```
-- users -
-- user stu_1, password stu_1;
-- user dean_acads, password dean1234;
-- user fac_1, password fac_1;

-- psql -U postgres -h localhost
-- create database - create database Student_Portal
-- create user - CREATE USER dean_acads WITH ENCRYPTED PASSWORD 'p_dean';
-- create user - CREATE USER stu_1 WITH ENCRYPTED PASSWORD 'stu_1';
-- create  faculty - CREATE USER fac_1 WITH ENCRYPTED PASSWORD 'fac_1';
-- list users - \du
-- choose your database - \c Student_Portal
-- list all tables of the database - \dt
-- grant privileges - GRANT ALL ON course_catalogue TO dean_acads;
-- to check the privileges on a table - \dp course_catalogue
-- login as dean_acads user - psql -h localhost -d Student_Portal -U dean_acads
-- login as stu_1 - psql -h localhost -d Student_Portal -U stu_1
-- list table entries - select * from table_name;
-- grant view(select) for stu_1 on course_catalogue - GRANT SELECT ON course_catalogue
TO stu_1;
-- GRANT VIEW(SELECT) For FAC_1 ON COURSE_CATALOGUE - GRANT SELECT ON
course_catalogue TO fac_1;
-- Faulty can rw - GRANT ALL ON course_offering TO fac_1;
-- GRANT SELECT ON course_offering TO stu_1;


-- GRANT SELECT ON course_offering TO dean_acads;
-- GRANT SELECT, UPDATE ON tickets_table TO dean_acads;
-- GRANT SELECT, UPDATE ON dean_tickets_table TO dean_acads;


--schema of course_catalogue
CREATE TABLE course_catalogue(
        course_id       VARCHAR(6) NOT NULL PRIMARY KEY,
        L                       INTEGER  NOT NULL,
        T                       INTEGER  NOT NULL,
        P                       INTEGER  NOT NULL,
        S                       INTEGER  NOT NULL,
        C                       INTEGER  NOT NULL,
        prereqs                 VARCHAR[]
);
```

```
INSERT INTO course_catalogue(course_id, L, T, P, S, C, prereqs) VALUES ('ge111', 3, 1, 0, 3,
3, NULL);
INSERT INTO course_catalogue(course_id, L, T, P, S, C, prereqs) VALUES ('cs301', 3, 1, 0, 3,
4, ARRAY['cs202', 'cs201']);
INSERT INTO course_catalogue(course_id, L, T, P, S, C, prereqs) VALUES ('hs202', 3, 1, 0, 3,
3, NULL);
INSERT INTO course_catalogue(course_id, L, T, P, S, C, prereqs) VALUES ('ge103', 3, 1, 0, 3,
3, ARRAY['cs301', 'cs201']);
INSERT INTO course_catalogue(course_id, L, T, P, S, C, prereqs) VALUES ('cs302', 3, 1, 0, 3,
4, ARRAY['ge102', 'cs201']);


--procedure for dean_acads to insert in course catalogue
CREATE PROCEDURE insert_in_catalogue(course_id      VARCHAR(6),
        L               INTEGER,
        T               INTEGER,
        P               INTEGER,
        S               INTEGER,
        C               INTEGER,
        prereqs                 VARCHAR[])
LANGUAGE plpgsql
AS $$
BEGIN
INSERT INTO course_catalogue(course_id, L, T, P, S, C, prereqs) VALUES (course_id, L, T, P,
S, C, prereqs);
END
$$;

--calling insert_in_catalogue
CALL insert_in_catalogue('cs305', 3, 1, 0, 3, 4, ARRAY['cs203', 'cs202']);


--course_offering schema

CREATE TABLE course_offering(
        course_id       VARCHAR(6) NOT NULL,
        semester    INTEGER  NOT NULL,
        year        INTEGER  NOT NULL,
        faculty_id  VARCHAR(6) NOT NULL,
        time_slot               INTEGER[],
        batch_list     INTEGER[],
        constraints         INTEGER,
        PRIMARY KEY(course_id, semester, year)
);
```

--procedure to add to course_offering

```
drop procedure insert_course_offering;
CREATE OR REPLACE PROCEDURE insert_course_offering(cour_id      VARCHAR(6),
        semester                INTEGER,
        year                    INTEGER,
        time_slot                       INTEGER[],
        batch_list              INTEGER[],
        constraints                     INTEGER
)
LANGUAGE plpgsql
AS $$
DECLARE
        fac_id          VARCHAR(6);
BEGIN
        SELECT current_user INTO fac_id;
        IF EXISTS (SELECT cour_id FROM course_catalogue c WHERE cour_id= c.course_id)
                THEN
                IF EXISTS (select * FROM time_table t WHERE t.slot=time_slot)
                THEN
                INSERT INTO course_offering(course_id, semester, year, faculty_id , time_slot,
batch_list, constraints) VALUES (cour_id, semester, year, fac_id , time_slot, batch_list,
constraints);
        ELSE raise notice 'The timetable slot could not be found';
        END IF;
        ELSE raise notice 'The course % could not be found', cour_id;
        END IF;
END;$$
```

--calling insert_course_offering

```
-- CALL insert_course_offering('ge101', 1, 2021, 20, ARRAY[110, 210, 310], NULL, NULL);

--  CALL insert_course_offering('hs475', 1, 2021, 1, ARRAY[110, 210, 310], ARRAY[2019,
