SUBANKAMO 300470761

QUESTION 1: Defining the Database

Your answer to Question 1 should include:
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.
2. A list of all your CREATE TABLE statements.
3. A justification for your choice of actions on delete or on update for each foreign key.
4. A brief justification for your choice of attribute constraints (other than the basic data)

Banks

Attributes: BankName, City, NoAccounts, Security
PrimaryKey: BankName, City
ForeignKey: None

PrimaryKey Justification: There can be duplicates in BankName (e.g ANZ), City (e.g multiple banks in the same city), NoAccounts, Security.
The combination of BankName and City generates a unique identifier as there is unlikely to be multiple banks with the same name in the same city.
'The banks are identified by their name and city rather than by an Id'.

Create Table Statement:

```
CREATE TABLE Banks (
   BankName VARCHAR(30) NOT NULL,
   City VARCHAR(30) NOT NULL,
   NoAccounts INT CONSTRAINT CheckNoAccountsPositive CHECK (NoAccounts>=0),
   Security VARCHAR(15) CONSTRAINT CheckSecurityValue CHECK (Security IN('weak', 'good', 'very good', 'excellent')),
   CONSTRAINT BanksPK PRIMARY KEY (BankName, City)
);
```

Actions:
None

Attributes constraint:
-Primary key cannot be null hence NOT NULL for BankName, City
-Logically bank account cannot be less than 0 hence CHECK (NoAccounts>=0)
-Logically security must be from category ('weak', 'good', 'very good', 'excellent') CHECK (Security IN('weak', 'good', 'very good', 'excellent')) - assumption made by reading _banks23

Robberies

Attributes: BankName, City, Date, Amount
PrimaryKey: BankName, City, Date
ForeignKey: Robberies{BankName, City} ⊆ Banks{BankName, City}

PrimaryKey Justification: There can be duplicates in BankName (e.g ANZ), City (e.g multiple banks in the same city), Date, Amount.
The combination of BankName, City and Date generates a unique identifier as there is unlikely to be more than one robbery at a specific bank on a specific date.

ForeignKey Justification: Each robbery in the Robberies table is associated with a bank identified by its name and city, and this information is stored in the Banks table. Therefore, the foreign key BankName, City in the Robberies table references the primary key BankName, City in the Banks table.

Create Table Statement:

```
CREATE TABLE Robberies (
   BankName VARCHAR(30) NOT NULL,
   City VARCHAR(30) NOT NULL,
   Date DATE NOT NULL,
   Amount DECIMAL CONSTRAINT CheckAmountPositive CHECK (Amount>=0),
   CONSTRAINT RobberiesPK PRIMARY KEY (BankName, City, Date),
   CONSTRAINT RobberiesFK FOREIGN KEY (BankName, City) REFERENCES
Banks(BankName, City) ON DELETE RESTRICT ON UPDATE CASCADE
);
```

Action:

- DELETE RESTRICT is used to prevent deletion of records in the Bank table if there is a record in the Robberies table using its attributes as a foreign key. this is to avoids referential integrity constraint violation and prevent accidental deletion of a record in the Banks table
-If a bank name or city is update in the bank table, records in the robberies table should be updated hence ON UPDATE CASCADE

Attributes constraint:

-Primary key cannot be null hence NOT NULL for BankName, City, Date
-Logically amount cannot be less than 0 hence CHECK (Amount>=0)

Plans

Attributes: BankName, City, NoRobbers, PlannedDate
PrimaryKey: BankName, City, PlannedDate
ForeignKey: Plans{BankName, City} ⊆ Banks{BankName, City}

PrimaryKey Justification: There can be duplicates in BankName (e.g ANZ), City (e.g multiple banks in the same city), NoRobbers, PlannedDate
The combination of BankName, City and PlannedDate generates a unique identifier as there is unlikely to be more than one robbery plan at a specific bank on a specific date.

ForeignKey Justification: Each plan in the Plans table is associated with a bank identified by its name and city, and this information is stored in the Banks table. Therefore, the foreign key BankName, City in the Plans table references the primary key BankName, City in the Banks table.

Create Table Statement:

```
CREATE TABLE Plans (
   BankName VARCHAR(30) NOT NULL,
   City VARCHAR(30) NOT NULL,
```

```
    PlannedDate DATE NOT NULL,
    NoRobbers INT CONSTRAINT CheckNoRobbersPositive CHECK (NoRobbers >=0),
    CONSTRAINT PlansPK PRIMARY KEY (BankName, City, PlannedDate),
    CONSTRAINT PlansFK FOREIGN KEY (BankName, City) REFERENCES
Banks(BankName, City) ON DELETE RESTRICT ON UPDATE CASCADE
);
```

Action:
- DELETE RESTRICT is used to prevent deletion of records in the Bank table if there is a record in the Plans table using its attributes as a foreign key. this is to avoids referential integrity constraint violation and prevent accidental deletion of a record in the Banks table
-If a bank name or city is update in the bank table, records in the robberies table should be updated hence ON UPDATE CASCADE

Attributes constraint:
-Primary key cannot be null hence NOT NULL for BankName, City, PlannedDate
-Logically NoRobbers cannot be less than 0 hence CHECK (NoRobbers>=0)

Robbers
Attributes: RobberId, Nickname, Age, NoYears
PrimaryKey: RobberId
ForeignKey: None

PrimaryKey Justification: There can be duplicates in Nickname, Age, NoYears. RobberId is unique for each robber.

Create Table Statement:

```
CREATE TABLE Robbers (
    RobberId SERIAL,
    Nickname VARCHAR(30),
    Age INT CONSTRAINT CheckAgePositive CHECK (Age >=0),
    NoYears INT CONSTRAINT CheckNoYearPositiveAndLessThanAge CHECK (NoYears
<= Age),
    CONSTRAINT RobbersPK PRIMARY KEY (RobberId)
);
```

Action:
None

Attributes constraint:
-Logically Age cannot be less than 0 hence CHECK (Age>=0)
-Logically NoYears cannot be less Age hence CHECK (NoYears <= Age)
>Serial is used to automatically generate a uniqueID for RobberId ‘it is possible to have two robbers with the same nickname. It would be better to give each robber a unique Id’

Skills

SUBANKAMO 300470761

Attributes: SkillId, Description
PrimaryKey: SkillId
ForeignKey: None

PrimaryKey Justification: There can be duplicates in Description.
SkillId is unique for each skill.

Create Table Statement:

```
CREATE TABLE Skills (
   SkillId SERIAL,
   Description VARCHAR(100) CONSTRAINT unique_column_constraint UNIQUE,
   CONSTRAINT SkillsPK PRIMARY KEY (SkillId)
);
```

Action:
None

Attributes constraint:
>Description must be Unique as we do not want skills with duplicate description hence CONSTRAINT unique_column_constraint UNIQUE
>Serial is used to automatically generate a uniqueID for SkillId - skills is being derived from existing hasskill table hence it does not currently have an ID ‘A better design can be achieved if we introduce an additional Skills table listing all the possible skills’

HasSkills
Attributes: RobberId, SkillId, Preference, Grade
PrimaryKey: RobberId, SkillId
ForeignKey: HasSkills{RobberId} ⊆ Robbers{RobberId}, HasSkills{SkillId} ⊆ Skills{SkillId}

PrimaryKey Justification: There can be duplicates in RobberId (e.g robber can have multiple skills), SkillId (e.g skill can belong to multiple robbers), Preference, Grade.
The combination of RobberId and SkillId uniquely identifies a record in the HasSkills table.

ForeignKey Justification: Each record in the HasSkills table corresponds to a particular robber and a particular skill, and this information is stored in the Robbers and Skills tables, respectively. Therefore, the foreign key RobberId in the HasSkills table references the primary key RobberId in the Robbers table, and the foreign key SkillId in the HasSkills table references the primary key SkillId in the Skills table.

