```
--------------------------------------------------------------------------------
-- Faculty Information part Tables
--------------------------------------------------------------------------------


--Create department table
create table department(
 dept_name varchar(6) PRIMARY KEY
);

-- Create faculty table -->
create table faculty(
 faculty_id int PRIMARY KEY,
 name varchar(20) not null,
 email varchar(20) not null,
 mobile_no varchar(10) not null,
 dept_name varchar(6) not null,
 FOREIGN KEY(dept_name) REFERENCES department(dept_name)
);

-- Create login table --
create table login_data(
 id SERIAL NOT NULL,
 username varchar(20) not null PRIMARY KEY,
 password varchar(20) default '123',
 faculty_id int,
 FOREIGN KEY(faculty_id) REFERENCES faculty(faculty_id)

);

-- create hod table
-- write a trigger to check if insertion in this table satisfy
-- by defaul hod will have period of 2 years
-- following condition :-
-- department of faculty is same as department of which he is been appointing as HoD
-- trigger_name -> verify_hod
-- another trigger on deletion from hod
-- which will store the deleting value in table old_hod
create table hod(
 faculty_id int not null,
 dept_name varchar(6) PRIMARY KEY,
 start_date DATE default CURRENT_DATE,
 end_date DATE default CURRENT_DATE + 730,
 FOREIGN KEY(faculty_id) REFERENCES faculty(faculty_id),
 FOREIGN KEY(dept_name) REFERENCES department(dept_name)
)

-- store all cross faculty role in an institution
create table cross_faculty_role(
 cross_role varchar(30) PRIMARY KEY
)

--cross cutting faculty appointed
-- each appointed role will have 2 year period default
```

```
create table cross_cutting_faculty(
 faculty_id int default null,
 cross_role varchar(30) PRIMARY KEY,
 start_date DATE default CURRENT_DATE,
 end_date DATE default CURRENT_DATE + 730,
 FOREIGN KEY (faculty_id) REFERENCES faculty(faculty_id),
 FOREIGN KEY (cross_role) REFERENCES cross_faculty_role(cross_role)
)

--Director table
-- each appointed role will have 4 year period default
create table director(
 faculty_id int default null,
 role varchar(10) default 'director',
 start_date DATE default CURRENT_DATE,
 end_date DATE default CURRENT_DATE + 730,
 FOREIGN KEY (faculty_id) REFERENCES faculty(faculty_id),
 PRIMARY KEY role
)

--max_leave (a faculty can take in a year) table
create table max_leave(
 max_leave int not null,
 current_year DATE default CURRENT_DATE
)

-- TABLE TO STORE remaining leave for each faculty
create table remaining_leave(
 faculty_id int PRIMARY KEY,
 remaining_leave int not null default 0,
 FOREIGN KEY (faculty_id) REFERENCES faculty(faculty_id)
)

----------------------------------------------------------------------------------
-- Leave Information and process  part Tables
----------------------------------------------------------------------------------

-- table to store leave application
-- leave_application table
create table leave_application(
 faculty_id int,
 application_id SERIAL not null unique,
 subject TEXT,
 description TEXT,
 start_date DATE,
 end_date DATE,
 applied_on TIMESTAMP default now(),
 status varchar(10) default 'pending',
 type varchar(1) default 'n',
 PRIMARY KEY (faculty_id, application_id)
)

-- table to store comment
create table comment(
 application_id int,
 comment_id SERIAL not null,
 commented_on TIMESTAMP,
 comment TEXT,
 --faculty_id should be either in faculty table or in old_faculty table
 faculty_id int not null,
 designation varchar(10) default 'self',
 PRIMARY KEY (application_id, comment_id),
 FOREIGN KEY (application_id) REFERENCES leave_application(application_id)
)


-- table to store route for normal leave application
create table route(
 initiator varchar not null,
 approved_by varchar not null,
 forward_to varchar not null,
 PRIMARY KEY (initiator, approved_by)
)

-- table to store route for retrospective leave application