2018], NULL);
```

```
-- CALL insert_course_offering('cs301', 1, 2021, 5, ARRAY[111, 211, 311], ARRAY[2019],
NULL);

-- CALL insert_course_offering('cs305', 1, 2021, 1, ARRAY[114,214,314], NULL, NULL);

-- create student_table

CREATE TABLE student_table(
        stu_id  VARCHAR(6) NOT NULL PRIMARY KEY,
        batch INTEGER NOT NULL);


--entering a student in student_table

CREATE OR REPLACE PROCEDURE student_entry(student_id   VARCHAR(6),
        batch                   INTEGER
)
LANGUAGE plpgsql
AS $$
DECLARE
        cur_fac_table refcursor;
        f record;
BEGIN
        INSERT INTO student_table(stu_id, batch) VALUES (student_id, batch);
        EXECUTE format('
        CREATE TABLE %I(
                course_id       VARCHAR(6) NOT NULL ,
                semester        INTEGER NOT NULL,
                year            INTEGER NOT NULL,
                grade           INTEGER,
PRIMARY KEY(course_id, semester, year))', 't_' || student_id);

        EXECUTE format ('
        CREATE USER %I WITH ENCRYPTED PASSWORD ''%I'',student_id, 'p_' ||
student_id);

        OPEN cur_fac_table FOR SELECT * FROM faculty_table;
                loop
                fetch cur_fac_table into f;
                exit when not found;
        EXECUTE format ('GRANT SELECT, UPDATE ON %I TO %I', 't_' ||
student_id,f.faculty_id);
        end loop;
```

```
        EXECUTE format ('GRANT SELECT ON %I TO %I', 't_' || student_id, student_id);
        EXECUTE format ('GRANT SELECT ON %I TO dean_acads', 't_' || student_id);
        EXECUTE format ('GRANT INSERT(course_id, semester, year) ON %I TO %I', 't_' ||
student_id, student_id);
        EXECUTE format ('GRANT SELECT ON course_offering TO %I', student_id);
        EXECUTE format ('GRANT SELECT ON course_catalogue TO %I', student_id);
EXECUTE format ('GRANT SELECT ON faculty_advisor TO %I', student_id);
        EXECUTE format ('GRANT SELECT ON faculty_table TO %I', student_id);
EXECUTE format ('GRANT SELECT ON time_table TO %I', student_id);
        EXECUTE format ('GRANT SELECT ON tickets_table TO %I', student_id);
EXECUTE format ('GRANT INSERT(student_id, course_id, semester, year) ON tickets_table
TO %I', student_id);
EXECUTE format ('GRANT SELECT, UPDATE on student_table TO %I',student_id);

END;$$




-- CALL student_entry('ee1221', 2018);
-- CALL student_entry('ch1001', 2020);
-- CALL student_entry('antara', 2019);


--grant all to dean_academics on student_table

-- GRANT ALL ON student_table TO dean_acads; //given previously
GRANT SELECT ON student_table TO dean_acads;


-- // create time slot table

CREATE TABLE time_table(
        slot INTEGER[] NOT NULL PRIMARY KEY
);

-- procedure to upload time table by dean_acads

CREATE PROCEDURE upload_time_slot(slot INTEGER[]
)
```

```
LANGUAGE plpgsql
AS $$
BEGIN
	INSERT INTO time_table(slot) VALUES(slot);
END;$$

--grant
GRANT ALL ON time_table TO dean_acads;

-- -Call statement -
-- CALL upload_time_slot(ARRAY[114, 214, 314]);

--checker

CREATE OR REPLACE FUNCTION checker(s_id VARCHAR(6),
                                                    sem INTEGER,
year INTEGER,
this_credit INTEGER)
RETURNS INTEGER
LANGUAGE plpgsql
AS $$
DECLARE
	bool_val INTEGER;
	prev_sem1  INTEGER;
	prev_sem2 INTEGER;
	prev_year1 INTEGER;
	prev_year2 INTEGER;
	sem_credits_record record;
	sem_credits refcursor;
	curr_credits DECIMAL;
	sem_credits_record1 record;
	sem_credits1 refcursor;
	prev_credits1 DECIMAL;
	sem_credits_record2 record;
	sem_credits2 refcursor;
	prev_credits2 DECIMAL;

BEGIN
	IF(sem = 1)
	THEN prev_sem1 = 2;
		prev_sem2=1;
		prev_year1 = year-1;
		prev_year2 = year-1;
```

```
        ELSE
                prev_sem1 = 1;
                prev_sem2=2;
                prev_year1 = year;
                prev_year2 = year-1;

        END IF;


        curr_credits = 0;
        EXECUTE format('
SELECT sum(cc.c)
FROM %I st, course_catalogue cc
WHERE st.course_id = cc.course_id
and st.semester=%s
and st.year=%s', 't_' || s_id, sem, year) into curr_credits;

        curr_credits = curr_credits + this_credit;

        prev_credits1 = 0;
        EXECUTE format('
SELECT sum(cc.c)
FROM %I st, course_catalogue cc
WHERE st.course_id = cc.course_id
and st.semester=%s
and st.year=%s', 't_' || s_id, prev_sem1, prev_year1) into prev_credits1;


        prev_credits2 = 0;
        EXECUTE format('
SELECT sum(cc.c)
FROM %I st, course_catalogue cc
WHERE st.course_id = cc.course_id
and st.semester=%s
and st.year=%s', 't_' || s_id, prev_sem2, prev_year2) into prev_credits2;