Create Table Statement:

```
CREATE TABLE HasSkills (
   RobberId INT NOT NULL,
   SkillId INT NOT NULL,
   Preference INT CONSTRAINT CheckPreferenceBetween1And3 CHECK (Preference
BETWEEN 1 AND 3),
   Grade VARCHAR(5),
   CONSTRAINT HasSkillsPK PRIMARY KEY (RobberId, SkillId),
```

```
   CONSTRAINT HasSkillsRobbersFK FOREIGN KEY (RobberId) REFERENCES
Robbers(RobberId) ON DELETE RESTRICT ON UPDATE CASCADE,
   CONSTRAINT HasSkillsSkillsFK FOREIGN KEY (SkillId) REFERENCES Skills(SkillId)
ON DELETE RESTRICT ON UPDATE CASCADE
);
```

Action:
- DELETE RESTRICT is used to prevent deletion of records in the Robber table if there is a record in the HasSkills table using its attributes as a foreign key. this is to avoids referential integrity constraint violation and prevent accidental deletion of a record in the Robber table
- DELETE RESTRICT is used to prevent deletion of records in the Skills table if there is a record in the HasSkills table using its attributes as a foreign key. this is to avoids referential integrity constraint violation and prevent accidental deletion of a record in the Skills table
-If a robber's robberID is update in the robber table, records in the hasskill table should be updated hence ON UPDATE CASCADE
-If a skill's skillID is update in the skill table, records in the hasskill table should be updated hence ON UPDATE CASCADE

Attributes constraint:
-Primary key cannot be null hence NOT NULL RobberId, SkillId
-Logically Preference must be between 1 and 3 hence CHECK (Preference>=1 AND Preference>=3)

HasAccounts
Attributes: RobberId, BankName, City
PrimaryKey: RobberId, BankName, City
ForeignKey: HasAccounts{RobberId} ⊆ Robbers{RobberId}, HasAccounts{BankName, City} ⊆ Banks{BankName, City}

PrimaryKey Justification: There can be duplicates in RobberId (e.g robber has account at multiple banks), BankName, City .
A robber can have an account at a specific bank only once. The combination of RobberId, BankName, and City uniquely identifies an account in the HasAccounts table.

ForeignKey Justification: Each record in the HasAccounts table corresponds to a particular robber and a particular bank, and this information is stored in the Robbers and Banks tables, respectively. Therefore, the foreign key RobberId in the HasAccounts table references the primary key RobberId in the Robbers table, and the foreign key BankName, City in the HasAccounts table references the primary key BankName, City in the Banks table.

Create Table Statement:

```
CREATE TABLE HasAccounts (
   RobberId INT NOT NULL,
   BankName VARCHAR(30) NOT NULL,
   City VARCHAR(30) NOT NULL,
   CONSTRAINT HasAccountsPK PRIMARY KEY (RobberId, BankName, City),
   CONSTRAINT HasAccountsRobbersFK FOREIGN KEY (RobberId) REFERENCES
Robbers(RobberId) ON DELETE RESTRICT ON UPDATE CASCADE,
```

CONSTRAINT HasAccountsBanksFK FOREIGN KEY (BankName, City) REFERENCES Banks(BankName, City) ON DELETE RESTRICT ON UPDATE CASCADE
);

Action:
- DELETE RESTRICT is used to prevent deletion of records in the Robber table if there is a record in the HasAccounts table using its attributes as a foreign key. this is to avoids referential integrity constraint violation and prevent accidental deletion of a record in the Robber table
- DELETE RESTRICT is used to prevent deletion of records in the Banks table if there is a record in the HasAccounts table using its attributes as a foreign key. this is to avoids referential integrity constraint violation and prevent accidental deletion of a record in the Banks table
-If a robber's robberID is update in the robber table, records in the HasAccounts table should be updated hence ON UPDATE CASCADE
-If a bank's name or city is update in the bank table, records in the HasAccounts table should be updated hence ON UPDATE CASCADE

Attributes constraint:
-Primary key cannot be null hence NOT NULL RobberId, BankName, City

Accomplices
Attributes: RobberId, BankName, City, Date, Share
PrimaryKey: RobberId, BankName, City, Date
ForeignKey: Accomplices{RobberId} ⊆ Robbers{RobberId}, Accomplices{BankName, City, Date} ⊆ Robberies{BankName, City, Date}

PrimaryKey Justification: RobberId (e.g involved in multiple robberies), BankName (e.g multiple robberies at bank), City (e.g multiple robberies in city), Date(e.g multiple robberies on same date), Share (e.g multiple robbers with same share) can be duplicated.
A robber can be an accomplice at a specific robbery at a specific bank on a specific date only once. The combination of RobberId, BankName, City, and Date uniquely identifies an account in the Accomplices table.

ForeignKey Justification: Each record in the Accomplices table corresponds to a particular robber and a particular robbery and the share of money obtained by the robber in the robbery. Therefore, the foreign key RobberId in the Accomplices table references the primary key RobberId in the Robbers table, and the foreign key BankName, City, Date in the Accomplices table references the primary key BankName, City, Date in the Robberies table.

Create Table Statement:
CREATE TABLE Accomplices (
RobberId INT NOT NULL,
BankName VARCHAR(30) NOT NULL,
City VARCHAR(30) NOT NULL,
Date DATE NOT NULL,
Share DECIMAL CONSTRAINT CheckAmountPositive CHECK (Amount>=0),
CONSTRAINT AccomplicesPK PRIMARY KEY (RobberId, BankName, City, Date),

CONSTRAINT AccomplicesRobbersFK FOREIGN KEY (RobberId) REFERENCES Robbers(RobberId) ON DELETE RESTRICT ON UPDATE CASCADE,
CONSTRAINT AccomplicesRobberiesFK FOREIGN KEY (BankName, City, Date) REFERENCES Robberies(BankName, City, Date) ON DELETE RESTRICT ON UPDATE CASCADE
);

Action:
- DELETE RESTRICT is used to prevent deletion of records in the Robber table if there is a record in the Accomplice table using its attributes as a foreign key. this is to avoids referential integrity constraint violation and prevent accidental deletion of a record in the Robber table
- DELETE RESTRICT is used to prevent deletion of records in the Robberies table if there is a record in the Accomplice table using its attributes as a foreign key. this is to avoids referential integrity constraint violation and prevent accidental deletion of a record in the Robberies table
-If a robber's robberID is update in the robber table, records in the Accomplices table should be updated hence ON UPDATE CASCADE
-If a robbery's bankname, city or date is update in the robberies table, records in the Accomplices table should be updated hence ON UPDATE CASCADE

Attributes constraint:
-Primary key cannot be null hence NOT NULL RobberId, BankName, City, Date
-Share robber receive cannot be negative (cannot pay/lose money) hence CONSTRAINT CheckAmountPositive CHECK (Amount>=0)

QUESTION 2: Populating your Database with Data

Your answer to Question 2 should include:
1. A description of how you performed all the data conversion, for example, a sequence of the PostgreSQL statements that accomplished the conversion.
2. A brief description of the order in which you have implemented the tables of the RobbersGang database. Justify your answer.