create table route_r(
 initiator varchar not null,
 approved_by varchar not null,
 forward_to varchar not null,
 PRIMARY KEY (initiator, approved_by)
)


-- table to store leave application in process
create table pending_leave_application(
 faculty_id int,
 application_id int,
 prev_level varchar not null,
 current_level varchar not null,
 current_level_faculty_id int not null,
 status varchar(10) default 'pending',
 date TIMESTAMP default now(),
 FOREIGN KEY (faculty_id) REFERENCES faculty(faculty_id),
 FOREIGN KEY (application_id) REFERENCES leave_application(application_id),
 PRIMARY KEY (faculty_id, application_id, prev_level)
);

-- table to store processed leave application
-- Entry in this table will come from pending_leave_application
-- which means faculty_id will always be valid
create table leave_application_hist(
 faculty_id int,
 application_id int,
 prev_level varchar not null,
 current_level varchar not null,
 current_level_faculty_id int not null,
 status varchar(10) default 'pending',
 remark text default ' ',
 date TIMESTAMP,
 FOREIGN KEY (application_id) REFERENCES leave_application(application_id),
 PRIMARY KEY (faculty_id, application_id, prev_level)
);


---------------------------------------------------------------------------------
-- Stored Procedures and functions
---------------------------------------------------------------------------------

-----------------------------------------------------------
-- This procedure will chack if there is change in year
-- if yes, then it will update remaining_leave for each faculty
-- This procedure will be called whenever someone will access
-- the database from leave portal
-----------------------------------------------------------

create or replace procedure check_for_year_change()
language plpgsql
as
$$
declare
 prev_date DATE;
 date_now DATE;
 prev_year int;

 year_now int;
 faculty_data record;
 max_leave_data int;

 leave_cur cursor for
  select *
  from remaining_leave;
begin
 date_now = CURRENT_DATE;
 select current_year into prev_date from max_leave;
 prev_year = EXTRACT(YEAR FROM DATE (prev_date));
 year_now = EXTRACT(YEAR FROM DATE (date_now));

 raise info 'year_now : % , prev_year: %', year_now, prev_year;

 open leave_cur;
 if year_now!=prev_year then
  select max_leave into max_leave_data from max_leave;
  loop
   fetch leave_cur into faculty_data;

   exit when not found;

   update remaining_leave set remaining_leave = remaining_leave+max_leave_data
    where remaining_leave.faculty_id=faculty_data.faculty_id;

  end loop;
 end if;
 update max_leave set current_year = date_now;

 close leave_cur;
end
$$



--------------------------------------------------
-- given input initiator and approved_by
-- this function will return whom to forward now.
--------------------------------------------------

create or replace function find_table_name(
       _initiator varchar,
       _approved_by varchar,
       type char(1))
returns varchar
language plpgsql
as
$$
declare
 forward_to varchar;
begin
 forward_to = null;
 if type='n' then
  select r.forward_to into forward_to from route r
   where r.initiator=_initiator and r.approved_by=_approved_by;

  if not found then
   return null;
  end if;
 else
  select r.forward_to into forward_to from route_r r
   where r.initiator=_initiator and r.approved_by=_approved_by;

  if not found then
   return null;
  end if;
 end if;

 return forward_to;
end
$$

--------------------------------------------------------
-- This procedure will check if the start date of leave
-- leave application is today and it is still pending for
-- approval then this application will be rejected by system
-- this takes care of mentioned constraint:
-- In case the leave is not approved/rejected before the start
-- date of the leave, then it is automatically “rejected by the
-- system.”
