
1: Defining the Relation Schema

Banks (banks_25.data)	**RECTANGLE**

```
CREATE TYPE SecurityLevel AS ENUM ('weak', 'good', 'very good', 'excellent');

CREATE TABLE Banks (
   BankName VARCHAR(50),
   City VARCHAR(50),
   NoAccounts INT NOT NULL CHECK (NoAccounts >= 0) DEFAULT 0,
   Security SecurityLevel NOT NULL,
   PRIMARY KEY (BankName, City)
);
```

- **Primary Key:** BankName and City go hand in hand. There can be many Banks of the same BankName and many Banks in the same City. However, the combination of BankName and City is strictly unique, making it a suitable choice as a Primary Key.
- **Security Attribute** is an ordinal categorical variable. Since it holds strictly weak, good, very good, and excellent, it would be suitable to set this as an ENUM. This saves the extra hassle of needing to declare a check constraint as ENUM provides this already.
- **NoAccounts Check**: The only necessary check is to ensure that NoAccounts is not negative as this makes no sense. Note that I allow NoAccounts to be 0 as this is possible in the event of a new Bank or under maintenance. We can complicate this by ensuring NoAccounts are greater than those referenced by robbers in hasaccounts. I believe this to be beyond scope but I mention it in case. Lastly there's no upper bound.
- **Not Null Constraint:** BankName and City are part of the PK so NOT NULL is applied automatically. NoAccounts and Security are NOT NULL as they provide crucial details.
- **Defaults**: NoAccounts could be 0 by default. Security is a different story as while weak stands out as a default, it seems precarious to assume security, thus I have left it blank
- **Data Types**: BankName and City both use VARCHAR, short for variable character. This means the field stores text of any length up to 50 characters. It saves space compared to CHAR(50) as CHAR uses a fixed amount of space via padding. While the BankNames and Cities are generally short, 50 is a reasonable and safe assumption.

Robberies (robberies_25.data) **DIAMOND**

```
CREATE TABLE Robberies (
   BankName VARCHAR(50),
   City VARCHAR(50),
   Date DATE,
   Amount DECIMAL(15, 2) NOT NULL CHECK (Amount >= 0) DEFAULT 0,
   PRIMARY KEY (BankName, City, Date),
   FOREIGN KEY (BankName, City) REFERENCES Banks(BankName, City)
      ON DELETE RESTRICT ON UPDATE CASCADE
);
```

- **Primary Key**: A Robbery can happen at the same BankName and same City. Adding Date adds extra specificity to the robbery, helping to identify them, and therefore will be a component of the Primary Key. Technically, it's possible for a robbery to target the same BankName, City, on the same Date, but I assume this never happens (unique).

- **Amount Attribute**: Amount is not an integer and is instead a decimal given that it can contain decimal places at most 2dp akin to currency. Likewise, the check >= 0 exists as it is impossible for amount to be negative. It can however, be 0, and I specify this as the default amount assuming an unsuccessful robbery.

- **Foreign Key**: The Robbery references the specific Bank it targets (BankName and City). While Date contributes to the Robbery PK, this doesn't make it an FK.

- **On Delete/Update:** If we remove a Bank, should the robbery for that bank also be deleted? Well no because robberies are historic events, a record should still be kept. Therefore, I argue ON DELETE RESTRICT is the better option. This prevents the deletion of banks altogether. I believe ON UPDATE CASCADE is alright as a name change is possible for a bank in which case the Robbery details should reflect this.

- **Not Null**: BankName, City, and Date are part of the PK so NOT NULL is automatically applied. Amount is given NOT NULL as it is an key attribute describing the robbery.

- **Data Types**: BankName and City both use VARCHAR, short for variable character. This means the field stores text of any length up to 50 characters. It saves space compared to CHAR(50) as CHAR uses a fixed amount of space via padding. While the BankNames and Cities are generally short, 50 is a reasonable and safe assumption. Date uses DATE as it only stores the year, month, and day, which matches our data.

Plans (plans_25.data) **DIAMOND**

```
CREATE TABLE Plans (
   BankName VARCHAR(50),
   City VARCHAR(50),
   PlannedDate DATE,
   NoRobbers INT NOT NULL CHECK (NoRobbers > 0) DEFAULT 1,
   PRIMARY KEY (BankName, City, PlannedDate),
   FOREIGN KEY (BankName, City) REFERENCES Banks(BankName, City)
      ON DELETE CASCADE ON UPDATE CASCADE
);
```

- **Primary Key:** Similar to Robberies, a plan can exist for the same BankName and City. The addition of PlannedDate adds the necessary uniqueness to complete the primary key. Again, I assume >1 robberies for the same Bank in the same day is not possible.

- **NoRobbers** Attribute: NoRobbers is an integer as this value is often quite small e.g. 6. It makes no sense for a plan to exist and there be no robbers in attendance, therefore a check constraint is used to ensure greater than 0 (no negatives also). Similarly it is fair to set the default NoRobbers as 1. It's in red to highlight it's not really critical to add

- **Not Null**: BankName, City, and PlannedDate are part of the primary key so NOT NULL is automatically applied. NoRobbers is Not Null as this is key info regarding the plan.

- **Foreign Key**: Similar to Robberies, Plans reference the Bank that is being targeted, hence (BankName and City). This matches the Primary Key used by the Banks table.

- **On Delete/Update**: Unlike Robberies, if a Bank gets deleted, it makes perfect sense that the plans to rob that bank should also be deleted. Thus ON DELETE CASCADE is appropriate. Likewise, if the bank is under a different identifier, the planned robbery can still occur in which case the plan details referencing the bank are simply updated.

