P_ID AND B.T_NO=C.T_NO;

-- 2. To display count of confirmed bookings of ‘Rajdhani Express’ on 01-01-2022.

CREATE OR REPLACE VIEW V2 AS

SELECT COUNT(TICKET_NO) AS CONFIRMED_BOOKINGS FROM TRAIN A, TRAIN_PASS_TICKET B WHERE A.T_NO=B.T_NO AND B.TICKET_STATUS='C' AND A.T_NAME='RAJDHANI EXPRESS' AND T_DATE='11/06/2022';

/* 1. Write a trigger after inserting the age in passenger table to display the message “Age above5

years will be charged the full fare” if the passenger's age is above 5 years.*/

```
CREATE OR REPLACE FUNCTION CHK_AGE() RETURNS TRIGGER AS'

BEGIN

 IF(NEW.AGE > 5) THEN

  RAISE NOTICE '' AGE ABOVE 5 YEARS WILL BE CHARGED THE FULL FARE'';

 ELSE

  RETURN NEW;

 END IF;

RETURN OLD;

END;

'LANGUAGE 'plpgsql';


CREATE TRIGGER CHK
```

```
AFTER INSERT ON PASSENGER

FOR EACH ROW

EXECUTE PROCEDURE CHK_AGE();


/*2. Write a stored function to display train wise bookings on 02-05-2020 whose ticket status is

waiting.*/

CREATE OR REPLACE FUNCTION BOOKING_DETAILS() RETURNS INT AS'

DECLARE

 TTX RECORD;

BEGIN

 RAISE INFO '' TICKET NO || TICKET AMOUNT || TRAIN NAME'';

 RAISE INFO ''--------------------------------------------------'';

 FOR TTX IN SELECT A.TICKET_NO, A.TICKET_AMT, B.T_NAME FROM TRAIN_PASS_TICKET A, TRAIN B
WHERE A.T_NO=B.T_NO AND T_DATE=''03/02/2022'' AND TICKET_STATUS=''W'' GROUP BY A.TICKET_NO,
A.TICKET_AMT,B.T_NAME

   LOOP

    RAISE INFO''   %          %        %'',TTX.TICKET_NO, TTX.TICKET_AMT, TTX.T_NAME;

    EXIT WHEN NOT FOUND;

   END LOOP;

  RETURN 1;

 END;

'LANGUAGE 'plpgsql';
```

Slip9

```
CREATE TABLE PROJECT(P_NO INT PRIMARY KEY, P_NAME VARCHAR(20) NOT NULL, P_TYPE
VARCHAR(15), DURATION INT);

CREATE TABLE EMPLOYEE(E_NO INT PRIMARY KEY, E_NAME VARCHAR(10), QUALIFICATION CHAR(10),
JOIN_DATE DATE);
```

```
CREATE TABLE PROJ_EMP(P_NO INT REFERENCES PROJECT(P_NO), E_NO INT REFERENCES
EMPLOYEE(E_NO), START_DATE DATE, NO_OF_HRS_WORKED INT);

-- 1. To display the project name, project type, and project start date, sorted by project start date.

CREATE OR REPLACE VIEW V1 AS

SELECT A.*, B.START_DATE FROM PROJECT A, PROJ_EMP B WHERE A.P_NO=B.P_NO ORDER BY
B.START_DATE;


-- 2. To display details of employees working on ‘Robotics’ project

CREATE OR REPLACE VIEW V2 AS

SELECT A.* FROM EMPLOYEE A, PROJECT B, PROJ_EMP C WHERE A.E_NO=C.E_NO AND B.P_NO=C.P_NO
AND P_NAME='ROBOTICS';

/*1. Write a trigger before inserting the duration into the project table and make sure that the

duration is always greater than zero. Display appropriate message.*/

CREATE OR REPLACE FUNCTION CHK_DUR() RETURNS TRIGGER AS'

BEGIN

 IF(NEW.DURATION < 0 OR NEW.DURATION = 0) THEN

   RAISE EXCEPTION ''DURATION  OF PROJECT MUST BE GREATER THAN 0'';

 ELSE

  RETURN NEW;

 END IF;

RETURN OLD;

END;

'LANGUAGE 'plpgsql';


CREATE TRIGGER CHECK_DURATION

BEFORE INSERT ON PROJECT
```

```
FOR EACH ROW

EXECUTE PROCEDURE CHK_DUR();


/*2. Write function using cursor to accept project name as an input parameter and display

names of employees working on that project.*/

CREATE OR REPLACE FUNCTION EMP_DET(PNM TEXT) RETURNS INT AS'

DECLARE

 JUS CURSOR FOR SELECT E_NAME, QUALIFICATION FROM EMPLOYEE A, PROJECT B, PROJ_EMP C WHERE
A.E_NO=C.E_NO AND B.P_NO=C.P_NO AND PNM=B.P_NAME;

 ENM TEXT;

 QUALI TEXT;

BEGIN

 RAISE INFO''EMPLOYEE NAME || QUALIFICATION'';

  RAISE INFO''-------------------------------'';

 OPEN JUS;

  LOOP

   FETCH JUS INTO ENM, QUALI;

   EXIT WHEN NOT FOUND;

    RAISE INFO ''   %          %'',ENM,QUALI;

 END LOOP;

RETURN 1;

END;

'LANGUAGE 'plpgsql';
```