        IF(curr_credits >1.25* (prev_credits2 + prev_credits1)/2 )
        THEN
                bool_val = 0;
        ELSE
                bool_val = 1;
        END IF;
```

```
        RETURN bool_val;
END;
$$




-- procedure to register students
-- drop procedure student_registration;
CREATE OR REPLACE PROCEDURE student_registration(
        cour_id VARCHAR(6),
        sem     INTEGER,
        yea             INTEGER
)
LANGUAGE plpgsql
AS $$
DECLARE
        student_id VARCHAR(6);
        arr VARCHAR[];
        slot_arr INTEGER[];
        bat_arr INTEGER[];
        z INTEGER;
        x varchar(6);
        is_present INTEGER;
        y INTEGER;
        slot_same INTEGER;
        stud_table VARCHAR(10);
        cgpa DECIMAL;
        credit INTEGER;
        bool_var INTEGER;
        bat INTEGER;
        is_batch INTEGER;
BEGIN

        SELECT current_user INTO student_id;

        SELECT c.prereqs
        INTO arr
        FROM course_catalogue c
        WHERE c.course_id = cour_id;
```

```
        SELECT co.batch_list
        INTO bat_arr
        FROM course_offering co
        WHERE co.course_id = cour_id
        and co.semester = sem
        and co.year = yea;

        SELECT batch INTO bat FROM student_table s WHERE s.stu_id=student_id;

        is_batch = 0;
        is_present := 0;
        slot_same := 0;
        stud_table := 't_' || student_id;

        cgpa = calculate_cg(student_id);

        SELECT cc.c INTO credit FROM course_catalogue cc WHERE cc.course_id=cour_id;

        bool_var = checker(student_id, sem, yea, credit);

        IF (bat_arr IS NOT NULL)
        THEN
        FOREACH z IN ARRAY bat_arr
                LOOP
                IF (z=bat)
                THEN
                is_batch = 1;
                END IF;
                END LOOP;
        ELSE
                is_batch = 1;
        END IF;

        IF (arr IS NOT NULL)
        THEN
        FOREACH x IN ARRAY arr
                LOOP
                EXECUTE Format ('SELECT count(*) FROM %I as tbl WHERE tbl.course_id=
"%I"', stud_table, x)
                INTO is_present;
                END LOOP;
        END IF;
```

```
IF NOT EXISTS(SELECT * FROM course_catalogue cc WHERE cc.course_id=cour_id)
THEN
raise notice 'The course % is not in course_catalogue', cour_id;
ELSE
IF EXISTS(SELECT *
                FROM course_offering c
                WHERE c.course_id = cour_id
                        and c.semester = sem
                        and c.year = yea)
        THEN
        IF(bool_var = 1)
        THEN
        IF (is_batch = 1)
        THEN
        IF EXISTS (SELECT * FROM course_offering c
                        WHERE c.course_id = cour_id
                        and c.semester = sem
                        and c.year = yea
                        and (c.constraints is NULL or cgpa>=c.constraints))
        THEN
        IF (arr IS NOT NULL and is_present = 0)
        THEN
        raise notice 'The prerequistes of the course % are not fullfilled', cour_id;
        ELSE
                SELECT time_slot
                INTO slot_arr
                FROM course_offering c
                WHERE c.course_id = cour_id
                and c.semester=sem
                and c.year=yea;

                FOREACH y IN ARRAY slot_arr
                        LOOP
                        EXECUTE Format ('SELECT count(*)
                                                FROM
                                                (SELECT st.course_id, st.semester,
st.year, co.time_slot
                                                FROM %I st, course_offering co
                                                WHERE st.course_id=co.course_id
                                                and st.semester=co.semester
                                                and st.year=co.year) as new_tbl
                                                WHERE new_tbl.year=%s and
new_tbl.semester=%s and %s = ANY(new_tbl.time_slot)'
                                                , stud_table, yea, sem, y )
```

```
                                INTO slot_same;
                                IF (slot_same<>0)
                                THEN
                                        EXIT;
                                END IF;
                                END LOOP;

                        IF (slot_same<>0)
                        THEN
                        raise notice 'The time slot cannot be same as of another course already
taken';
                        ELSE
                        EXECUTE format('INSERT INTO %I(course_id, semester, year)