Creating Database and table

```
postgres=# CREATE DATABASE RobbersGang;
CREATE DATABASE
```

```
postgres=# \c robbersgang
psql (14.4, server 15.1)
WARNING: psql major version 14, server major version 15.
         Some psql features might not work.
You are now connected to database "robbersgang" as user "postgres".
robbersgang=#
```

SUBANKAMO 300470761

```
robbersgang=# CREATE TABLE Banks (
robbersgang(#     BankName VARCHAR(30) NOT NULL,
robbersgang(#     City VARCHAR(30) NOT NULL,
robbersgang(#     NoAccounts INT CONSTRAINT CheckNoAccountsPositive CHECK (NoAccounts>=0),
robbersgang(#     Security VARCHAR(15) CONSTRAINT CheckSecurityValue CHECK (Security IN('weak', 'good', 'very good', 'ex
cellent')),
robbersgang(#     CONSTRAINT BanksPK PRIMARY KEY (BankName, City)
robbersgang(# );
CREATE TABLE
```

```
robbersgang=# CREATE TABLE Robberies (
robbersgang(#     BankName VARCHAR(30) NOT NULL,
robbersgang(#     City VARCHAR(30) NOT NULL,
robbersgang(#     Date DATE NOT NULL,
robbersgang(#     Amount DECIMAL CONSTRAINT CheckAmountPositive CHECK (Amount>=0),
robbersgang(#     CONSTRAINT RobberiesPK PRIMARY KEY (BankName, City, Date),
robbersgang(#     CONSTRAINT RobberiesFK FOREIGN KEY (BankName, City) REFERENCES Banks(BankName, City) ON DELETE RESTRIC
T ON UPDATE CASCADE
robbersgang(# );
CREATE TABLE
```

```
robbersgang=# CREATE TABLE Plans (
robbersgang(#     BankName VARCHAR(30) NOT NULL,
robbersgang(#     City VARCHAR(30) NOT NULL,
robbersgang(#
robbersgang(#     PlannedDate DATE NOT NULL,
robbersgang(#     NoRobbers INT CONSTRAINT CheckNoRobbersPositive CHECK (NoRobbers >=0),
robbersgang(#     CONSTRAINT PlansPK PRIMARY KEY (BankName, City, PlannedDate),
robbersgang(#     CONSTRAINT PlansFK FOREIGN KEY (BankName, City) REFERENCES Banks(BankName, City) ON DELETE RESTRICT ON
 UPDATE CASCADE
robbersgang(# );
CREATE TABLE
```

```
robbersgang=# CREATE TABLE Robbers (
robbersgang(#     RobberId SERIAL,
robbersgang(#     Nickname VARCHAR(30),
robbersgang(#     Age INT CONSTRAINT CheckAgePositive CHECK (Age >=0),
robbersgang(#     NoYears INT CONSTRAINT CheckNoYearPositiveAndLessThanAge CHECK (NoYears <= Age),
robbersgang(#     CONSTRAINT RobbersPK PRIMARY KEY (RobberId)
robbersgang(# );
CREATE TABLE
```

```
postgres=# CREATE TABLE Skills (
postgres(#     SkillId SERIAL,
postgres(#     Description VARCHAR(100) CONSTRAINT unique_column_constraint UNIQUE,
postgres(#     CONSTRAINT SkillsPK PRIMARY KEY (SkillId)
postgres(# );
CREATE TABLE
postgres=#
```

```
robbersgang=# CREATE TABLE HasSkills (
robbersgang(#     RobberId INT NOT NULL,
robbersgang(#     SkillId INT NOT NULL,
robbersgang(#     Preference INT CONSTRAINT CheckPreferenceBetween1And3 CHECK (Preference BETWEEN 1 AND 3),
robbersgang(#     Grade VARCHAR(5),
robbersgang(#     CONSTRAINT HasSkillsPK PRIMARY KEY (RobberId, SkillId),
robbersgang(#     CONSTRAINT HasSkillsRobbersFK FOREIGN KEY (RobberId) REFERENCES Robbers(RobberId) ON DELETE RESTRICT O
N UPDATE CASCADE,
robbersgang(#     CONSTRAINT HasSkillsSkillsFK FOREIGN KEY (SkillId) REFERENCES Skills(SkillId) ON DELETE RESTRICT ON UP
DATE CASCADE
robbersgang(# );
CREATE TABLE
```

```
robbersgang=# CREATE TABLE HasAccounts (
robbersgang(#     RobberId INT NOT NULL,
robbersgang(#     BankName VARCHAR(30) NOT NULL,
robbersgang(#     City VARCHAR(30) NOT NULL,
robbersgang(#     CONSTRAINT HasAccountsPK PRIMARY KEY (RobberId, BankName, City),
robbersgang(#     CONSTRAINT HasAccountsRobbersFK FOREIGN KEY (RobberId) REFERENCES Robbers(RobberId) ON DELETE RESTRICT
 ON UPDATE CASCADE,
robbersgang(#     CONSTRAINT HasAccountsBanksFK FOREIGN KEY (BankName, City) REFERENCES Banks(BankName, City) ON DELETE
 RESTRICT ON UPDATE CASCADE
robbersgang(# );
CREATE TABLE
```

```
robbersgang=# CREATE TABLE Accomplices (
robbersgang(#     RobberId INT NOT NULL,
robbersgang(#     BankName VARCHAR(30) NOT NULL,
robbersgang(#     City VARCHAR(30) NOT NULL,
robbersgang(#     Date DATE NOT NULL,
robbersgang(#     Share DECIMAL,
robbersgang(#     CONSTRAINT AccomplicesPK PRIMARY KEY (RobberId, BankName, City, Date),
robbersgang(#     CONSTRAINT AccomplicesRobbersFK FOREIGN KEY (RobberId) REFERENCES Robbers(RobberId
) ON DELETE RESTRICT ON UPDATE CASCADE,
robbersgang(#     CONSTRAINT AccomplicesRobberiesFK FOREIGN KEY (BankName, City, Date) REFERENCES Ro
bberies(BankName, City, Date) ON DELETE RESTRICT ON UPDATE CASCADE
robbersgang(# );
CREATE TABLE
```

```
robbersgang=# \dt
            List of relations
 Schema |    Name     | Type  |  Owner
--------+-------------+-------+----------
 public | accomplices | table | postgres
 public | banks       | table | postgres
 public | hasaccounts | table | postgres
 public | hasskills   | table | postgres
 public | plans       | table | postgres
 public | robberies   | table | postgres
 public | robbers     | table | postgres
 public | skills      | table | postgres
(8 rows)
```

Part 1 Populating table/Data Conversion:

1. Bank

```
robbersgang-# \copy Banks FROM C:/Users/msuban01/Downloads/datafiles/banks_23.data
COPY 20
```

Data output | Messages | Notifications

	bankname [PK] character varying (30)	city [PK] character varying (30)	noaccounts integer	security character varying (15)
1	Bad Bank	Chicago	6000	weak
2	Bankrupt Bank	Evanston	444000	weak
3	Dollar Grabbers	Chicago	56005	very good
4	Dollar Grabbers	Evanston	909090	good
5	Gun Chase Bank	Burbank	1999	weak
6	Gun Chase Bank	Evanston	656565	excellent
7	Hidden Treasure	Chicago	999999	excellent
8	Inter-Gang Bank	Chicago	100000	excellent
9	Inter-Gang Bank	Evanston	555555	excellent
10	Loanshark Bank	Chicago	121212	excellent
11	Loanshark Bank	Deerfield	3456789	very good
12	Loanshark Bank	Evanston	7654321	excellent
13	NXP Bank	Chicago	1593311	very good
14	NXP Bank	Evanston	656565	excellent
15	Outside Bank	Chicago	5000	good

Total rows: 20 of 20 | Query complete 00:00:00.282

2. Robberies

```
robbersgang-# \copy Robberies FROM C:/Users/msuban01/Downloads/datafiles/robberies_23.data
COPY 21
```

Data output | Messages | Notifications

	bankname [PK] character varying (30)	city [PK] character varying (30)	date [PK] date	amount numeric
1	Bad Bank	Chicago	2017-02-02	6020
2	Dollar Grabbers	Evanston	2017-06-28	3580
3	Dollar Grabbers	Evanston	2017-11-08	4380
4	Gun Chase Bank	Evanston	2016-04-30	18131.3
5	Inter-Gang Bank	Evanston	2016-02-16	72620
6	Inter-Gang Bank	Evanston	2017-03-13	92620
7	Inter-Gang Bank	Evanston	2018-02-14	52619
8	Loanshark Bank	Chicago	2017-11-09	41000
9	Loanshark Bank	Chicago	2019-03-30	21005
10	Loanshark Bank	Evanston	2016-04-20	20880
11	Loanshark Bank	Evanston	2017-04-20	10990
12	Loanshark Bank	Evanston	2019-02-28	19990
13	NXP Bank	Chicago	2019-01-08	34302.3
14	Penny Pinchers	Chicago	2016-08-30	900
15	Penny Pinchers	Evanston	2016-08-30	99000.8