--------------------------------------------------------
```

```
create or replace procedure auto_reject_bydate()
language plpgsql
as
$$
declare
 row record;

 application_cur cursor for
  select *
  from leave_application l;
begin
 open application_cur;

 loop
  fetch application_cur into row;

  exit when not found;

  if row.start_date = CURRENT_DATE and row.type = 'n' then
   update pending_leave_application set (status, date) = ('rejected', now())
    where status='pending' and faculty_id=row.faculty_id and application_id = row.application_id;

   update leave_application_hist set remark = 'rejected by system'
    where status = 'rejected' and faculty_id=row.faculty_id and application_id = row.application_id;
  end if;
 end loop;

 close application_cur;
end;
$$

--------------------------------------------------------------------------
-- procedure to auto reject application when faculty (who initiated application)
-- changed his role (e.g. from faculty to hod) before getting final approval
-- for his leave application
--------------------------------------------------------------------------

create or replace procedure auto_reject_when_faculty_change(facultyid int)
language plpgsql
as
$$
declare
 row record;
 applicationid int;

 application_cur cursor for
  select *
  from leave_application l;
begin
 -- find application id
 select distinct(p.application_id) into applicationid from pending_leave_application p
  where p.faculty_id=facultyid;

 if found then
```

```
  update pending_leave_application set (status, date) = ('rejected', now())
   where status='pending' and faculty_id=facultyid and application_id = applicationid;

  update leave_application_hist set remark = 'rejected by system'
   where status = 'rejected' and faculty_id=facultyid and application_id = applicationid;
 end if;


end;
$$

---------------------------------------------------------------------------
-- function to check whether a faculty has leave_application is in process or not
-- It will also check if given faculty has currently ongoing leave
---------------------------------------------------------------------------

create or replace function check_for_ongoing_leave_application(facultyid int)
returns int
language plpgsql
as
$$
declare
 today_date DATE;

 dummy record;
begin
 today_date = CURRENT_DATE;

 select * into dummy from remaining_leave r where r.faculty_id = facultyid;

 if dummy.remaining_leave <= 0 then
  return 1;
 end if;

 select * into dummy from leave_application l where l.faculty_id=facultyid and (l.status='pending');

 if found then
  return 1;
 end if;

 select * into dummy from leave_application l where l.faculty_id=facultyid and (l.status='approved') and l.end_date>
=today_date;

 if found then
  return 1;
 else
  return 0;
 end if;
end
$$


-----------------------------------------------------------------------------------
-- Triggers
```

```
--------------------------------------------------------------------------------

------------------------------------------------------------------------
-- Trigger for verifying insertion in HoD table
-- Trigger on deletion/update from HoD
-- this trigger will preform following operation
-- > If operation is delete then it will store information of HoD in old_hod table for record keeping purpose
-- > in case of insert
--  > this will check if faculty's department is same as HoD deparment (for which he is being appointed)
--  > It will check that appointing faculty should not be holding any other special position (Dean or director)
--  > It will reject the leave application applied by this faculty . Since, After becoming HoD route of application will
change
-- > in case of update
--  > It will perform all the operation of insert case
--  > it will store information of Previous HoD in old_hod table for record keeping purpose
--  > It will reject the leave application applied by previous HoD and new HoD . Since, After becoming HoD route o
f application will change
--  > This will also forward all the leave application to new HoD, which were pending approval at previous HoD
------------------------------------------------------------------------

-- table to store data of retired HoD
create table old_hod(
 id SERIAL NOT NULL PRIMARY KEY,
 faculty_id int not null,
 dept_name varchar(6),
 start_date DATE,
 end_date DATE,
 leaved_on TIMESTAMP(6),
 FOREIGN KEY(dept_name) REFERENCES department(dept_name)
)


create or replace function verify_hod_fun()
RETURNS TRIGGER
language plpgsql
as
$$
declare
 faculty_data record;

 update_cur cursor for
  select *
  from pending_leave_application p;
 row record;

 dummy record;
begin

 -- if insertion ir update then check the department
 -- of faculty and HoD department
 if TG_OP='INSERT' or TG_OP='UPDATE' then
  select *
  into faculty_data
  from faculty f