                                                VALUES ("%I", %s, %s)', 't_' || student_id, cour_id,
sem, yea);
                        END IF;
                END IF;
        ELSE
        raise notice 'CGPA constraint not cleared';
        END IF;
        ELSE
        raise notice 'batch not allowed';
        END IF;
        ELSE
        raise notice 'credit limit exceeded';
        END IF;
        ELSE
        raise notice 'The course % is not offered in % sem % year', cour_id, sem, yea;
        END IF;
        END IF;

END;$$
```

```
-- cgpa calculate procedure
```

```
-- DROP PROCEDURE calculate_cg;

CREATE OR REPLACE FUNCTION calculate_cg(s_id VARCHAR(6)
)
RETURNS DECIMAL
LANGUAGE plpgsql
AS $$
DECLARE
	cg DECIMAL;
BEGIN
	cg = 0;

	EXECUTE format ('
	SELECT ROUND(((sum(st.grade*cc.c)*1.0)/(sum(cc.c)*1.0))::numeric,2)
	FROM %I as st, course_catalogue as cc
	WHERE cc.course_id=st.course_id and st.grade IS NOT NULL',   't_' || s_id)
	INTO cg;

	IF cg IS NULL
	THEN
	cg = 0;
	END IF;

	RAISE NOTICE 'The cgpa of student with student ID % is %', s_id, cg;


	RETURN cg;

END;$$




-- CALL calculate_cg('ch1001');
```

```
-- procedure to insert in tickets_table by a student
DROP PROCEDURE insert_ticket;

CREATE OR REPLACE PROCEDURE insert_ticket(cour_id        VARCHAR(6),
        sem             INTEGER,
        yea             INTEGER
)
LANGUAGE plpgsql
AS $$
DECLARE
        stu_id VARCHAR(6);
BEGIN
        SELECT current_user INTO stu_id;
        IF EXISTS (SELECT * FROM course_offering co WHERE co.course_id = cour_id and
co.semester = sem and co.year = yea)
        THEN
        EXECUTE FORMAT('INSERT INTO tickets_table(student_id, course_id, semester, year)
VALUES ("%I" , "%I", %s, %s)', stu_id, cour_id, sem, yea);
        ELSE
        RAISE NOTICE '% course not offered in % sem % year',cour_id, sem, yea;
        END IF;
END;$$
```

```
--table to keep track of the faculty advisor of the batches

CREATE TABLE faculty_advisor(
	fac_id VARCHAR(6) NOT NULL,
	batch INTEGER NOT NULL PRIMARY KEY
);


--insert procedure for faculty advisor

CREATE OR REPLACE PROCEDURE insert_fac_advisor(fac_id VARCHAR(6),
batch INTEGER
)
LANGUAGE plpgsql
AS $$
BEGIN
	INSERT INTO faculty_advisor(fac_id, batch) VALUES (fac_id, batch);
END; $$

-- CALL insert_fac_advisor('f1', 2019);



-- // create table for all the tickets

CREATE TABLE tickets_table (
	student_id	VARCHAR(6) NOT NULL,
	course_id	VARCHAR(6) NOT NULL,
	semester		INTEGER  NOT NULL,
	year			INTEGER  NOT NULL,
	ins_app			INTEGER,
fa_app				INTEGER,
dean_app		INTEGER,
	PRIMARY KEY(student_id, course_id, semester, year)
);

GRANT UPDATE on tickets_table TO dean_acads;
```

```
CREATE OR REPLACE PROCEDURE insert_fac_advisor(fac_id VARCHAR(6),
batch INTEGER
)
LANGUAGE plpgsql
AS $$
BEGIN
	INSERT INTO faculty_advisor(fac_id, batch) VALUES (fac_id, batch);
END; $$


GRANT ALL ON  faculty_advisor TO dean_acads;



-- CALL insert_fac_advisor('f1', 2019);









CREATE TABLE dean_tickets_table (
	student_id	VARCHAR(6) NOT NULL,
	course_id	VARCHAR(6) NOT NULL,
	semester		INTEGER  NOT NULL,
	year			INTEGER  NOT NULL,
	credits			INTEGER NOT NULL,
dean_app		INTEGER,
	PRIMARY KEY(student_id, course_id, semester, year)
);

CREATE OR REPLACE PROCEDURE insert_fac_advisor(fac_id VARCHAR(6),
batch INTEGER
)
LANGUAGE plpgsql
AS $$
BEGIN
	INSERT INTO faculty_advisor(fac_id, batch) VALUES (fac_id, batch);
```

```
END; $$



CREATE TABLE faculty_table (
        faculty_id      VARCHAR(6) NOT NULL PRIMARY KEY
);



CREATE OR REPLACE PROCEDURE faculty_entry (faculty_id    VARCHAR(6) )
LANGUAGE plpgsql
AS $$
DECLARE
cur_stu_table refcursor;
f record;
BEGIN
        INSERT INTO faculty_table (faculty_id) VALUES (faculty_id);
        EXECUTE format('
        CREATE TABLE %I(
                student_id              VARCHAR(6) NOT NULL,
                course_id               VARCHAR(6) NOT NULL,
                semester                INTEGER NOT NULL,
                year                    INTEGER NOT NULL,
                credits                 INTEGER NOT NULL,
                if_faculty_advisor      INTEGER NOT NULL,
                ins_app                 INTEGER,
                PRIMARY KEY(student_id, course_id, semester, year,if_faculty_advisor))', 'f_' ||
faculty_id);

EXECUTE format ('
        CREATE USER %I WITH ENCRYPTED PASSWORD ''%I'',faculty_id, 'p_' || faculty_id);


                OPEN cur_stu_table FOR SELECT * FROM student_table;
                loop
                fetch cur_stu_table into f;
                exit when not found;
