

1(a). Create a trigger on Table of your choice to check if the Primary key ID already exists or not before inserting a new record. & Send a custom reply instead of an error message.

```
CREATE OR REPLACE FUNCTION "cf_db".Tri_student()
RETURNS trigger
LANGUAGE 'plpgsql'
AS $$
DECLARE
i record;
BEGIN
    FOR i in (select * from "cf_db"."students")
    LOOP
        IF(NEW.s_id=i.s_id) then
            raise notice 'Student with same ID exists';
            RETURN NULL;
        END IF;
    END LOOP;

RETURN NEW;
END
$$;

CREATE TRIGGER "Trigger_ins"
BEFORE INSERT ON "cf_db".students
FOR EACH ROW EXECUTE PROCEDURE "cf_db".Tri_student();

INSERT INTO "cf_db".students(s_id) VALUES (9);
```

The screenshot displays the pgAdmin 4 web interface. On the left, the 'Browser' pane shows the database structure for 'PostgreSQL 10', including a database named '201801015_db' with a schema 'cf_db' containing a table 'students'. The 'Query Editor' pane on the right shows the following SQL script:

```
1 CREATE OR REPLACE FUNCTION "cf_db".Tri_student()
2 RETURNS trigger
3 LANGUAGE 'plpgsql'
4 AS $$
5 DECLARE
6 i record;
7 BEGIN
8     FOR i in (select * from "cf_db"."students")
9     LOOP
10         IF(NEW.s_id=i.s_id) then
11             raise notice 'Student with same ID exists';
12             RETURN NULL;
13         END IF;
14     END LOOP;
15
16 RETURN NEW;
17 END
18 $$;
19
20 CREATE TRIGGER "Trigger_ins"
21 BEFORE INSERT ON "cf_db".students
22 FOR EACH ROW EXECUTE PROCEDURE "cf_db".Tri_student();
23
24 INSERT INTO "cf_db".students(s_id) VALUES (9);
```

Below the query editor, the 'Data Output' pane shows the results of the execution:

```
NOTICE: Student with same ID exists
INSERT 0 0

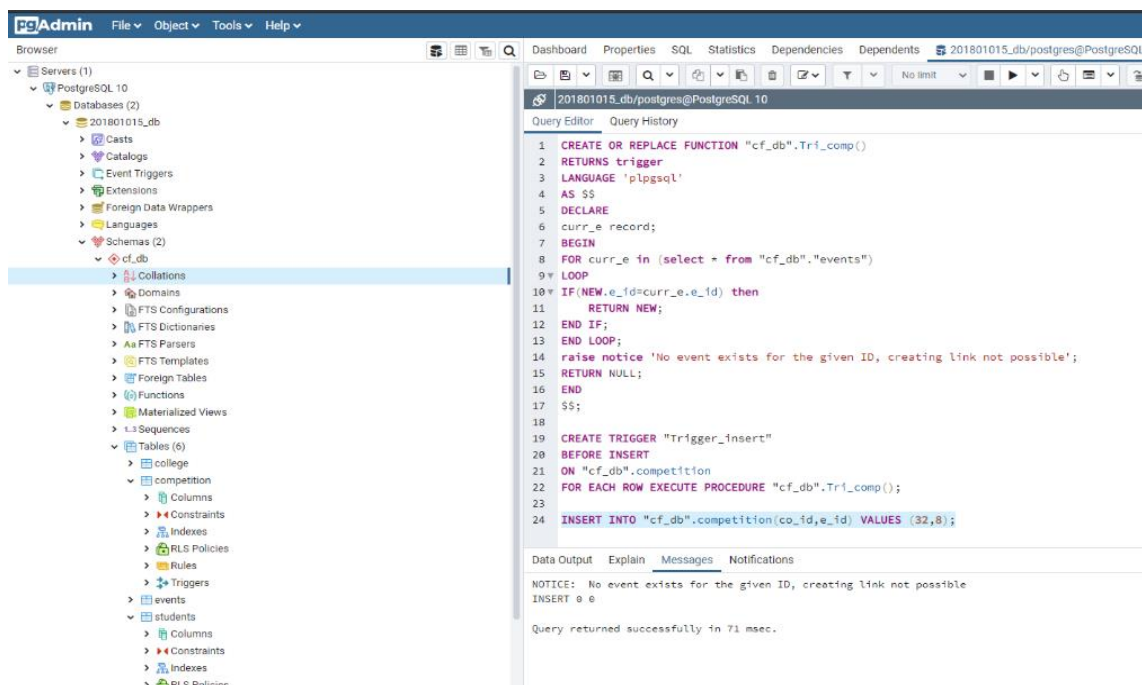
Query returned successfully in 99 msec.
```

1(b). Create a trigger on Table of your choice to check if the Foreign key ID already exists or not before inserting a new record. & Send a custom reply instead of an error message.

```
CREATE OR REPLACE FUNCTION "cf_db".Tri_comp()
RETURNS trigger
LANGUAGE 'plpgsql'
AS $$
DECLARE
curr_e record;
BEGIN
FOR curr_e in (select * from "cf_db"."events")
LOOP
IF(NEW.e_id=curr_e.e_id) then
RETURN NEW;
END IF;
END LOOP;
raise notice 'No event exists for the given ID, creating link not possible';
RETURN NULL;
END
$$;
```

```
CREATE TRIGGER "Trigger_insert"
BEFORE INSERT
ON "cf_db".competition
FOR EACH ROW EXECUTE PROCEDURE "cf_db".Tri_comp();

INSERT INTO "cf_db".competition(co_id,e_id) VALUES (32,8);
```



2 (Stored Procedure) 1.

