## QUESTION 1: Defining the Database [16 marks]

**banks** (bank_name, city, no_accounts, security)

**Primary key** {bank_name, city}

**Constraints:**

1. The values in *no_accounts* cannot negative value (CHECK)
2. The values in *security* must be one of this input: {weak, excellent, very good, good} to ensure data consistency (CHECK)

The primary key is {bank_name, city} as we cannot take any attributes away and still remain a minimal superkey.

**robberies** (bank_name, city, robbery_date, amount)

**Primary key** {bank_name, city, robbery_date}

**Foreign key** bank_name, city ⊆ banks[bank_name, city]

Robberies can occur in a bank many times, so the combination of {bank_name, city} is not a minimal superkey. Thus, we add the attribute *robbery_date* as the composite primary key can uniquely identify each tuple in the relation. The one and only foreign key is [bank_name, city] which refers to the primary key in their parent table, banks.

**plans** (bank_name, city, no_robbers, planned_date)

**Primary key** {bank_name, city, planned_date}

**Foreign key** bank_name, city ⊆ banks[bank_name, city]

**Constraints:**

1. The values in *no_robbers* must not be negative value

A bank could be robbed many times, so the combination {bank_name, city} is not a minimal superkey. Thus, we add the attribute *planned_date* to the composite primary key as the possibility of same bank to be robbed twice on the same date is highly unlikely. The foreign key is same with the robberies table, which refers to their parent table, banks.

**robbers (**robber_id, nickname, age, no_years)

**Primary key** {robber_id}

**Constraints:**

1. The values in *age* column must be larger than zero (CHECK)
2. The values in *no_years* must be smaller or equal to age, as it is impossible to be in prison for more years than you have been alive. (CHECK)

The date type of the *robber_id* is serial as we want auto incrementing the integer when a new tuple is inserted into the table. Since the values of *nickname*, *age* and *no_years* can be duplicated in the columns, the only suitable primary key is {robber_id}.

**skills** (skill_id, description)

**Primary key** {skill_id}

**Constraints:**

1. To ensure the uniqueness of values in column *description* so that there is no duplicate skill's description with different skill_id. (UNIQUE)

The date type of the *skill_id* is serial as we want auto incrementing the integer when a new tuple is inserted into the table. The skill_id is a minimal superkey as removing the description we can still uniquely identify each tuple and it is redundant to have both attributes as primary key.

**has_skills** (robber_id, skill_id, preference, grade)

**Primary key** {robber_id, skill_id}

**Foreign key** skill_id ⊆ skills[skill_id]
robber_id ⊆ robbers[robber_id]

**Constraints:**

1. The values in preference must be larger than zero (CHECK)
2. The preferences of each robbery must be in ranking order, e.g. a robber has two skills: driving, guarding and his preferences could be first is driving and second is guarding and both skills cannot have same preference number. Thus, the preference numbers must be 1 and 2 (CHECK)

A robber can have many skills, which means *robber_id* can be duplicated. Thus, we have to have composite primary key to uniquely identify each tuple. The foreign keys are both references to the primary keys in their parent tables. The [robber_id] refers to the column *robber_id* in *robbers* table, while [skill_id] refers to the column *skill_id* in *skills* table.

**has_accounts** (robber_id, bank_name, city)

**Primary key** {robber_id, bank_name, city}

**Foreign key** bank_name, city ⊆ banks[bank_name, city]
robber_id ⊆ robbers[robber_id]

A robber can have many bank accounts and a bank can have many accounts, so the subsets of {bank_name, city}, {robber_id, bank_name}, {robber_id, city} are not suitable primary keys. Thus, the only suitable primary key is the combination of all three attributes.

**accomplices** (robber_id, bank_name, city, robbery_date, share)

**Primary key** {robber_id, bank_name, city, robbery_date}

**Foreign key** bank_name, city ⊆ banks[bank_name, city]
robber_id ⊆ robbers[robber_id]

A robber could involved in many robberies, and a robber could also robbed a same bank multiple times. Thus, any subsets of the attributes {robber_id, bank_name, city, share} are not suitable candidate keys, and the only remaining set is {robber_id, bank_name, city, robbery_date} as we could remove the attribute *share* from this {robber_id, bank_name, city, robbery_date, share} set and still remain a minimal superkey.