        EXECUTE format ('GRANT SELECT,UPDATE ON %I TO %I', 't_' || f.stu_id, faculty_id);
        end loop;

                EXECUTE format ('GRANT SELECT ON %I TO dean_acads', 'f_' || faculty_id);
```

```
        EXECUTE format ('GRANT UPDATE, INSERT, SELECT ON %I TO %I' , 'f_' || faculty_id,
faculty_id);
        EXECUTE format ('GRANT SELECT ON course_catalogue TO %I', faculty_id);
        EXECUTE format ('GRANT SELECT ON time_table TO %I', faculty_id);
        EXECUTE format ('GRANT SELECT, INSERT ON course_offering TO %I', faculty_id);
        EXECUTE format ('GRANT UPDATE on tickets_table TO %I', faculty_id);
        EXECUTE format ('GRANT SELECT on tickets_table TO %I', faculty_id);
        EXECUTE format ('GRANT SELECT ON faculty_advisor TO %I', faculty_id);
        EXECUTE format ('GRANT SELECT ON faculty_table TO %I', faculty_id);
        EXECUTE format ('GRANT SELECT on student_table TO %I', faculty_id);
        EXECUTE format ('GRANT pg_read_server_files TO %I;', faculty_id);


END;$$
```

```
CREATE OR REPLACE PROCEDURE insert_in_ftable(new_student_id VARCHAR(6),
new_course_id VARCHAR(6), new_semester INTEGER, new_year INTEGER)
LANGUAGE PLPGSQL
SECURITY DEFINER
AS $$
DECLARE
fac_id VARCHAR(6);
credit INTEGER;

BEGIN

SELECT faculty_id
INTO fac_id
FROM course_offering c
WHERE c.course_id=new_course_id
and c.semester = new_semester
and c.year=new_year ;

SELECT c INTO credit
FROM course_catalogue cc
WHERE cc.course_id=new_course_id;
```

```
EXECUTE format('INSERT INTO %I(student_id, course_id,  semester, year, credits,
if_faculty_advisor,  ins_app) VALUES( "%I", "%I", %s, %s, %s, 0, NULL)', 'f_' || fac_id,
new_student_id, new_course_id, new_semester, new_year, credit);

END;
$$




CREATE OR REPLACE PROCEDURE insert_in_advisor_table(new_student_id VARCHAR(6),
new_course_id VARCHAR(6), new_semester INTEGER, new_year INTEGER)
LANGUAGE PLPGSQL
SECURITY DEFINER
AS $$
DECLARE
advisor_id VARCHAR(6);
credit INTEGER;
bat INTEGER;
is_present INTEGER;

BEGIN

SELECT batch
INTO bat
FROM student_table s
WHERE new_student_id = s.stu_id;

SELECT f.fac_id
INTO advisor_id
FROM faculty_advisor f
WHERE f.batch = bat;

SELECT c INTO credit
FROM course_catalogue cc
WHERE cc.course_id=new_course_id;
```

```
EXECUTE format('INSERT INTO %I(student_id, course_id,  semester, year, credits,
if_faculty_advisor,  ins_app) VALUES( "%I" , "%I" , %s, %s, %s, 1, NULL)', 'f_' || advisor_id,
new_student_id, new_course_id, new_semester, new_year, credit);

END;
$$
```

```
CREATE OR REPLACE PROCEDURE insert_in_dean_table(new_student_id VARCHAR(6),
new_course_id VARCHAR(6), new_semester INTEGER, new_year INTEGER)
 LANGUAGE PLPGSQL
SECURITY DEFINER
AS $$
DECLARE
credit INTEGER;
is_present INTEGER;

BEGIN
SELECT c INTO credit
FROM course_catalogue cc
WHERE cc.course_id=new_course_id;

EXECUTE FORMAT('INSERT INTO dean_tickets_table(student_id, course_id,  semester, year,
credits, dean_app) VALUES( "%I" , "%I" , %s, %s, %s, NULL)', new_student_id,
new_course_id, new_semester, new_year, credit);
END;
$$
```

```
CREATE OR REPLACE FUNCTION  insert_in_fad_table ()
RETURNS TRIGGER
LANGUAGE plpgsql
```

```
AS $$
BEGIN
        CALL   insert_in_ftable(NEW.student_id, NEW.course_id, NEW.semester, NEW.year);
        CALL   insert_in_advisor_table(NEW.student_id, NEW.course_id, NEW.semester,
NEW.year);
        CALL   insert_in_dean_table(NEW.student_id, NEW.course_id, NEW.semester,
NEW.year);
        RETURN NEW;
END; $$



CREATE TRIGGER trig_insert_tickets_table
  AFTER INSERT
  ON tickets_table
  FOR EACH ROW
  EXECUTE PROCEDURE insert_in_fad_table();




CREATE OR REPLACE PROCEDURE generate_transcript(student_id      VARCHAR(6))
LANGUAGE plpgsql
AS $$

DECLARE
transcript_record record;
transcript refcursor;
cgpa DECIMAL;

BEGIN
                OPEN transcript FOR EXECUTE ('SELECT t.course_id, t.semester, t.year, cc.c,
t.grade
                FROM ' || quote_ident('t_' || student_id) || ' t, course_catalogue cc
                WHERE t.course_id = cc.course_id and t.grade is not NULL');
                LOOP
                        fetch transcript into transcript_record;
                        exit when not found;
                        raise notice '%',transcript_record;
                END LOOP;
                cgpa  = calculate_cg(student_id);
```

```
END
$$;

-- CALL generate_transcript('antara');



--import_csv procedure imports the grades with the attributes student_id and their grades
-- grade_in_student_table procedure writes the grades in the individual student tables. It should be called after the import_csv procedure.