Total rows: 21 of 21 | Query complete 00:00:00.300

3. Plans

```
robbersgang=# \copy plans FROM C:/Users/msuban01/Downloads/datafiles/plans_23.data
COPY 11
```

Data output | Messages | Notifications

	bankname [PK] character varying (30)	city [PK] character varying (30)	planneddate [PK] date	norobbers integer
1	Bad Bank	Chicago	2020-02-02	2
2	Dollar Grabbers	Chicago	2019-12-10	3
3	Gun Chase Bank	Evanston	2019-10-30	6
4	Hidden Treasure	Chicago	2020-01-11	5
5	Inter-Gang Bank	Evanston	2019-12-31	4
6	Loanshark Bank	Deerfield	2019-11-15	4
7	NXP Bank	Chicago	2019-10-10	5
8	NXP Bank	Chicago	2019-10-30	5
9	PickPocket Bank	Chicago	2020-03-10	2
10	PickPocket Bank	Deerfield	2019-11-30	6
11	PickPocket Bank	Deerfield	2019-12-15	6

Total rows: 11 of 11 | Query complete 00:00:01.526

4. Robbers

```
robbersgang=# \copy Robbers(nickname,age,noyears) From C:/Users/msuban01/Downloads/datafiles/robbers_23.data
COPY 24
```

Data output | Messages | Notifications

	robberid [PK] integer	nickname character varying (30)	age integer	noyears integer
1	1	Al Capone	31	2
2	2	Bugsy Malone	42	15
3	3	Lucky Luchiano	42	15
4	4	Anastazia	48	15
5	5	Mimmy The Mau Mau	18	0
6	6	Tony Genovese	28	16
7	7	Dutch Schulz	64	31
8	8	Clyde	20	0
9	9	Calamity Jane	44	3
10	10	Bonnie	19	0
11	11	Meyer Lansky	34	6
12	12	Moe Dalitz	41	3
13	13	Mickey Cohen	24	3
14	14	Kid Cann	14	0
15	15	Boo Boo Hoff	54	13

Total rows: 24 of 24 | Query complete 00:00:00.394

5. Skills

As mentioned in brief:
‘The list of robber skills uses the descriptions of the skills to uniquely identify them’ ‘a. A better design can be achieved if we introduce an additional Skills table listing all the possible skills and define a constraint on the HasSkills table to ensure that every skill there is also in the new Skills table’

We must derive the skill record from the hasSkill records through identification of unique description of skills.
To do this we create a tempskill table which holds the hasSkill data.
We then insert unique description from tempSkill table to the Skill table

```
robbersgang=# CREATE TABLE TEMPSKILLS(
robbersgang(# NickName VARCHAR(20),
robbersgang(# Description VARCHAR(20),
robbersgang(# Preference INT,
robbersgang(# Grade VARCHAR(2));
CREATE TABLE
robbersgang=# \copy tempskills From C:/Users/msuban01/Downloads/datafiles/hasskills_23.data
COPY 38
```

```
robbersgang=# INSERT INTO SKILLS(Description) SELECT DISTINCT Description FROM tempskills;
INSERT 0 12
```

Data output | Messages | Notifications

	skillid [PK] integer	description character varying (100)
1	1	Explosives
2	2	Guarding
3	3	Planning
4	4	Cooking
5	5	Gun-Shooting
6	6	Lock-Picking
7	7	Safe-Cracking
8	8	Preaching
9	9	Driving
10	10	Eating
11	11	Scouting
12	12	Money Counting

Total rows: 12 of 12 | Query complete 00:00:00.436

6. HasSkills

To populate the hasSkill table I join the skill table using the description attribute and the robbers table using the nickname attribute in tempskill and insert the value into hasSkill. After I delete the tempSkill table.

```
robbersgang=# INSERT INTO HASSKILLS(Robberid,skillid,Preference,Grade)
robbersgang-# SELECT ROBBERS.RobberId, SKILLS.SkillId, Preference, Grade FROM TEMPSKILLS
robbersgang-# INNER JOIN ROBBERS ON ROBBERS.NickName = TEMPSKILLS.NickName
robbersgang-# INNER JOIN SKILLS ON SKILLS.Description = TEMPSKILLS.Description;
INSERT 0 38
```

```
robbersgang=# DROP TABLE TEMPSKILLS
```

Data output | Messages | Notifications

	robberid [PK] integer	skillid [PK] integer	preference integer	grade character varying (5)
1	1	3	1	A+
2	1	7	2	C+
3	1	8	3	A+
4	2	1	1	A
5	3	6	1	B+
6	3	9	2	B+
7	4	2	1	A
8	5	3	1	A+
9	5	9	2	C
10	6	10	1	B+
11	7	6	1	A+
12	7	9	2	C+
13	8	3	3	C
14	8	6	1	C+
15	8	11	2	C+

Total rows: 38 of 38 Query complete 00:00:00.367

7. HasAccounts

To populate the HasAccounts table I created a tempaccount table to temporarily hold the data in hasaccount_23.
I then join the robbers table using nicknames from tempaccounts and insert the values into HasAccounts.
After I delete the tempaccount table.

```
robbersgang=# CREATE TABLE TEMPACCOUNT
robbersgang(# NickName VARCHAR(30),
robbersgang(# BankName VarCHAR(30),
robbersgang(# City VARCHAR(20));
CREATE TABLE
```

```
robbersgang=# \copy tempaccount FROM C:\users\msuban01\downloads\datafiles\hasaccounts_23.data
COPY 31
```

```
robbersgang=# INSERT INTO HASACCOUNTS(RobberId, BankName, City)
robbersgang-# SELECT ROBBERS.RobberId, BankName, City
robbersgang-# FROM tempaccount
robbersgang-# INNER JOIN ROBBERS ON ROBBERS.NickName = tempaccount.NickName;
INSERT 0 31
```

```
robbersgang=# DROP TABLE TEMPAccount;
DROP TABLE
```

Data output | Messages | Notifications

	robberid [PK] integer	bankname [PK] character varying (30)	city [PK] character varying (30)
1	1	Bad Bank	Chicago
2	1	Inter-Gang Bank	Evanston
3	1	NXP Bank	Chicago
4	2	Loanshark Bank	Chicago
5	2	Loanshark Bank	Deerfield
6	3	Bankrupt Bank	Evanston
7	3	NXP Bank	Chicago
8	4	Loanshark Bank	Evanston
9	5	Inter-Gang Bank	Evanston
10	5	Loanshark Bank	Evanston
11	7	Inter-Gang Bank	Chicago
12	8	Penny Pinchers	Evanston
13	9	Bad Bank	Chicago
14	9	Dollar Grabbers	Chicago
15	9	PickPocket Bank	Chicago

