Project 2 README

Eesun Moon (em3907), Chaeeun Ryu (cr3413)

PostgreSQL account name: em3907

Database name: proj1part2

Description of the Extensions in Project 2

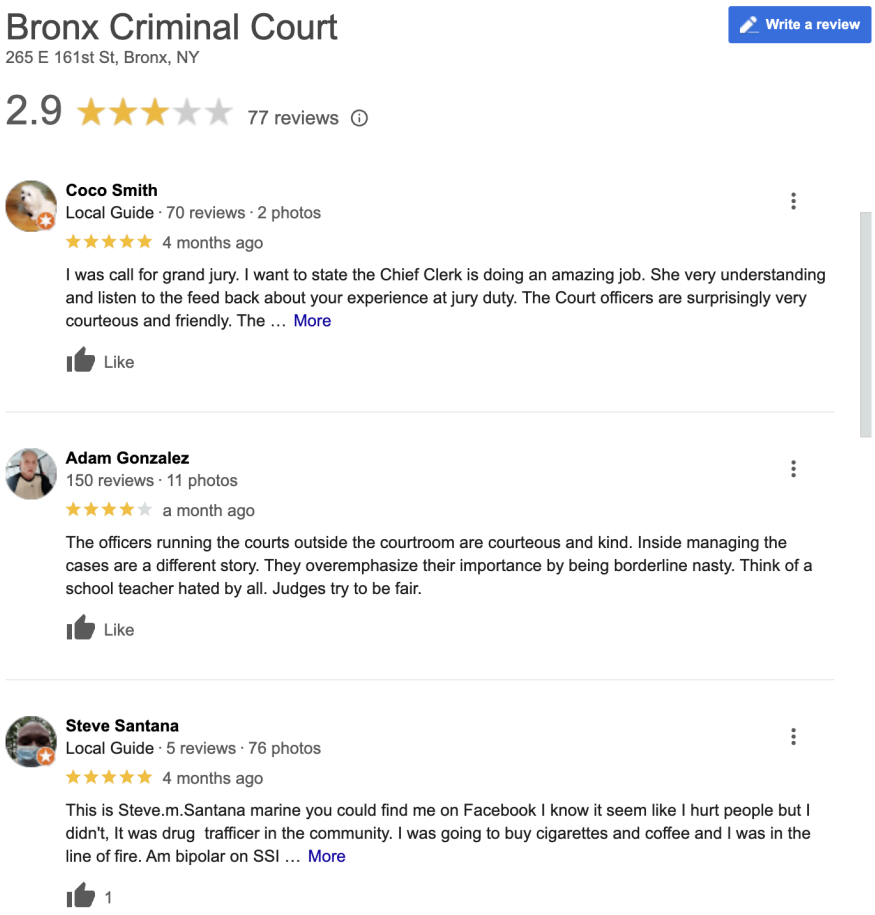
1. Array Attribute

According to the official website of Transit District, for each transit district, excluding the representative officer, there was more contact information about other officers for the transit district. Through the website, we were able to collect the names and email addresses of the officers listed in the website. Thus, we made a two dimensional array containing the contact information of multiple officers working in the particular transit district and updated the existing table 'Transit_Police' and added the new variable 'Sub_officer_info' containing the two dimensional array with the contact information. The following table explains the sample contact information that we embedded in a two dimensional array for Transit District 12 we added to our database.

Officer	Email
Sgt Sean Casey	Sean.Casey2@nypd.org
PO Dionisio Cruz	Dionisio.Cruz@nypd.org
PO Danauris Baez	Danauris.Baez@nypd.org
PO Janet Vega	Janet.Vega@nypd.org
PO Jillian Deluna	Jillian.Deluna@nypd.org
PO Nancy Thomas-Martinez	Nancy.ThomasMartinez@nypd.org
PO Reginald Minott	Reginald.Minott@nypd.org

2. Text

We collected google reviews about the courts in New York City and created a new attribute named ‘review’ for the table ‘court’ as an one dimensional array containing TEXT type values. The following screenshot is an example of how reviews look like for the ‘Bronx Criminal Court’.