I did not add any CASCADE constraints on the foreign keys, as I think delete operations should be restricted unless the data need to be updating/deleting all the time and we could always alter the tables if we need to delete a tuple and its child tuples altogether. Using ALTER TABLE table_name ADD [constraint_name] [constraint_type] [constraint_condition];.

```
-- Instead of using the text data type, we can use the citext (case insensitive text) type.
-- First we need to enable the citext extension: CREATE EXTENSION IF NOT EXISTS citext;
-- Unfortunately, the PostgreSQL in the lab could not open extension control file as no such file added.

CREATE TABLE banks(
  bank_name VARCHAR(80) NOT NULL,
  city VARCHAR(25) NOT NULL,
  no_accounts int DEFAULT 0,
  security VARCHAR(12),
  PRIMARY KEY (bank_name, city),
  CONSTRAINT check_account_num CHECK (no_accounts >= 0),
  CONSTRAINT check_security_input CHECK (security IN ('weak', 'excellent', 'very good', 'good'))
);

CREATE TABLE robberies(
  bank_name VARCHAR(80) NOT NULL,
  city VARCHAR(25) NOT NULL,
  robbery_date DATE NOT NULL,
  amount NUMERIC(12, 2),
  PRIMARY KEY (bank_name, city, robbery_date),
  FOREIGN KEY (bank_name, city) REFERENCES banks(bank_name, city)
);

CREATE TABLE plans(
  bank_name VARCHAR(80) NOT NULL,
  city VARCHAR(25) NOT NULL,
  planned_date DATE NOT NULL,
  no_robbers int,
  PRIMARY KEY (bank_name, city, planned_date),
  FOREIGN KEY (bank_name, city) REFERENCES banks(bank_name, city),
  CONSTRAINT check_no_robbers CHECK (no_robbers >= 0)
);

CREATE TABLE robbers(
  robber_id SERIAL NOT NULL,
  nickname VARCHAR(70) NOT NULL UNIQUE,
  age int,
  no_years int DEFAULT 0,
  PRIMARY KEY (robber_id),
  CONSTRAINT check_age CHECK (age > 0),
  CONSTRAINT check_prison_years CHECK (no_years <= age)
);

CREATE TABLE has_accounts(
  robber_id int NOT NULL,
  bank_name VARCHAR(80) NOT NULL,
  city VARCHAR(25) NOT NULL,
  PRIMARY KEY (robber_id, bank_name, city),
  FOREIGN KEY (robber_id) REFERENCES robbers(robber_id),
  FOREIGN KEY (bank_name, city) REFERENCES banks(bank_name, city)
);
```

```
CREATE TABLE skills(
  skill_id SERIAL NOT NULL,
  description VARCHAR(25) NOT NULL UNIQUE,
  PRIMARY KEY (skill_id),
  CONSTRAINT must_be_different UNIQUE (description)
);

-- This function returns the maximum preference number (ranking order) of a robber.
CREATE FUNCTION get_max(id int) RETURNS int AS
$max$
   DECLARE
      total int;
   BEGIN
      SELECT COUNT(preference)
      FROM has_skills
      WHERE robber_id = id
      GROUP BY robber_id INTO total;
      RETURN total;
   END;
$max$
LANGUAGE plpgsql;

CREATE TABLE has_skills(
  robber_id int NOT NULL,
  skill_id int NOT NULL,
  preference int,
  grade CHAR(2),
  PRIMARY KEY (robber_id, skill_id),
  FOREIGN KEY (robber_id) REFERENCES robbers(robber_id),
  FOREIGN KEY (skill_id) REFERENCES skills(skill_id),
  CONSTRAINT check_preference_input CHECK (preference > 0),
  CONSTRAINT check_preference_num CHECK (preference = get_max(robber_id)+1)
);

CREATE TABLE accomplices(
  robber_id int NOT NULL,
  bank_name VARCHAR(80) NOT NULL,
  city VARCHAR(25) NOT NULL,
  robbery_date DATE NOT NULL,
  share NUMERIC(12,2),
  PRIMARY KEY (robber_id, bank_name, city, robbery_date),
  FOREIGN KEY (robber_id) REFERENCES robbers(robber_id),
  FOREIGN KEY (bank_name, city) REFERENCES banks(bank_name, city)
);
```

