Project 1

Question 1: Defining the Database

1. **A list of the primary keys and foreign keys for each relation, along with a brief justification for your choice of keys and foreign keys.**

Banks

Primary: {BankName, City}

The banks are specified by the name of the bank and the city where the branch is located in.

Foreign: No foreign keys needed.

Robberies

Primary: {BankName, City, Date}

Assuming that no bank would be robbed more than once a day, the pair of BankName and City determines a bank.

Foreign: {BankName, City} -> Banks[{BankName, City}]

BankName and City are related to a specified bank in Banks table.

Plans

Primary: {BankName, City, PlannedDate}

The plans are specified by the name of the bank, the city where the branch is located in and the planned date. No branch will be planned to rob more than once on the same day.

Foreign: {BankName, City} -> Banks[{BankName, City}]

BankName and City are related to a specified bank in Banks table.

Robbers

Primary: {RobberID}

The RobberID is unique for each robber. Some robbers may have the same NickNames, but every Robber has their unique IDs.

Foreign: No foreign keys needed.

Skills

Primary: {SkillID}

The skillID is unique for each skill.

Foreign: No foreign keys needed.

HasSkills

Primary: {RobberID, SkillID}

The tuples in HasSkills table are specified by the RobberID and the SkillID. One robber cannot have duplicate skills.

Foreign: {RobberID} -> Robbers[{RobberID}]

{SkillID} -> Skills[{SkillID}]

One RobberID is related to an unique tuple in Robbers. Same for SkillID.

HasAccounts

Primary: {RobberID, BankName, City}

The tuples in HasAccount table are specified by the RobberID, the name of the bank and the location. One robber can only have one account in a bank branch.

Foreign: {RobberID} > Robbers[{RobberID}]

{BankName, City} > Banks[{BankName, City}]

One RobberID is related to an unique tuple in Robbers.

Accomplices

Primary: {RobberID, BankName, City, RobberyDate}

The tuples in Accomplices table are specified by the RobberID, the name of the bank, the location and the robbery date. One robber can only rob a bank branch at most once on a same day.

Foreign: {RobberID} > Robbers[{RobberID}]

{BankName, City} > Banks[{BankName, City}]

One RobberID is related to an unique tuple in Robbers.

2. A list of all your CREATE TABLE statements.

```
/* Banks */
CREATE TABLE Banks
(
    BankName varchar(255),
    City varchar(255),
    NoAccounts integer NOT NULL,
    Security varchar(255) NOT NULL,
    PRIMARY KEY (BankName, City),
    CONSTRAINT non_negative CHECK (NoAccounts >= 0),
    CONSTRAINT city_range CHECK (City IN ('Chicago', 'Evanston', 'Deerfield')),
    CONSTRAINT security_range CHECK (Security IN ('excellent', 'very good', 'good', 'weak'))
);
```

```
/* Robberies */
CREATE TABLE Robberies
(
    BankName varchar(255),
    City varchar(255),
    Date date,
    Amount numeric,
    PRIMARY KEY (BankName, City, Date),
    FOREIGN KEY (BankName, City) REFERENCES Banks(BankName, City),
    CONSTRAINT non_negative CHECK (Amount >= 0)
);

/* Plans */
CREATE TABLE Plans
(
    BankName varchar(255),
    City varchar(255),
    NoRobbers integer NOT NULL,
    PlannedDate date,
    PRIMARY KEY (BankName, City, PlannedDate),
    FOREIGN KEY    (BankName, City) REFERENCES Banks(BankName, City),
    CONSTRAINT non_negative CHECK (NoRobbers >= 0)
);

/* Robbers */
CREATE TABLE Robbers
(
    RobberID serial,
    NickName varchar(255) NOT NULL,
    Age integer,
    NoYears integer,
    PRIMARY KEY (RobberID),
    CONSTRAINT non_negative CHECK (Age >= 0),
    CONSTRAINT NYs_not_larger_than_Age CHECK (NoYears <= Age)
);

/* Skills */
CREATE TABLE Skills
(
    SkillID serial,
    Description varchar(255) NOT NULL UNIQUE,
    PRIMARY KEY (SkillID)
);
```

/* **HasSkills** */
```
CREATE TABLE HasSkills
(
    RobberID integer,
    SkillID integer,
    Preference integer NOT NULL,
    Grade varchar(2) NOT NULL,
    PRIMARY KEY (RobberID, SkillID),
    FOREIGN KEY (RobberID) REFERENCES Robbers(RobberID),
    FOREIGN KEY (SkillID) REFERENCES Skills(SkillID),
    CONSTRAINT preference_range CHECK (Preference >= 1 AND Preference <= 3),
    CONSTRAINT grade_range CHECK (Grade ~ '[A-C][\+\-]?')
);
```

/* **HasAccounts** */
```
CREATE TABLE HasAccounts
(
    RobberID integer,
    BankName varchar(255),
    City varchar(255),
    PRIMARY KEY    (RobberID, BankName, City),
    FOREIGN KEY (RobberID) REFERENCES Robbers(RobberID),
    FOREIGN KEY (BankName, City) REFERENCES Banks(BankName, City)
);
```

/* **Accomplices** */
```
CREATE TABLE Accomplices
(
    RobberID integer,
    BankName varchar(255),
    City varchar(255),
    RobberyDate date,
    Share numeric,
    PRIMARY KEY    (RobberID, BankName, City, RobberyDate),
    FOREIGN KEY (RobberID) REFERENCES Robbers(RobberID),
    FOREIGN KEY (BankName, City) REFERENCES Banks(BankName, City),
    CONSTRAINT non_negative CHECK (Share >= 0)
);
```

3. A justification for your choice of actions on delete or on update for each foreign key.

Robberies
{BankName, City} -> Banks[{BankName, City}]
ON DELETE:
The bank branch cannot be deleted from Banks table if there are any robberies related to the branch.
ON UPDATE CASCADE:
If the bank branch makes some changes to its name or location, the corresponding robbery records should be changed as well.

Plans
{BankName, City} -> Banks[{BankName, City}]
ON DELETE CASCADE:
If a bank branch is deleted from Banks table, then the planned robbery of that branch should also be deleted since the branch is not there any more.
ON UPDATE CASCADE:
If the bank branch makes some changes to its name or location, the corresponding planned robbery records should be changed as well.

HasSkills
1. {RobberID} -> Robbers[{RobberID}]
ON DELETE CASCADE:
If a robber tuple is deleted from Robbers table, then the skills the robber posses should also be deleted since the robber is not there any more.
ON UPDATE CASCADE:
If there are some changes in a RobberID in Robbers table, the corresponding HasSkills tuples should be changed as well.
2. {SkillID} -> Skills[{SkillID}]
ON DELETE:
The skill cannot be deleted from Skills table if there are any robbers currently have that skill.
ON UPDATE CASCADE:
If there are some changes in a SkillID in Skills table, the corresponding HasSkills tuples should be changed as well.

HasAccounts
1. {RobberID} -> Robbers[{RobberID}]
ON DELETE CASCADE:
If a robber tuple is deleted from Robbers table, then the accounts the robber posses should also be deleted since the robber is not there any more.
ON UPDATE CASCADE:
If there are some changes in a RobberID in Robbers table, the corresponding HasAccount tuples should be changed as well.

2. {BankName, City} -> Banks[{BankName, City}]
ON DELETE:
The bank branch cannot be deleted from Banks table if there are any accounts related to the branch.
ON UPDATE CASCADE:
If the bank branch makes some changes to its name or location, the corresponding HasAccount records should be changed as well.