3. Trigger

In the previous part (Project 1 - Part 3), we already mentioned as below:

Without ASSERTIONs, we cannot map the model:
- The constraints of *Transit_Police* and *Precinct* cover *NYPD*,
- The constraints of *Transit_Police* and *Precinct* do not overlap,
- The participation constraints between *Near* and *Hospital*, between *Arrive* and *Crime_Scene*, between *Arrive* and *Dispatch_Duration*, between *Incident* and *Monitor*, between *Send* and *Incident*, and between *Incident* and *Occured*.

Therefore, in this project, we map the model that we could not design before by using Triggers.

SQL Statements

1. Array Attribute

```
ALTER TABLE transit_police
ADD COLUMN sub_officers_info text[][];
```

2. Text

```
ALTER TABLE court
ADD COLUMN review TEXT[];
```

3. Trigger

To start with, we list all the triggers we have implemented:

```
                trigger_name                |          table_name          | tgtype | tgenabled
--------------------------------------------+------------------------------+--------+-----------
 on_delete_user                             | as6322.users                 |      9 | O
 user_id_generation_trigger                 | lss2205.users                |      7 | O
 updatepointstrigger                        | rta2125.match                |     21 | O
 playsduringtrigger                         | jaw2292.plays_during         |     11 | O
 validate_parental_income_and_education_trigger | jt3363.users             |     23 | O
 validate_transaction_id_on_update_trigger  | jt3363.initiated_transaction |     19 | O
 totalparthostcrim                          | hospital                     |      5 | O
 totalpartincinypd                          | incident                     |      7 | O
 totalpartdisparri                          | dispatch_duration            |     23 | O
 totalpartcrimarri                          | crime_scene                  |     23 | O
 totalpartincisend                          | incident                     |     23 | O
 totalpartincioccur                         | incident                     |     23 | O
 trigger_check_nypd_pid_in_precinct         | precinct                     |     23 | O
 trigger_check_nypd_pid_in_transit          | transit_police               |     23 | O
 inserttotalpartcompanyindustryin           | ogc2111.companies            |      5 | O
(15 rows)
```

Participation Constraints

(1) Between *Near* and *Hospital*

```
CREATE OR REPLACE FUNCTION enforce_hospital_participation()
RETURNS TRIGGER AS $$
BEGIN
IF NOT EXISTS (
SELECT 1
FROM Near
WHERE Near.HospitalID = NEW.HospitalID ) THEN
```

```
RAISE EXCEPTION 'Participation constraint violated: HospitalID % must
exist in the Near relationship.', NEW.HospitalID;
END IF;

RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TotalPartHostCrim
AFTER INSERT ON Hospital
FOR EACH ROW
EXECUTE FUNCTION enforce_hospital_participation();
```

(2) Between *Monitor* and *Incident*

```
CREATE OR REPLACE FUNCTION enforce_incident_participation()
RETURNS TRIGGER AS $$
BEGIN
IF NOT EXISTS (
SELECT 1
FROM Monitor
WHERE Monitor.INCIDENT_PID = NEW.INCIDENT_PID )
THEN
RAISE EXCEPTION 'Participation constraint violated: INCIDENT_PID %
must exist in the Monitor relationship.', NEW.INCIDENT_PID;
END IF;

RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TotalPartInciNYPD
BEFORE INSERT ON Incident
FOR EACH ROW EXECUTE FUNCTION enforce_incident_participation();
```