- **Data Types**: BankName and City both use VARCHAR, short for variable character. This means the field stores text of any length up to 50 characters. It saves space compared to CHAR(50) as CHAR uses a fixed amount of space via padding. While the BankNames and Cities are generally short, 50 is a reasonable and safe assumption. PlannedDate uses DATE as it stores year, month, and day, which matches our data.

Robbers (robbers_25.data) **RECTANGLE**

```
CREATE TABLE Robbers (
   RobberId SERIAL PRIMARY KEY,
   Nickname VARCHAR(50) NOT NULL,
   Age INT NOT NULL CHECK (Age >= 0 AND Age <= 150),
   NoYears INT NOT NULL CHECK (NoYears >= 0 AND NoYears <= Age) DEFAULT 0
);
```

- **Observation**: Bucky Malone and Lucky Luchiano, likewise Grasy Guzik and Lepke Buchalter have the same age and NoYears, it is possible they are the same person, it is also possible they are partners, same age, and starting time. (However, see below)

- **Primary Key**: How are we to determine an identifier if a robber can be under different aliases? Well, it says in the brief that we can safely assume this has not happened yet. All Robber aliases are uniquely separate people. In future however, it is not the case. Thus, a new variable RobberID is to be added using the Serial Type for uniqueness.

- **Age Attribute**: This is an integer that has the check constraint >=0 to stop negatives while allowing 0 as months are a valid age. It also has an upper limit based on lifespan

- **NoYears Attribute**: This is also an integer that has an important check constraint to ensure that experience is not greater than age besides the basic not negative check. Note that a robber can rob for the first time. This does not mean they have 1 year of experience, therefore 0 is definitely a possible outcome and is thus the default value.

- **Not Null Constraint**: RobberID will be auto generated by PostgreSQL and as it is a PK, NOT NULL is automatically applied. Attributes Nickname, Age, and NoYears are NOT NULL as they are essential information that describe the robbers.

- **Data Types**: Nickname uses VARCHAR, short for variable character. This means the field stores text of any length up to 50 characters. It saves space compared to CHAR(50) as CHAR uses a fixed amount of space via padding. While Nicknames are generally short, 50 is a reasonable and safe assumption in case Robbers get creative.

Skills (Created in 2) **RECTANGLE**

```
CREATE TABLE Skills (
   SkillId SERIAL PRIMARY KEY,
   Description VARCHAR(50) NOT NULL UNIQUE
);
```

- **Observation**: It seems that HasSkills skills can take a value from either Planning, Safe-Cracking, Preaching, Driving, Guarding, Explosives, Gun-Shooting, Lock-Picking, Scouting, Money Counting, Eating, and Cooking. This will become the Description

- **Primary Key**: Similar to Robbers, a new variable SkillId is to be added using Postgre. It will be auto generated following each row of this table. For uniqueness, the Serial data type will be used. Serial is a sequential integer starting at 1 for easy identification.

- **Unique Constraint**: Description is the first variable thus far to utilize the Unique constraint. This is because Description does not compose the Primary Key but must still be unique as this is a referenced table. Note that Robbers is also a referenced table but does not use Unique for Nickname. This is because Nicknames can be shared and there's Age and NoYears which provide a way to uniquely identify robbers.

- **Not Null**: You may think that Description being Unique automatically implies Not Null similar to Primary Key, but this is not the case. Not Null must still be declared as Null is allowed underneath Unique. Description must be Not Null as it is a critical attribute.

- **Data Types**: Description uses VARCHAR, short for variable character. This means the field stores text of any length up to 50 characters. It saves space compared to CHAR(50) as CHAR uses a fixed amount of space via padding. While Descriptions are generally short, 50 is a reasonable and safe assumption in case of hyper-specific skills

HasSkills (hasskills_25.data) **DIAMOND**

```
CREATE TYPE GradeLevel AS ENUM ('C', 'C+', 'B', 'B+', 'A', 'A+');

CREATE TABLE HasSkills (
   RobberId SERIAL,
   SkillId SERIAL,
   Preference INT NOT NULL CHECK (Preference IN (1, 2, 3)) DEFAULT 1,
   Grade GradeLevel NOT NULL,
   PRIMARY KEY (RobberId, SkillId),
   FOREIGN KEY (RobberId) REFERENCES Robbers(RobberId)
      ON DELETE RESTRICT ON UPDATE CASCADE,
   FOREIGN KEY (SkillId) REFERENCES Skills(SkillId)
      ON DELETE RESTRICT ON UPDATE CASCADE
);
```

- **Primary Key**: A robber can have many skills, therefore RobberID cannot solely be PK. Likewise, many robbers can have the same skill, therefore skill alone cannot be PK. As a result, the primary is composed of both RobberID and SkillID (a joining table).

- **Preference Attribute**: This takes values which must be either 1, 2, or 3. While ENUM is possible, I experimented using an INT data type with a Check Constraint to ensure the value is in the set (1, 2, 3). We can overcomplicate this by ensuring a preference of 3 cannot exist if the same RobberID has not listed a preference of 2 and 1. I assume this is beyond scope and this feature is not required or tested in later questions. As a basic implementation I have set the default value as 1 to prevent immediate 2 or 3.

- **Grade Attribute**: This is an ordinal categorical variable consisting of values that take either C, C+, B, B+, A, or A+ and as a result, ENUM is an appropriate data type. Similar to Security for Banks, deciding a default value is precarious to assume. Therefore, unlike Preference, I have not given Grade a default value.