Slip10

-- 1. To display employee details and it should be sorted by employee’s joining date.

```
CREATE OR REPLACE VIEW V1 AS

SELECT * FROM EMPLOYEE ORDER BY JOIN_DATE;


-- 2. To display employee and project details where employees worked less than 100 hours.

CREATE OR REPLACE VIEW V2 AS

SELECT A., B. FROM EMPLOYEE A, PROJECT B, PROJ_EMP C WHERE A.E_NO=C.E_NO AND
B.P_NO=C.P_NO AND C.NO_OF_HRS_WORKED < 10;

/*1. Write a trigger before inserting joining date into employee table, check joining date should be

always less than current date. Display appropriate message.*/

CREATE OR REPLACE FUNCTION CHK_DT() RETURNS TRIGGER AS'

BEGIN

 IF(NEW.JOIN_DATE > CURRENT_DATE) THEN

   RAISE EXCEPTION ''JOINING DATE CAN NEVER BE FUTURE DATE'';

 ELSE

   RETURN NEW;

 END IF;

RETURN OLD;

END;

'LANGUAGE 'plpgsql';


CREATE TRIGGER CK_DT

BEFORE INSERT ON EMPLOYEE

FOR EACH ROW

EXECUTE PROCEDURE CHK_DT();
```

```
/*2. Write a stored function to accept project name as an input parameter and returns the number

of employees working on that project. Raise an exception for an invalid project name.*/

CREATE OR REPLACE FUNCTION NO_OF_EMPLOYEES(PNM TEXT) RETURNS INT AS'

DECLARE

 CNT INT;

BEGIN

  SELECT INTO CNT COUNT(A.E_NO) FROM EMPLOYEE A, PROJECT B, PROJ_EMP C WHERE
A.E_NO=C.E_NO AND B.P_NO=C.P_NO AND PNM=B.P_NAME;

   IF (CNT=0) THEN

    RAISE EXCEPTION ''INVALID PROJECT NAME'';

  END IF;

  RETURN CNT;

END;

'LANGUAGE 'plpgsql';
```