## QUESTION 2: Populating your Database with Data [15 marks]

1. A description of how you performed all the data conversion, for example, a sequence of the PostgreSQL statements that accomplished the conversion. [12 points]

The data conversion for banks, plans, robberies and robbers are simple and easy. We could just simply use the \copy command to insert the data as tuples into the tables. Below is the following code I use to insert the data, we could start either with the banks or robbers data, as they both do not have any foreign key constraints. Thus, the banks data must inserted before the plans and robberies data.
**banks** => \copy banks from 'banks_17.data'
**robberies** => \copy robberies from 'robberies_17.data'
**plans** => \copy plans (bank_name, city, planned_date, no_robbers) from 'plans_17.data'
**robbers** => \copy robbers(nickname, age, no_years) from 'robbers_17.data

To insert data into the has_skills table, first I have to insert the data from the file into a raw data table called has_skills_raw. After that, I inserted the values of skill description into the skills table using the skill column in the has_skills_raw table. Next, I updated the skill_id and robber_id columns using the skills and robbers table respectively. Before inserting the tuples into the has_skill table, I used SELECT statement to ensure there is no null values in the skill_id and robber_id columns. Lastly, I inserted all the tuples in the has_skills_raw into the has_skills table. As for the accomplices and has_accounts data, I applied the same method that I mentioned above by creating has_account raw and accomplices_raw tables. Below is the sequence of PostgreSQL statements that I used to perform data conversion for has_skills, skills, has_accounts and accomplices tables.

```
-- create three extra tables for the raw data
CREATE TABLE has_skills_raw(
  robber_id int,
  nickname VARCHAR(70) NOT NULL,
  skill VARCHAR(25) NOT NULL,
  skill_id int,
  preference int,
  grade CHAR(2)
);

CREATE TABLE has_accounts_raw(
  robber_id int,
  nickname VARCHAR(70) NOT NULL,
  bank_name VARCHAR(80) NOT NULL,
  city VARCHAR(25) NOT NULL,
  FOREIGN KEY (bank_name, city) REFERENCES banks(bank_name, city)
);

CREATE TABLE accomplices_raw(
  nickname VARCHAR(70) NOT NULL,
  robber_id int,
  bank_name VARCHAR(80) NOT NULL,
  city VARCHAR(25) NOT NULL,
  robbery_date DATE NOT NULL,
  share NUMERIC(12,2),
  FOREIGN KEY (bank_name, city) REFERENCES banks(bank_name, city)
);
```

**Below is the data conversion for has_skills and skills:**

```
-- insert the data from file “has_skills.data” into the has_skills_raw
\copy has_skills_raw(nickname, skill, preference, grade) from 'hasskills_17.data'

-- insert the skill description into the skills table
INSERT INTO skills (description) (SELECT skill FROM has_skills_raw GROUP BY skill);

-- insert robber_id from robbers table into the column robber_id of has_skills_raw table
UPDATE has_skills_raw
SET robber_id = (SELECT robber_id
                 FROM robbers WHERE has_skills_raw.nickname = nickname);

-- to ensure all the rows in the column robber_id and skill_id is not NULL and the result must return 0
SELECT * FROM has_skills_raw WHERE robber_id IS NULL;
SELECT * FROM has_skills_raw WHERE skill_id IS NULL;

-- insert skill_id from skills table into the column skill_id of has_skills_raw table
UPDATE has_skills_raw
SET skill_id = (SELECT skill_id FROM skills WHERE has_skills_raw.skill = description);

-- insert all tuples from table has_skills_raw into has_skills table
INSERT INTO has_skills (SELECT DISTINCT robber_id, skill_id, preference, grade
                        FROM has_skills_raw ORDER BY robber_id, preference);
```

**Below is the data conversion for has_accounts:**

```
-- insert has_accounts data into table has_accounts_raw
\copy has_accounts_raw(nickname, bank_name, city) from 'hasaccounts_17.data'

-- check if all the tuple contain a robber_id
SELECT * FROM has_accounts_raw WHERE robber_id IS NULL;

-- update the robber_id column in has_accounts_raw using a subquery from robbers table:
UPDATE has_accounts_raw SET robber_id = (SELECT robber_id FROM robbers
                                         WHERE has_accounts_raw.nickname = nickname);
```

