# Project 1 Part 3 Submission

Eesun Moon (em3907), Chaeeun Ryu (cr3413)

**PostgreSQL account name**: em3907

**Database name:** proj1part2

## Description of the extensions

*Names of the entities are written in Italics and newly created entities and relationships are in bold.

We built an E/R diagram on the crime database with the advice from our supportive TA, Quan Fang. All of the data we loaded onto the server are real-world data, collected from various official sources including 'NYC Open Data' and official websites of NYPD. The content of our submission follows the order: 1. Brief listings of entities and relationships we had as our initial proposal before receiving the feedback 2. How we modified the pre-proposed diagrams according to the TA's feedback 3. Description of our extension.

1. Initial Proposal: Entities and Relationships - Part 1

   Our initial project design consisted of seven entities and eights relationships, as follows:

- Entities: *NYPD*, *Radio_Code*, *Dispatch_Duration*, *Crime_Location*, *Incident*, *Outbreak_Time*
- Relationships: *Send* (a ternary relationship between *Incident*, *NYPD* and *Radio_code*), *Add_System*/*Call*/*Close*/*Monitor* (between *NYPD* and *Incident*), *Arrive* (a ternary relationship between *NYPD*, *Dispatch_Duration*, and *Crime_Location*), *Located_at* (between *Incident* and *Crime_Location*), and *Happend_at* (between *Incident* and *Outbreak_Time*).

2. Revised Design: Entities and Relationships (after Feedback - Part 2)

Following the TA's feedback, "the exactly one cases, including *Add_System*, *Call*, and *Close*, can associate with the *Incident*, so these can be combined them into a single table, simplifying the structure and ensuring that each incident maintains a direct one-to-one relationship with each of these actions or locations," we simplified and refined our diagrams to ensure clarity and consistency. We tried to combine the diagrams that have similar meanings as one diagram.

Specifically, we combined *Outbreak_Time* and *Crime_Location* into a single entity, ***Crime_Scence***, to incorporate both time and location details. Also, we merged the relationships, such as *Add_System*, *Call*, *Close*, and *Monitor*, into a single ***Monitor*** relationship as well as converted this relationship to 'at least one' case since each *Incident* can be monitored from several *NYPD*. Just like we

did for ***Crime_Scene***, we also integrated *Located_at* and *Happened_at* relationships into a single one-to-one ***Occurred*** relationship, ensuring that each *Incident* is tied to a specific *Crime_Scence*, and vice versa also applies. For the ***Send*** relationship, we added a key constraint to ensure that each combination of *NYPD* and *Incident* is linked to at most one *Radio_Code*.

- Entities: ***Crime_Scene*** (Integration of *Crime_Location* and *Outbreak_Time*), *NYPD*, *Radio_Code*, *Dispatch_Duration*, and *Incident*
- Relationships: ***Monitor*** (between *NYPD* and *Incident* - Integration of *Add_System*, *Call*, *Close*, and *Monitor*), ***Occurred*** (between *Incident* and *Crime_Scence* - Integration of *Located_at* and *Happend_at*), *Send* (a ternary relationship between *Incident*, *NYPD* and *Radio_code*), and *Arrive* (a ternary relationship between *NYPD*, *Dispatch_Duration*, and *Crime_Location*)

## 3. Extended Design: Entities and Relationships - Part 3

While in Part 1, the diagram focused on the basic information about the crime incident itself such as locations, nypd, radio code and etc, in Part3, we expanded our design to include additional entities and relationships that focus on stakeholders that are involved. To be specific, the stakeholders involved in crime incidents are two types: people (e.g., ***Victims*** and ***Suspects***) and institutions (e.g., ***Courts*** and ***Hospitals***). Additionally, to ensure comprehensive coverage, we incorporated detailed ISA hierarchies about *NYPD*, by including the subtypes of the supertype NYPD, which are ***Transit Police*** and ***Precinct***.

The *NYPD* entity includes two hierarchical subtypes: ***Transit_Police*** specializes in overseeing transit systems, such as railways, bus lines, and subway systems, and ***Precinct*** manages specific geographical regions across the five boroughs of NYC. Each subset has distinct roles and responsibilities, with no overlap in coverage. These relationships (i.e., ***Transit_Police*** and ***Precinct***), initially represented the descriptive attribute JURIS_DESC (description of jurisdiction) in our prior diagram and we expanded it to represent NYPD more comprehensively. To do so, we collected additional data from different sources for the two entities respectively. We also added to our schema with the check constraint that attribute 'Borough' in the *NYPD* entity should be defined as one of the five boroughs: Bronx, Brooklyn, Manhattan, Queens, or Staten Island.

