1) HOD is considered as a cross cutting faculty.

Procedures to handle Route updates

```
CREATE OR REPLACE FUNCTION route_update()
RETURNS TRIGGER AS $$
DECLARE
      temp_int INTEGER;
BEGIN
      BEGIN;
      UPDATE facult.leave SET pending_id=NEW.next_post WHERE
pending_id=OLD.next_post AND status=0;
      RETURN NEW;
END;$$
LANGUAGE PLPGSQL;
CREATE TRIGGER route_trigger
BEFORE UPDATE
ON facult.next_guy
FOR EACH ROW
EXECUTE PROCEDURE route_update();

CREATE OR REPLACE FUNCTION s_route_update()
RETURNS TRIGGER AS $$
DECLARE
      next_post_guy VARCHAR(20);
BEGIN
      SELECT INTO next_post_guy next_post FROM facult.next_guy WHERE
start_post=OLD.start_post;
      UPDATE facult.leave SET pending_id=next_post_guy FROM facult.faculty
WHERE  leave.status=0 AND faculty.email=leave.sender_id AND
faculty.post=OLD.start_post;
      COMMIT;
      RETURN NEW;

END;$$
LANGUAGE PLPGSQL;

CREATE TRIGGER s_route_trigger
BEFORE UPDATE
ON facult.start_end
FOR EACH ROW
EXECUTE PROCEDURE s_route_update();
```

Procedure to handle faculty post updates, and faculty deletions.

```
CREATE OR REPLACE FUNCTION cross_cutting_resign_email()
RETURNS TRIGGER AS $$
DECLARE
      temp_row RECORD;
      current_post_holder VARCHAR(30);
      temp_int INTEGER;
      temp_date DATE;
      next_post_guy VARCHAR(20);
BEGIN
      IF OLD.POST!=NEW.POST THEN
            IF OLD.POST!='faculty' THEN
            SELECT INTO temp_date doj from facult.current_cross_cutting where
faculty_id=OLD.email;
            INSERT INTO facult.archive_cross_cutting
(faculty_id,post,doj)VALUES(OLD.email,OLD.post,temp_date);
            DELETE from facult.current_cross_cutting where post=OLD.post AND
faculty_id=OLD.email;
            END IF;
            IF NEW.POST !='faculty' THEN
            IF NEW.POST='hod' THEN
                  SELECT INTO temp_int COUNT( *) from
facult.current_cross_cutting,facult.faculty WHERE current_cross_cutting.post
='hod' AND current_cross_cutting.faculty_id =faculty.email AND
faculty.department =OLD.department;
                  IF temp_int>0 THEN
                  SELECT INTO current_post_holder faculty.email from
facult.current_cross_cutting,facult.faculty WHERE current_cross_cutting.post
='hod' AND current_cross_cutting.faculty_id =faculty.email AND
faculty.department =OLD.department;
                  SELECT INTO temp_date doj from facult.current_cross_cutting
where faculty_id=current_post_holder;

                  UPDATE facult.faculty SET post='faculty' WHERE
faculty.email=current_post_holder;
                  DELETE from facult.current_cross_cutting where post=NEW.post
AND faculty_id=current_post_holder;

                  END IF;
            ELSE
```

```
                    SELECT INTO temp_int COUNT( *) from
facult.current_cross_cutting WHERE current_cross_cutting.post =NEW.post;
                    IF temp_int>0 THEN
                    SELECT INTO current_post_holder
current_cross_cutting.faculty_id from facult.current_cross_cutting WHERE
current_cross_cutting.post =NEW.POST;
                         SELECT INTO temp_date doj from
facult.current_cross_cutting where faculty_id=current_post_holder;

                    UPDATE facult.faculty SET post='faculty' WHERE
faculty.email=current_post_holder;
                    DELETE from facult.current_cross_cutting where post=NEW.post
AND faculty_id=current_post_holder;

                    END IF;
            END IF;
             INSERT INTO facult.current_cross_cutting(faculty_id,post)
VALUES(OLD.email,NEW.post);
            END IF;
            SELECT INTO next_post_guy next_post FROM facult.next_guy WHERE
start_post=NEW.post;
            UPDATE facult.leave SET pending_id=next_post_guy WHERE
sender_id=OLD.email AND status=0;
      END IF;

RETURN NEW;
END;$$
LANGUAGE PLPGSQL;


CREATE TRIGGER cross_resign_email
BEFORE UPDATE
ON facult.faculty
FOR EACH ROW
EXECUTE PROCEDURE cross_cutting_resign_email();

DROP function cross_cutting_resign_email() CASCADE;
UPDATE facult.faculty SET post='hod' WHERE email='a@a';


DROP function normal_faculty_retires() CASCADE;
CREATE OR REPLACE FUNCTION normal_faculty_retires()
RETURNS TRIGGER AS $$
DECLARE
      temp_date DATE;
      is_cross_cutting INTEGER :=0;
BEGIN
```

```
	INSERT INTO facult.archive_faculty
VALUES(OLD.email,OLD.department,OLD.doj);
	INSERT INTO facult.archive_leave SELECT
leave_id,sender_id,pending_id,doa,dos,doe,status from facult.leave where
leave.sender_id = OLD.email;
	INSERT INTO facult.archive_comments SELECT * from facult.comments where
comments.sender_id=OLD.email;
	SELECT INTO is_cross_cutting COUNT(*) from facult.current_cross_cutting
where faculty_id=OLD.email;
	IF is_cross_cutting>0 THEN
	SELECT INTO temp_date doj from facult.current_cross_cutting where
faculty_id=OLD.email;
	INSERT INTO facult.archive_cross_cutting(faculty_id,post,doj)
VALUES(OLD.email,OLD.post,temp_date);
	DELETE from facult.current_cross_cutting where faculty_id=OLD.email;
	END IF;
	DELETE from facult.comments where comments.sender_id=OLD.email;
	DELETE from facult.leave where leave.sender_id = OLD.email;

	RETURN OLD;
END;$$
LANGUAGE PLPGSQL;

CREATE TRIGGER faculty_retirement
BEFORE DELETE
ON facult.faculty
FOR EACH ROW
EXECUTE PROCEDURE normal_faculty_retires();
```