- **Foreign Key:** RobberID references the RobberID found in robbers. Likewise SkillID will reference the SkillID found in the newly generated Skills table within Question 2.

- **On Delete/Update**: This is up to interpretation. I lean more on the side similar to Robberies where this is historic data in which Hasskills should not be deleted. By extension, Robber and Skill should not be deleted under ON DELETE RESTRICT. Of course if a Robber didn't exist, their hasskill should also not exist. But a robber is a valuable piece of information for our fellow police and thus should not be deleted. Note that Cascade is applicable for Update as changes to Robber/Skill should be reflected.

Note: One idea could be to make robberID, skillID, and Preference to be a unique identifier to prevent duplicate preferences from the same person.

HasAccounts (hasaccounts_25.data) **DIAMOND**

```
CREATE TABLE HasAccounts (
   RobberId SERIAL,
   BankName VARCHAR(50),
   City VARCHAR(50),
   PRIMARY KEY (RobberId, BankName, City),
   FOREIGN KEY (RobberId) REFERENCES Robbers(RobberId)
      ON DELETE CASCADE ON UPDATE CASCADE,
   FOREIGN KEY (BankName, City) REFERENCES Banks(BankName, City)
      ON DELETE CASCADE ON UPDATE CASCADE
);
```

- **Primary Key**: A robber can have many accounts in BankName and City, likewise the Same BankName and City can hold the account of many robbers. As a result, our PK consists of all three attributes (BankName, City, RobberID). Note that the current data files do not have RobberID. This will be established and ironed out in Question 2.

- **Foreign Key**: HasAccounts is akin to a linker table. RobberID references the RobberID in Robbers. BankName and City references those attributes in Banks.

- **On Delete/Update**: You no longer have an account if the bank no longer exists, similar for Robbers. Unlike Robberies, Accounts are a present situation, not historic, which means it is acceptable for ON DELETE CASCADE to be used. On a similar note, if the Bank or Robber were to be updated, the HasAccounts record should reflect this.

- **No Null:** BankName, City, and robberID compose the primary key so NOT NULL is automatically applied. These three attributes are also the only attributes in this relation

- **Data Types**: BankName and City both use VARCHAR, short for variable character. This means the field stores text of any length up to 50 characters. It saves space compared to CHAR(50) as CHAR uses a fixed amount of space via padding. While the BankNames and Cities are generally short, 50 is a reasonable and safe assumption.

Accomplices (accomplices_25.data) **DIAMOND**

```
CREATE TABLE Accomplices (
   RobberId SERIAL,
   BankName VARCHAR(50),
   City VARCHAR(50),
   Date DATE,
   Share DECIMAL(15, 2) NOT NULL CHECK (Share >= 0) DEFAULT 0,
   PRIMARY KEY (RobberId, BankName, City, Date),
   FOREIGN KEY (RobberId) REFERENCES Robbers(RobberId)
      ON DELETE RESTRICT ON UPDATE CASCADE,
   FOREIGN KEY (BankName, City, Date) REFERENCES Robberies(BankName, City, Date)
      ON DELETE RESTRICT ON UPDATE CASCADE
);
```

- **Primary Key:** Robberies can have many Robbers and a Robber can partake in many Robberies. As a result, the primary key constitutes the primary key of both Robberies and Robbers, hence (RobberID, BankName, City, Date).

- **Share Attribute:** Share is a Decimal variable as it contains at most 2 decimal places akin to that of currency. Share follows an identical structure as Robberies(Amount) in which it is possible one robber can have the entire share which means someone can have none of the share in which 0 is a possible option. Therefore 0 is a valid default.

- **Foreign Key:** The RobberID references the RobberID present in Robbers. BankName, City, and Date references these same attributes present in Robberies

- **On Delete/Update**: Like Robberies, this is also historic data, a past record of an accomplice for a robbery. ON DELETE RESTRICT is applicable here to prevent the deletion of an accomplice row and by extension the deletion of a Robber and Robbery. Oppositely, I still argue that ON UPDATE CASCADE is possible if a new identifier is used for either Robber or Robbery in which case the details here should be reflected.

- **Data Types**: BankName and City both use VARCHAR, short for variable character. This means the field stores text of any length up to 50 characters. It saves space compared to CHAR(50) as CHAR uses a fixed amount of space via padding. While the BankNames and Cities are generally short, 50 is a reasonable and safe assumption. Date uses DATE as it only stores the year, month, and day, which matches our data.

2: Populating the Database with Data

1. \copy Banks FROM banks_25.data
2. \copy Robberies FROM robberies_25.data
3. \copy Plans FROM plans_25.data
4. \copy Robbers(Nickname, Age, NoYears) FROM robbers_25.data

5. CREATE TEMP TABLE TempHasSkills (
    Nickname VARCHAR(50),
    Description VARCHAR(50),
    Preference INT NOT NULL CHECK (Preference IN (1, 2, 3)) DEFAULT 1,
    Grade GradeLevel NOT NULL
);
5. \copy TempHasSkills FROM hasskills_25.data

6. INSERT INTO Skills (Description)
SELECT DISTINCT Description FROM TempHasSkills;

7. INSERT INTO HasSkills (RobberId, SkillId, Preference, Grade)
SELECT r.RobberId, s.SkillId, t.Preference, t.Grade
FROM TempHasSkills t
JOIN Robbers r ON r.Nickname = t.Nickname
JOIN Skills s ON s.Description = t.Description;