The ***Court*** entity represents NYC's judicial system and includes attributes such as court type (e.g., Supreme Court, Family Court, Civil Court), description about the court, address, and phone number. Each *Incident* can be judged by at most one court, but each court can oversee multiple incidents. This relationship is represented through the ***Judge*** relationship, which includes a key constraint between *Judge* and *Court*.

The ***Hospital*** entity includes NYC hospitals' names, types, contact information, and geographic details (borough and location). A ***Near*** relationship connects the *Crime_Scence* entity to nearby *Hospitals*. Since every borough has at least one hospital, and each hospital can serve multiple crime scenes, a total participation constraint is applied to maintain consistency. To ensure geographic accuracy, the latitude and longitude ranges follow those of New York City, specifically between 40 and 41 for latitude, and between -73 and -75 for longitude.

The ***Victim_Type*** and ***Suspect_Type*** entities represent individuals's demographic details (e.g., race and gender) involved in incidents. Since a victim can exist only when associated with a suspect, *Victim_Type* is modeled as a weak entity dependent on *Suspect_Type* through the ***Harm*** identifying relationship set. The ***Harm*** relationship enforces a total participation between *Victim_Type* and *Harm* and

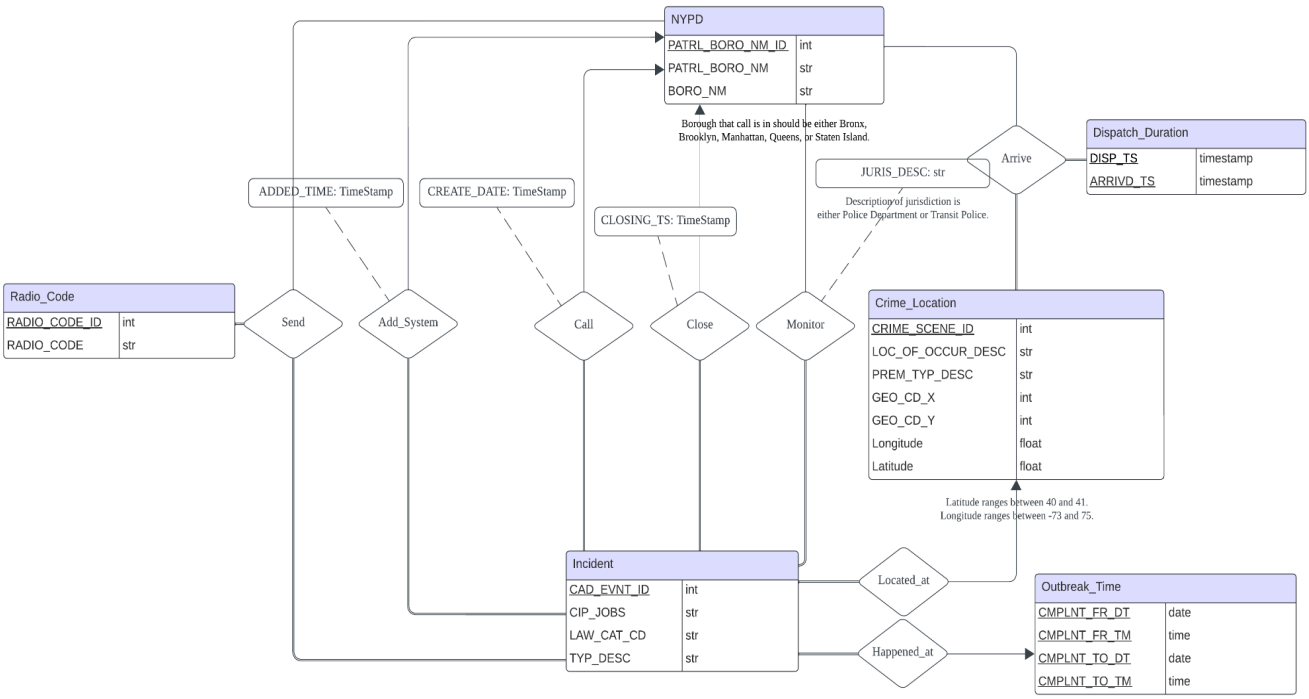
a key constraint between *Suspect_Type* and *Harm*. Aggregation is used in the ***Harm*** relationship to represent personal details collectively.