Accomplices
1. {RobberID} -> Robbers[{RobberID}]
ON DELETE:
A robber tuple cannot be deleted from Robbers table if there are any accomplish tuples related to that robber.
ON UPDATE CASCADE:
If there are some changes in a RobberID in Robbers table, the corresponding accomplish tuples should be changed as well.
2. {BankName, City} -> Banks[{BankName, City}]
ON DELETE NO ACTION:
A robbery tuple cannot be deleted from Robberies table if there are any accomplish tuples related to that robbery.
ON UPDATE CASCADE:
If the bank branch makes some changes to its name or location or the robbery date has been changed, the corresponding accomplish records should be changed as well.

4. A brief justification for your choice of attribute constraints (other than the basic data).

Banks
Number of accounts should be greater or equal to 0, should not be null.
City should be in range (Chicago, Evanston, Deerfield).
Security should be in range (excellent, very good, good, weak).

Robberies
Amount should be greater or equal to 0.

Plans
Number of robbers should be greater than 0.

Robbers
Age should be greater or equal to 0.
NoYears should be greater or equal to 0.
Age should be greater than NoYears. No one can be in prison longer than the age of that person.

Skills
Description should not be null.
Description should be unique. One skill can not related to more than one SkillID.

HasSkills
Preference should not be null.
Preference should be greater or equal to 1, and less or equal than 3.
grade should be in range (A+, A, A-, B+, B, B-, C+, C, C-)

Accomplices
Share should be greater or equal to 0.

QUESTION 2: Populating your Database with Data

1. **A description of how you performed all the data conversion, for example, a sequence of the PostgreSQL statements that accomplished the conversion.**

```
/*insert into banks*/
\copy Banks FROM data/banks_16.data

/*insert into robberis*/
\copy Robberies FROM data/robberies_16.data

/*insert into plans*/
\copy Plans(BankName, City, PlannedDate, NoRobbers) FROM data/plans_16.data

/*insert into robbers*/
\copy Robbers(NickName, Age, NoYears) FROM data/robbers_16.data

/*create a temporary hasSkills table*/
CREATE TABLE TempHasSkills
(
    NickName varchar(255) NOT NULL,
    Skill varchar(255) NOT NULL,
    Preference integer NOT NULL,
    Grade varchar(2) NOT NULL,
    primary key (NickName, Skill)
);

/*insert hasskills.data into TempHasSkills*/
\copy TempHasSkills FROM data/hasskills_16.data
```

```
/*insert into Skills table using TempHasSkills table*/
INSERT INTO Skills (description)
SELECT DISTINCT Skill
FROM TempHasSkills;

/*insert into HasSkills table=*/
INSERT INTO HasSkills
SELECT RobberId, SkillId, Preference, Grade
FROM TempHasSkills
JOIN Robbers ON TempHasSkills.NickName = Robbers.NickName
JOIN Skills ON TempHasSkills.skill = Skills.description;

/*Drop the temp table*/
DROP TABLE TempHasSkills;

/*create a temporary hasAccounts table*/
CREATE TABLE TempHasAccounts
(
    NickName varchar(255) NOT NULL,
    BankName varchar(255) NOT NULL,
    City varchar(255) NOT NULL,
    primary key (NickName, BankName, City)
);

/*insert hasaccounts.data into TempHasAccounts*/
\copy TempHasAccounts FROM data/hasaccounts_16.data

/*insert into HasAccounts table*/
INSERT INTO HasAccounts
SELECT RobberId, BankName, City
FROM TempHasAccounts
JOIN Robbers ON TempHasAccounts.NickName = Robbers.NickName;

/*Drop the temp table*/
DROP TABLE TempHasAccounts;

/*create a temporary Accomplices table*/
CREATE TABLE TempAccomplices
(
    NickName varchar(255) NOT NULL,
    BankName    varchar(255) NOT NULL,
    City        varchar(255) NOT NULL,
    RobberyDate  date NOT NULL,
    Share       real,
    primary key (NickName, BankName, City, RobberyDate)
);
```

*/*insert accomplices.data into TempAccomplices*/*

```
\copy TempAccomplices FROM data/accomplices_16.data
```

*/*insert into Accomplices table*/*

```
INSERT INTO Accomplices
SELECT RobberID, BankName, City, RobberyDate, Share
FROM TempAccomplices
JOIN Robbers ON TempAccomplices.NickName = Robbers.NickName;
```

*/*Drop the temp table*/*

```
DROP TABLE TempAccomplices;
```

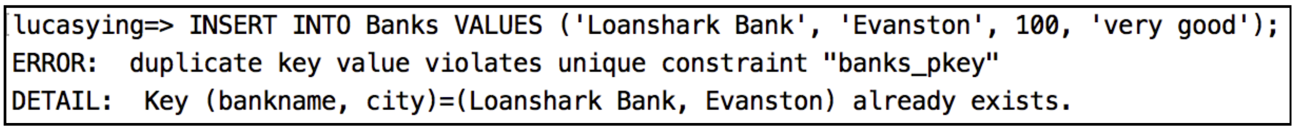
2. A brief explanation of what enforced a partial order in your implementation of RobbersGang database.

The table which is referenced by another table must be imported data earlier than the referencing table.
For example, since the table Banks does not have any foreign keys, it could import data first. However, the table Plans, which contains a foreign key references Banks, must import data after inputing data to Banks.

QUESTION 3: Checking your Database

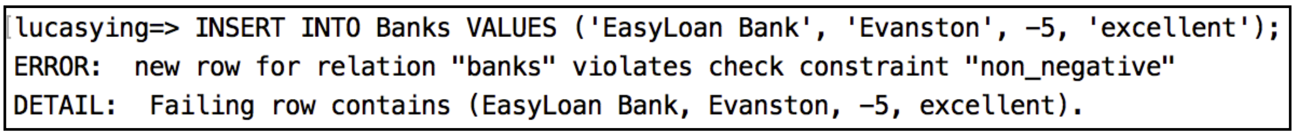
1. Insert the following tuples into the Banks table:

a. (‘Loanshark Bank’, ‘Evanston’, 100, ‘very good’)



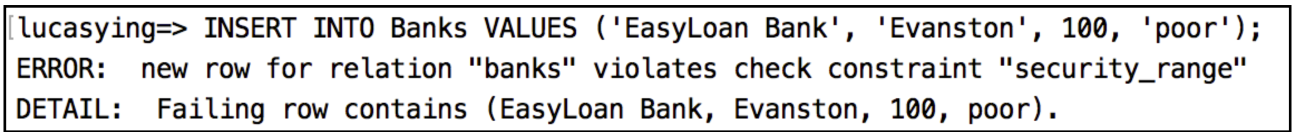
```
lucasying=> INSERT INTO Banks VALUES ('Loanshark Bank', 'Evanston', 100, 'very good');
ERROR:  duplicate key value violates unique constraint "banks_pkey"
DETAIL:  Key (bankname, city)=(Loanshark Bank, Evanston) already exists.
```

b. (‘EasyLoan Bank’, ‘Evanston’, -5, ‘excellent’)



```
lucasying=> INSERT INTO Banks VALUES ('EasyLoan Bank', 'Evanston', -5, 'excellent');
ERROR:  new row for relation "banks" violates check constraint "non_negative"
DETAIL:  Failing row contains (EasyLoan Bank, Evanston, -5, excellent).
```