8. CREATE TEMP TABLE TempHasAccounts (
    Nickname VARCHAR(50),
    BankName VARCHAR(50),
    City VARCHAR(50)
);
8. \copy TempHasAccounts FROM hasaccounts_25.data

9. INSERT INTO HasAccounts (RobberId, BankName, City)
SELECT r.RobberId, t.BankName, t.City
FROM TempHasAccounts t
JOIN Robbers r ON r.Nickname = t.Nickname;

10. CREATE TEMP TABLE TempAccomplices (
    Nickname VARCHAR(50),
    BankName VARCHAR(50),
    City VARCHAR(50),
    Date DATE,
    Share DECIMAL(15, 2)
);
10. \copy TempAccomplices FROM accomplices_25.data

11. INSERT INTO Accomplices (RobberId, BankName, City, Date, Share)
SELECT r.RobberId, t.BankName, t.City, t.Date, t.Share
FROM TempAccomplices t
JOIN Robbers r ON r.Nickname = t.Nickname;

Description Explaining/Justifying Above Data Conversion:

- The Skills relation does not have a data file and therefore needs to be generated using postgre. We do however, have the hasskills data file which holds all current skills in use.

- The hasskills data file, however, does not follow the intended relation schema as it does not have RobberID and SkillID. Therefore, a TempHasSkills is created to match and import the hasskills data file which uses Nickname and Description.
- Using the TempHasSkills, all distinct skill descriptions are inserted into the Skills relation. This action auto generates the SkillID, the same for RobberID when Robbers was copied

- HasSkills relation can now be inserted with appropriate data: Preference and Grade from TempHasSkills, RobberID by joining Robbers on matching Nickname, and SkillID by joining Skills on matching Description. This completes the data conversion for Hasskills.

- Next, the HasAccounts data file also fails to follow our relation schema as it does not use RobberID. Therefore TempHasAccounts is created to match and import our current data file. As Robbers is already created, HasAccounts can be inserted with BankName and City from TempHasAccount, and RobberID by joining Robbers on matching Nickname.

- Lastly, the Accomplices data file also fails to match the intended schema as it doesn't use RobberID. Therefore TempAccomplices is created to match and import the data file. Again, as Robbers is set up, Accomplices can be inserted with BankName, City, Date, and Share from TempAccomplices, and RobberID by joining Robbers on Nickname.

Description Explaining/Justifying Order:

- Banks must be done prior to Robberies and Plans as it holds BankName and City which these two relations reference. These are my first three which complete the banks cluster.
- Robbers can be done anytime at the start as it is an Entity relation which is independent of the Banks cluster. Importantly, it holds the RobberID which is needed by Hasskills, HasAccounts, and Accomplices, and thus needs to be completed prior to those relations.

- On a similar note, Skills must be completed to obtain SkillID needed for HasSkills. However, TempHasSkills must be before Skills in order to even create the skills relation. After this, the Hasskills relation can be done considering the Robbers table is also done.
- Technically, prior to skills/hasskills, HasAccounts and Accomplices could have been done as they only require Robbers. Instead I complete these now to satisfy the order as per the question 1 brief. Importantly Temp tables are created before the actual tables.

3: Checking the database

1. INSERT INTO Skills(SkillId, Description) VALUES (21, 'Driving');

ERROR: duplicate key value violates unique constraint "unique_description"
DETAIL: Key (description)=(Driving) already exists.

2. INSERT INTO Banks VALUES ('Loanshark Bank', 'Evanston', 100, 'very good');

ERROR: duplicate key value violates unique constraint "banks_pkey"
DETAIL: Key (bankname, city)=(Loanshark Bank, Evanston) already exists.

INSERT INTO Banks VALUES ('EasyLoan Bank', 'Evanston', -5, 'excellent');

ERROR: new row for relation "banks" violates check constraint "banks_noaccounts check"
DETAIL: Failing row contains (EasyLoan Bank, Evanston, -5, excellent).

INSERT INTO Banks VALUES ('EasyLoan Bank', 'Evanston', 100, 'poor');

ERROR: invalid input value for enum securitylevel: "poor"
LINE 1: ...NTO Banks VALUES ('EasyLoan Bank', 'Evanston', 100, 'poor');

3. INSERT INTO Robberies VALUES ('NXP Bank', 'Chicago', '2019-01-08', 1000);

ERROR: duplicate key value violates unique constraint "robberies_pkey"
DETAIL: Key (bankname, city, date)=(NXP Bank, Chicago, 2019-01-08) already exists.

4. DELETE FROM Skills WHERE SkillId = 1 AND Description = 'Driving';
SkillID = 1 and Description = ‘Driving’ doesn’t exist. A successful query but nothing is deleted.

But if we replace**: DELETE FROM Skills WHERE SkillId = 1 AND Description = 'Explosives';**
ERROR: update or delete on table "skills" violates foreign key constraint "hasskills_skillid_fkey" on table "hasskills" DETAIL: Key (skillid)=(1) is still referenced from table "hasskills".

5. DELETE FROM Banks WHERE BankName = 'PickPocket Bank' AND City = 'Evanston' AND NoAccounts = 2000 AND Security = 'very good';

ERROR: update or delete on table "banks" violates foreign key constraint "robberies bankname_city_fkey" on table "robberies" DETAIL: Key (bankname, city)=(PickPocket Bank, Evanston) is still referenced from table "robberies".

6.DELETE FROM Robberies WHERE BankName = 'Loanshark Bank' AND City = 'Chicago';

ERROR: update or delete on table "robberies" violates foreign key constraint "accomplices bankname_city_date_fkey" on table "accomplices" DETAIL: Key (bankname, city, date)=(Loanshark Bank, Chicago, 2017-11-09) is still referenced from table "accomplices".
I make the assumption that Date & Amount are just not specified. It seems to work as intended.