The ***Occurred*** relationship links an *Incident* (record), *Crime_Scene* (geographic and temporal details), and *Harm* (aggregating personal details). A key constraint between *Occurred* and *Crime_Scene* ensures that the unique combination of each *Incident* and Individual information (*Victim* and *Suspect*) is linked at a specific (exactly one) *Crime_Scene*.

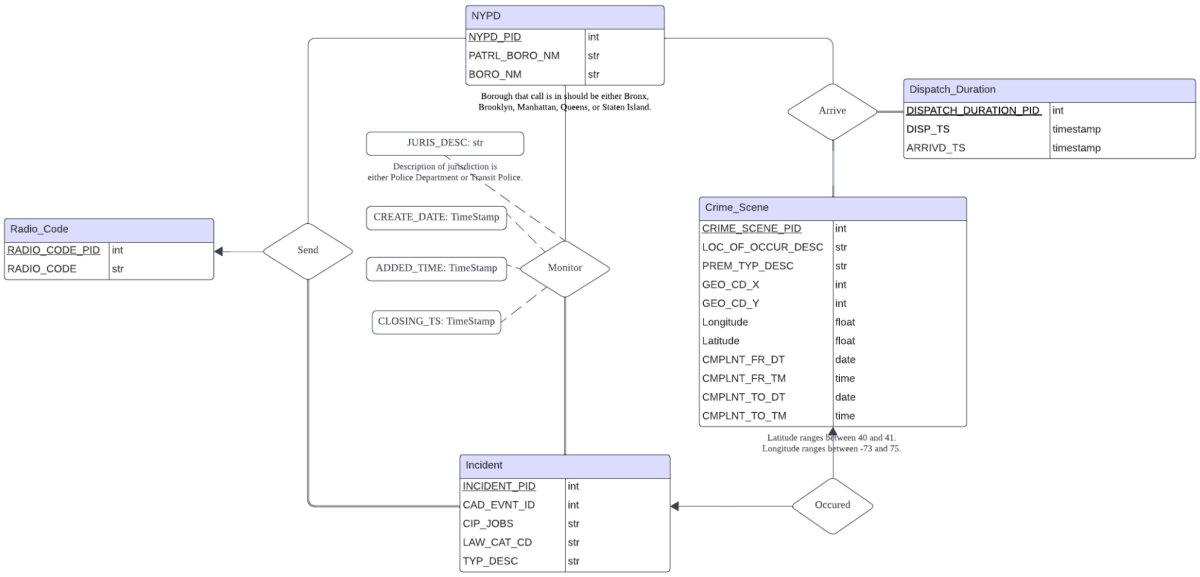
- Entities: *NYPD* (hierarchy: ***Transit_Police*** and ***Precinct***), ***Hospital***, ***Suspected_Type***, ***Victim_Type*** (weak entity set tied to *Suspected_Type*), ***Court***, *Radio_Code*, *Dispatch_Duration*, *Crime_Scene*, and *Incident*
- Relationships: ***Harm*** (Identifying relationship between *Suspected_Type* and *Victim_Type*), ***Occurred*** (a ternary relationship between *Incident*, *Crime_Scence*, and aggregation of ***Harm***), ***Near*** (relationship between *Crime_Scence* and *Hospital*), ***Judge*** (between *Incident* and *Court*), *Send* (a ternary relationship between *Incident*, *NYPD* and *Radio_code*), *Monitor* (between *NYPD* and *Incident*), and *Arrive* (a ternary relationship between *NYPD*, *Dispatch_Duration*, and *Crime_Location*)