Create user defined function 'winning_competitions' with one input argument that show the list of winning competitions for particular team name. (Consider top 3 place for winners)

```
create or replace function "cf_db"."winners_competition"(s character varying(250))
returns table(b character varying(250))
language 'plpgsql'
as $$
declare
    i RECORD;
begin
    for i in (select competition.co_name from CF_DB.competition join (select teams.t_name,
    winners.co_id from CF_DB.winners join CF_DB.teams on winners.team_id = teams.team_id
    where winners.position <= 3 and teams.t_name = s) as M on competition.co_id = M.co_id)
    loop b:=(i.co_name);
    return next;
    end loop;
end;
$$

select "cf_db"."winners_competition"('IITBA')
```

3 rows

The screenshot shows the pgAdmin 4 interface. On the left, the 'Servers' tree is expanded to show the 'cf_db' database. The 'Functions' folder is highlighted. The main pane displays the 'Query Editor' with the following SQL code:

```
1 create or replace function "cf_db"."winners_competition"(s character varying(250))
2 returns table(b character varying(250))
3 language 'plpgsql'
4 as $$
5 declare
6     i RECORD;
7 begin
8     for i in (select competition.co_name from CF_DB.competition join (select teams.t_name, winners.co_id from CF_DB.winners j
9 CF_DB.teams on winners.team_id=teams.team_id where winners.position<=3 and teams.t_name=s) as M on
10 competition.co_id=M.co_id)
11     loop b:=(i.co_name);
12     return next;
13     end loop;
14 end;
15 $$
16
17 select "cf_db"."winners_competition"('IITBA')
```

Below the query editor, the 'Data Output' tab is active, showing the results of the query. The results are displayed in a table with the following structure:

winners_competition	character varying
1	Battle Of Band
2	INC
3	I Relay

2 (Stored Procedure) 2.

Create user defined Function 'events_competition_list' that showing records for event's competitions for particular event name. Competitions list must show like (co_name,winning_price, co_date,co_time)

```
create or replace function "cf_db"."events_competition_list"(s character varying(250))
returns table(b character varying(250),price integer,date_1 character varying(100),time_1
character varying(100))
language 'plpgsql'
as $$
declare
    i RECORD;
begin
    for i in (select competition.co_name, competition.win_price, competition.date,
        competition.time from CF_DB.competition join CF_DB.events on
        events.e_id=competition.e_id
        where events.e_name=s)
    loop b:=(i.co_name); price:=(i.win_price); date_1:=(i.date); time_1:=(i.time);
        return next;
    end loop;
end;
$$;
select "cf_db"."events_competition_list"('Synapse');
```

19 rows

The screenshot displays the pgAdmin 4 web interface. On the left, the 'Browser' pane shows the database structure: Servers (1) > PostgreSQL 10 > Databases (2) > 201801015_db > Schemas (2) > cf_db. The 'Query Editor' pane on the right contains the SQL code for creating and executing the function 'events_competition_list'. Below the editor, the 'Data Output' pane shows the results of the function call, displaying 19 rows of competition data for the event 'Synapse'.

events_competition_list
14 (Rampage,4000,2020-02-23,9:00 PM)
15 (Encore,3000,2020-02-23,4:00 PM)
16 (Rhapsody,5000,2020-02-24,10:00 AM)
17 (Shalles,5500,2020-02-24,2:00 PM)
18 (Showdown,6000,2020-02-24,4:00 PM)
19 (Symposium,4000,2020-02-24,9:00 PM)