Slip11

```
CREATE TABLE PROJECT(P_NO INT PRIMARY KEY, P_NAME VARCHAR(20) NOT NULL, P_TYPE
VARCHAR(15), DURATION INT);

CREATE TABLE EMPLOYEE(E_NO INT PRIMARY KEY, E_NAME VARCHAR(10), QUALIFICATION CHAR(10),
JOIN_DATE DATE);

CREATE TABLE PROJ_EMP(P_NO INT REFERENCES PROJECT(P_NO), E_NO INT REFERENCES
EMPLOYEE(E_NO), START_DATE DATE, NO_OF_HRS_WORKED INT);


-- 1. To display employee details working on ‘ERP’ Project.

CREATE OR REPLACE VIEW V1 AS

SELECT A.* FROM EMPLOYEE A, PROJECT B, PROJ_EMP C WHERE A.E_NO=C.E_NO AND B.P_NO=C.P_NO
AND B.P_NAME='ERP';
```

```
-- 2. To display employee and project details where employees worked more than 100 hours.

CREATE OR REPLACE VIEW V2 AS

SELECT A., B. FROM EMPLOYEE A, PROJECT B, PROJ_EMP C WHERE A.E_NO=C.E_NO AND
B.P_NO=C.P_NO AND C.NO_OF_HRS_WORKED > 10;

/*1. Write a trigger after deleting Project record from Project table. Display the message "Project record
is being deleted".*/

CREATE OR REPLACE FUNCTION DEL_REC() RETURNS TRIGGER AS'

BEGIN

 RAISE NOTICE ''PROJECT RECORD IS BEING DELETED'';

RETURN NULL;

END;

'LANGUAGE 'plpgsql';


CREATE TRIGGER DEL_RC

AFTER DELETE ON PROJECT

FOR EACH ROW

EXECUTE PROCEDURE DEL_REC();


/*2. Write a function to find the number of employees whose date of joining is before 03-10-2022.*/

CREATE OR REPLACE FUNCTION NO_OF_EMP() RETURNS INT AS'

DECLARE

 CNT INT;

BEGIN

  SELECT COUNT(E_NO) INTO CNT FROM EMPLOYEE WHERE JOIN_DATE<(''10/03/2022'');

RETURN CNT;
```

```
END;

'LANGUAGE 'plpgsql';
```

slip12

```
CREATE TABLE STUDENT(S_NO INT PRIMARY KEY, S_NAME CHAR(10), S_CLASS CHAR(7), ADDR VARCHAR(10));


CREATE TABLE TEACHER(T_NO INT PRIMARY KEY, T_NAME CHAR(10), QUALIFICATION CHAR(7), EXPERIENCE_YEARS INT);


CREATE TABLE STUD_TEACH(S_NO INT REFERENCES STUDENT(S_NO), T_NO INT REFERENCES TEACHER(T_NO), SUBJECT CHAR(10));
```

1. To display student names who are taught by most experienced teacher.

```
CREATE OR REPLACE VIEW V1 AS

SELECT A.* FROM STUDENT A, TEACHER B, STUD_TEACH C WHERE B.EXPERIENCE_YEARS IN (SELECT MAX(EXPERIENCE_YEARS) FROM TEACHER) AND A.S_NO=C.S_NO AND B.T_NO=C.T_NO;


-- 2. To display subjects taught by each teacher.

CREATE OR REPLACE VIEW V2 AS

SELECT DISTINCT SUBJECT, T_NAME FROM STUD_TEACH A, TEACHER B WHERE A.T_NO=B.T_NO ORDER BY A.SUBJECT;

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

/* 1. Write a trigger before inserting the student record. If the sno is less than or equal to zero, then

display the message 'Invalid student number'.*/

CREATE OR REPLACE FUNCTION VLD_NO() RETURNS TRIGGER AS'

BEGIN

 IF(NEW.S_NO < 0 OR NEW.S_NO = 0) THEN
```

```
  RAISE EXCEPTION ''INVALID STUDENT NUMBER !!'';
 ELSE
  RETURN NEW;
 END IF;
RETURN OLD;
END;
'LANGUAGE 'plpgsql';


CREATE TRIGGER CHK_SNO
BEFORE INSERT ON STUDENT
FOR EACH ROW
EXECUTE PROCEDURE VLD_NO();


/*2. Write a stored function to count the number of students studying a subject named ‘ ’ (Accept
the subject's name as an input parameter). Display error message for invalid subject name.*/
CREATE OR REPLACE FUNCTION STD_CNT(SNM TEXT) RETURNS INT AS'
DECLARE
 JAZ INT;
BEGIN
 SELECT INTO JAZ COUNT(A.S_NO) FROM STUDENT A, STUD_TEACH B WHERE A.S_NO=B.S_NO AND
SNM=B.SUBJECT;
  IF (JAZ=0) THEN
   RAISE EXCEPTION ''INVALID SUBJECT NAME'';
  END IF;
RETURN JAZ;
```

```
END;
'LANGUAGE 'plpgsql';
```