  where new.faculty_id = f.faculty_id;

  if not found then
   raise exception 'faculty does not exist';
  end if;

  select * into dummy from cross_cutting_faculty c where c.faculty_id=new.faculty_id;
  if found then
   raise exception 'Faculty already hold cross faculty role. Cannot be appointed as HoD';
  end if;

  select * into dummy from director d where d.faculty_id=new.faculty_id;
  if found then
   raise exception 'Faculty already hold Director position. Cannot be appointed as HoD';
  end if;

  if faculty_data.dept_name = new.dept_name then
   raise info '% is now new HoD of department %', faculty_data.name, faculty_data.dept_name;
  else
   raise exception 'faculty department is different from appointing HoD deparment';
  end if;

  call auto_reject_when_faculty_change(new.faculty_id);

  if TG_OP='UPDATE' then
   insert into old_hod (faculty_id, dept_name, start_date, end_date, leaved_on)
    values (old.faculty_id, old.dept_name, old.start_date, old.end_date, now());

   open update_cur;

   loop
    fetch update_cur into row;
    exit when not found;

    if row.current_level='hod' then
     update pending_leave_application set current_level_faculty_id = new.faculty_id
      where status='pending' and current_level = row.current_level;
    end if;
   end loop;

   call auto_reject_when_faculty_change(old.faculty_id);

   close update_cur;
  end if;

  return new;
 end if;

 -- if delete or update
 -- then store this old value in old_hod table
 if TG_OP='DELETE' then
  insert into old_hod (faculty_id, dept_name, start_date, end_date, leaved_on)
   values (old.faculty_id, old.dept_name, old.start_date, old.end_date, now());
 end if;
```

```
 return old;
end
$$;

drop trigger if exists verify_hod
on hod;

create trigger verify_hod
before insert or update or delete
on hod for each row
execute procedure verify_hod_fun();

--------------------------------------------------------------------------
-- Trigger for verifying insertion in cross_cutting_faculty table
-- Trigger on deletion/update from cross_cutting_faculty
-- this trigger will preform following operation
-- > If operation is delete then it will store information of cross_cutting_faculty in old_cross_cutting_faculty table fo
r record keeping purpose
-- > in case of insert
--  > It will check that appointing faculty should not be holding any other special position (HoD or director)
--  > It will reject the leave application applied by this faculty . Since, After becoming Cross faculty,  route of applic
ation will change
-- > in case of update
--  > It will perform all the operation of insert case
--  > it will store information of Previous cross_cutting_faculty in old_cross_cutting_faculty table for record keeping
 purpose
--  > It will reject the leave application applied by previous new HoD cross_cutting_faculty. Since, After becoming/r
emoving cross_faculty route of application will change
--  > This will also forward all the leave application to new Dean, which were pending approval at previous Dean
--------------------------------------------------------------------------

-- Table to store old_cross_cutting_faculty
create table old_cross_cutting_faculty(
 id SERIAL NOT NULL PRIMARY KEY,
 faculty_id int not null,
 cross_role varchar(30),
 start_date DATE,
 end_date DATE,
 leaved_on TIMESTAMP(6),
 FOREIGN KEY (cross_role) REFERENCES cross_faculty_role(cross_role)
)

-- will insert old value in old_cross_cutting_faculty whenever there is
-- update or deletion on table cross_cuttong_faculty
create or replace function verify_cross_cutting_faculty_fun()
returns trigger
language plpgsql
as
$$
declare
 cc_data record;

 update_cur cursor for
```

```
  select *
  from pending_leave_application p;
 row record;
 dummy record;
begin
 if TG_OP='UPDATE' or TG_OP='INSERT' then
  select *
  into cc_data
  from faculty f
  where new.faculty_id = f.faculty_id;

  if not found then
   raise exception 'faculty does not exist';
  end if;

  select * into dummy from hod h where h.faculty_id=new.faculty_id;
  if found then
   raise exception 'Faculty already hold HoD role. Cannot be appointed as cross cutting faculty';
  end if;

  select * into dummy from director d where d.faculty_id=new.faculty_id;
  if found then
   raise exception 'Faculty already hold Director position. Cannot be appointed as cross cutting faculty';
  end if;

  call auto_reject_when_faculty_change(new.faculty_id);

  if TG_OP='UPDATE' then
   insert into old_cross_cutting_faculty(faculty_id, cross_role, start_date, end_date, leaved_on)
    values (old.faculty_id, old.cross_role, old.start_date, old.end_date, now());

   open update_cur;

   loop
    fetch update_cur into row;
    exit when not found;

    if row.current_level='dean faculty affair' then
     update pending_leave_application set current_level_faculty_id = new.faculty_id