7INSERT INTO Robbers(RobberId, Nickname, Age, NoYears) VALUES (1, 'Shotgun', 70, 0);

ERROR: duplicate key value violates unique constraint "robbers_pkey"
DETAIL: Key (robberid)=(1) already exists.

INSERT INTO Robbers(RobberId, Nickname, Age, NoYears) VALUES (333, 'Jail Mouse', 25, 35);

ERROR: new row for relation "robbers" violates check constraint "robbers_check"
DETAIL: Failing row contains (333, Jail Mouse, 25, 35). # NoYears is > Age

8. INSERT INTO HasSkills VALUES (1, 7, 1, 'A+');

ERROR: duplicate key value violates unique constraint "hasskills_pkey"
DETAIL: Key (robberid, skillid)=(1, 7) already exists.

INSERT INTO HasSkills VALUES (1, 2, 0, 'A');

ERROR: new row for relation "hasskills" violates check constraint "hasskills_preference_check"
DETAIL: Failing row contains (1, 2, 0, A). # Preference must be of either 1, 2, or 3

INSERT INTO HasSkills VALUES (333, 1, 1, 'B-');

ERROR: invalid input value for enum gradelevel: "B-"
LINE 1: INSERT INTO HasSkills VALUES (333, 1, 1, 'B-');

INSERT INTO HasSkills VALUES (3, 20, 3, 'B+');

ERROR: insert or update on table "hasskills" violates foreign key constraint "hasskills_skillid fkey" DETAIL: Key (skillid)=(20) is not present in table "skills".

9. DELETE FROM Robbers
WHERE RobberId = 1 AND Nickname = 'Al Capone' AND Age = 31 AND NoYears = 2;

ERROR: update or delete on table "robbers" violates foreign key constraint "hasskills_robberid fkey" on table "hasskills" DETAIL: Key (robberid)=(1) is still referenced from table "hasskills".

4: Simple Database Queries

1. Retrieve BankName and City of all banks that have never been robbed. [4 marks]

```
SELECT BankName, City
FROM Banks
WHERE BankName NOT IN (SELECT DISTINCT BankName FROM Robberies);
```

```
    bankname     |   city
-----------------+----------
 Bankrupt Bank   | Evanston
 Hidden Treasure | Chicago
 Outside Bank    | Chicago
(3 rows)
```

2. Retrieve RobberId, Nickname and the Number of Years not spent in prison for all robbers who spent more than half of their life in prison. [4 marks]

```
SELECT RobberID, Nickname, (Age - NoYears) AS YearsNotPrison
FROM Robbers
WHERE NoYears > Age / 2;
```

```
 robberid |   nickname    | yearsnotprison
----------+---------------+----------------
        6 | Tony Genovese |             12
       16 | King Solomon  |             31
(2 rows)
```

3. Retrieve RobberId, Nickname, Age, and all skill descriptions of all robbers who are not younger than 35 years. [4 marks]

```
SELECT r.RobberID, r.Nickname, r.Age, s.Description
FROM Hasskills AS h
JOIN Robbers AS r ON r.RobberID = h.RobberID
JOIN Skills AS s ON s.SkillID = h.SkillID
WHERE r.Age >= 35;
```

```
 robberid |    nickname    | age |  description
----------+----------------+-----+---------------
       18 | Vito Genovese  |  66 | Cooking
       17 | Bugsy Siegel   |  48 | Driving
        3 | Lucky Luchiano |  42 | Driving
        7 | Dutch Schulz   |  64 | Driving
       20 | Longy Zwillman |  35 | Driving
       18 | Vito Genovese  |  66 | Eating
       24 | Sonny Genovese |  39 | Explosives
```

```
 2 | Bugsy Malone    |  42 | Explosives
 4 | Anastazia       |  48 | Guarding
17 | Bugsy Siegel    |  48 | Guarding
 9 | Calamity Jane   |  44 | Gun-Shooting
 3 | Lucky Luchiano  |  42 | Lock-Picking
 7 | Dutch Schulz    |  64 | Lock-Picking
24 | Sonny Genovese  |  39 | Lock-Picking
19 | Mike Genovese   |  35 | Money Counting
15 | Boo Boo Hoff    |  54 | Planning
16 | King Solomon    |  74 | Planning
24 | Sonny Genovese  |  39 | Safe-Cracking
12 | Moe Dalitz      |  41 | Safe-Cracking
18 | Vito Genovese   |  66 | Scouting
(20 rows)
```

4. Retrieve BankName and city of all banks where Al Capone has an account. The answer should list every bank at most once. [4 marks]

SELECT DISTINCT h.BankName, h.City
FROM HasAccounts h
JOIN Robbers AS r ON r.RobberID = h.RobberID
WHERE r.Nickname = 'Al Capone';

```
    bankname     |   city
-----------------+----------
 Bad Bank        | Chicago
 Inter-Gang Bank | Evanston
 NXP Bank        | Chicago
(3 rows)
```

5. Retrieve RobberId, Nickname and individual total “earnings” of those robbers who have earned at least $50,000 by robbing banks. Sorted in decreasing total earnings. [4 marks]

SELECT a.RobberID, r.Nickname, SUM(a.Share) AS Earnings
FROM Accomplices AS a
JOIN Robbers AS r ON r.RobberID = a.RobberID
GROUP BY a.RobberId, r.Nickname
HAVING SUM(a.Share) >= 50000
ORDER BY Earnings DESC;

```
 robberid |     nickname      | earnings
----------+-------------------+----------
        5 | Mimmy The Mau Mau | 70000.00
       15 | Boo Boo Hoff      | 61447.61
       16 | King Solomon      | 59725.80
       17 | Bugsy Siegel      | 52601.10
(4 rows)
```

6. Retrieve the Description of all skills together with RobberId and NickName of all robbers who possess this skill. The answer should be ordered by skill description.