-- Updated code - checked if the course was offered.
CREATE OR REPLACE PROCEDURE import_csv(course_id       VARCHAR(6),
                                       semester INTEGER,
                                       year INTEGER,
                                       path_of_file
VARCHAR(100))
LANGUAGE plpgsql
AS $$

DECLARE
count_var INTEGER;
faculty_id VARCHAR(6);



BEGIN

SELECT current_user INTO faculty_id;

EXECUTE format('SELECT count(*) FROM course_offering co WHERE co.faculty_id = "%I"
and co.course_id = "%I" and co.semester = %s and co.year = %s',faculty_id, course_id,
semester, year) into count_var;
IF(count_var <> 0)
THEN
EXECUTE format('
        CREATE TABLE %I(
                student_id      VARCHAR(6) NOT NULL PRIMARY KEY,
                grade           INTEGER NOT NULL)', course_id || '_' || semester || '_' || year);

EXECUTE FORMAT('COPY %I(student_id, grade)
FROM ''' || path_of_file || '''
DELIMITER ","
CSV HEADER', course_id || '_' || semester || '_' || year);
```

```
EXECUTE format('GRANT SELECT ON %I to dean_acads',  course_id || '_' || semester || '_' ||
year);
ELSE
RAISE NOTICE '% course not offered in % sem, % year by % faculty', course_id, semester,
year, faculty_id;
END IF;


END
$$;









-- CALL import_csv('hs475',1,2021,'C:\Users\Public\folder\myfile.csv');




-- update_on_dean procedure is used to update the deans approval in the dean_ticcket_table
and also in the dean column of the tickets_table




CREATE OR REPLACE PROCEDURE update_on_dean(s_id VARCHAR(6),
c_id VARCHAR(6),
sem INTEGER,
yea INTEGER,
approval INTEGER)
LANGUAGE plpgsql
AS $$
BEGIN
        UPDATE dean_tickets_table d
        SET dean_app = approval
WHERE d.student_id = s_id and
d.course_id = c_id and
d.semester = sem and
```

```
d.year = yea;

UPDATE tickets_table tb
        SET dean_app = approval
WHERE tb.student_id = s_id and
tb.course_id = c_id and
tb.semester = sem and
tb.year = yea;



END; $$



--when dean updates his/her decision, student table gets a course registered.
CREATE TRIGGER update_by_dean
  AFTER UPDATE OF dean_app
  ON dean_tickets_table
  FOR EACH ROW
  EXECUTE PROCEDURE insert_in_stable_by_ticket();




CREATE OR REPLACE FUNCTION  insert_in_stable_by_ticket()
RETURNS TRIGGER
SECURITY DEFINER
LANGUAGE plpgsql
AS $$
DECLARE

        arr VARCHAR[];
        x varchar(6);
        stud_table VARCHAR(10);
        is_present INTEGER;
        slot_arr INTEGER[];
        y INTEGER;
        slot_same INTEGER;

BEGIN
```

```
slot_same := 0;
is_present := 0;
stud_table := 't_' || NEW.student_id;


SELECT c.prereqs
INTO arr
FROM course_catalogue c
WHERE c.course_id = NEW.course_id;

IF (arr IS NOT NULL)
        THEN
        FOREACH x IN ARRAY arr
                LOOP
                EXECUTE Format ('SELECT count(*) FROM %I as tbl WHERE tbl.course_id=
"%I"', stud_table, x)
                INTO is_present;
                END LOOP;
        END IF;



IF(NEW.dean_app=1)
THEN
        IF (arr IS NOT NULL and is_present = 0)
                THEN
                raise notice 'The prerequistes of the course % are not fullfilled', NEW.course_id;
                ELSE
                        SELECT time_slot
                        INTO slot_arr
                        FROM course_offering c
                        WHERE c.course_id = NEW.course_id
                        and c.semester=NEW.semester
                        and c.year=NEW.year;

                        FOREACH y IN ARRAY slot_arr
                                LOOP
                                EXECUTE Format ('SELECT count(*)
                                                        FROM
                                                        (SELECT st.course_id, st.semester,
st.year, co.time_slot
                                                        FROM %I st, course_offering co
```

```
                                                    WHERE st.course_id=co.course_id
                                                    and st.semester=co.semester
                                                    and st.year=co.year) as new_tbl
                                                    WHERE new_tbl.year=%s and
new_tbl.semester=%s and %s = ANY(new_tbl.time_slot)'
                                                    , stud_table, NEW.year,
NEW.semester, y )
                                INTO slot_same;
                                IF (slot_same<>0)
                                THEN
                                        EXIT;
                                END IF;
                                END LOOP;

                        IF (slot_same<>0)
                        THEN
                        raise notice 'The time slot cannot be same as of another course already
taken';
                        ELSE
                        EXECUTE format('INSERT INTO %I(course_id, semester, year)
                                                VALUES ("%I", %s, %s)', 't_' || NEW.student_id,
New.course_id, NEW.semester, NEW.year);
                        END IF;
                END IF;


END IF;



                RETURN NEW;
END; $$
```