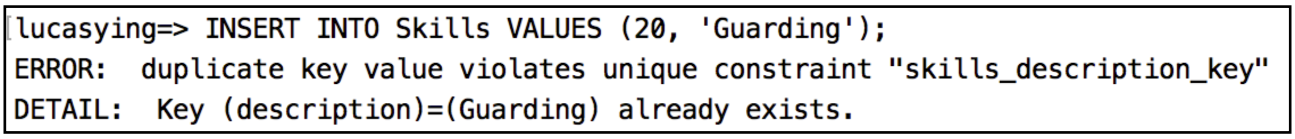
c. (‘EasyLoan Bank’, ‘Evanston’, 100, ‘poor’)



```
[lucasying=> INSERT INTO Banks VALUES ('EasyLoan Bank', 'Evanston', 100, 'poor');
ERROR:  new row for relation "banks" violates check constraint "security_range"
DETAIL:  Failing row contains (EasyLoan Bank, Evanston, 100, poor).
```

2. Insert the following tuple into the Skills table:

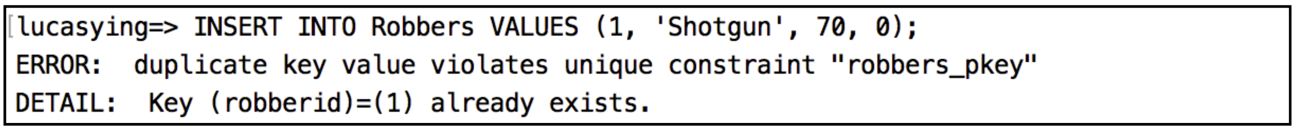
a. (20, ‘Guarding’)



```
[lucasying=> INSERT INTO Skills VALUES (20, 'Guarding');
ERROR:  duplicate key value violates unique constraint "skills_description_key"
DETAIL:  Key (description)=(Guarding) already exists.
```

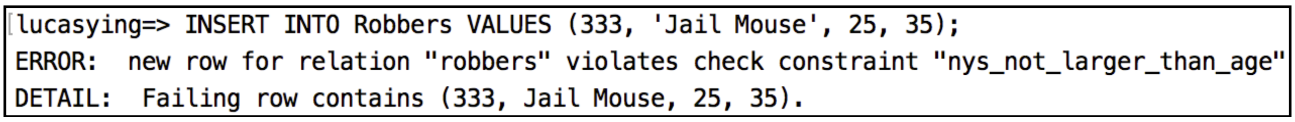
3. Insert the following tuples into the Robbers table:

a. (1, ‘Shotgun’, 70, 0)



```
[lucasying=> INSERT INTO Robbers VALUES (1, 'Shotgun', 70, 0);
ERROR:  duplicate key value violates unique constraint "robbers_pkey"
DETAIL:  Key (robberid)=(1) already exists.
```

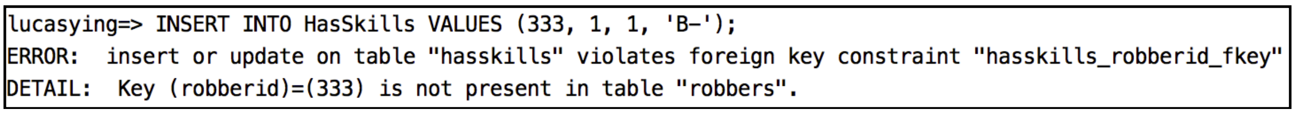
b. (333, ‘Jail Mouse’, 25, 35)



```
[lucasying=> INSERT INTO Robbers VALUES (333, 'Jail Mouse', 25, 35);
ERROR:  new row for relation "robbers" violates check constraint "nys_not_larger_than_age"
DETAIL:  Failing row contains (333, Jail Mouse, 25, 35).
```

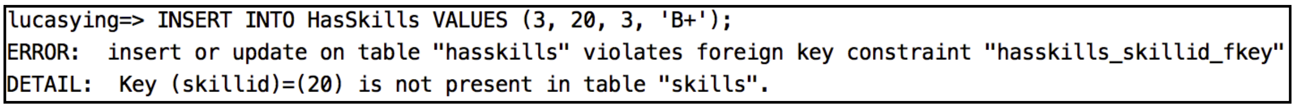
4. Insert the following tuples into the HasSkills table:

a. (333, 1, 1, ‘B-’)



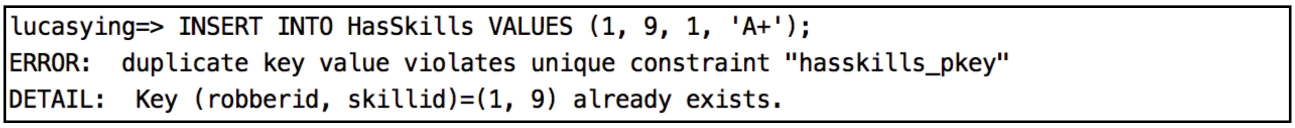
```
lucasying=> INSERT INTO HasSkills VALUES (333, 1, 1, 'B-');
ERROR:  insert or update on table "hasskills" violates foreign key constraint "hasskills_robberid_fkey"
DETAIL:  Key (robberid)=(333) is not present in table "robbers".
```

b. (3, 20, 3, ‘B+’)



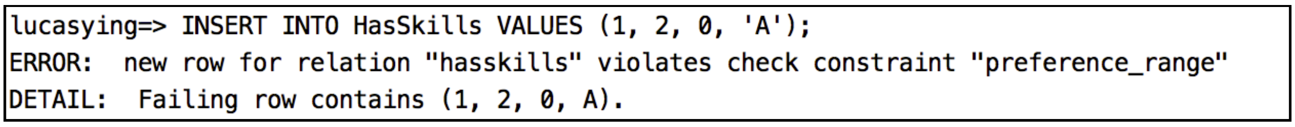
```
lucasying=> INSERT INTO HasSkills VALUES (3, 20, 3, 'B+');
ERROR:  insert or update on table "hasskills" violates foreign key constraint "hasskills_skillid_fkey"
DETAIL:  Key (skillid)=(20) is not present in table "skills".
```

c. (1, 9, 1, ‘A+’)



```
lucasying=> INSERT INTO HasSkills VALUES (1, 9, 1, 'A+');
ERROR:  duplicate key value violates unique constraint "hasskills_pkey"
DETAIL:  Key (robberid, skillid)=(1, 9) already exists.
```

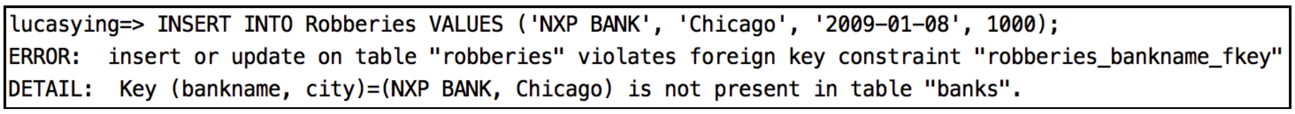
d. (1, 2, 0, ‘A ’)



```
lucasying=> INSERT INTO HasSkills VALUES (1, 2, 0, 'A');
ERROR:  new row for relation "hasskills" violates check constraint "preference_range"
DETAIL:  Failing row contains (1, 2, 0, A).
```

5. Insert the following tuple into the Robberies table:

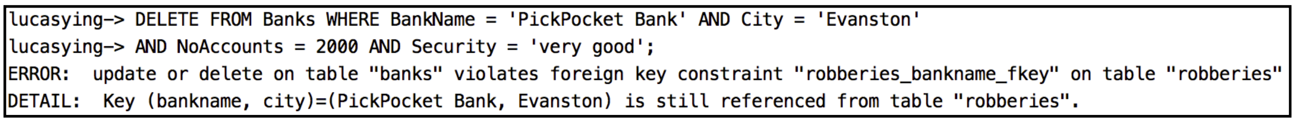
a. (‘NXP Bank’, ‘Chicago’, ‘2009-01-08’, 1000)



```
lucasying=> INSERT INTO Robberies VALUES ('NXP BANK', 'Chicago', '2009-01-08', 1000);
ERROR:  insert or update on table "robberies" violates foreign key constraint "robberies_bankname_fkey"
DETAIL:  Key (bankname, city)=(NXP BANK, Chicago) is not present in table "banks".
```