SLIP 13

1. To display teacher details having qualification as ‘Ph.D.’.

```
CREATE OR REPLACE VIEW V1 AS
SELECT * FROM TEACHER WHERE QUALIFICATION='PHD';
```

2. To display student details living in ‘Pune’.

```
CREATE OR REPLACE VIEW V2 AS
SELECT * FROM STUDENT WHERE ADDR='PUNE';
```

```
/*1. Write a trigger before inserting experience into a teacher table; experience should be
minimum 5 years. Display appropriate message. */
CREATE OR REPLACE FUNCTION CHK_EXP() RETURNS TRIGGER AS'
BEGIN
 IF(NEW.EXPERIENCE_YEARS < 5) THEN
  RAISE EXCEPTION ''TEACHER EXPERIENCE MUST BE AT LEAST 5 YEARS !'';
 ELSE
  RETURN NEW;
 END IF;
RETURN OLD;
END;
'LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER CK_EXP

BEFORE INSERT ON TEACHER

FOR EACH ROW

EXECUTE PROCEDURE CHK_EXP();


/*2. Write a cursor to list the details of the teachers who are teaching to a student named ‘ __’.

(Accept student name as an input parameter). */

CREATE OR REPLACE FUNCTION TEACHER_DETAILS(SNM TEXT) RETURNS INT AS'

DECLARE

 THO CURSOR FOR SELECT DISTINCT A.T_NAME, A.QUALIFICATION, A.EXPERIENCE_YEARS FROM
TEACHER A, STUDENT B, STUD_TEACH C WHERE A.T_NO=C.T_NO AND B.S_NO=C.S_NO AND
SNM=B.S_NAME;

 TNM VARCHAR;

 QUALI VARCHAR;

 EXP INT;

BEGIN

 RAISE INFO ''TEACHER NAME || QUALIFICATION || EXPERIENCE'';

 RAISE INFO '' ------------------------------------------------------'';

 OPEN THO;

 LOOP

 FETCH THO INTO TNM,QUALI,EXP;

  EXIT WHEN NOT FOUND;

 RAISE INFO ''  %           %           %'',TNM,QUALI,EXP;

 END LOOP;

 CLOSE THO;

RETURN NULL;
```

```
END;

'LANGUAGE 'plpgsql';
```