-- update_on_facultyprocedure is used to update the faculty approval in the ‘f_facultyid’ ticket table and also in the ins/fac column of the tickets_table

```
CREATE OR REPLACE PROCEDURE update_on_faculty(s_id VARCHAR(6),
c_id VARCHAR(6),
sem INTEGER,
yea INTEGER,
approval INTEGER)
LANGUAGE plpgsql
AS $$
DECLARE
        faculty_id VARCHAR(6);
   fac_table  VARCHAR(10);
        count_faculty INTEGER;
        count_fa INTEGER;
BEGIN
        SELECT current_user INTO faculty_id;
        fac_table := 'f_' || faculty_id;
        EXECUTE format ('UPDATE %I d
                                        SET ins_app = %s
                        WHERE d.student_id = "%I" and
                d.course_id = "%I" and
                d.semester = %s and
                d.year = %s', fac_table, approval, s_id, c_id, sem, yea );
        EXECUTE format ('UPDATE %I d
                                        SET ins_app = %s
                        WHERE d.student_id = "%I" and
                d.course_id = "%I" and
                d.semester = %s and
                d.year = %s', fac_table, approval, s_id, c_id, sem, yea );

        EXECUTE format ('SELECT count(*)  FROM %I f
                                WHERE f.if_faculty_advisor=0 and
                                f.student_id ="%I" and
                                f.semester = %s and
                                f.year= %s',fac_table, s_id, sem, yea ) into count_faculty;

        IF (count_faculty <> 0)
        THEN
        UPDATE tickets_table tb
        SET ins_app = approval
```

```
            WHERE tb.student_id = s_id and
            tb.course_id = c_id and
            tb.semester = sem and
            tb.year = yea;
        END IF;

        EXECUTE format ('SELECT count(*)  FROM %I f
                            WHERE f.if_faculty_advisor=1 and
                            f.student_id = ''%I''  and
                            f.semester = %s and
                            f.year= %s',fac_table, s_id, sem, yea) into count_fa;

        IF (count_fa <> 0)

THEN
        UPDATE tickets_table tb
        SET fa_app= approval
            WHERE tb.student_id = s_id and
            tb.course_id = c_id and
            tb.semester = sem and
            tb.year = yea;
        END IF;

END; $$
```

```
-- grade_in_student_table procedure writes the grades in the individual student tables. It should be called after the import_csv procedure.
CREATE OR REPLACE PROCEDURE grade_in_student_table(cour_id    VARCHAR(6),
                                                   semester INTEGER,
                                                   year INTEGER)
LANGUAGE plpgsql
AS $$
DECLARE
grade_record  record;
grades refcursor;
stud_table VARCHAR(10);
BEGIN
        OPEN grades FOR EXECUTE ('SELECT * FROM ' || quote_ident(cour_id || '_' ||
semester || '_' || year));
```

```
        LOOP
                fetch grades into grade_record ;
                exit when not found;
                stud_table := 't_' || grade_record.student_id;
                EXECUTE format ('UPDATE %I s
                SET grade = %s
                WHERE  s.course_id = "%I"
                and s.semester = %s
                and s.year = %s', stud_table,grade_record.grade,cour_id,semester,year);

        END LOOP;
END
$$;


-- call grade_in_student_table('hs475', 1, 2021);
```

--try

```
CREATE OR REPLACE PROCEDURE student_registration(
	cour_id VARCHAR(6),
	sem	INTEGER,
	yea		INTEGER
)
LANGUAGE plpgsql
AS $$
DECLARE
	student_id VARCHAR(6);
	arr VARCHAR[];
	slot_arr INTEGER[];
	bat_arr INTEGER[];
	z INTEGER;
	x varchar(6);
	is_present INTEGER;
	y INTEGER;
	slot_same INTEGER;
	stud_table VARCHAR(10);
	cgpa DECIMAL;
	credit INTEGER;
	bool_var INTEGER;
	bat INTEGER;
	is_batch INTEGER;
	is_p INTEGER;
	g INTEGER;
BEGIN

	SELECT current_user INTO student_id;

	SELECT c.prereqs
	INTO arr
	FROM course_catalogue c
	WHERE c.course_id = cour_id;

	SELECT co.batch_list
	INTO bat_arr
	FROM course_offering co
	WHERE co.course_id = cour_id
	and co.semester = sem
	and co.year = yea;

	SELECT batch INTO bat FROM student_table s WHERE s.stu_id=student_id;

	is_batch = 0;
```

```
        is_present := 0;
        slot_same := 0;
        stud_table := 't_' || student_id;

        cgpa = calculate_cg(student_id);

        SELECT cc.c INTO credit FROM course_catalogue cc WHERE cc.course_id=cour_id;

        bool_var = checker(student_id, sem, yea, credit);

        IF (bat_arr IS NOT NULL)
        THEN
        FOREACH z IN ARRAY bat_arr
                LOOP
                IF (z=bat)
                THEN
                is_batch = 1;
                END IF;
                END LOOP;
        ELSE
                is_batch = 1;
        END IF;

        IF (arr IS NOT NULL)
        THEN
        FOREACH x IN ARRAY arr
                LOOP
                is_p = 0;
                g = 0;
                EXECUTE Format ('SELECT count(*) FROM %I as tbl WHERE tbl.course_id=
"%I"', stud_table, x)
                INTO is_p;
                IF (is_p = 1)
                THEN
                is_present = 1;