SELECT s.Description, h.RobberID, r.Nickname
FROM Hasskills AS h
JOIN Skills AS s ON s.SkillID = h.SkillID
JOIN Robbers AS r ON r.RobberID = h.RobberID
ORDER BY s.Description ASC;

description	robberid	nickname
Cooking	18	Vito Genovese
Driving	17	Bugsy Siegel
Driving	3	Lucky Luchiano
Driving	5	Mimmy The Mau Mau
Driving	23	Lepke Buchalter
Driving	7	Dutch Schulz
Driving	20	Longy Zwillman
Eating	6	Tony Genovese
Eating	18	Vito Genovese
Explosives	24	Sonny Genovese
Explosives	2	Bugsy Malone
Guarding	4	Anastazia
Guarding	17	Bugsy Siegel
Guarding	23	Lepke Buchalter
Gun-Shooting	9	Calamity Jane
Gun-Shooting	21	Waxey Gordon
Lock-Picking	8	Clyde
Lock-Picking	3	Lucky Luchiano
Lock-Picking	7	Dutch Schulz
Lock-Picking	22	Greasy Guzik
Lock-Picking	24	Sonny Genovese
Money Counting	13	Mickey Cohen
Money Counting	14	Kid Cann
Money Counting	19	Mike Genovese
Planning	15	Boo Boo Hoff
Planning	8	Clyde
Planning	5	Mimmy The Mau Mau
Planning	1	Al Capone
Planning	16	King Solomon
Preaching	22	Greasy Guzik
Preaching	10	Bonnie
Preaching	1	Al Capone
Safe-Cracking	1	Al Capone
Safe-Cracking	24	Sonny Genovese
Safe-Cracking	12	Moe Dalitz
Safe-Cracking	11	Meyer Lansky
Scouting	8	Clyde
Scouting	18	Vito Genovese

(38 rows)

5: Complex Database Queries

1. Retrieve RobberId, Nickname and individual total “earnings” of those robbers who participated in the robbery with the highest amount. Sorted in decreasing total earnings. [5]

STEPWISE

```
CREATE VIEW MaxRobberyAmount AS
SELECT MAX(Amount) AS MaxAmount
FROM Robberies;
```

```
 maxamount
-----------
  99000.80
(1 row)
```

```
CREATE VIEW MaxRobberyDetails AS
SELECT BankName, City, Date
FROM Robberies AS r
JOIN MaxRobberyAmount AS m ON m.MaxAmount = r.Amount;
```

```
    bankname     |   city    |    date
-----------------+-----------+------------
 Penny Pinchers | Evanston | 2016-08-30
(1 row)
```

```
CREATE VIEW MaxRobberyRobbers AS
SELECT r.RobberID, r.Nickname
FROM Accomplices AS a
JOIN Robbers AS r ON r.RobberId = a.RobberId
JOIN MaxRobberyDetails AS b ON b.BankName = a.BankName AND b.City = a.City AND
b.Date = a.Date;
```

```
 robberid |    nickname
----------+----------------
        3 | Lucky Luchiano
        4 | Anastazia
        8 | Clyde
       10 | Bonnie
       16 | King Solomon
       17 | Bugsy Siegel
(6 rows)
```

(Final Query Below)

```
SELECT r.RobberID, r.Nickname, SUM(a.Share) AS TotalEarnings
FROM Accomplices AS a
JOIN Robbers AS r ON r.RobberId = a.RobberId
JOIN MaxRobberyRobbers AS m ON m.RobberID = a.RobberID
GROUP BY r.RobberID, r.Nickname
ORDER BY TotalEarnings DESC;
```

robberid	nickname	totalearnings
16	King Solomon	59725.80
17	Bugsy Siegel	52601.10
3	Lucky Luchiano	42667.00
10	Bonnie	40085.00
4	Anastazia	39169.62
8	Clyde	31800.00

(6 rows)

SINGLE NESTED

```
SELECT a.RobberID, rb.Nickname, SUM(a.Share) AS TotalEarnings
FROM Accomplices AS a
JOIN Robbers AS rb ON rb.RobberID = a.RobberID
JOIN (
   SELECT a.RobberID
   FROM Accomplices AS a
   JOIN Robbers AS rb ON rb.RobberID = a.RobberID
   JOIN (
      SELECT BankName, City, Date
      FROM Robberies r
      WHERE r.Amount = (SELECT MAX(Amount) FROM Robberies)
   ) AS b ON b.BankName = a.BankName AND b.City = a.City AND b.Date = a.Date
) AS r ON r.RobberID = a.RobberID
GROUP BY a.RobberID, rb.Nickname
ORDER BY TotalEarnings DESC;
```

robberid	nickname	totalearnings
16	King Solomon	59725.80
17	Bugsy Siegel	52601.10
3	Lucky Luchiano	42667.00
10	Bonnie	40085.00
4	Anastazia	39169.62
8	Clyde	31800.00

(6 rows)

2. Retrieve RobberId, Nickname, and Description of the first preferred skill of all robbers who have two or more skills. [5 marks]