Total rows: 31 of 31 Query complete 00:00:00.603

8. Accomplices

To populate the Accomplices table I created a tempAccomplices table to temporarily hold the data in accomplices _23.
I then join the robbers table using nicknames from tempAccomplices and insert the values into Accomplices.
After I delete the tempAccomplices table.

```
robbersgang=# CREATE TABLE TEMPACCOMPLICES(
robbersgang(# NickName VARCHAR(30),
robbersgang(# BankName VarCHAR(30),
robbersgang(# City VARCHAR(30),
robbersgang(# Date DATE,
robbersgang(# Share DECIMAL);
CREATE TABLE
```

```
robbersgang=# \copy tempaccomplices FROM C:\users\msuban01\downloads\datafiles\accomplices_23.data
COPY 76
```

```
robbersgang=# INSERT INTO ACCOMPLICES(RobberId, BankName, City, Date, Share)
robbersgang-# SELECT ROBBERS.RobberId, BankName, City, Date, Share
robbersgang-# FROM TEMPACCOMPLICES
robbersgang-# INNER JOIN ROBBERS ON ROBBERS.NickName = TEMPACCOMPLICES.NickName;
INSERT 0 76
```

```
robbersgang=# DROP TABLE TEMPACCOMPLICES;
DROP TABLE
```

SUBANKAMO 300470761

Data output | Messages | Notifications

	robberid [PK] integer	bankname [PK] character varying (30)	city [PK] character varying (30)	date [PK] date	share numeric
1	1	Bad Bank	Chicago	2017-02-02	3010
2	1	Inter-Gang Bank	Evanston	2016-02-16	12103
3	1	Inter-Gang Bank	Evanston	2018-02-14	8769
4	1	Loanshark Bank	Chicago	2019-03-30	4201
5	1	Loanshark Bank	Evanston	2019-02-28	4997
6	1	NXP Bank	Chicago	2019-01-08	6406
7	2	NXP Bank	Chicago	2019-01-08	2300
8	3	Inter-Gang Bank	Evanston	2018-02-14	8769
9	3	Loanshark Bank	Chicago	2017-11-09	8200
10	3	Loanshark Bank	Chicago	2019-03-30	4201
11	3	Loanshark Bank	Evanston	2019-02-28	4997
12	3	Penny Pinchers	Evanston	2016-08-30	16500
13	4	Gun Chase Bank	Evanston	2016-04-30	3291.3
14	4	Inter-Gang Bank	Evanston	2018-02-14	8769
15	4	Loanshark Bank	Chicago	2019-03-30	4201

Total rows: 76 of 76 Query complete 00:00:00.484

Part 2 Order of Table Creation/Population:
I began by creating/populating tables that were parents of another table or did not require any joins (Bank, Robberies, Plans, Robbers, Skills).
This is so when I went to create tables (hasSkills, hasAccounts, Accomplices) that require information from these tables e.g robberID I could use joins to get these values from the parent table e.g nickname in accomplices table used to get robberID in robber table.
If I had done it the other way around it would be more time consuming as I would be missing data to populate the attributes of the child table.

SUBANKAMO 300470761

QUESTION 3: Checking your Database

Your answer to Question 3 should include: Your SQL statements for each task, the feedback from PostgreSQL, and the constraint that has been violated in case of an error message.

1. Insert the following tuple into the Skills table:
   a. (21, ‘Driving’)

```
robbersgang=# INSERT INTO SKILLS VALUES (21, 'Driving');
ERROR:  duplicate key value violates unique constraint "unique_column_constraint"
DETAIL:  Key (description)=(Driving) already exists.
```

2. Insert the following tuples into the Banks table:
   a. (‘Loanshark Bank’, ‘Evanston’, 100, ‘very good’)

```
robbersgang=# INSERT INTO Banks VALUES (‘Loanshark Bank’, ‘Evanston’, 100, ‘very good’);
ERROR:  duplicate key value violates unique constraint "bankspk"
DETAIL:  Key (bankname, city)=(Loanshark Bank, Evanston) already exists.
```

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

```
robbersgang=# INSERT INTO Banks VALUES (‘EasyLoan Bank’, ‘Evanston’, -5, ‘excellent’);
ERROR:  new row for relation "banks" violates check constraint "checknoaccountspositive"
DETAIL:  Failing row contains (EasyLoan Bank, Evanston, -5, excellent).
```

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

```
robbersgang=# INSERT INTO Banks VALUES (‘EasyLoan Bank’, ‘Evanston’, 100, ‘poor’);
ERROR:  new row for relation "banks" violates check constraint "checksecurityvalue"
DETAIL:  Failing row contains (EasyLoan Bank, Evanston, 100, poor).
```

3. Insert the following tuple into the Robberies table:
   a. (‘NXP Bank’, ‘Chicago’, ‘2019-01-08’, 1000)

```
robbersgang=# INSERT INTO Robberies VALUES (‘NXP Bank’, ‘Chicago’, ‘2019-01-08’, 1000);
ERROR:  duplicate key value violates unique constraint "robberiespk"
DETAIL:  Key (bankname, city, date)=(NXP Bank, Chicago, 2019-01-08) already exists.
```

4. Delete the following tuple from the Skills table:
   a. (1, ‘Driving’)
   >deleting does nothing when value 1, ‘Driving’ as ‘Driving’ has id=9

```
robbersgang=# DELETE FROM SKILLS WHERE SKillId=1 AND Description='Driving';
DELETE 0
robbersgang=# DELETE FROM SKILLS WHERE SKillId=9 AND Description='Driving';
ERROR:  update or delete on table "skills" violates foreign key constraint "hasskillsskillsfk" on ta
ble "hasskills"
DETAIL:  Key (skillid)=(9) is still referenced from table "hasskills".
```

5. Delete the following tuples from the Banks table: a. (‘PickPocket Bank’, ‘Evanston’, 2000, ‘very good’)

```
robbersgang=# 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 "hasaccountsbanksfk" on ta
ble "hasaccounts"
DETAIL:  Key (bankname, city)=(PickPocket Bank, Evanston) is still referenced from table "hasaccount
s".
robbersgang=#
```

6. Delete the following tuple from the Robberies table:
   a. (‘Loanshark Bank’, ‘Chicago’, ‘’, ‘’)

```
robbersgang=# DELETE FROM ROBBERIES WHERE BankName='Loanshark Bank' AND City='Chicago';
ERROR:  update or delete on table "robberies" violates foreign key constraint "accomplicesrobberiesfk" on table "accompl
ices"
DETAIL:  Key (bankname, city, date)=(Loanshark Bank, Chicago, 2017-11-09) is still referenced from table "accomplices".
robbersgang=#
```

7. Insert the following tuples into the Robbers table:

a. (1, ‘Shotgun’, 70, 0)

```
robbersgang=# INSERT INTO Robbers VALUES (1, ‘Shotgun’, 70, 0);
ERROR:  duplicate key value violates unique constraint "robberspk"
DETAIL:  Key (robberid)=(1) already exists.
robbersgang=#
```

b. (999, ‘Jail Mouse’, 25, 35)

```
robbersgang=# INSERT INTO Robbers VALUES (999, ‘Jail Mouse’, 25, 35);
ERROR:  new row for relation "robbers" violates check constraint "checknoyearpositiveandlessthanage"
DETAIL:  Failing row contains (999, Jail Mouse, 25, 35).
robbersgang=#
```

8. Insert the following tuples into the HasSkills table

a. (1, 7, 1, ‘A+’)

```
robbersgang=# INSERT INTO HASSKILLS VALUES (
robbersgang(# 1, 7, 1, ‘A+’);
ERROR:  duplicate key value violates unique constraint "hasskillspk"
DETAIL:  Key (robberid, skillid)=(1, 7) already exists.
```

b. (1, 2, 0, ‘A’)

```
robbersgang=# INSERT INTO HASSKILLS VALUES (
robbersgang(# 1, 2, 0, ‘A’);
ERROR:  new row for relation "hasskills" violates check constraint "checkpreferencebetween1and3"
DETAIL:  Failing row contains (1, 2, 0, A).
```

c. (999, 1, 1, ‘B-’)

```
robbersgang=# INSERT INTO HASSKILLS VALUES (
robbersgang(# 999, 1, 1, ‘B-’);
ERROR:  insert or update on table "hasskills" violates foreign key constraint "hasskillsrobbersfk"
DETAIL:  Key (robberid)=(999) is not present in table "robbers".
```

d. (3, 20, 3, ‘B+’)

```
robbersgang=# INSERT INTO HASSKILLS VALUES (
robbersgang(# 3, 20, 3, ‘B+’);
ERROR:  insert or update on table "hasskills" violates foreign key constraint "hasskillsskillsfk"
DETAIL:  Key (skillid)=(20) is not present in table "skills".
```

9. Delete the following tuple from the Robbers table:

a. (1, ‘Al Capone’, 31, 2)

```
robbersgang=# 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 "hasaccountsrobbersfk" on table "hasaccounts
"
DETAIL:  Key (robberid)=(1) is still referenced from table "hasaccounts".
```

SUBANKAMO 300470761

QUESTION 4: Simple Database Queries

Your answer to Question 4 should include:

• Your SQL statement for each task, and the answer from PostgreSQL.