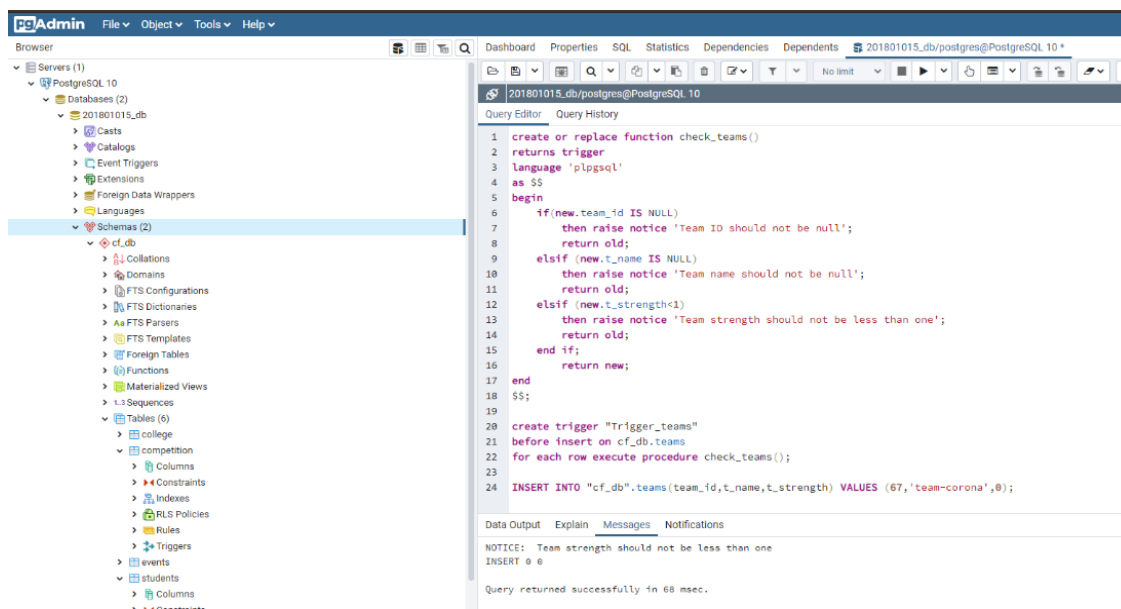
2 (Trigger) 1

Create Trigger name for 'teams' table that check following conditions before insert record into this teams table. 1. check team id is null or not, 2. check team name is null or not, 3. check team strength less than 1 or not. For these conditions trigger shows respective error message like 'team id should not be null'.

```
create or replace function check_teams()
returns trigger
language 'plpgsql'
as $$
begin
    if(new.team_id IS NULL)
        then raise notice 'Team ID should not be null';
        return old;
    elsif (new.t_name IS NULL)
        then raise notice 'Team name should not be null';
        return old;
    elsif (new.t_strength<1)
        then raise notice 'Team strength should not be less than one';
        return old;
    end if;
    return new;
end
$$;

create trigger "Trigger_teams"
before insert on cf_db.teams
for each row execute procedure check_teams();

INSERT INTO "cf_db".teams(team_id,t_name,t_strength) VALUES (67,'team-corona',0);
```



2 (Trigger) 2.

Create Trigger for checking on table 'competition' for after insert data you need to check new added competition's winning price is less than 2000 or not. If yes than you have to updated winning price with default value 2000.

```
create or replace function check_price()
returns trigger
language 'plpgsql'
as $$
begin
    if(new.win_price<2000)
    then UPDATE cf_db.competition SET win_price = 2000 WHERE co_id = new.co_id;
    raise notice 'The Win price entered is less than 2000, updated to default price
money';
    end if;

    return new;
end
$$;
```

```
create trigger "Winning_price_check"
after insert on cf_db.competition
for each row execute procedure check_price();
```

```
INSERT INTO "cf_db".competition(co_id,e_id,co_name,win_price,date,time) VALUES
(72,3,'corona_1',1000,'2020-02-21','6:00 PM');
```

```
select * from cf_db.competition
```

42 rows

The screenshot shows the pgAdmin 4 interface. On the left, the 'Servers' tree is expanded to show the 'cf_db' database. The 'Tables' folder is expanded, and the 'competition' table is selected. The main window displays the SQL query editor with the following code:

```
1 create or replace function check_price()
2 returns trigger
3 language 'plpgsql'
4 as $$
5 begin
6     if(new.win_price<2000)
7     then UPDATE cf_db.competition SET win_price = 2000 WHERE co_id = new.co_id;
8     raise notice 'The Win price entered is less than 2000, updated to default price money';
9     end if;
10    return new;
11 end
12 $$;
13
14 create trigger "Winning_price_check"
15 after insert on cf_db.competition
16 for each row execute procedure check_price();
17
18 INSERT INTO "cf_db".competition(co_id,e_id,co_name,win_price,date,time) VALUES (72,3,'corona_1',1000,'2020-02-21','6:00 PM');
19
20 select * from cf_db.competition
21
```

Below the query editor, the 'Data Output' tab is active, showing the results of the query. The table has 7 columns: co_id (PK) integer, e_id integer, co_name text, win_price integer, date date, and time text. The results show 42 rows, with the last row (row 42) highlighted in blue, showing the data for the 'corona_1' entry.

co_id (PK) integer	e_id integer	co_name text	win_price integer	date date	time text
38	38	3 i.Decipher	6000	2019-10-13	12:00 AM
39	39	3 i.Biz	4000	2019-10-13	2:00 PM
40	40	3 i.Ganith	5000	2019-10-13	4:00 PM
41	41	3 TreasureHunt	4500	2019-10-13	12:00 AM
42	72	3 corona_1	2000	2020-02-21	6:00 PM