STEPWISE

```
CREATE VIEW RobbersWithSkills AS
SELECT RobberID
FROM HasSkills
GROUP BY RobberID
HAVING COUNT(*) >= 2;

SELECT r.RobberID, r.Nickname, s.Description
FROM HasSkills AS h
JOIN Robbers AS r ON r.RobberID = h.RobberID
JOIN Skills AS s ON s.SkillID = h.SkillID
JOIN RobbersWithSkills AS rs ON rs.RobberID = h.RobberID
WHERE h.Preference = 1;
```

```
 robberid
----------
       22
        3
       17
        5
        7
       24
        1
       18
       23
        8
(10 rows)
```

SINGLE NESTED

```
SELECT h.RobberID, r.Nickname, s.Description
FROM (SELECT * FROM HasSkills WHERE Preference = 1) AS h
JOIN Robbers AS r ON r.RobberID = h.RobberID
JOIN Skills AS s ON s.SkillID = h.SkillID
WHERE h.RobberID IN (
   SELECT RobberID
   FROM HasSkills
   GROUP BY RobberID
   HAVING COUNT(*) >= 2
);
```

```
 robberid |     nickname      | description
----------+-------------------+-------------
       17 | Bugsy Siegel      | Driving
       23 | Lepke Buchalter   | Driving
       24 | Sonny Genovese    | Explosives
        8 | Clyde             | Lock-Picking
        3 | Lucky Luchiano    | Lock-Picking
        7 | Dutch Schulz      | Lock-Picking
        5 | Mimmy The Mau Mau | Planning
        1 | Al Capone         | Planning
       22 | Greasy Guzik      | Preaching
       18 | Vito Genovese     | Scouting
(10 rows)
```

Note that the image above is from my main database. Prior to submitting, all queries were tested on a new database and the output for above was in sorted order by robberid.

3. Retrieve BankName and City of all banks that were not robbed in the year, in which there were robbery plans for that bank. [5 marks]

STEPWISE

```
CREATE VIEW RobberiesInPlanned AS
SELECT DISTINCT r.BankName, r.City, EXTRACT(YEAR FROM r.Date) AS Year
FROM Robberies AS r
JOIN Plans AS p ON p.BankName = r.BankName AND p.City = r.City
AND EXTRACT(YEAR FROM p.PlannedDate) = EXTRACT(YEAR FROM r.Date);
```

```
 bankname |  city   | year
----------+---------+------
 NXP Bank | Chicago | 2019
(1 row)
```

```
SELECT DISTINCT p.BankName, p.City
FROM  Plans AS p
LEFT JOIN RobberiesInPlanned AS r ON r.BankName = p.BankName AND r.City = p.City
AND r.Year = EXTRACT(YEAR FROM p.PlannedDate)
WHERE r.BankName IS NULL;
```

SINGLE NESTED

```
SELECT DISTINCT p.BankName, p.City
FROM Plans p
WHERE NOT EXISTS (
SELECT * FROM Robberies r
WHERE r.BankName = p.BankName AND r.City = p.City
AND EXTRACT(YEAR FROM r.Date) = EXTRACT(YEAR FROM p.PlannedDate)
);
```

```
    bankname     |   city
-----------------+-----------
 Bad Bank        | Chicago
 Gun Chase Bank  | Evanston
 Inter-Gang Bank | Evanston
 PickPocket Bank | Chicago
 Hidden Treasure | Chicago
 PickPocket Bank | Deerfield
 Loanshark Bank  | Deerfield
 Dollar Grabbers | Chicago
(8 rows)
```

4. Retrieve RobberId and Nickname of all robbers who never robbed the banks at which they have an account. [5 marks]

STEPWISE

```
CREATE VIEW RobbersRobAccount AS
SELECT DISTINCT h.RobberID
FROM HasAccounts AS h
JOIN Accomplices AS a ON a.RobberID = h.RobberID
AND a.BankName = h.BankName AND a.City = h.City;

SELECT DISTINCT rb.RobberID, rb.Nickname
FROM HasAccounts AS h
JOIN Robbers AS rb ON rb.RobberID = h.RobberID
WHERE h.RobberID NOT IN (
SELECT RobberID FROM RobbersRobAccount);
```

```
 robberid
----------
        1
        5
        8
       11
       17
       18
       20
       22
(8 rows)
```

SINGLE NESTED

```
SELECT DISTINCT h.RobberID, r.NickName
FROM HasAccounts AS h
JOIN Robbers AS r ON r.RobberID = h.RobberID
WHERE h.RobberID NOT IN (
SELECT DISTINCT h.RobberID
FROM HasAccounts AS h
JOIN Accomplices AS a ON a.RobberID = h.RobberID AND a.BankName = h.BankName AND
a.City = h.City);
```

```
 robberid |    nickname
----------+-----------------
       14 | Kid Cann
       13 | Mickey Cohen
       24 | Sonny Genovese
       19 | Mike Genovese
        2 | Bugsy Malone
       12 | Moe Dalitz
       21 | Waxey Gordon
        7 | Dutch Schulz
       15 | Boo Boo Hoff
        4 | Anastazia
        9 | Calamity Jane
        3 | Lucky Luchiano
       23 | Lepke Buchalter
(13 rows)
```

Note that the image above is from my main database. Prior to submitting, all queries were tested on a new database and the output for above was in sorted order by robberid.