• Also, submit your SQL queries, with each query (just SQL code) as a separate .sql file. Name files in the following way: Question4_TaskX.sql, where X stands for the task number 1, 2, …

1. Retrieve BankName and City of all banks that have never been robbed.

```
robbersgang=# SELECT BankName, City From BANKS
robbersgang-# WHERE (BankName, City)
robbersgang-# NOT IN(SELECT BankName, City FROM ROBBERIES);
    bankname     |   city
-----------------+----------
 Bankrupt Bank   | Evanston
 Loanshark Bank  | Deerfield
 Inter-Gang Bank | Chicago
 NXP Bank        | Evanston
 Dollar Grabbers | Chicago
 Gun Chase Bank  | Burbank
 PickPocket Bank | Deerfield
 Hidden Treasure | Chicago
 Outside Bank    | Chicago
(9 rows)
```

2. Retrieve RobberId, Nickname, Age, and all skill descriptions of all robbers who are older than 40 years old.

```
robbersgang=# SELECT RobberId, NickName, Age, Description
robbersgang-# From ROBBERS
robbersgang-# NATURAL JOIN HASSKILLS
robbersgang-# NATURAL JOIN SKILLS
robbersgang-# WHERE Age > 40;
 robberid |    nickname    | age |  description
----------+----------------+-----+---------------
        4 | Anastazia      |  48 | Guarding
       15 | Boo Boo Hoff   |  54 | Planning
        2 | Bugsy Malone   |  42 | Explosives
       17 | Bugsy Siegel   |  48 | Driving
       17 | Bugsy Siegel   |  48 | Guarding
        9 | Calamity Jane  |  44 | Gun-Shooting
        7 | Dutch Schulz   |  64 | Lock-Picking
        7 | Dutch Schulz   |  64 | Driving
       16 | King Solomon   |  74 | Planning
        3 | Lucky Luchiano |  42 | Lock-Picking
        3 | Lucky Luchiano |  42 | Driving
       12 | Moe Dalitz     |  41 | Safe-Cracking
       18 | Vito Genovese  |  66 | Scouting
       18 | Vito Genovese  |  66 | Cooking
       18 | Vito Genovese  |  66 | Eating
(15 rows)
```

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

```
robbersgang=# SELECT BankName, City From BANKS NATURAL JOIN HASACCOUNTS NATURAL JOIN ROBBERS WHERE NickName = 'Al Capon
e';
    bankname     |   city
-----------------+----------
 Bad Bank        | Chicago
 Inter-Gang Bank | Evanston
 NXP Bank        | Chicago
(3 rows)
```

4. Retrieve BankName and City and NoAccounts of all banks with no branch in Chicago. The answer should be sorted in increasing order of the number of accounts.

```
robbersgang=# SELECT BankName, City, NoAccounts
robbersgang-# From BANKS
robbersgang-# WHERE BankName NOT IN (SELECT BankName FROM BANKS WHERE City =
robbersgang(# 'Chicago')
robbersgang-# ORDER BY NoAccounts;
    bankname     |   city    | noaccounts
-----------------+-----------+------------
 Gun Chase Bank | Burbank   |       1999
 Bankrupt Bank  | Evanston  |     444000
 Gun Chase Bank | Evanston  |     656565
(3 rows)
```

5. Retrieve RobberId, Nickname and individual total “earnings” of those robbers who have earned more than $40,000 by robbing banks. The answer should be sorted in decreasing order of the total earnings.

```
robbersgang=# SELECT RobberId, NickName, Earning
robbersgang-# From (SELECT RobberId, SUM(Share) AS Earning FROM ACCOMPLICES GROUP BY
robbersgang(# RobberId) AS Total
robbersgang-# NATURAL JOIN ROBBERS
robbersgang-# WHERE Earning > 40000
robbersgang-# ORDER BY Earning DESC;
 robberid |      nickname      | earning
----------+--------------------+----------
        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
(6 rows)
```

6. Retrieve RobberId, NickName, and the Number of Years in prison for all robbers who were imprisoned for more than ten years.

```
robbersgang=# SELECT RobberId, NickName, NoYears
robbersgang-# From ROBBERS
robbersgang-# WHERE NoYears>10;
 robberid |    nickname     | noyears
----------+-----------------+---------
        2 | Bugsy Malone    |      15
        3 | Lucky Luchiano  |      15
        4 | Anastazia       |      15
        6 | Tony Genovese   |      16
        7 | Dutch Schulz    |      31
       15 | Boo Boo Hoff    |      13
       16 | King Solomon    |      43
       17 | Bugsy Siegel    |      13
(8 rows)
```

7. 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.

```
robbersgang=# SELECT RobberId, NickName, (Age - NoYears) AS NoYearsNotInPrison
robbersgang-# From ROBBERS
robbersgang-# WHERE (Age/2)<NoYears;
 robberid |   nickname    | noyearsnotinprison
----------+---------------+--------------------
        6 | Tony Genovese |                 12
       16 | King Solomon  |                 31
(2 rows)
```

8. 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.

```
robbersgang=# SELECT Description, RobberId, NickName
robbersgang-# From ROBBERS
robbersgang-# NATURAL JOIN HASSKILLS
robbersgang-# NATURAL JOIN SKILLS
robbersgang-# ORDER BY Description;
  description   | robberid |     nickname
----------------+----------+------------------
 Cooking        |       18 | Vito Genovese
 Driving        |        3 | Lucky Luchiano
 Driving        |        7 | Dutch Schulz
 Driving        |        5 | Mimmy The Mau Mau
 Driving        |       23 | Lepke Buchalter
 Driving        |       20 | Longy Zwillman
 Driving        |       17 | Bugsy Siegel
 Eating         |       18 | Vito Genovese
 Eating         |        6 | Tony Genovese
 Explosives     |       24 | Sonny Genovese
 Explosives     |        2 | Bugsy Malone
 Guarding       |       23 | Lepke Buchalter
 Guarding       |        4 | Anastazia
 Guarding       |       17 | Bugsy Siegel
 Gun-Shooting   |        9 | Calamity Jane
 Gun-Shooting   |       21 | Waxey Gordon
 Lock-Picking   |        8 | Clyde
 Lock-Picking   |       22 | Greasy Guzik
 Lock-Picking   |       24 | Sonny Genovese
 Lock-Picking   |        3 | Lucky Luchiano
 Lock-Picking   |        7 | Dutch Schulz
 Money Counting |       14 | Kid Cann
 Money Counting |       13 | Mickey Cohen
 Money Counting |       19 | Mike Genovese
 Planning       |        5 | Mimmy The Mau Mau
 Planning       |       15 | Boo Boo Hoff
 Planning       |        1 | Al Capone
 Planning       |        8 | Clyde
 Planning       |       16 | King Solomon
 Preaching      |       22 | Greasy Guzik
 Preaching      |        1 | Al Capone
 Preaching      |       10 | Bonnie
 Safe-Cracking  |       11 | Meyer Lansky
 Safe-Cracking  |       12 | Moe Dalitz
 Safe-Cracking  |        1 | Al Capone
 Safe-Cracking  |       24 | Sonny Genovese
 Scouting       |        8 | Clyde
 Scouting       |       18 | Vito Genovese
(38 rows)
```

SUBANKAMO 300470761

QUESTION 5: Complex Database Queries

Your answer to Question 5 should include:

• Your SQL statement for each task, and the answer from PostgreSQL.