# E/R Diagram

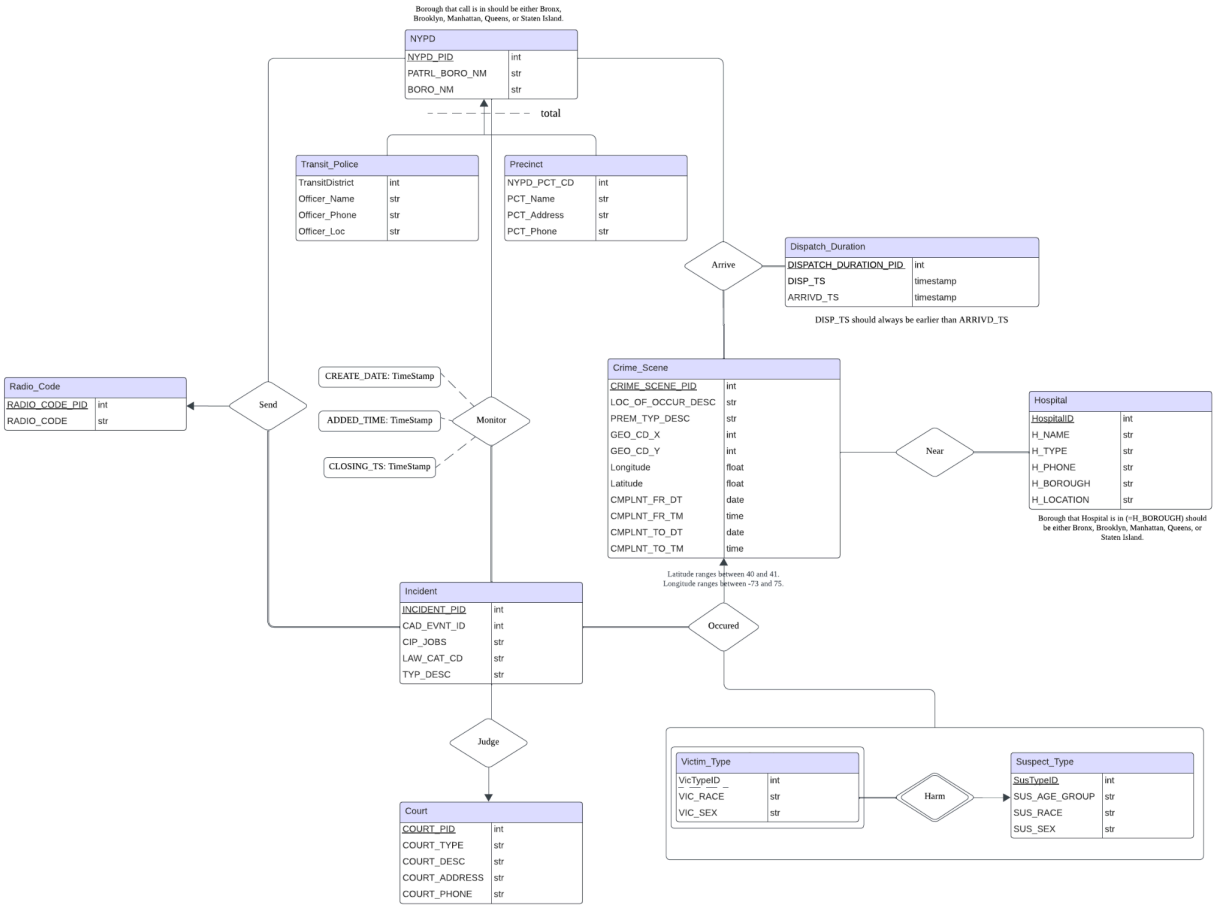
## Initial E/R Diagram - Part 1



## Revised E/R Diagram after applying feedback - Part 2




## Expanded E/R Diagram - Part 3



## Mapping E/R Diagram to SQL Schema

For the entities of ***Court*** and *Incident*, and their corresponding ***Judge*** relationship (which involves one key constraint), we chose the approach of using two tables, as discussed in class. The ***Court*** entity was mapped to the *Court* table, which stores court-related information, and the *Incident* entity was mapped to the *Judge_Incident* table, which combines the *Incident* entity and ***Judge*** relationship. For the primary key to uniquely identify each table, the COURT_PID was designed as the primary key in the ***Court*** table, while the INCIDENT_PID was used as the primary key in the ***Judge_Incident*** table. Also, the ***Judeg_Incident*** table includes a COURT_PID that serves as a foreign key, referencing the primary key in the ***Court*** table. This setup enforces the 'exactly one' case, where each incident is associated with a single court, while a court can handle multiple incidents.

We designated ***Victim_Type*** as the weak entity associated with ***Suspect_Type***. Therefore, the ***Victim_Type*** weak entity set and the ***Harm*** identifying relationship set were combined into a single table, ***Victim_Who_Harmed***. This table uses a composite primary key consisting of the partial key (VicTypeID) from the weak entity set (***Vicim_Type***) and the primary key (SusTypeID) in the identifying entity set (***Suspect_Type***). Additionally, we applied the ON DELETE CASCADE and ON UPDATE CASCADE options to ensure that any deletion or update of the identifying entity (***Suspect_Type***) is propagated to the corresponding weak entities. For the ternary relationship ***Occured***, which connects the crime record (***Incident***), temporal and geographical crime details (***Crime_Scene***), and individual details (***Victims*** and ***Suspects***), we utilized the aggregation represented in the ***Harm*** table to represent comprehensive information of individuals involved in the corresponding crimes. The ***Harm*** table uniquely identifies victim-suspect pairs by combining their respective keys. In the ***Occured*** table, we then linked all three tables: *Incident*, *Crime_Scene*, and ***Harm***. To enforce the key constraint between ***Occured*** and *Crime_Scene*, we defined the primary key of the ***Occured*** table as a combination of the keys from *Incident* and ***Harm***, while setting CRIME_SCENE_PID (the primary key of *Crime_Scene*) as NOT NULL. This ensures that every occurrence is tied to a valid crime scene, however, without using ASSERTIONs, it is not possible to capture the participation constraint between *Occured* and *Incident* tables in SQL.