slip15

```
CREATE TABLE STUDENTS(ROLL_NO INT PRIMARY KEY, S_NAME VARCHAR (15) NOT NULL, CITY
VARCHAR(15), CLASS VARCHAR(7));

CREATE TABLE SUBJECTS(S_CODE VARCHAR(10) PRIMARY KEY, SUB_NAME VARCHAR(15));

CREATE TABLE STUD_SUB(ROLL_NO INT REFERENCES STUDENTS(ROLL_NO), S_CODE VARCHAR(10)
REFERENCES SUBJECTS(S_CODE), MARKS_SCORED INT);


-- 1. To display names of students class ‘FYBCA’.

CREATE OR REPLACE VIEW V1 AS

SELECT * FROM STUDENTS WHERE CLASS='FYBCA';


-- 2. To display students name, subject and marks who has scored more than 90 marks.

CREATE OR REPLACE VIEW V2 AS

SELECT S_NAME, SUB_NAME, MARKS_SCORED FROM STUDENTS A, SUBJECTS B, STUD_SUB C WHERE
A.ROLL_NO=C.ROLL_NO AND B.S_CODE=C.S_CODE AND C.MARKS_SCORED>90;


/*1. Write a trigger before inserting Rollno into Student table. Display error message if entered

Rollno less than equal to zero.*/

CREATE OR REPLACE FUNCTION CHK_ROLL() RETURNS TRIGGER AS'

BEGIN

 IF(NEW.ROLL_NO < 0 OR NEW.ROLL_NO = 0) THEN

  RAISE EXCEPTION ''ROLL NUMBER CANNOT BE LESS THAN OR EQUAL TO 0 !'';

 ELSE
```

```
  RETURN NEW;

 END IF;

RETURN OLD;

END;

'LANGUAGE 'plpgsql';


CREATE TRIGGER CK_ROLL

BEFORE INSERT ON STUDENTS

FOR EACH ROW

EXECUTE PROCEDURE CHK_ROLL();
```

/2. Write a function using cursor, to calculate total marks of each student and display it./

```
CREATE OR REPLACE FUNCTION TOTAL_MARKS() RETURNS INT AS'

DECLARE

 LLB CURSOR FOR SELECT S_NAME, SUM(MARKS_SCORED) FROM STUDENTS A, STUD_SUB B WHERE
A.ROLL_NO=B.ROLL_NO GROUP BY A.S_NAME;

 SNM VARCHAR;

 TOTAL INT;

BEGIN

 RAISE INFO ''STUDENT NAME || TOTAL MARKS'';

 RAISE INFO ''-----------------------------'';

  OPEN LLB;

  LOOP

   FETCH LLB INTO SNM, TOTAL;
   EXIT WHEN NOT FOUND;
   RAISE INFO ''   %           %'',SNM,TOTAL;
  END LOOP;
 CLOSE LLB;
RETURN 1;
END;
'LANGUAGE 'plpgsql';
```

Slip14

1. To display details of teachers having experience > 5 years.

```
CREATE OR REPLACE VIEW V1 AS
SELECT * FROM TEACHER WHERE EXPERIENCE_YEARS > 5;
```

2. To display details of teachers whose name start with the letter ‘S’.

```
CREATE OR REPLACE VIEW V2 AS
SELECT * FROM TEACHER WHERE T_NAME LIKE 'S%';
```

```
/* 1. Write a trigger before update a student’s class from student table. Display appropriate
message.*/
CREATE OR REPLACE FUNCTION UPD_CLS() RETURNS TRIGGER AS'
BEGIN
 IF (NEW.S_CLASS != OLD.S_CLASS) THEN
  RAISE NOTICE ''UPDATING STUDENT CLASS'';
 ELSE

   RETURN NEW;

 END IF;

RETURN OLD;

END;

'LANGUAGE 'plpgsql';


CREATE TRIGGER UPD_CLASS

BEFORE UPDATE ON STUDENT

FOR EACH ROW

EXECUTE PROCEDURE UPD_CLS();



/*2. Write a function to count the number of teachers who are teaching to a student named'____

'. (Accept student name as an input parameter).*/

CREATE OR REPLACE FUNCTION TEACHER_COUNT(SNM TEXT) RETURNS INT AS'

DECLARE

 OMG INT;

BEGIN

 SELECT INTO OMG COUNT(A.T_NO) FROM TEACHER A, STUDENT B, STUD_TEACH C WHERE
A.T_NO=C.T_NO AND B.S_NO=C.S_NO AND SNM=B.S_NAME;

 IF(OMG=0) THEN

   RAISE NOTICE ''INVALID STUDENT NAME !!'';

 END IF;

RETURN OMG;
```

```
END;

'LANGUAGE 'plpgsql';
```