• Also, submit your SQL queries, with each query (just SQL code) as a separate .sql file. Name files in the following way: Question4_TaskX.sql, where X stands for the task number 1, 2, …

1. Retrieve BankName and City of all banks that were not robbed in the year, in which there were robbery plans for that bank.

```
robbersgang=# SELECT BankName,City FROM(
robbersgang(# SELECT BankName, City, (SELECT EXTRACT (YEAR FROM PlannedDate)) FROM PLANS
robbersgang(# EXCEPT
robbersgang(# SELECT BankName, City, (SELECT EXTRACT (YEAR FROM Date)) FROM ROBBERIES)
robbersgang-# AS Foo;
    bankname     |   city
-----------------+----------
 Hidden Treasure | Chicago
 Gun Chase Bank  | Evanston
 Loanshark Bank  | Deerfield
 Dollar Grabbers | Chicago
 Inter-Gang Bank | Evanston
 PickPocket Bank | Chicago
 PickPocket Bank | Deerfield
 Bad Bank        | Chicago
(8 rows)
```

2. Retrieve RobberId and Nickname of all robbers who never robbed the banks at which they have an account.

```
robbersgang=# SELECT RobberId, NickName FROM ROBBERS
robbersgang-# EXCEPT
robbersgang-# SELECT RobberId, NickName FROM HASACCOUNTS
robbersgang-# NATURAL JOIN ACCOMPLICES
robbersgang-# NATURAL JOIN ROBBERS;
 robberid |    nickname
----------+-----------------
       14 | Kid Cann
       16 | King Solomon
       21 | Waxey Gordon
        7 | Dutch Schulz
       23 | Lepke Buchalter
       10 | Bonnie
       13 | Mickey Cohen
        6 | Tony Genovese
       24 | Sonny Genovese
       19 | Mike Genovese
        2 | Bugsy Malone
       12 | Moe Dalitz
       15 | Boo Boo Hoff
        4 | Anastazia
        9 | Calamity Jane
        3 | Lucky Luchiano
(16 rows)
```

3. Retrieve RobberId, Nickname, and Description of the first preferred skill of all robbers who have two or more skills.

```
robbersgang=# SELECT RobberId, NickName, Description
robbersgang-# FROM (SELECT RobberId FROM HASSKILLS GROUP BY(RobberId) HAVING COUNT(RobberId) >=2) as TwoSkill
robbersgang-# NATURAL JOIN ROBBERS
robbersgang-# NATURAL JOIN SKILLS
robbersgang-# NATURAL JOIN HASSKILLS
robbersgang-# WHERE Preference = 1;
 robberid |     nickname      | description
----------+-------------------+-------------
        1 | Al Capone         | Planning
        5 | Mimmy The Mau Mau | Planning
       17 | Bugsy Siegel      | Driving
        8 | Clyde             | Lock-Picking
        7 | Dutch Schulz      | Lock-Picking
       22 | Greasy Guzik      | Preaching
       23 | Lepke Buchalter   | Driving
        3 | Lucky Luchiano    | Lock-Picking
       24 | Sonny Genovese    | Explosives
       18 | Vito Genovese     | Scouting
(10 rows)
```

4. Retrieve BankName, City and Date of all robberies in the city that observes the highest Share among all robberies.

```
robbersgang=# SELECT BankName, City, Date FROM ROBBERIES WHERE (City, Amount) = ANY(SELECT City, MAX(Amount) FROM ROBBE
RIES GROUP BY(City));
    bankname    |   city   |    date
----------------+----------+------------
 Penny Pinchers | Evanston | 2016-08-30
 Loanshark Bank | Chicago  | 2017-11-09
(2 rows)
```

5. Retrieve BankName and City of all banks that were robbed by all robbers.

```
robbersgang=# SELECT BankName, City FROM ACCOMPLICES
robbersgang-# GROUP BY(BankName, City)
robbersgang-# HAVING COUNT(DISTINCT RobberId) = (SELECT COUNT(RobberId) FROM ROBBERS);
 bankname | city
----------+------
(0 rows)
```

SUBANKAMO 300470761

QUESTION 6: Even More Database Queries

Your answer to Question 5 should include:

• A sequence of SQL statements for the basic queries and the views/tables you created, and the output of the final query.

• A single nested SQL query, with its output from PostgreSQL (hopefully the same).

• Also, submit your SQL nested queries, with each nested query (just SQL code) as a separate sql file. Name files in the following way: Query6_TaskX.sql, where X stands for the task number 1, 2, …

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.

SEQUENCE Part 1:

Step 1:

Create an unpenalisedRobbers view which contains all robbers and their robberies who have not been in jail.

To do this we join the Robbers and accomplices table using common columns, filter only records where NoYears is 0 and save it to unpenalisedRobbers view

```
robbersgang=# CREATE VIEW UNPENALISEDROBBERS AS SELECT * from ACCOMPLICES NATURAL JOIN ROBBERS
robbersgang-# WHERE NoYears = 0;
CREATE VIEW
```

```
robbersgang=# SELECT * FROM UNPENALISEDROBBERS
robbersgang-# ;
 robberid |    bankname     |   city   |    date    | share |     nickname      | age | noyears
----------+-----------------+----------+------------+-------+-------------------+-----+---------
        5 | Inter-Gang Bank | Evanston | 2017-03-13 | 60000 | Mimmy The Mau Mau |  18 |       0
        5 | Loanshark Bank  | Evanston | 2016-04-20 | 10000 | Mimmy The Mau Mau |  18 |       0
        8 | Penny Pinchers  | Evanston | 2016-08-30 | 16500 | Clyde             |  20 |       0
        8 | Penny Pinchers  | Chicago  | 2016-08-30 |   450 | Clyde             |  20 |       0
        8 | Loanshark Bank  | Evanston | 2017-04-20 |  2747 | Clyde             |  20 |       0
        8 | Inter-Gang Bank | Evanston | 2016-02-16 | 12103 | Clyde             |  20 |       0
       10 | Penny Pinchers  | Evanston | 2016-08-30 | 16500 | Bonnie            |  19 |       0
       10 | Loanshark Bank  | Chicago  | 2017-11-09 |  8200 | Bonnie            |  19 |       0
       10 | Inter-Gang Bank | Evanston | 2016-02-16 | 12103 | Bonnie            |  19 |       0
       10 | Gun Chase Bank  | Evanston | 2016-04-30 |  3282 | Bonnie            |  19 |       0
       14 | Dollar Grabbers | Evanston | 2017-06-28 |  1790 | Kid Cann          |  14 |       0
       18 | Dollar Grabbers | Evanston | 2017-06-28 |  1790 | Vito Genovese     |  66 |       0
       18 | Bad Bank        | Chicago  | 2017-02-02 |  3010 | Vito Genovese     |  66 |       0
       18 | Dollar Grabbers | Evanston | 2017-11-08 |  2000 | Vito Genovese     |  66 |       0
       21 | Penny Pinchers  | Evanston | 2019-05-30 | 3250.1 | Waxey Gordon     |  15 |       0
       21 | Loanshark Bank  | Evanston | 2019-02-28 |  4997 | Waxey Gordon      |  15 |       0
       21 | Loanshark Bank  | Chicago  | 2017-11-09 |  8200 | Waxey Gordon      |  15 |       0
       24 | PickPocket Bank | Evanston | 2018-01-30 |   500 | Sonny Genovese    |  39 |       0
       24 | PickPocket Bank | Evanston | 2016-03-30 |  2000 | Sonny Genovese    |  39 |       0
       24 | PickPocket Bank | Chicago  | 2015-09-21 |   681 | Sonny Genovese    |  39 |       0
       24 | Penny Pinchers  | Evanston | 2017-10-30 |  3000 | Sonny Genovese    |  39 |       0
       24 | Loanshark Bank  | Chicago  | 2019-03-30 |  4201 | Sonny Genovese    |  39 |       0
       24 | Gun Chase Bank  | Evanston | 2016-04-30 |  3282 | Sonny Genovese    |  39 |       0
(23 rows)
```