                EXECUTE Format ('SELECT tbl.grade FROM %I as tbl WHERE tbl.course_id=
"%I"', stud_table, x) INTO g;
                IF (g IS NULL or g < 5)
                THEN
                is_present = 0;
                EXIT;
                END IF;
                END IF;
                END LOOP;
```

```
        END IF;


        IF NOT EXISTS(SELECT * FROM course_catalogue cc WHERE cc.course_id=cour_id)
        THEN
        raise notice 'The course % is not in course_catalogue', cour_id;
        ELSE
        IF EXISTS(SELECT *
                        FROM course_offering c
                        WHERE c.course_id = cour_id
                                        and c.semester = sem
                                        and c.year = yea)
                THEN
                IF(bool_var = 1)
                THEN
                IF (is_batch = 1)
                THEN
                IF EXISTS (SELECT * FROM course_offering c
                                        WHERE c.course_id = cour_id
                                        and c.semester = sem
                                        and c.year = yea
                                        and (c.constraints is NULL or cgpa>=c.constraints))
                THEN
                IF (arr IS NOT NULL and is_present = 0)
                THEN
                raise notice 'The prerequistes of the course % are not fullfilled', cour_id;
                ELSE
                        SELECT time_slot
                        INTO slot_arr
                        FROM course_offering c
                        WHERE c.course_id = cour_id
                        and c.semester=sem
                        and c.year=yea;

                        FOREACH y IN ARRAY slot_arr
                                LOOP
                                EXECUTE Format ('SELECT count(*)
                                                        FROM
                                                        (SELECT st.course_id, st.semester,
st.year, co.time_slot
                                                        FROM %I st, course_offering co
                                                        WHERE st.course_id=co.course_id
                                                        and st.semester=co.semester
                                                        and st.year=co.year) as new_tbl
```

```
                                                        WHERE new_tbl.year=%s and
new_tbl.semester=%s and %s = ANY(new_tbl.time_slot)'
                                                        , stud_table, yea, sem, y )
                                INTO slot_same;
                                IF (slot_same<>0)
                                THEN
                                        EXIT;
                                END IF;
                                END LOOP;

                        IF (slot_same<>0)
                        THEN
                        raise notice 'The time slot cannot be same as of another course already
taken';
                        ELSE
                        EXECUTE format('INSERT INTO %I(course_id, semester, year)
                                                VALUES ("%I", %s, %s)', 't_' || student_id, cour_id,
sem, yea);
                        END IF;
                END IF;
        ELSE
        raise notice 'CGPA constraint not cleared';
        END IF;
        ELSE
        raise notice 'batch not allowed';
        END IF;
        ELSE
        raise notice 'credit limit exceeded';
        END IF;
        ELSE
        raise notice 'The course % is not offered in % sem % year', cour_id, sem, yea;
        END IF;
        END IF;

END;$$


CREATE TABLE course_offering(
        course_id       VARCHAR(6) NOT NULL,
        semester    INTEGER  NOT NULL,
        year        INTEGER  NOT NULL,
        faculty_id  VARCHAR(6) NOT NULL,
        time_slot                   INTEGER[],
        batch_list     INTEGER[],
```

```
        constraints         DECIMAL,
        PRIMARY KEY(course_id, semester, year)
);



--procedure to add to course_offering

drop procedure insert_course_offering;
CREATE OR REPLACE PROCEDURE insert_course_offering(cour_id      VARCHAR(6),
        semester                INTEGER,
        year                    INTEGER,
        time_slot                       INTEGER[],
        batch_list              INTEGER[],
        constraints                     DECIMAL
)
LANGUAGE plpgsql
AS $$
DECLARE
        fac_id          VARCHAR(6);
BEGIN
        SELECT current_user INTO fac_id;
        IF EXISTS (SELECT cour_id FROM course_catalogue c WHERE cour_id= c.course_id)
                THEN
                IF EXISTS (select * FROM time_table t WHERE t.slot=time_slot)
                THEN
                INSERT INTO course_offering(course_id, semester, year, faculty_id , time_slot,
batch_list, constraints) VALUES (cour_id, semester, year, fac_id , time_slot, batch_list,
constraints);
        ELSE raise notice 'The timetable slot could not be found';
        END IF;
        ELSE raise notice 'The course % could not be found', cour_id;
        END IF;
END;$$
```