(3) Between *Arrive* and *Crime_Scene*

```
CREATE OR REPLACE FUNCTION enforce_crime_arrive_participation()
```

```
RETURNS TRIGGER AS $$
BEGIN
IF NOT EXISTS (
SELECT 1
FROM Arrive
WHERE Arrive.CRIME_SCENE_PID = NEW.CRIME_SCENE_PID
) THEN
RAISE EXCEPTION 'Participation constraint violated: CRIME_SCENE_PID %
must exist in the Arrive table.', NEW.CRIME_SCENE_PID; END IF;

RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TotalPartCrimArri
BEFORE INSERT OR UPDATE ON Crime_Scene
FOR EACH ROW
EXECUTE FUNCTION enforce_crime_arrive_participation();
```

(4) Between *Arrive* and *Dispatch_Duration*

```
CREATE OR REPLACE FUNCTION enforce_dispatch_arrive_participation()
RETURNS TRIGGER AS $$
BEGIN
IF NOT EXISTS (
SELECT 1
FROM Arrive
WHERE Arrive.DISPATCH_DURATION_PID = NEW.DISPATCH_DURATION_PID
) THEN
RAISE EXCEPTION 'Participation constraint violated:
DISPATCH_DURATION_PID % must exist in the Arrive table.',
NEW.DISPATCH_DURATION_PID;
END IF;

RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER TotalPartDispArri
BEFORE INSERT OR UPDATE ON Dispatch_Duration
FOR EACH ROW
EXECUTE FUNCTION enforce_dispatch_arrive_participation();
```

(5) Between *Send* and *Incident*

```
CREATE OR REPLACE FUNCTION enforce_incident_send_participation()
RETURNS TRIGGER AS $$
BEGIN
IF NOT EXISTS (
SELECT 1
FROM Send
WHERE Send.INCIDENT_PID = NEW.INCIDENT_PID
) THEN
RAISE EXCEPTION 'Participation constraint violated: INCIDENT_PID %
must exist in the Send relationship.', NEW.INCIDENT_PID;
END IF;

RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TotalPartInciSend
BEFORE INSERT OR UPDATE ON Incident
FOR EACH ROW
EXECUTE FUNCTION enforce_incident_send_participation();
```

(6) Between *Occured* and *Incident*

```
CREATE OR REPLACE FUNCTION enforce_incident_occurred_participation()
RETURNS TRIGGER AS $$
BEGIN
IF NOT EXISTS (
SELECT 1
FROM Occurred
WHERE Occurred.INCIDENT_PID = NEW.INCIDENT_PID
) THEN
```

```
RAISE EXCEPTION 'Participation constraint violated: INCIDENT_PID %
must exist in the Occurred relationship.', NEW.INCIDENT_PID;
END IF;

RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TotalPartInciOccur
BEFORE INSERT OR UPDATE ON Incident
FOR EACH ROW
EXECUTE FUNCTION enforce_incident_occurred_participation();
```

Constraints of No Overlap and Coverage in the ISA Hierarchies

(1) *Transit_Police* and *Precinct* **do not overlap**

```
CREATE OR REPLACE FUNCTION enforce_nypd_pid_uniqueness_in_precinct()
RETURNS TRIGGER AS $$
BEGIN
   IF EXISTS (SELECT 1 FROM transit_police WHERE nypd_pid =
NEW.nypd_pid) THEN
       RAISE EXCEPTION 'nypd_pid % already exists in transit_police',
NEW.nypd_pid;
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_check_nypd_pid_in_precinct
BEFORE INSERT OR UPDATE ON precinct
FOR EACH ROW
EXECUTE FUNCTION enforce_nypd_pid_uniqueness_in_precinct();


CREATE OR REPLACE FUNCTION enforce_nypd_pid_uniqueness_in_transit()
RETURNS TRIGGER AS $$
BEGIN
   IF EXISTS (SELECT 1 FROM precinct WHERE nypd_pid = NEW.nypd_pid)
```

```
THEN
       RAISE EXCEPTION 'nypd_pid % already exists in precinct',
NEW.nypd_pid;
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_check_nypd_pid_in_transit
BEFORE INSERT OR UPDATE ON transit_police
FOR EACH ROW
EXECUTE FUNCTION enforce_nypd_pid_uniqueness_in_transit();
```