Step 2:

Create an robbersStatistic view which contains the total number of robberies committed by the robbers and the average

To do this we group the record by robberID and nickname and create records containing RobberId, NickName, numbers of robberies committed, total amount stolen and the average robberies committed by every robber. We then save these records to robbersStatistic view.

```
robbersgang=# CREATE VIEW robbersStatistic AS SELECT RobberId, NickName, COUNT(RobberId) as robberyCount, SUM(Share)
 totalShare, AVG(COUNT(RobberId)) over () AS averageRobberiesCommited
robbersgang-# from unpenalisedRobbers GROUP BY(RobberId, NickName);
CREATE VIEW
robbersgang=# SELECT * FROM RobbersStatistic
robbersgang-# ;
 robberid |     nickname      | robberycount | totalshare | averagerobberiescommited
----------+-------------------+--------------+------------+-------------------------
        5 | Mimmy The Mau Mau |            2 |      70000 |       3.2857142857142857
        8 | Clyde             |            4 |      31800 |       3.2857142857142857
       10 | Bonnie            |            4 |      40085 |       3.2857142857142857
       14 | Kid Cann          |            1 |       1790 |       3.2857142857142857
       18 | Vito Genovese     |            3 |       6800 |       3.2857142857142857
       21 | Waxey Gordon      |            3 |    16447.1 |       3.2857142857142857
       24 | Sonny Genovese    |            6 |      13664 |       3.2857142857142857
(7 rows)
```

Step 3:
Create an mostActiveunpenalisedRobber view which contains the nickname of robbers who have committed more robberies than the average and have not been penalised and ordered by a descending amount of money stolen.
To do this we select records from robbersStatistic where the number of robberies committed is greater than average robberies committed and then order by descending amount of total amount stolen.

```
robbersgang=# CREATE VIEW MostActiveUnpenalisedRobber AS SELECT NickName from robbersStatistic
robbersgang-# WHERE robberyCount > averagerobberiescommited
robbersgang-# ORDER BY totalshare DESC;
CREATE VIEW
```

```
robbersgang=# SELECT * FROM MostActiveUnpenalisedRobber;
    nickname
----------------
 Bonnie
 Clyde
 Sonny Genovese
(3 rows)
```

NESTED Part 2:

```
robbersgang=# SELECT NickName FROM( SELECT RobberId, NickName, COUNT(RobberId) as robberyCount, SUM(Share) as totalShar
e,AVG(COUNT(RobberId)) over () AS averageRobberiesCommitted from ACCOMPLICES
robbersgang(# NATURAL JOIN ROBBERS
robbersgang(# WHERE NoYears = 0
robbersgang(# GROUP BY(RobberId, NickName)) AS unpenalisedRobbersStatistic
robbersgang-# WHERE robberyCount > averageRobberiesCommitted
robbersgang-# ORDER BY totalShare 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 to 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.

SEQUENCE Part 1:

Step 1:

Create a BankRobberiesStatistic view which contains all bank and robberies information.

To do this we join the Banks and accomplices table using common columns and save it to BankRobberiesStatistic view.

```
robbersgang=# CREATE VIEW BankRobberiesStatistic AS SELECT * FROM BANKS
robbersgang-# NATURAL JOIN ROBBERIES;
CREATE VIEW
robbersgang=# SELECT * FROM BankRobberiesStatistic;
    bankname     |   city   | noaccounts | security  |    date    | amount
-----------------+----------+------------+-----------+------------+---------
 NXP Bank        | Chicago  |    1593311 | very good | 2019-01-08 | 34302.3
 Loanshark Bank  | Evanston |    7654321 | excellent | 2019-02-28 |   19990
 Loanshark Bank  | Chicago  |     121212 | excellent | 2019-03-30 |   21005
 Inter-Gang Bank | Evanston |     555555 | excellent | 2018-02-14 |   52619
 Penny Pinchers  | Chicago  |     156165 | weak      | 2016-08-30 |     900
 Penny Pinchers  | Evanston |     130013 | excellent | 2016-08-30 | 99000.8
 Gun Chase Bank  | Evanston |     656565 | excellent | 2016-04-30 | 18131.3
 PickPocket Bank | Evanston |       2000 | very good | 2016-03-30 | 2031.99
 PickPocket Bank | Chicago  |     130013 | weak      | 2018-02-28 |     239
 Loanshark Bank  | Evanston |    7654321 | excellent | 2017-04-20 |   10990
 Inter-Gang Bank | Evanston |     555555 | excellent | 2016-02-16 |   72620
 Penny Pinchers  | Evanston |     130013 | excellent | 2017-10-30 |  9000.5
 PickPocket Bank | Evanston |       2000 | very good | 2018-01-30 |  542.99
 Loanshark Bank  | Chicago  |     121212 | excellent | 2017-11-09 |   41000
 Penny Pinchers  | Evanston |     130013 | excellent | 2019-05-30 | 13000.4
 PickPocket Bank | Chicago  |     130013 | weak      | 2015-09-21 |    2039
 Loanshark Bank  | Evanston |    7654321 | excellent | 2016-04-20 |   20880
 Inter-Gang Bank | Evanston |     555555 | excellent | 2017-03-13 |   92620
 Dollar Grabbers | Evanston |     909090 | good      | 2017-11-08 |    4380
 Dollar Grabbers | Evanston |     909090 | good      | 2017-06-28 |    3580
 Bad Bank        | Chicago  |       6000 | weak      | 2017-02-02 |    6020
(21 rows)
```

Step 2:

Create a security statistic view which contains security level, total number of robberies committed at banks with the security level and the average amount stolen from banks with the security level.

To do this we group the records in BankRobberiesStatistic by security we then calculate the total robberies and average amount stolen for each group and save it to securityStatistic view.

```
robbersgang=# CREATE VIEW SecurityStatistics AS SELECT Security AS SecurityLevel, COUNT(Security) as totalRobberies, AV
G(Amount) AS averageAmountStolen FROM BankRobberiesStatistic
robbersgang-# GROUP BY Security;
CREATE VIEW
```

```
robbersgang=# SELECT * FROM SecurityStatistics;
 securitylevel | totalrobberies |  averageamountstolen
---------------+----------------+-----------------------
 weak          |              4 |  2299.5000000000000000
 good          |              2 |  3980.0000000000000000
 very good     |              3 | 12292.4266666666666667
 excellent     |             12 |     39238.083333333333
(4 rows)
```

NESTED Part 2:

```
robbersgang=# SELECT Security AS SecurityLevel, COUNT(Security) as totalRobberies, AVG(Amount) AS averageAmountStolen F
ROM BANKS
robbersgang-# NATURAL JOIN ROBBERIES
robbersgang-# GROUP BY Security;
 securitylevel | totalrobberies |  averageamountstolen
---------------+----------------+-----------------------
 weak          |              4 |  2299.5000000000000000
 good          |              2 |  3980.0000000000000000
 very good     |              3 | 12292.4266666666666667
 excellent     |             12 |     39238.083333333333
(4 rows)
```