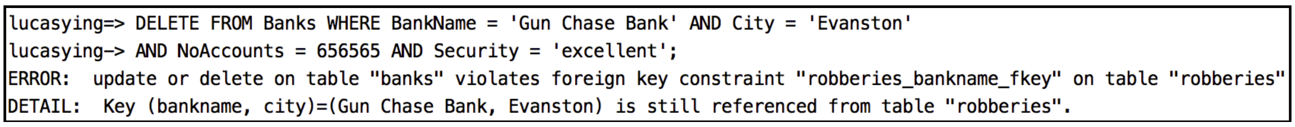
6. Delete the following tuples from the Banks table:

a. (‘PickPocket Bank’, ‘Evanston’, 2000, ‘very good’)



```
lucasying-> DELETE FROM Banks WHERE BankName = 'PickPocket Bank' AND City = 'Evanston'
lucasying-> AND NoAccounts = 2000 AND Security = 'very good';
ERROR:  update or delete on table "banks" violates foreign key constraint "robberies_bankname_fkey" on table "robberies"
DETAIL:  Key (bankname, city)=(PickPocket Bank, Evanston) is still referenced from table "robberies".
```

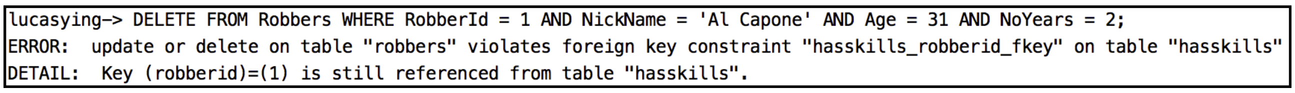
b. (‘Gun Chase Bank’, ‘Evanston’, 656565, ‘excellent’)



```
lucasying=> DELETE FROM Banks WHERE BankName = 'Gun Chase Bank' AND City = 'Evanston'
lucasying-> AND NoAccounts = 656565 AND Security = 'excellent';
ERROR:  update or delete on table "banks" violates foreign key constraint "robberies_bankname_fkey" on table "robberies"
DETAIL:  Key (bankname, city)=(Gun Chase Bank, Evanston) is still referenced from table "robberies".
```

7. Delete the following tuple from the Robbers table:

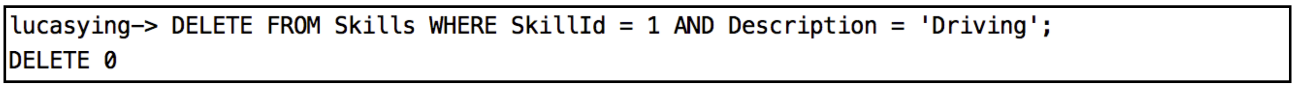
a. (1, ‘Al Capone’, 31, 2).



```
lucasying-> 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".
```

8. Delete the following tuple from the Skills table:

a. (1, ‘Driving’)

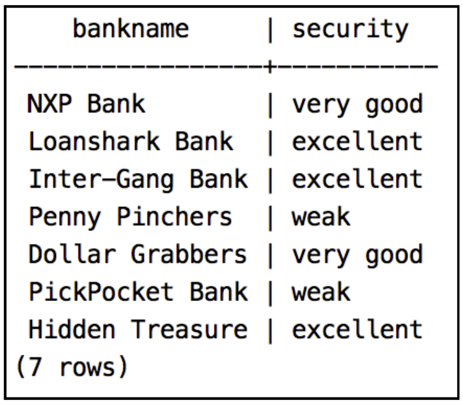


```
lucasying-> DELETE FROM Skills WHERE SkillId = 1 AND Description = 'Driving';
DELETE 0
```

QUESTION 4: Simple Database Queries

1. **Retrieve BankName and Security for all banks in Chicago that have more than 9000 accounts.**

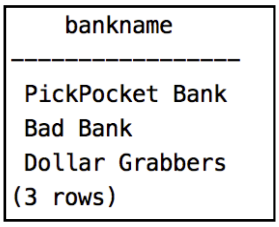
SELECT BankName, Security
FROM Banks
WHERE City = 'Chicago' AND NoAccounts > 9000;



```
     bankname     | security
------------------+-----------
 NXP Bank         | very good
 Loanshark Bank   | excellent
 Inter-Gang Bank  | excellent
 Penny Pinchers   | weak
 Dollar Grabbers  | very good
 PickPocket Bank  | weak
 Hidden Treasure  | excellent
(7 rows)
```

2. **Retrieve BankName of all banks where Calamity Jane has an account. The answer should list every bank at most once.**

```
SELECT DISTINCT BankName
FROM HasAccounts
WHERE HasAccounts.RobberID IN (
    SELECT RobberID
    FROM Robbers
    WHERE NickName = 'Calamity Jane'
);
```

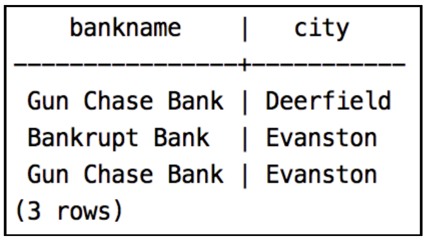


```
    bankname
-----------------
 PickPocket Bank
 Bad Bank
 Dollar Grabbers
(3 rows)
```

3. **Retrieve BankName and City of all bank branches that have no branch in Chicago. The answer should be sorted in increasing order of the number of accounts.**

```
SELECT BankName, City
FROM Banks b
WHERE NOT EXISTS (
    SELECT BankName
    FROM Banks
    WHERE City = 'Chicago' AND b.BankName = BankName
```

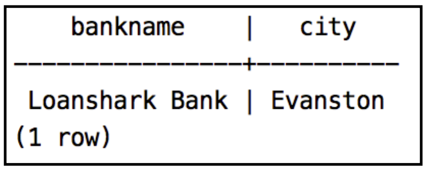
)
ORDER BY NoAccounts;



```
    bankname     |   city
-----------------+-----------
 Gun Chase Bank | Deerfield
 Bankrupt Bank  | Evanston
 Gun Chase Bank | Evanston
(3 rows)
```

4. Retrieve BankName and City of the first bank branch that was ever robbed by the gang.

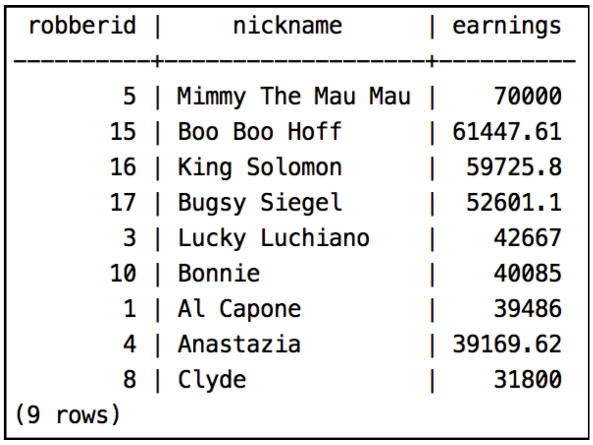
```
SELECT BankName, City
FROM Robberies
ORDER BY Date
LIMIT 1;
```



```
    bankname     |   city
-----------------+-----------
 Loanshark Bank | Evanston
(1 row)
```

5. Retrieve RobberId, Nickname and individual total “earnings” of those robbers who have earned more than $30,000 by robbing banks. The answer should be sorted in decreasing order of the total earnings.