Slip16

-- 1. To display the students name who scored more than 80 marks in ‘DBMS’ Subject.

```
CREATE OR REPLACE VIEW V1 AS

SELECT A.* FROM STUDENTS A, SUBJECTS B, STUD_SUB C WHERE A.ROLL_NO=C.ROLL_NO AND B.S_CODE=C.S_CODE AND C.MARKS_SCORED>60 AND B.SUB_NAME='DBMS';
```

-- 2. To display student details of class ‘TYBCA’.

```
CREATE OR REPLACE VIEW V2 AS

SELECT * FROM STUDENTS WHERE CLASS='TYBCA';
```

/1. Write a trigger after deleting a student record from the student table. Display the message “student record is being deleted”./

```
CREATE OR REPLACE FUNCTION DEL_REC() RETURNS TRIGGER AS'

BEGIN

 RAISE NOTICE ''STUDENT RECORD IS BEING DELETED  '';

RETURN NULL;

END;

'LANGUAGE 'plpgsql';


CREATE TRIGGER REC_DEL

AFTER DELETE ON STUDENTS

FOR EACH ROW

EXECUTE PROCEDURE DEL_REC();
```

/2. Write a stored function to accept student name as an input parameter and display their subject information./

```
CREATE OR REPLACE FUNCTION SUBJECT_INFO(SNM TEXT) RETURNS INT AS'

DECLARE

 REV RECORD;

BEGIN

 RAISE INFO '' SUBJECT CODE || SUBJECT NAME'';

 RAISE INFO ''--------------------------------------'';

 FOR REV IN SELECT * FROM SUBJECTS A, STUDENTS B, STUD_SUB C WHERE SNM=B.S_NAME AND
B.ROLL_NO=C.ROLL_NO AND A.S_CODE=C.S_CODE

 LOOP

  RAISE INFO ''    %           %'',REV.S_CODE,REV.SUB_NAME;

 END LOOP;

RETURN NULL;

END;

'LANGUAGE 'plpgsql';
```

Slip17

--1. To display details of students whose name starts with the letter ‘A’.

```
CREATE OR REPLACE VIEW V1 AS

SELECT * FROM STUDENTS WHERE S_NAME LIKE 'S%';
```

-- 2. To display details of students who has scored less than 40 marks.

```
CREATE OR REPLACE VIEW V2 AS

SELECT DISTINCT A.* FROM STUDENTS A, STUD_SUB B WHERE B.MARKS_SCORED<40 AND
A.ROLL_NO=B.ROLL_NO;


/*1. Write a trigger to ensure that the marks entered for a student with respect to a subject is

never < 0 and greater than 100.*/

CREATE OR REPLACE FUNCTION VLD_MKS() RETURNS TRIGGER AS'

BEGIN

 IF (NEW.MARKS_SCORED < 0 OR NEW.MARKS_SCORED > 100) THEN

  RAISE WARNING ''MARKS SHOULD BE GREATER THAN 0 AND LESS THAN 100 !'';

 ELSE

  RETURN NEW;

 END IF;

RETURN OLD;

END;

'LANGUAGE 'plpgsql';


CREATE TRIGGER CHK_MKS

BEFORE INSERT ON STUD_SUB

FOR EACH ROW

EXECUTE PROCEDURE VLD_MKS();


/2. Write a stored function to accept city as an input parameter and display student details./

CREATE OR REPLACE FUNCTION STD_DET(CTY TEXT) RETURNS INT AS'

DECLARE
```

```
 VAR RECORD;
BEGIN
 RAISE NOTICE ''ROLL NO || STUDENT NAME ||   CITY   || CLASS  '';
 RAISE NOTICE ''----------------------------------------------'';
 FOR VAR IN SELECT * FROM STUDENTS WHERE CITY=CTY
 LOOP
  RAISE NOTICE ''  %          %       %        % '',VAR.ROLL_NO,VAR.S_NAME,VAR.CITY,VAR.CLASS;
 END LOOP;
RETURN NULL;
END;
'LANGUAGE 'plpgsql';
```