(2) *Transit_Police* and *Precinct* **cover** *NYPD*

```
CREATE OR REPLACE FUNCTION enforce_coverage()
RETURNS TRIGGER AS $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Transit_Police WHERE NYPD_PID =
NEW.NYPD_PID) AND
       NOT EXISTS (SELECT 1 FROM Precinct WHERE NYPD_PID =
NEW.NYPD_PID) THEN
        RAISE EXCEPTION 'Coverage constraint violated: NYPD_PID %
must exist in either Transit_Police or Precinct.', NEW.NYPD_PID;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_coverage_on_nypd
AFTER INSERT OR UPDATE ON NYPD
FOR EACH ROW
EXECUTE FUNCTION enforce_coverage();
```

Explanation of Triggers & Events from the executed trigger

Triggers for the Participation Constraints

We have six participation constraints in the E/R diagram, such as (1) *Hospital* - *Near*, (2) *Incident* - *Monitor*, (3) *Crime_Scence* - *Arrive*, (4) *Dispatch_Duration* - *Arrive*, (5) *Send* - *Incident*, and (6) *Occured* - *Incident*. This participation constraints can be mapped using triggers, denoting as TotalPartHostCrim, TotalPartInciNYPD, TotalPartCrimArri, TotalPartDispArri, TotalPartInciSend, and TotalPartInciOccur, respectively.

For example, one of these triggers, TotalPartHostCrim, enforces the participation constraint between the *Hospital* and *Near* tables. Specifically, every HospitalID in the *Hospital* table must exist in the *Near* table, ensuring that a *Hospital* is always associated with at least one *Near* relationship, as required by the participation constraint. This trigger is executed after insert into the *Hospital* table. When a new row is inserted into the *Hospital* table, the trigger checks if the HospitalID exists in the *Near* table. If the HospitalID does not exist in *Near*, the trigger raises an exception, preventing the invalid insert. The example shown below represents the situation where the trigger raises an exception due to the attempt to insert a *Hospital* without a corresponding row in *Near*:

```
INSERT INTO Hospital (HospitalID, H_NAME, H_TYPE, H_PHONE, H_BOROUGH,
H_LOCATION)
VALUES (1000, 'City Hospital', 'Private', '987-654-3210', 'Brooklyn',
'456 Elm St');
```

```
proj1part2=> select HospitalID from Near where HospitalID=1000;
 hospitalid
------------
(0 rows)

proj1part2=> INSERT INTO Hospital (HospitalID, H_NAME, H_TYPE, H_PHONE, H_BOROUGH, H_LOCATION)
proj1part2-> VALUES (1000, 'City Hospital', 'Private', '987-654-3210', 'Brooklyn', '456 Elm St');
ERROR:  Participation constraint violated: HospitalID 1000 must exist in the Near relationship.
CONTEXT:  PL/pgSQL function enforce_hospital_participation() line 8 at RAISE
proj1part2=>
```

To show the event where the trigger is executed, we first check whether the element HospitalID = 1000 exists in the *Near* table. When attempting to insert a specific value in the *Hospital* table, the TotalPartHostCrim trigger checks if HospitalID = 1000 exists in the *Near* table. Since HospitalID = 1000 does not exist in the *Near* table, the trigger raises an exception. Therefore, the insertion operation is rolled back, and no row is added to the *Hospital* table.

Similarly, other triggers such as TotalPartInciNYPD, TotalPartCrimArri, TotalPartDispArri, TotalPartInciSend, and TotalPartInciOccur operate in the same manner. Each trigger is designed to enforce its respective participation constraint by checking the existence of the referenced entity in the associated table and raising an expectation if the constraint is violated. This ensures database integrity by preventing the insertion of invalid data.