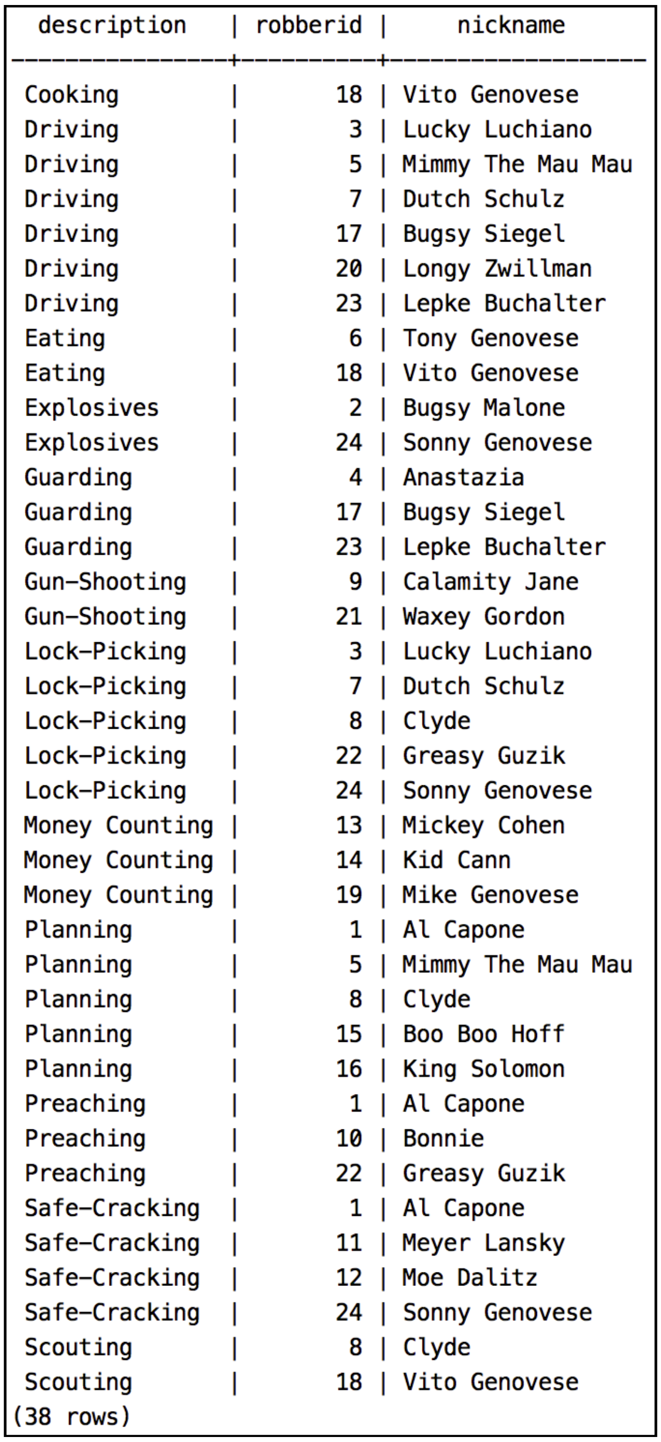
```
SELECT a.RobberID, NickName, SUM(Share) AS Earnings
FROM Accomplices a
JOIN Robbers r on a.RobberID = r.RobberID
GROUP BY a.RobberID, r.NickName
HAVING SUM(Share) > 30000
ORDER BY Earnings DESC;
```



```
 robberid |      nickname       | earnings
----------+---------------------+----------
        5 | Mimmy The Mau Mau |    70000
       15 | Boo Boo Hoff        | 61447.61
       16 | King Solomon        |  59725.8
       17 | Bugsy Siegel        |  52601.1
        3 | Lucky Luchiano      |    42667
       10 | Bonnie              |    40085
        1 | Al Capone           |    39486
        4 | Anastazia           | 39169.62
        8 | Clyde               |    31800
(9 rows)
```

6. **Retrieve the Descriptions of all skills together with the RobberId and NickName of all robbers that possess this skill. The answer should be grouped by skill description.**

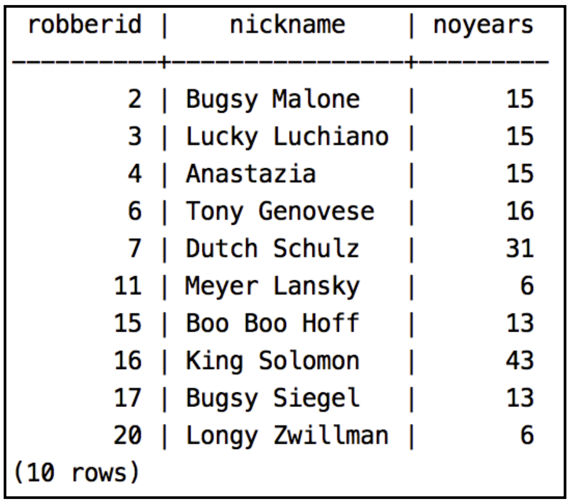
```
SELECT s.Description, r.RobberID, r.NickName
FROM Skills s
JOIN HasSkills hs on s.SkillID = hs.SkillID
JOIN Robbers r on hs.RobberID = r.RobberID
ORDER BY s.Description, r.RobberID;
```



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

7. **Retrieve RobberId, NickName, and the Number of Years in Prison for all robbers who were in prison for more than three years.**

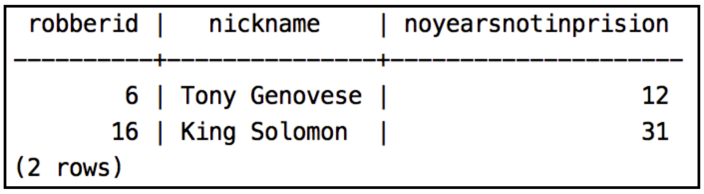
SELECT RobberID, NickName, NoYears
FROM Robbers
WHERE NoYears > 3;



```
 robberid |    nickname     | noyears
----------+-----------------+---------
        2 | Bugsy Malone    |      15
        3 | Lucky Luchiano  |      15
        4 | Anastazia       |      15
        6 | Tony Genovese   |      16
        7 | Dutch Schulz    |      31
       11 | Meyer Lansky    |       6
       15 | Boo Boo Hoff    |      13
       16 | King Solomon    |      43
       17 | Bugsy Siegel    |      13
       20 | Longy Zwillman  |       6
(10 rows)
```

8. **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.**

SELECT RobberID, NickName, Age - NoYears AS NoYearsNotInPrision
FROM Robbers
WHERE NoYears > Age / 2;



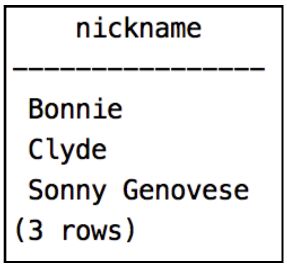
```
 robberid |   nickname    | noyearsnotinprision
----------+---------------+---------------------
        6 | Tony Genovese |                  12
       16 | King Solomon  |                  31
(2 rows)
```

QUESTION 5: Complex Database Queries

1. **The police department wants to know which robbers are most active, but were never penalised. Construct a view that contains the Nicknames of all robbers who participated in more robberies than the average, but spent no time in prison. The answer should be sorted in decreasing order of the individual total “earnings” of the robbers.**

Stepwise:

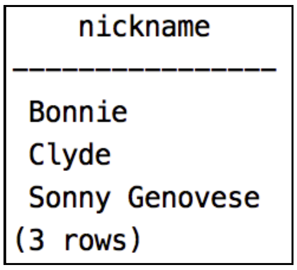
```
CREATE VIEW TempRobbers AS
SELECT RobberID, COUNT(*) AS NoAccomplices, SUM(Share) AS Earnings
FROM Accomplices
GROUP BY RobberID;

CREATE VIEW AvgNoAccomplices AS
SELECT AVG(NoAccomplices) AS Average
FROM TempRobbers;

CREATE VIEW ProcessedRobbers AS
SELECT RobberID, Earnings
FROM TempRobbers
JOIN AvgNoAccomplices ON NoAccomplices > Average;

CREATE VIEW Task1Final AS
SELECT NickName
FROM Robbers
NATURAL JOIN ProcessedRobbers
WHERE NoYears = 0
ORDER BY Earnings DESC;
```



```
    nickname
----------------
 Bonnie
 Clyde
 Sonny Genovese
(3 rows)
```

Nested:

```
CREATE VIEW Q5Task1 AS
SELECT Nickname
FROM Robbers
NATURAL JOIN (
    SELECT RobberID, Earnings
    FROM (
        SELECT RobberID, COUNT(RobberID) AS NoAccomplices, SUM(Share) AS Earnings
        FROM Accomplices
        GROUP BY RobberID
    ) AS TempRobbers
    WHERE NoAccomplices > (
        SELECT AVG(NoAccomplices) AS Average
        FROM (
            SELECT RobberID, COUNT(RobberID) AS NoAccomplices
            FROM Accomplices
            GROUP BY RobberID
        ) AS Average
    )
) AS FinalRobbers
WHERE NoYears = 0
ORDER BY Earnings DESC;
```

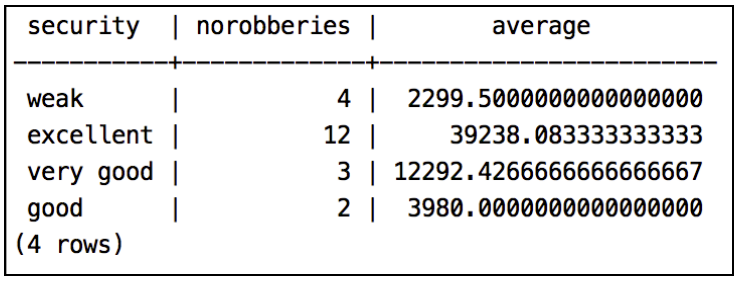


```
    nickname
-----------------
 Bonnie
 Clyde
 Sonny Genovese
(3 rows)
```

2. The police department wants to know whether bank branches with lower security levels are more attractive for robbers than those with higher security levels. Construct a view containing the Security level, the 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.

Stepwise:

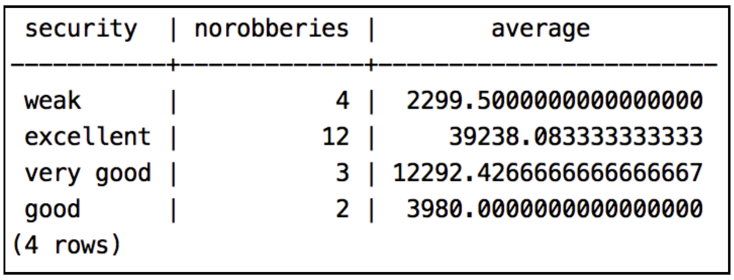
```
CREATE VIEW BankWithRobbery AS
SELECT Security, Amount
FROM Banks
NATURAL JOIN Robberies;

CREATE VIEW Task2Final AS
SELECT Security, COUNT(*) AS NoRobberies, AVG(Amount) AS Average
FROM BankWithRobbery
GROUP BY Security;
```



```
 security  | norobberies |        average
-----------+-------------+------------------------
 weak      |           4 |  2299.5000000000000000
 excellent |          12 |    39238.083333333333
 very good |           3 | 12292.426666666666667
 good      |           2 |  3980.0000000000000000
(4 rows)
```

Nested:

```
CREATE VIEW Q5Task2 AS
SELECT Security, COUNT(*) AS NoRobberies, AVG(Amount) AS Average
FROM (
    SELECT Security, Amount
    FROM Banks
    NATURAL JOIN Robberies
) AS Instances
GROUP BY Security;
```

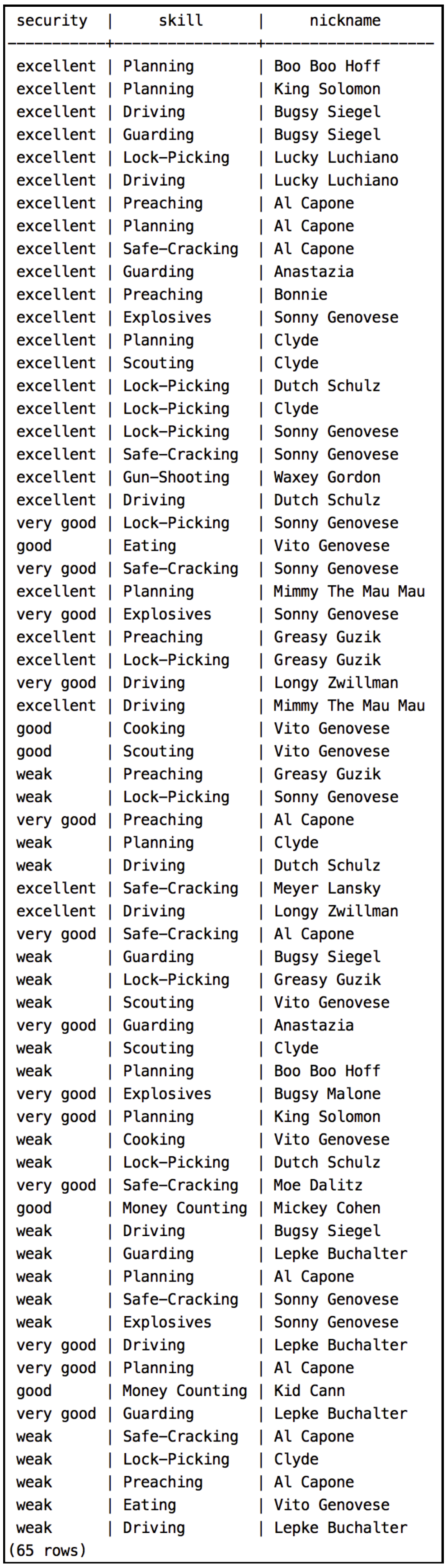


```
 security  | norobberies |        average
-----------+-------------+------------------------
 weak      |           4 |  2299.5000000000000000
 excellent |          12 |     39238.083333333333
 very good |           3 | 12292.426666666666667
 good      |           2 |  3980.0000000000000000
(4 rows)
```

3. **The police department wants to know which robbers are most likely to attack a particular bank branch. Robbing bank branches with a certain security level might require certain skills. For example, maybe every robbery of a branch with "excellent" security requires a robber with "Explosives" skill. Construct a view containing Security level, Skill, and Nickname showing for each security level all those skills that were possessed by some participating robber in each robbery of a bank branch of the respective security level, and the nicknames of all robbers who have that skill.**