Slip 18

```
CREATE TABLE MOVIE(M_NO INT PRIMARY KEY, M_NAME VARCHAR(20), RELEASE_YEAR INT, BUDGET
MONEY);

CREATE TABLE ACTOR(A_NO INT PRIMARY KEY, A_NAME VARCHAR(20), ROLE VARCHAR(10), CHARGES
MONEY, ADDR VARCHAR(20));

CREATE TABLE PRODUCER(P_NO INT PRIMARY KEY, P_NAME VARCHAR(20),P_ADDR VARCHAR(20));

CREATE TABLE MOV_ACT_PRO(M_NO INT REFERENCES MOVIE(M_NO), A_NO INT REFERENCES
ACTOR(A_NO), P_NO INT REFERENCES PRODUCER(P_NO));
```

```
-- 1. To display actor names who lives in Mumbai.

CREATE OR REPLACE VIEW V1 AS

SELECT A_NAME FROM ACTOR WHERE ADDR='MUMBAI';


-- 2. To display actors information in each movie.

CREATE OR REPLACE VIEW V2 AS

SELECT DISTINCT  A.A_NAME, A.ROLE, B.M_NAME FROM ACTOR A, MOVIE B, MOV_ACT_PRO C WHERE
A.A_NO=C.A_NO AND B.M_NO=C.M_NO ORDER BY B.M_NAME;




/*1. Write a trigger before inserting budget into a movie table. Budget should be minimum 60

lakh. Display appropriate message.*/

CREATE OR REPLACE FUNCTION CHK_BUD() RETURNS TRIGGER AS'

BEGIN

 IF(NEW.BUDGET < ''6000000'') THEN

  RAISE EXCEPTION ''MOVIE BUDGET MUST BE AT LEAST 60 LAKHS !'';

 ELSE

  RETURN NEW;

 END IF;

RETURN OLD;

END;

LANGUAGE 'plpgsql';


CREATE TRIGGER CK_BUDGET

BEFORE INSERT ON MOVIE
```

```
FOR EACH ROW

EXECUTE PROCEDURE CHK_BUD();
```

```
/*2. Write a stored function to accept producer name as an input parameter and display count of

movies that producer has produced.*/

CREATE OR REPLACE FUNCTION MOVIE_COUNT(PNM TEXT) RETURNS INT AS'

DECLARE

 GDT INT;

BEGIN

 SELECT INTO GDT COUNT(A.M_NO) FROM MOV_ACT_PRO A, PRODUCER B WHERE A.P_NO=B.P_NO
AND PNM=B.P_NAME;

  IF(GDT=0) THEN

   RAISE NOTICE '' INVALID PRODUCER NAME !'';

  END IF;

 RETURN GDT;

END;

LANGUAGE 'plpgsql';
```