Triggers in the ISA Hierarchies: The constraints of Coverage and No Overlap

We have a generalization / specialization hierarchy in our E/R diagram, where *NYPD* acts as the superclass and *Transit_Police* and *Precinct* act as subclasses. Additionally, this relationship has the constraints that *Transit_Police* and *Precinct* together must cover all entries in *NYPD*, but there is no overlap between the two subclasses. To implement this in PostgreSQL, in the previous project (Project 1 - part 3), we already created the tables of subclasses (*Transit_Police* and *Precinct*) as they have their own tables and reference *NYPD* via foreign keys. To represent these constraints in PostgreSQL, we use triggers:

(1) Trigger for Coverage (Total Participation) Constraint

The enforce_coverage trigger ensures that every NYPD_PID in the *NYPD* table is referenced in at least one of the subclasses (*Transit_Police* or *Precinct*). This enforces the total participation constraint in the ISA hierarchy.

Specifically, this trigger is attached to the *NYPD* table. After an INSERT or UPDATE on the *NYPD* table, the trigger checks if the NYPD_PID in the *NYPD* table is referenced in either *Transit_Police* or *Precinct*. If the NYPD_PID is not referenced in either table, the trigger raises an exception. The example shown below represents the situation where the trigger raises an exception:

```
INSERT INTO NYPD (NYPD_PID, PATRL_BORO_NM, BORO_NM)
VALUES (2580, 'PATROL BORO MAN NORTH', 'MANHATTAN');
```

```
proj1part2=> INSERT INTO NYPD (NYPD_PID, PATRL_BORO_NM, BORO_NM)
proj1part2-> VALUES (2580, 'PATROL BORO MAN NORTH', 'MANHATTAN');
ERROR:  Coverage constraint violated: NYPD_PID 2580 must exist in either Transit_Police or Precinct.
CONTEXT:  PL/pgSQL function enforce_coverage() line 5 at RAISE
```

When the query is executed (INSERT INTO NYPD), the *enforce_coverage* trigger checks the *Transit_Police* and *Precinct* tables to see if NYPD_PID = 2580 exists in either table. Since NYPD_PID = 2580 is not present in either *Transit_Police* or *Precinct*, the trigger raises an exception. Therefore, no modifications are made to the database as the insert operation is rolled back.

(2) Trigger for No Overlap Constraint

The two triggers *trigger_check_nypd_pid_in_precinct* and *trigger_check_nypd_pid_in_transit* are implemented to assure that there is no overlap between *nypd_pid* in *transit_police* table and *precinct* table. The trigger trigger_check_nypd_pid_in_precinct is attached to the precinct table and *trigger_check_nypd_pid_in_transit* is attached to the *transit_police* table. When updating or inserting either one of the tables, it calls the trigger and checks if the updated/inserted nypd_pid

exists in the other table. And if it doesn't exist in the other table, it enables insertion/updates. Otherwise, it throws an error message saying that the nypd_pid the user is currently trying to insert/update already exists in the other table.

```
INSERT INTO transit_police (nypd_pid, transitdistrict, officer_name,
officer_phone, officer_loc)
VALUES (1, 20, 'Officer A', '555-5678', 'Station A'); -- This will
raise an exception
```

There already exists nypd_pid of value 1 in the precinct table. So running the above code should give us an error based on our trigger. When running the code above, we get the following output:

```
proj1part2=>
proj1part2=> INSERT INTO transit_police (nypd_pid, transitdistrict, officer_name, officer_phone, officer_loc)
proj1part2-> VALUES (1, 20, 'Officer A', '555-5678', 'Station A'); -- This will raise an exception
ERROR:  nypd_pid 1 already exists in precinct
CONTEXT:  PL/pgSQL function enforce_nypd_pid_uniqueness_in_transit() line 5 at RAISE
proj1part2=>
```

Here, we could see that we confronted an error saying that nypid_pid = 1 already exists in our 'precinct' table.