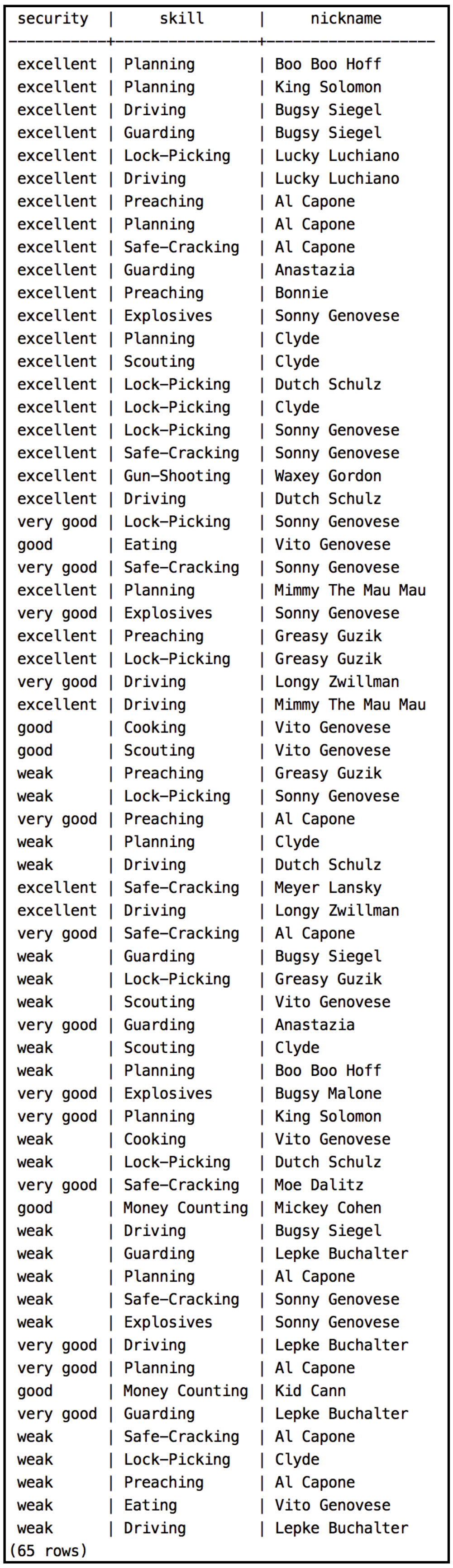
Stepwise:

```
CREATE VIEW Task3Temp1 AS
SELECT Security, Description AS Skill, NickName, COUNT(*) AS NoSkills
FROM Accomplices
NATURAL JOIN Banks
NATURAL JOIN HasSkills
NATURAL JOIN Skills
NATURAL JOIN Robbers
GROUP BY Security, Skill, NickName
ORDER BY NoSkills DESC;

CREATE VIEW Task3Final AS
SELECT Security, Skill, NickName
FROM Task3Temp1;
```



```
 security  |      skill      |      nickname
-----------+-----------------+--------------------
 excellent | Planning        | Boo Boo Hoff
 excellent | Planning        | King Solomon
 excellent | Driving         | Bugsy Siegel
 excellent | Guarding        | Bugsy Siegel
 excellent | Lock-Picking    | Lucky Luchiano
 excellent | Driving         | Lucky Luchiano
 excellent | Preaching       | Al Capone
 excellent | Planning        | Al Capone
 excellent | Safe-Cracking   | Al Capone
 excellent | Guarding        | Anastazia
 excellent | Preaching       | Bonnie
 excellent | Explosives      | Sonny Genovese
 excellent | Planning        | Clyde
 excellent | Scouting        | Clyde
 excellent | Lock-Picking    | Dutch Schulz
 excellent | Lock-Picking    | Clyde
 excellent | Lock-Picking    | Sonny Genovese
 excellent | Safe-Cracking   | Sonny Genovese
 excellent | Gun-Shooting    | Waxey Gordon
 excellent | Driving         | Dutch Schulz
 very good | Lock-Picking    | Sonny Genovese
 good      | Eating          | Vito Genovese
 very good | Safe-Cracking   | Sonny Genovese
 excellent | Planning        | Mimmy The Mau Mau
 very good | Explosives      | Sonny Genovese
 excellent | Preaching       | Greasy Guzik
 excellent | Lock-Picking    | Greasy Guzik
 very good | Driving         | Longy Zwillman
 excellent | Driving         | Mimmy The Mau Mau
 good      | Cooking         | Vito Genovese
 good      | Scouting        | Vito Genovese
 weak      | Preaching       | Greasy Guzik
 weak      | Lock-Picking    | Sonny Genovese
 very good | Preaching       | Al Capone
 weak      | Planning        | Clyde
 weak      | Driving         | Dutch Schulz
 excellent | Safe-Cracking   | Meyer Lansky
 excellent | Driving         | Longy Zwillman
 very good | Safe-Cracking   | Al Capone
 weak      | Guarding        | Bugsy Siegel
 weak      | Lock-Picking    | Greasy Guzik
 weak      | Scouting        | Vito Genovese
 very good | Guarding        | Anastazia
 weak      | Scouting        | Clyde
 weak      | Planning        | Boo Boo Hoff
 very good | Explosives      | Bugsy Malone
 very good | Planning        | King Solomon
 weak      | Cooking         | Vito Genovese
 weak      | Lock-Picking    | Dutch Schulz
 very good | Safe-Cracking   | Moe Dalitz
 good      | Money Counting  | Mickey Cohen
 weak      | Driving         | Bugsy Siegel
 weak      | Guarding        | Lepke Buchalter
 weak      | Planning        | Al Capone
 weak      | Safe-Cracking   | Sonny Genovese
 weak      | Explosives      | Sonny Genovese
 very good | Driving         | Lepke Buchalter
 very good | Planning        | Al Capone
 good      | Money Counting  | Kid Cann
 very good | Guarding        | Lepke Buchalter
 weak      | Safe-Cracking   | Al Capone
 weak      | Lock-Picking    | Clyde
 weak      | Preaching       | Al Capone
 weak      | Eating          | Vito Genovese
 weak      | Driving         | Lepke Buchalter
(65 rows)
```

Stepwise



```
 security  |      skill      |      nickname
-----------+-----------------+--------------------
 excellent | Planning        | Boo Boo Hoff
 excellent | Planning        | King Solomon
 excellent | Driving         | Bugsy Siegel
 excellent | Guarding        | Bugsy Siegel
 excellent | Lock-Picking    | Lucky Luchiano
 excellent | Driving         | Lucky Luchiano
 excellent | Preaching       | Al Capone
 excellent | Planning        | Al Capone
 excellent | Safe-Cracking   | Al Capone
 excellent | Guarding        | Anastazia
 excellent | Preaching       | Bonnie
 excellent | Explosives      | Sonny Genovese
 excellent | Planning        | Clyde
 excellent | Scouting        | Clyde
 excellent | Lock-Picking    | Dutch Schulz
 excellent | Lock-Picking    | Clyde
 excellent | Lock-Picking    | Sonny Genovese
 excellent | Safe-Cracking   | Sonny Genovese
 excellent | Gun-Shooting    | Waxey Gordon
 excellent | Driving         | Dutch Schulz
 very good | Lock-Picking    | Sonny Genovese
 good      | Eating          | Vito Genovese
 very good | Safe-Cracking   | Sonny Genovese
 excellent | Planning        | Mimmy The Mau Mau
 very good | Explosives      | Sonny Genovese
 excellent | Preaching       | Greasy Guzik
 excellent | Lock-Picking    | Greasy Guzik
 very good | Driving         | Longy Zwillman
 excellent | Driving         | Mimmy The Mau Mau
 good      | Cooking         | Vito Genovese
 good      | Scouting        | Vito Genovese
 weak      | Preaching       | Greasy Guzik
 weak      | Lock-Picking    | Sonny Genovese
 very good | Preaching       | Al Capone
 weak      | Planning        | Clyde
 weak      | Driving         | Dutch Schulz
 excellent | Safe-Cracking   | Meyer Lansky
 excellent | Driving         | Longy Zwillman
 very good | Safe-Cracking   | Al Capone
 weak      | Guarding        | Bugsy Siegel
 weak      | Lock-Picking    | Greasy Guzik
 weak      | Scouting        | Vito Genovese
 very good | Guarding        | Anastazia
 weak      | Scouting        | Clyde
 weak      | Planning        | Boo Boo Hoff
 very good | Explosives      | Bugsy Malone
 very good | Planning        | King Solomon
 weak      | Cooking         | Vito Genovese
 weak      | Lock-Picking    | Dutch Schulz
 very good | Safe-Cracking   | Moe Dalitz
 good      | Money Counting  | Mickey Cohen
 weak      | Driving         | Bugsy Siegel
 weak      | Guarding        | Lepke Buchalter
 weak      | Planning        | Al Capone
 weak      | Safe-Cracking   | Sonny Genovese
 weak      | Explosives      | Sonny Genovese
 very good | Driving         | Lepke Buchalter
 very good | Planning        | Al Capone
 good      | Money Counting  | Kid Cann
 very good | Guarding        | Lepke Buchalter
 weak      | Safe-Cracking   | Al Capone
 weak      | Lock-Picking    | Clyde
 weak      | Preaching       | Al Capone
 weak      | Eating          | Vito Genovese
 weak      | Driving         | Lepke Buchalter
(65 rows)
```