With the notion that victims would be sent to the nearest available hospital when injured due to the crime incident, we added entity ***Hospital*** in our E/R diagram and mapped the relationship ***Near*** between the entity ***Hospital*** and *Crime_Scene*. Since there are multiple hospitals in New York, and thus multiple hospitals may exist near the location of crime incidents, we used the participation constraint. However, without ASSERTIONs, we cannot capture this participation constraint between ***Near*** and ***Hospital*** in SQL. The entity ***Hospital*** has attributes HospitalID, H_NAME, H_TYPE, H_PHONE, H_BOROUGH, H_Location representing Hospital ID, the name of the hospital, type of the hospital, phone number of the hospital, and the borough that the hospital is located at, respectively. For the Hospital entity, HospitalID is the primary key. We also made a check constraint to assure that the borough for the hospital is among the values of Manhattan, Bronx, Queens, Brooklyn, and Staten Island. For the relationship ***NEAR***, we have both CRIME_SCENE_PID, which is the primary key of the entity CRIME_SCEHE, and HospitalID, which is the primary key of the entity ***Hospital***, as the primary key.

We mapped the ISA Hierarchy of the entity *NYPD* by having the subtypes of *NYPD* which are two entities: ***Transit_Police*** and ***Precinct***. The two entities ***Transit_Police*** and ***Precinct*** do not overlap and provide the total coverage of *NYPD*. However, without ASSERTIONs, we cannot map these constraints of both No Overlap and Coverage in SQL. Since the two entities are lower-level entities of the higher level entity *NYPD*, they both have their primary keys inherited from the primary key of *NYPD* and they have referential actions 'ON DELETE CASCADE' and 'ON UPDATE CASCADE' to make sure that they also get updated or deleted when the parent table (*NYPD*) is updated or deleted. The attributes of the ***Transit_Police*** are TransitDistrict, Officer_Name, Officer_Phone, and Officer_Loc which represents the unique district number of Transit, name of the officer representing the transit district, phone number of the officer representing the district, and the location of the officer responsible for the transit district, respectively. For the entity ***Precinct***, its attributes NYPD_PCT_CD, PCT_Name, PCT_Address, and PCT_Phone represent the unique number of the precinct, the official name of the precinct, its address, and its phone number, respectively.