Slip19

```
-- 1. To display actor details acted in movie Sholey.

CREATE OR REPLACE VIEW V1 AS

SELECT A.* FROM ACTOR A, MOVIE B, MOV_ACT_PRO C WHERE A.A_NO=C.A_NO AND
B.M_NO=C.M_NO AND B.M_NAME='KGF 2';


-- 2. To display producer name who have produced more than two movies.
```

```
CREATE OR REPLACE VIEW V2 AS

SELECT DISTINCT A.P_NAME FROM PRODUCER A, MOV_ACT_PRO B WHERE A.P_NO=B.P_NO GROUP BY
A.P_NAME HAVING COUNT(B.M_NO)>2;

/*1. Write a trigger before inserting charges into relationship table. Charges should not be more

than 30 lakh. Display appropriate message. */

CREATE OR REPLACE FUNCTION VALID_CHARGES() RETURNS TRIGGER AS'

BEGIN

 IF(NEW.CHARGES > ''3000000'') THEN

  RAISE WARNING '' ACTOR CHARGES MUST BE LESS THAN 30 LAKHS !'';

 ELSE

  RETURN NEW;

 END IF;

RETURN OLD;

END;

LANGUAGE 'plpgsql';


CREATE TRIGGER CK_CHARGES

BEFORE INSERT ON ACTOR

FOR EACH ROW

EXECUTE PROCEDURE VALID_CHARGES();




/*2. Write a stored function to accept actor name as an input parameter and display names of

movies in which that actor has acted. Display error message for an invalid actor name.*/

CREATE OR REPLACE FUNCTION MOVIE_NAMES(ANM TEXT) RETURNS INT AS'
```

```
DECLARE

 CMD RECORD;

BEGIN

 RAISE INFO ''    MOVIE NAME     '';

 RAISE INFO ''-----------------'';

 FOR CMD IN SELECT DISTINCT M_NAME FROM MOVIE A, ACTOR B, MOV_ACT_PRO C WHERE
A.M_NO=C.M_NO AND B.A_NO=C.A_NO AND ANM=B.A_NAME

  LOOP

   RAISE INFO ''     %    '',CMD.M_NAME;

 END LOOP;

 IF NOT FOUND THEN

   RAISE EXCEPTION '' % INVALID ACTOR NAME !'',ANM;

  END IF;

RETURN NULL;

END;

LANGUAGE 'plpgsql';
```

Slip20

-- 1. To display movie names produced by Mr. Subhash Ghai.

```
CREATE OR REPLACE VIEW V1 AS

SELECT DISTINCT M_NAME FROM MOVIE A, PRODUCER B, MOV_ACT_PRO C WHERE A.M_NO=C.M_NO
AND B.P_NO=C.P_NO AND B.P_NAME='VIJAY';
```

-- 2. To display actor names who do not live in Mumbai or Pune city.

```
CREATE OR REPLACE VIEW V2 AS

SELECT A_NAME FROM ACTOR WHERE ADDR NOT IN('MUMBAI','PUNE');
```

```
/*1. Write a trigger before inserting record into movie table; check release_year should not be

greater than current year. Display appropriate message.*/

CREATE OR REPLACE FUNCTION CHK_YR() RETURNS TRIGGER AS'

BEGIN

 IF(NEW.RELEASE_YEAR > DATE_PART(''YEAR'',(SELECT CURRENT_TIMESTAMP))) THEN

  RAISE WARNING '' REALEASE YEAR CANNOT BE FUTURE YEAR !!'';

 ELSE

  RETURN NEW;

 END IF;

RETURN OLD;

END;

LANGUAGE 'plpgsql';


CREATE TRIGGER CK_YR

BEFORE INSERT ON MOVIE

FOR EACH ROW

EXECUTE PROCEDURE CHK_YR();



/2. Write a cursor using function to list movie-wise charges of Amitabh Bachchan./

CREATE OR REPLACE FUNCTION ACT_CHARGES() RETURNS INT AS'

DECLARE

 BUT CURSOR FOR SELECT DISTINCT A.M_NAME, B.CHARGES FROM MOVIE A, ACTOR B, MOV_ACT_PRO
C WHERE A.M_NO=C.M_NO AND B.A_NO=C.A_NO AND B.A_NAME=''RANBIR KAPOOR'';
```

```
 MOV_NM VARCHAR;
 CG MONEY;
BEGIN
 RAISE INFO ''    MOVIE NAME     ||      ACTOR CHARGES    '';
 RAISE INFO ''--------------------------------------------'';
 OPEN BUT;
 LOOP
  FETCH BUT INTO MOV_NM,CG;
  EXIT WHEN NOT FOUND;
   RAISE INFO ''  %          %'',MOV_NM,CG;
  END LOOP;
 RETURN NULL;
END;
LANGUAGE 'plpgsql';
```