6: Even more Database Queries

1. Query that finds the average share of all robberies in Chicago, and average share of robberies in the other city with the largest average share. Average share of a bank robbery can be determined based on the number of participating robbers. [8 marks]

STEPWISE

```
CREATE OR REPLACE VIEW RobberyShareAverage AS
SELECT r.BankName, r.City, r.Date, r.Amount, COUNT(*), (r.Amount / COUNT(*)) AS avg
FROM Robberies AS r
JOIN Accomplices AS a ON a.BankName = r.BankName AND a.City = r.City AND a.Date =
r.Date
GROUP BY r.BankName, r.City, r.Date;
```

```
    bankname     |   city   |    date    |  amount  | count |          avg
-----------------+----------+------------+----------+-------+-----------------------
 PickPocket Bank | Chicago  | 2015-09-21 |  2039.00 |     3 |  679.6666666666666667
 Loanshark Bank  | Evanston | 2016-04-20 | 20880.00 |     3 | 6960.0000000000000000
 Gun Chase Bank  | Evanston | 2016-04-30 | 18131.30 |     5 | 3626.2600000000000000
 Inter-Gang Bank | Evanston | 2017-03-13 | 92620.00 |     3 |    30873.333333333333
 Dollar Grabbers | Evanston | 2017-06-28 |  3580.00 |     2 | 1790.0000000000000000
 NXP Bank        | Chicago  | 2019-01-08 | 34302.30 |     6 | 5717.0500000000000000
 Inter-Gang Bank | Evanston | 2016-02-16 | 72620.00 |     6 |    12103.333333333333
 Penny Pinchers  | Evanston | 2016-08-30 | 99000.80 |     6 |    16500.133333333333
 Loanshark Bank  | Evanston | 2019-02-28 | 19990.00 |     4 | 4997.5000000000000000
 Bad Bank        | Chicago  | 2017-02-02 |  6020.00 |     2 | 3010.0000000000000000
 Penny Pinchers  | Chicago  | 2016-08-30 |   900.00 |     2 |  450.0000000000000000
 PickPocket Bank | Evanston | 2018-01-30 |   542.99 |     2 |  271.4950000000000000
 Inter-Gang Bank | Evanston | 2018-02-14 | 52619.00 |     6 | 8769.8333333333333333
 Penny Pinchers  | Evanston | 2019-05-30 | 13000.40 |     4 | 3250.1000000000000000
 Loanshark Bank  | Chicago  | 2019-03-30 | 21005.00 |     5 | 4201.0000000000000000
 PickPocket Bank | Evanston | 2016-03-30 |  2031.99 |     2 | 1015.9950000000000000
 Penny Pinchers  | Evanston | 2017-10-30 |  9000.50 |     3 | 3000.1666666666666667
 Loanshark Bank  | Evanston | 2017-04-20 | 10990.00 |     3 | 3663.3333333333333333
 Dollar Grabbers | Evanston | 2017-11-08 |  4380.00 |     2 | 2190.0000000000000000
 PickPocket Bank | Chicago  | 2018-02-28 |   239.00 |     2 |  119.5000000000000000
 Loanshark Bank  | Chicago  | 2017-11-09 | 41000.00 |     5 | 8200.0000000000000000
(21 rows)
```

```
SELECT City, ROUND(AVG(avg), 2) AS Average
FROM RobberyShareAverage
GROUP BY City;
```

SINGLE NESTED

```
SELECT r.City, ROUND(AVG(r.avg), 2) AS Average
FROM (SELECT r.BankName, r.City, r.Date, r.Amount, COUNT(*),
(r.Amount / COUNT(*)) AS avg
FROM Robberies AS r
JOIN Accomplices AS a ON a.BankName = r.BankName AND
a.City = r.City AND a.Date = r.Date
GROUP BY r.BankName, r.City, r.Date) AS r
GROUP BY r.City;
```

```
   city   | average
----------+---------
 Evanston | 7072.25
 Chicago  | 3196.75
(2 rows)
```

2. The police department wants to know whether bank branches with lower security levels are more attractive to robbers than those with higher security levels.

Query to retrieve Security level, total Number of robberies that occurred in bank branches of that security level, and the average Amount of money that was stolen during these robberies. [8]

STEPWISE

```
CREATE VIEW RobberiesWithSecurity AS
SELECT b.Security, r.Amount
FROM Robberies r
JOIN Banks b ON r.BankName = b.BankName
AND r.City = b.City;

SELECT Security, COUNT(*),
ROUND((SUM(Amount) / COUNT(*)), 2) AS Average
FROM RobberiesWithSecurity
GROUP BY Security;
```

```
 security  |  amount
-----------+----------
 very good | 34302.30
 excellent | 19990.00
 excellent | 21005.00
 excellent | 52619.00
 weak      |   900.00
 excellent | 99000.80
 excellent | 18131.30
 very good |  2031.99
 weak      |   239.00
 excellent | 10990.00
 excellent | 72620.00
 excellent |  9000.50
 very good |   542.99
 excellent | 41000.00
 excellent | 13000.40
 weak      |  2039.00
 excellent | 20880.00
 excellent | 92620.00
 good      |  4380.00
 good      |  3580.00
 weak      |  6020.00
(21 rows)
```

SINGLE NESTED

```
SELECT b.security, COUNT(*),
ROUND(AVG(Amount), 2) AS Average
FROM Robberies AS r
JOIN Banks AS b ON b.BankName = r.BankName
AND b.City = r.City
GROUP BY b.security;
```

```
 security  | count | average
-----------+-------+----------
 good      |     2 |  3980.00
 very good |     3 | 12292.43
 weak      |     4 |  2299.50
 excellent |    12 | 39238.08
(4 rows)
```