## SQL Schema

Existed Tables in Part 2 are as follows. For Part 3, we modified the *Monitor* relationship by dropping the JURIS_DESC attribute (the description of jurisdiction) since we represented detailed jurisdiction as hierarchical structures in *NYPD*.

```
CREATE TABLE Crime_Scene (
CRIME_SCENE_PID INTEGER,
LOC_OF_OCCUR_DESC CHAR(10),
PREM_TYP_DESC VARCHAR(30),
GEO_CD_X REAL CHECK (GEO_CD_X>0),
GEO_CD_Y REAL CHECK (GEO_CD_Y>0),
Longitude REAL CHECK (Longitude>=-75 AND Longitude<=-73),
Latitude REAL CHECK (Latitude>=40 AND Latitude<=41),
CMPLNT_FR_DT DATE,
CMPLNT_FR_TM TIME,
CMPLNT_TO_DT DATE,
CMPLNT_TO_TM TIME,
CHECK (CMPLNT_FR_DT<=CMPLNT_TO_DT),
PRIMARY KEY (CRIME_SCENE_PID) );

CREATE TABLE Incident (
INCIDENT_PID INTEGER,
CAD_EVNT_ID INTEGER,
CIP_JOBS CHAR(20),
LAW_CAT_CD CHAR(15),
TYP_DESC VARCHAR(100),
PRIMARY KEY (INCIDENT_PID));
```

```
CREATE TABLE Monitor (
NYPD_PID INTEGER,
INCIDENT_PID INTEGER,
JURIS_DESC CHAR(20),
CREATE_DATE TIMESTAMP,
ADDED_TIME TIMESTAMP,
CLOSING_TS TIMESTAMP,
CHECK (JURIS_DESC IN ('N.Y. POLICE DEPT','N.Y. TRANSIT POLICE')),
PRIMARY KEY (NYPD_PID, INCIDENT_PID),
FOREIGN KEY (NYPD_PID) REFERENCES NYPD (NYPD_PID),
FOREIGN KEY (INCIDENT_PID) REFERENCES Incident (INCIDENT_PID) );

ALTER TABLE Monitor
DROP COLUMN JURIS_DESC;

CREATE TABLE Radio_Code (
RADIO_CODE_PID INTEGER,
RADIO_CODE CHAR(5),
PRIMARY KEY (RADIO_CODE_PID) );

CREATE TABLE Send (
INCIDENT_PID INTEGER,
NYPD_PID INTEGER,
RADIO_CODE_PID INTEGER NOT NULL,
PRIMARY KEY (INCIDENT_PID, NYPD_PID),
FOREIGN KEY (INCIDENT_PID) REFERENCES Incident (INCIDENT_PID),
FOREIGN KEY (NYPD_PID) REFERENCES NYPD (NYPD_PID),
FOREIGN KEY (RADIO_CODE_PID) REFERENCES Radio_Code (RADIO_CODE_PID) );

CREATE TABLE Dispatch_Duration (
DISPATCH_DURATION_PID INTEGER,
DISP_TS TIMESTAMP,
ARRIVD_TS TIMESTAMP,
CHECK (DISP_TS <= ARRIVD_TS),
PRIMARY KEY (DISPATCH_DURATION_PID) );

CREATE TABLE Arrive (
CRIME_SCENE_PID INTEGER,
NYPD_PID INTEGER,
DISPATCH_DURATION_PID INTEGER,
PRIMARY KEY (CRIME_SCENE_PID, NYPD_PID, DISPATCH_DURATION_PID),
FOREIGN KEY (CRIME_SCENE_PID) REFERENCES Crime_Scene (CRIME_SCENE_PID),
FOREIGN KEY (NYPD_PID) REFERENCES NYPD (NYPD_PID),
```

```
FOREIGN KEY (DISPATCH_DURATION_PID) REFERENCES Dispatch_Duration
(DISPATCH_DURATION_PID) );
```

We wrote new SQL queries to map the expanded E/R Diagram, as follows:

```
CREATE TABLE NYPD (
NYPD_PID INTEGER,
PATRL_BORO_NM_ID INTEGER,
PATRL_BORO_NM CHAR(50),
BORO_NM CHAR(20) CHECK (BORO_NM IN
('MANHATTAN','BRONX','QUEENS','BROOKLYN','STATEN ISLAND')),
PRIMARY KEY (NYPD_PID) );

CREATE TABLE Transit_Police(
NYPD_PID INTEGER,
TransitDistrict INTEGER,
Officer_Name CHAR(50),
Officer_Phone CHAR(40),
Officer_Loc CHAR(200),
PRIMARY KEY (NYPD_PID),
FOREIGN KEY (NYPD_PID) REFERENCES NYPD (NYPD_PID)
ON DELETE CASCADE ON UPDATE CASCADE );

CREATE TABLE Precinct(
NYPD_PID INTEGER,
NYPD_PCT_CD INTEGER,
PCT_Name CHAR(40),
PCT_Address CHAR(50),
PCT_Phone CHAR(20),
PRIMARY KEY (NYPD_PID),
FOREIGN KEY (NYPD_PID) REFERENCES NYPD (NYPD_PID)
ON DELETE CASCADE ON UPDATE CASCADE );
```

```
CREATE TABLE Victim_Who_Harmed (
VicTypeID INTEGER,
VIC_RACE CHAR(40),
VIC_SEX CHAR(1),
SusTypeID INTEGER,
PRIMARY KEY (VicTypeID, SusTypeID),
FOREIGN KEY (SusTypeID) REFERENCES Suspect_Type(SusTypeID)
ON DELETE CASCADE ON UPDATE CASCADE );
```

```
CREATE TABLE Suspect_Type(
SusTypeID INTEGER,
SUS_AGE_GROUP CHAR(20),
SUS_RACE CHAR(40),
SUS_SEX CHAR(1),
PRIMARY KEY (SusTypeID) );

CREATE TABLE Harm (
VicTypeID INTEGER,
SusTypeID INTEGER,
PRIMARY KEY (VicTypeID, SusTypeID),
FOREIGN KEY (VicTypeID, SusTypeID) REFERENCES Victim_Who_Harmed(VicTypeID,
SusTypeID),
FOREIGN KEY (SusTypeID) REFERENCES Suspect_Type(SusTypeID) );

CREATE TABLE Court(
COURT_PID INTEGER,
COURT_TYPE CHAR(30),
COURT_DESC VARCHAR(300),
COURT_ADDRESS VARCHAR(70),
COURT_PHONE CHAR(15),
PRIMARY KEY (COURT_PID) );

CREATE TABLE Judged_Incident(
INCIDENT_PID INTEGER,
CAD_EVNT_ID INTEGER,
CIP_JOBS CHAR(20),
LAW_CAT_CD CHAR(15),
TYP_DESC VARCHAR(100),
COURT_PID INTEGER,
PRIMARY KEY (INCIDENT_PID),
FOREIGN KEY (COURT_PID) REFERENCES Court (COURT_PID) );

CREATE TABLE Hospital(
HospitalID INTEGER,
H_NAME CHAR(80),
H_TYPE CHAR(30),
H_PHONE CHAR(25),
H_BOROUGH CHAR(15),
H_Location CHAR(100),
CHECK (H_BOROUGH IN ('Manhattan','Bronx','Queens','Brooklyn','Staten
Island')),
PRIMARY KEY (HospitalID) );
```

```
CREATE TABLE NEAR (
CRIME_SCENE_PID INTEGER,
HospitalID INTEGER,
PRIMARY KEY (CRIME_SCENE_PID,HospitalID),
FOREIGN KEY (CRIME_SCENE_PID) REFERENCES Crime_Scene (CRIME_SCENE_PID),
FOREIGN KEY (HospitalID) REFERENCES Hospital (HospitalID) );

CREATE TABLE Occured(
INCIDENT_PID INTEGER,
VicTypeID INTEGER,
SusTypeID INTEGER,
CRIME_SCENE_PID INTEGER NOT NULL,
PRIMARY KEY (INCIDENT_PID, VicTypeID, SusTypeID),
FOREIGN KEY (INCIDENT_PID) REFERENCES Incident (INCIDENT_PID),
FOREIGN KEY (VicTypeID,SusTypeID) REFERENCES Harm (VicTypeID,SusTypeID),
FOREIGN KEY (CRIME_SCENE_PID) REFERENCES Crime_Scene (CRIME_SCENE_PID) );
```