Nested

Nested:

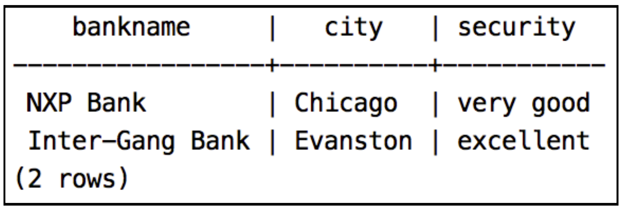
```
CREATE VIEW Q5Task3 AS
SELECT Security, Skill, NickName
FROM (
    SELECT Security, Description AS Skill, NickName, COUNT(*) AS NoSkills
    FROM Accomplices
    NATURAL JOIN Banks
    NATURAL JOIN HasSkills
    NATURAL JOIN Skills
    NATURAL JOIN Robbers
    GROUP BY Security, Skill, NickName
    ORDER BY NoSkills DESC;
) AS Temp;
```

Output at Previous Page - Page 19

4. **The police department wants to increase security at those bank branches that are most likely to be victims in the near future. Construct a view containing the BankName, the City, and Security level of all bank branches that have not been robbed in the previous year, but where plans for a robbery next year are known. The answer should be sorted in decreasing order of the number of robbers who have accounts in that bank branch.**

Stepwise:

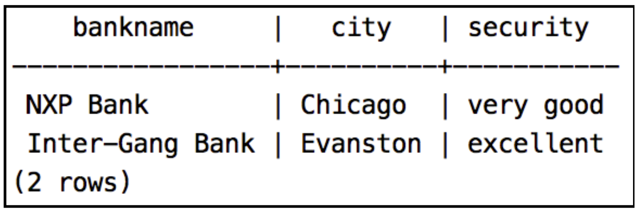
```
CREATE VIEW Robberies2015 AS
SELECT BankName, City
FROM Robberies
WHERE Date > '2014-12-31' AND Date < '2016-1-1';

CREATE VIEW Plans2017 AS
SELECT BankName, City
FROM Plans
WHERE PlannedDate > '2016-12-31' AND PlannedDate < '2018-1-1';

CREATE VIEW NotRobbed AS
SELECT BankName, City, Security
FROM Banks b
WHERE NOT EXISTS (
    SELECT *
    FROM Robberies2015
    WHERE b.BankName = BankName
);

CREATE VIEW InPlan AS
SELECT BankName, City, Security, COUNT(RobberID) AS NoAccounts
FROM NotRobbed
NATURAL JOIN Plans2017
NATURAL JOIN HasAccounts
GROUP BY BankName, City, Security
ORDER BY NoAccounts DESC;

CREATE VIEW Task4Final AS
SELECT BankName, City, Security
FROM InPlan;
```



```
    bankname     |   city   | security
-----------------+----------+-----------
 NXP Bank        | Chicago  | very good
 Inter-Gang Bank | Evanston | excellent
(2 rows)
```

Nested:

```
CREATE VIEW Q5Task4 AS
SELECT BankName, City, Security
FROM (
    SELECT BankName, City, Security, COUNT(RobberID) AS NoAccounts
    FROM (
        SELECT BankName, City, Security
        FROM Banks b
        WHERE NOT EXISTS (
            SELECT BankName, City
            FROM Robberies
            WHERE Date > '2014-12-31'
            AND Date < '2016-1-1'
            AND b.BankName = BankName
        )
    ) AS InPlan
    NATURAL JOIN (
        SELECT BankName, City
        FROM Plans
        WHERE PlannedDate > '2016-12-31'
        AND PlannedDate < '2018-1-1'
    ) AS Temp
    NATURAL JOIN HasAccounts
    GROUP BY BankName, City, Security
    ORDER BY NoAccounts DESC
) AS Result;
```

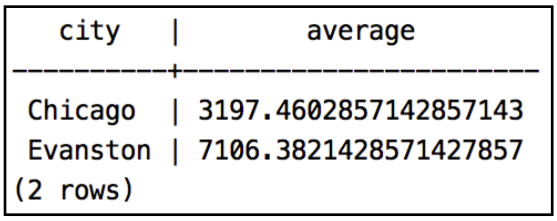


```
    bankname     |   city   | security
-----------------+----------+-----------
 NXP Bank        | Chicago  | very good
 Inter-Gang Bank | Evanston | excellent
(2 rows)
```

5. **The police department has a theory that bank robberies in Chicago are more profitable than in any other city in their district. Construct a view that shows the average share of all robberies in Chicago, and the average share of all robberies for that city (other than Chicago) that observes the highest average share. The average share of a robbery is computed based on the number of participants in that particular robbery.**

Stepwise:

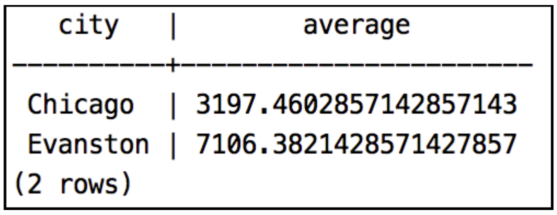
```
CREATE VIEW RobberiesAvg AS
SELECT BankName, City, RobberyDate AS Date, AVG(Share) AS Average
FROM Accomplices
GROUP BY BankName, City, Date
ORDER BY City, Date;

CREATE VIEW Task5Final AS
SELECT City, AVG(Average) AS Average
FROM RobberiesAvg
GROUP BY City;
```



```
   city   |       average
----------+---------------------
 Chicago  | 3197.4602857142857143
 Evanston | 7106.3821428571427857
(2 rows)
```

Nested:

```
CREATE VIEW Q5Task5 AS
SELECT City, AVG(Average) AS Average
FROM (
    SELECT BankName, City, RobberyDate AS Date, AVG(Share) AS Average
    FROM Accomplices
    GROUP BY BankName, City, Date
    ORDER BY City, Date
) AS Result
GROUP BY City;
```



```
   city   |       average
----------+---------------------
 Chicago  | 3197.4602857142857143
 Evanston | 7106.3821428571427857
(2 rows)
```