      where status='pending' and current_level = row.current_level;
    end if;
   end loop;

   close update_cur;

   call auto_reject_when_faculty_change(old.faculty_id);

  end if;

  raise info 'new % has been appointed', new.cross_role;
  return new;
 end if;

 if TG_OP='DELETE' then
```

```
  insert into old_cross_cutting_faculty(faculty_id, cross_role, start_date, end_date, leaved_on)
   values (old.faculty_id, old.cross_role, old.start_date, old.end_date, now());

  raise info 'faculty has been removed from % position', old.cross_role;
  return old;
 end if;

end
$$

drop trigger if exists verify_cross_cutting_faculty
on cross_cutting_faculty;

create trigger verify_cross_cutting_faculty
before insert or update or delete
on cross_cutting_faculty for each row
execute procedure verify_cross_cutting_faculty_fun();

--------------------------------------------------------------------------
--Trigger on insertion/deletion/update from director table
-- this trigger will preform following operation
-- > If operation is delete then it will store information of director in old_director table for record keeping purpose
-- > in case of update
--  > it will store information of Previous director in old_director table for record keeping purpose
--  > This will also forward all the leave application to new Director, which were pending approval at previous Direc
tor
--------------------------------------------------------------------------

-- Table to store previous director
create table old_director(
 id SERIAL NOT NULL PRIMARY KEY,
 faculty_id int not null,
 cross_role varchar(30) default 'director',
 start_date DATE,
 end_date DATE,
 leaved_on TIMESTAMP(6)
)

-- will insert old value in old_director whenever there is
-- update or deletion on table director
create or replace function verify_director_fun()
returns trigger
language plpgsql
as
$$
declare
 cc_data record;

 update_cur cursor for
  select *
  from pending_leave_application p;
 row record;
begin
 if TG_OP='UPDATE' or TG_OP='INSERT' then
```

```
  select *
  into cc_data
  from faculty f
  where new.faculty_id = f.faculty_id;

  if not found then
   raise exception 'faculty does not exist';
  end if;

  if TG_OP='UPDATE' then
   insert into old_director(faculty_id, start_date, end_date, leaved_on)
    values (old.faculty_id, old.start_date, old.end_date, now());

   open update_cur;

   loop
    fetch update_cur into row;
    exit when not found;

    if row.current_level='director' then
     update pending_leave_application set current_level_faculty_id = new.faculty_id
      where status='pending' and current_level=row.current_level;
    end if;
   end loop;

   close update_cur;
  end if;

  raise info 'new director has been appointed';
  return new;
 end if;

 if TG_OP='DELETE' then
  insert into old_director(faculty_id, start_date, end_date, leaved_on)
   values (old.faculty_id, old.start_date, old.end_date, now());

  raise info 'faculty has been removed from director position';
  return old;
 end if;

end
$$

drop trigger if exists verify_cross_cutting_faculty
on cross_cutting_faculty;

create trigger verify_director
before insert or update or delete
on director for each row
execute procedure verify_director_fun();

--------------------------------------------------------------------------
--Trigger on deletion from faculty table
-- It will :
```

```
-- >delete corresponding record from login_data table
-- >delete corresponding record from remaining_leave table
-- >delete corresponding record from director table (if he was a director)
-- >delete corresponding record from cross_cutting_faculty table (if he was a cross cutting faculty)
-- >delete corresponding record from hod table (if he was a HoD)
------------------------------------------------------------------------

-- Table to store retired/leaved faculty data
create table old_faculty(
 faculty_id int PRIMARY KEY,
 name varchar(20) not null,
 email varchar(20) not null,
 mobile_no varchar(10) not null,
 dept_name varchar(6) not null,
 leaved_on TIMESTAMP(6),
 FOREIGN KEY(dept_name) REFERENCES department(dept_name)
);

create or replace function delete_from_faculty_fun()
returns trigger
language plpgsql
as
$$
declare
 faculty_data record;
begin
 insert into old_faculty(faculty_id, name, email, mobile_no, dept_name, leaved_on)
  values (old.faculty_id, old.name, old.email, old.mobile_no, old.dept_name, now());

 delete from login_data
 where faculty_id=old.faculty_id;

 delete from remaining_leave
 where faculty_id=old.faculty_id;

 select * into faculty_data
 from director d
 where d.faculty_id=old.faculty_id;

 if found then
  delete from director d
  where d.faculty_id=old.faculty_id;
 end if;

 select * into faculty_data
 from cross_cutting_faculty c
 where c.faculty_id=old.faculty_id;

 if found then
  delete from cross_cutting_faculty c