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*.

## Three "interesting" SQL queries over our database

1. The percentage distribution of victims based on their race and gender

**[Query]**

```
SELECT
    VT.VIC_RACE AS Victim_Race,
    VT.VIC_SEX AS Victim_Gender,
    ROUND((COUNT(*) * 100.0) / SUM(COUNT(*)) OVER (), 2) AS Victim_Percentage
FROM Victim_Who_Harmed VT
JOIN Harm H ON VT.VicTypeID = H.VicTypeID
GROUP BY Victim_Race, Victim_Gender
ORDER BY Victim_Percentage DESC;
```

**[Output]**

```
              victim_race                | victim_gender | victim_percentage
-----------------------------------------+---------------+------------------
 WHITE HISPANIC                          | M             |             22.22
 BLACK                                   | M             |             11.11
 BLACK                                   | F             |             11.11
 AMERICAN INDIAN/ALASKAN NATIVE          | F             |             11.11
 BLACK HISPANIC                          | M             |             11.11
 WHITE HISPANIC                          | F             |             11.11
 WHITE                                   | M             |             11.11
 ASIAN / PACIFIC ISLANDER                | F             |             11.11
(8 rows)
```

**[Explanation]**

This query calculates the percentage distribution of victims based on their race and gender. It groups victims by their race (`VIC_RACE`) and gender (`VIC_SEX`) and determines what proportion each group represents out of the total number of victims. The results are sorted in descending order, showing the groups with the highest percentage first. This provides a concise view of victim demographic trends across the dataset. From the screen shot above, we could infer that the 'WHITE HISPANIC' and 'MALE' person was the most frequently targeted demographic to be a victim.

2. The Most Common Court and Crime Scene Pair for Judged Incidents

**[Query]**

```
WITH CourtCrimeSceneCount AS (
    SELECT C.COURT_TYPE AS Court, CS.PREM_TYP_DESC AS Crime_Scene,
           COUNT(J.INCIDENT_PID) AS Incident_Count
    FROM Court C
    JOIN Judged_Incident J ON C.COURT_PID = J.COURT_PID
    JOIN Crime_Scene CS ON J.INCIDENT_PID = CS.CRIME_SCENE_PID
    GROUP BY C.COURT_TYPE, CS.PREM_TYP_DESC
)
SELECT Court, Crime_Scene, MAX(Incident_Count) AS Max_Incident_Count
FROM CourtCrimeSceneCount
GROUP BY Court, Crime_Scene
ORDER BY Max_Incident_Count DESC;
```