Likewise, we would try inserting the nypd_pid in the transit_police table to precinct table.

```
INSERT INTO precinct (nypd_pid, pct_name, pct_address, pct_phone,
nypd_pct_cd)
VALUES (2, 'Precinct 1', '123 Main St', '555-1234', 100);
```

When we try to insert nypd_pid of value 2 that exists in the transit_police table into the precinct table using the code above, we get the following output:

```
proj1part2=> INSERT INTO precinct (nypd_pid, pct_name, pct_address, pct_phone, nypd_pct_cd)
proj1part2-> VALUES (2, 'Precinct 1', '123 Main St', '555-1234', 100);
ERROR:  nypd_pid 2 already exists in transit_police
CONTEXT:  PL/pgSQL function enforce_nypd_pid_uniqueness_in_precinct() line 5 at RAISE
proj1part2=>
```

From the output above, we could see that due to the trigger which we implemented so as to guarantee 'no overlap' property between the precinct and the transit police, inserting *nypd_pid* =

2 into *precinct* is not possible and we get an error that 2 already exists as *nypd_pid* in transit_police.

Two Meaningful SQL queries over our database

1. Query accessing elements in the **Array**

SQL Query

```
SELECT DISTINCT sub_officers_info
FROM transit_police
WHERE transitdistrict = 2;
```

Output

```
                                                                                                sub_officers_info


--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------
 {{"Sgt Kamil Choinski",Kamil.Choinski@nypd.org},{"DET Emmauel Tsividakis",Emmanuel.Tsivida
kis@nypd.org},{"PO Ranfier Villar",Ranfier.Villar@nypd.org},{"PO Jonathan Acosta",Jonathan.
Acosta@nypd.org},{"PO Alyssa Young",Alyssa.Young@nypd.org},{"PO Joseph Cammarato",Joseph.Ca
mmarato@nypd.org},{"PO Geoffrey Richard",Geoffrey.Richard@nypd.org}}
(1 row)
```

About: This query prints out the contact details of officers belonging to the Transit District = 2. From the above screenshot, we can see that the *sub_officers_info* attribute stores a two dimensional array with each value representing the name and email address of the officer. For example, we could see that an officer named Sgt Kamil Choinski with email Kamil.Choinski@nypd.org works for transit district 2.

2. Query using **full-text search**

SQL Query

```
SELECT
    c.court_address AS "Court Address",
    r.review_text AS "Review"
FROM court c
JOIN LATERAL unnest(c.review) AS r(review_text) ON true
WHERE r.review_text ILIKE '%rude%'
ORDER BY c.court_pid, r.review_text;
```

Output

```
                             Court Address                                 |
                                                                                                                       Review

----------------------------------------------------------------------+----------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------
---------------------------
 851 Grand Concourse, Bronx, NY 10451                                 | The height of bureaucracy. None of the phone numbers work, and the o
nes that do go to answering machines that haven't been set up yet. Lunch hours change from day to day. This is like if an author wanted to w
rite about a government office from hell. Of course, this review will accomplish nothing because the inconvenience, rudeness, and lack of ac
cess is the point: It's here to punish you. \n\nTheir lunch hours are not posted online, so it can be anywhere from 12-1 or 1-2 depending on
 how they feel. Don't expect any politeness. Don't even bother calling, no one will pick up. Just go there, take a sick day, bring a book an
d rough it out. Good luck.
 Bronx, Brooklyn, Midtown, Manhattan, Queens, Red Hook, Staten Island | I encountered very rude, indifferent and irresponsible customer serv
ice, I encountered very rude, indifferent and irresponsible customer service
 Bronx, Brooklyn, Midtown, Manhattan, Queens, Red Hook, Staten Island | Rude people on the phone, called about a ticket, was hung up on twic
e. Everyone there is a "supervisor", did not answer questions. Told me they could not help me with the court address, overall the rudest, mo
st inept people and worst experience on a phone call ever. Was yelled at multiple times.
(3 rows)
```

About: Above screenshot shows the output for the query to search through the court review and print out the review that contains ‘rude’ in its review along with the corresponding address of the court that had the review.