  where c.faculty_id=old.faculty_id;
 end if;

 select * into faculty_data
 from hod h
 where h.faculty_id=old.faculty_id;

 if found then
  delete from hod h
  where h.faculty_id=old.faculty_id;
 end if;

 return old;
end
$$

drop trigger if exists delete_from_faculty
on faculty;

create trigger delete_from_faculty
before delete
on faculty for each row
execute procedure delete_from_faculty_fun();


--------------------------------------------------------
-- TRIGGER on insertion into faculty table
-- This will insert corresponding data in remaining_leave table
--------------------------------------------------------

create or replace function insert_into_remaining_leave_fun()
returns trigger
language plpgsql
as
$$
declare
 max_leave_data record;
begin
 select max_leave into max_leave_data from max_leave;

 insert into remaining_leave(faculty_id, remaining_leave)
  values(new.faculty_id, max_leave_data.max_leave);
 return new;

end
$$

drop trigger if exists insert_into_remaining_leave
on faculty;

create trigger insert_into_remaining_leave
after insert
on faculty for each row
execute procedure insert_into_remaining_leave_fun();

------------------------------------------------------------------------
-- Trigger for insert in leave_application
-- this will initiate leave application
-- this trigger will insert corresponding value in pending_leave_application
-- and hence will initiate leave_application
---------------------------------------------------------------------------

create or replace function initiate_application_fun()
returns trigger
language plpgsql
as
$$
declare
 initiator_faculty varchar;
 current_faculty varchar;
 table_name varchar;
 _dept_name varchar;
 current_faculty_id int;
 forward_to varchar;

 dummy record;
begin
 select * into dummy from faculty f where f.faculty_id=new.faculty_id;

 if not found then
  raise exception 'faculty does not exist';
 end if;

 -- insert into pending leave application
 -- seacrh into director
 -- director is not allowed to apply for leave
 select * into dummy from director d where d.faculty_id=new.faculty_id;

 if found then
  raise exception 'Director cannot apply for leave';
 end if;

 -- search into cross_cutting_faculty
 select * into dummy from cross_cutting_faculty c
  where c.faculty_id = new.faculty_id;

 if found then
  initiator_faculty = 'dean faculty affair';
  current_faculty = initiator_faculty;
 else
  -- search into hod
  select * into dummy from hod h where h.faculty_id=new.faculty_id;

  if found then
   initiator_faculty = 'hod';
   current_faculty = 'hod';
  else
   initiator_faculty = 'faculty';
   current_faculty = 'faculty';

   select f.dept_name into _dept_name from faculty f where f.faculty_id=new.faculty_id;
  end if;
 end if;

 table_name = find_table_name(initiator_faculty, current_faculty, new.type);

 if table_name='director' then
  select d.faculty_id into current_faculty_id from director d;
 elseif table_name = 'dean faculty affair' then
  select c.faculty_id into current_faculty_id from cross_cutting_faculty c
   where c.role = 'Dean Faculty Affairs';
 elseif table_name='hod' then
  select h.faculty_id into current_faculty_id from hod h where h.dept_name=_dept_name;
 else
  table_name=NULL;
 end if;

 if table_name is not null then
  insert into pending_leave_application(faculty_id, application_id, prev_level, current_level,
        current_level_faculty_id, date) values (new.faculty_id, new.application_id,
              initiator_faculty, table_name, current_faculty_id, now());
 end if;

 return new;

end;
$$;

drop trigger if exists initiate_application
on leave_application;

create trigger initiate_application
after insert on leave_application
for each row execute procedure initiate_application_fun();

------------------------------------------------------------------------
-- Trigger on delete in pending_leave_application
-- This will delete row from pending_leave_application and will insert
-- into leave_application_hist table
------------------------------------------------------------------------

create or replace function transfer_to_leave_application_hist_fun()
returns trigger
language plpgsql
as
$$
declare
 row record;

 pending_leave_cur cursor(facultyid int , applicationid int) for
  select *
  from pending_leave_application p
  where p.faculty_id=facultyid and p.application_id=applicationid;
begin
 open pending_leave_cur(old.faculty_id, old.application_id);
 --loop
 --fetch pending_leave_cur into row;

 --exit when not found;

 insert into leave_application_hist(faculty_id, application_id, prev_level, current_level,
  current_level_faculty_id, status, date) values (old.faculty_id, old.application_id, old.prev_level,
            old.current_level, old.current_level_faculty_id, old.status, old.date);
 --end loop;

 close pending_leave_cur;

 return old;
end
$$;