**[Output]**

```
             court               |      crime_scene       | max_incident_count
---------------------------------+------------------------+--------------------
 The Supreme Criminal Court      | BANK                   |                866
 Criminal Court                  | TRANSIT - NYC SUBWAY   |                726
 The Supreme Criminal Court      | TRANSIT - NYC SUBWAY   |                397
 The Supreme Criminal Court      | STREET                 |                178
 Criminal Court                  | COMMERCIAL BUILDING    |                167
 The Supreme Criminal Court      | FOOD SUPERMARKET       |                154
 The Supreme Criminal Court      | GYM/FITNESS FACILITY   |                 82
 The Supreme Criminal Court      | RESIDENCE - APT. HOUSE |                 10
(8 rows)
```

**[Explanation]**

This query identifies the maximum incident count for each *Court-Crime_Scene* combination. To achieve this, we defined a new table by using the WITH clause, which calculates the number of *incidents* for each combination of court type (`COURT_TYPE`) and crime scene type (`PREM_TYP_DESC`).

The new table leverages JOIN operations to combine the *Court*, *Judged_Incident*, and *Crime_Scene* tables. These joins ensure a connection between court types, incidents, and their associated crime scene types, as well as the COUNT aggregate function is used to count the number of incidents for each unique court-crime scene combination, while the GROUP BY operation organizes the results by court type and crime scene type.

In the second part of the query, the MAX function identifies the highest incident count for each court-crime scene pair, and the GROUP BY clause ensures that these counts are grouped correctly. The ORDER BY Max_Incident_Count DESC clause sorts the results in descending order, highlighting the combinations with the highest incident counts at the top.

3. The Fastest and Slowest Officers

**[Query]**

```
SELECT Officer_Name, Fastest_Response_Time_Minutes
FROM (
    SELECT
        COALESCE(TP.Officer_Name, P.PCT_Name) AS Officer_Name,
        MIN(EXTRACT(EPOCH FROM DD.ARRIVD_TS - DD.DISP_TS) / 60) AS
Fastest_Response_Time_Minutes
    FROM Arrive A
    JOIN Dispatch_Duration DD ON A.DISPATCH_DURATION_PID = DD.DISPATCH_DURATION_PID
    LEFT JOIN Transit_Police TP ON A.NYPD_PID = TP.NYPD_PID
    LEFT JOIN Precinct P ON A.NYPD_PID = P.NYPD_PID
    GROUP BY COALESCE(TP.Officer_Name, P.PCT_Name)
    ORDER BY Fastest_Response_Time_Minutes ASC LIMIT 1
) AS Fastest

UNION ALL

SELECT Officer_Name, Slowest_Response_Time_Minutes
FROM (
    SELECT
        COALESCE(TP.Officer_Name, P.PCT_Name) AS Officer_Name,
        MAX(EXTRACT(EPOCH FROM DD.ARRIVD_TS - DD.DISP_TS) / 60) AS
Slowest_Response_Time_Minutes
    FROM Arrive A
    JOIN Dispatch_Duration DD ON A.DISPATCH_DURATION_PID = DD.DISPATCH_DURATION_PID
```

```
    LEFT JOIN Transit_Police TP ON A.NYPD_PID = TP.NYPD_PID
    LEFT JOIN Precinct P ON A.NYPD_PID = P.NYPD_PID
    GROUP BY COALESCE(TP.Officer_Name, P.PCT_Name)
    ORDER BY Slowest_Response_Time_Minutes DESC LIMIT 1
) AS Slowest;
```

**[Output]**

```
                officer_name                 | fastest_response_time_minutes
---------------------------------------------+-------------------------------
 Captain Robert Rios                         |        0.00000000000000000000
 Deputy Inspector Eric S. Sandseth           |          529.9166666666666667
(2 rows)
```

**[Explanation]**

This SQL query is designed to identify and return the officer or precinct with the fastest and slowest response times to crime scenes based on dispatch and arrival times. The goal is to compute the response time for officers (or precincts if no officer is assigned) and determine the fastest and slowest based on that time. From the screenshot with the output above, we could know that there is a nearly 8 hour gap between the fastest officer and the slowest officer.

## References of data


- [NYPD Calls for Service (Year to Date)]
- [NYPD Complaint Data Historic]
- [Hospitals in NYC]
- [Transit (NYPD)]
- [Precinct (NYPD)]
- [Court]