drop trigger if exists transfer_to_leave_application_hist
on pending_leave_application;

create trigger transfer_to_leave_application_hist
before delete
on pending_leave_application for each row
execute procedure transfer_to_leave_application_hist_fun();

-----------------------------------------------------------------------
-- Trigger on update in pending_leave_application
-- this will take care of application for work. And if there is nowhere to forward
-- then it will store corresponding data in leave_application_hist and delete that
-- from pending_leave_application
-----------------------------------------------------------------------

create or replace function update_leave_application_fun()
returns trigger
language plpgsql
as
$$
declare
 type char;
 forward_to varchar;
 initiator_faculty varchar;
 _dept_name varchar;
 current_faculty_id int;

 dummy record;
 leaveUsed int = 0;
 remainingLeave int;
begin
 if new.status != old.status then

  if new.status = 'rejected' then
   delete from pending_leave_application
    where faculty_id = new.faculty_id and application_id = new.application_id;

   update leave_application set status = new.status
    where faculty_id=new.faculty_id and application_id=new.application_id;

  elseif new.status = 'approved' then
   -- first find whom to forward
   -- type of application
   select l.type into type from leave_application l where
    l.faculty_id = new.faculty_id and l.application_id= new.application_id;

   -- find the level of initiator
   -- seacrh into cross_cutting_faculty
   select * into dummy from cross_cutting_faculty c
    where c.faculty_id = new.faculty_id;

   if found then
    initiator_faculty = 'dean faculty affair';
   else
    -- search into hod
    select * into dummy from hod h where h.faculty_id=new.faculty_id;

    if found then
     initiator_faculty = 'hod';
    else
     initiator_faculty = 'faculty';

     select f.dept_name into _dept_name from faculty f where f.faculty_id=new.faculty_id;
    end if;
   end if;

   forward_to = find_table_name(initiator_faculty,  new.current_level, type);

   if forward_to is null then
    select * into dummy from leave_application l where l.faculty_id = new.faculty_id
     and l.application_id=new.application_id;

    leaveUsed = (dummy.end_date - dummy.start_date) + 1;

    select r.remaining_leave into remainingLeave from remaining_leave r where r.faculty_id = new.faculty_id;

    update remaining_leave set remaining_leave = (remainingLeave - leaveUsed)
     where faculty_id = new.faculty_id;

    update leave_application set status = new.status
     where faculty_id=new.faculty_id and application_id=new.application_id;

    delete from pending_leave_application
     where faculty_id = new.faculty_id and application_id = new.application_id;
   else
    if forward_to='director' then
     select d.faculty_id into current_faculty_id from director d;
    elseif forward_to = 'dean faculty affair' then
     select c.faculty_id into current_faculty_id from cross_cutting_faculty c
      where c.cross_role = 'Dean Faculty Affairs';
    elseif forward_to='hod' then
     select h.faculty_id into current_faculty_id from hod h where h.dept_name = _dept_name;
```

```
    else
     forward_to=null;
    end if;

    if forward_to is not null then
     insert into pending_leave_application(faculty_id, application_id, prev_level, current_level,
            current_level_faculty_id, date) values (new.faculty_id, new.application_id,
                   new.current_level, forward_to, current_faculty_id, now());
    end if;

   end if;
  end if;
 end if;

 return new;
end
$$;

drop trigger if exists update_leave_application
on pending_leave_application;

create trigger update_leave_application
after update
on pending_leave_application for each row
execute procedure update_leave_application_fun();

------------------------------------------------------------
--TABLE ROUTE (TO STORE ROUTE OF AN LEAVE APPLICATION)
------------------------------------------------------------
      initiator       |     approved_by     |     forward_to
---------------------+---------------------+---------------------
 faculty             | faculty             | hod
 faculty             | hod                 | dean faculty affair
 hod                 | hod                 | director
 dean faculty affair | dean faculty affair | director


-- Table route_r to store route for retrospective leave application
      initiator       |     approved_by     |     forward_to
---------------------+---------------------+---------------------
 faculty             | faculty             | hod
 faculty             | hod                 | dean faculty affair
 faculty             | dean faculty affair | director
 hod                 | hod                 | director
 dean faculty affair | dean faculty affair | director
```