**Below is the data conversion for accomplices:**

```
-- insert accomplices data into table accomplices_raw
\copy accomplices_raw(nickname, bank_name, city, robbery_date, share) from 'accomplices_17.data'

-- update the robber_id in the dummy table accomplices_raw using the robbers table
UPDATE accomplices_raw
SET robber_id = (SELECT robber_id
                 FROM robbers WERE accomplices_raw.nickname = nickname);

-- To ensure all the rows in the column robber_id is not NULL and the result must return 0 rows
SELECT * FROM accomplices_raw WHERE robber_id IS NULL;

-- insert all rows from the dummy table accomplices_raw into the original accomplices table
-- in the SELECT statement only the required columns are included in the query
INSERT INTO accomplices (SELECT DISTINCT robber_id, bank_name, city, robbery_date, share
                         FROM accomplices_raw);
```

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

When not every pair of relations in the database need to be comparable, for example, the banks = {bank_name, city, no_accounts, security}, the robbers = {robber_id, nickname, age, no_years} and the skills = {skill_id, description} are incomparable with each other. Also, when the relations, such as *robberies, has_accounts*, and *accomplices* contain foreign keys = {bank_name, city} which refers to primary key in their parent tables, *banks*. Because primary key must come before foreign key, and foreign keys can be used only if the primary key is existed in the database (antisymmetric). Thus, the table *banks* with primary key {bank_name, city} must be implemented before the *robberies*, *has_accounts* and *accomplices* tables.

## QUESTION 3: Checking your Database [10 marks]

1. Insert the following tuples into the Banks table:

a. ('Loanshark Bank', 'Evanston', 100, 'very good')

**INSERT INTO banks VALUES ('Loanshark Bank', 'Evanston', 100, 'very good');**

**ERROR:** duplicate key value violates unique constraint "banks_pkey"

**DETAIL:** Key (bank_name, city)=(Loanshark Bank, Evanston) already exists.

UNIQUE constraint on primary key {bank_name, city} is violated as the table contains tuple with the same primary key.

b. ('EasyLoan Bank', 'Evanston', -5, 'excellent')

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

**ERROR:** new row for relation "banks" violates check constraint "check_account_num"

**DETAIL:** Failing row contains (EasyLoan Bank, Evanston, -5, excellent).

CHECK constraint on the attribute *no_accounts* is violated as the value cannot be negative value.

c. ('EasyLoan Bank', 'Evanston', 100, 'poor')

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

**ERROR:** new row for relation "banks" violates check constraint "check_security_input"

**DETAIL:** Failing row contains (EasyLoan Bank, Evanston, 100, poor).

CHECK constraint on the attribute *security* is violated as the value must be one of the fixed values: 'weak', 'excellent', 'very good', 'good'.

2. Insert the following tuple into the Skills table:

a. (20, 'Guarding')

**INSERT INTO skills VALUES (20, 'Guarding');**

**ERROR:** duplicate key value violates unique constraint "must_be_different"

**DETAIL:** Key (description)=(Guarding) already exists.

UNIQUE constraint on the attribute *description* is violated as the value 'Guarding' already existed in the column *description*.

3. Insert the following tuples into the Robbers table:

a. (1, 'Shotgun', 70, 0)

**INSERT INTO robbers VALUES (1, 'Shotgun', 70, 0);**

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

UNIQUE constraint on primary key {skill_id} is violated as the table contains tuple with the same primary key.

b. (333, 'Jail Mouse', 25, 35)

**INSERT INTO robbers VALUES (333, 'Jail Mouse', 25, 35);**

**ERROR:** new row for relation "robbers" violates check constraint "check_prison_years"
**DETAIL:** Failing row contains (333, Jail Mouse, 25, 35).

CHECK constraint on the attribute *no_years* (number of prison years) is violated as the value cannot be larger than the robber's age.

4. Insert the following tuples into the HasSkills table:

a. (333, 1, 1, 'B-')

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

**ERROR:** insert or update on table "has_skills" violates foreign key constraint "has_skills_robber_id_fkey"
**DETAIL:** Key (robber_id)=(333) is not present in table "robbers".

CHECK constraint on the attribute *no_years* (number of prison years) is violated as the value cannot be larger than the robber's age.

b. (3, 20, 3, 'B+')

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

**ERROR:** insert or update on table "has_skills" violates foreign key constraint "has_skills_skill_id_fkey"
**DETAIL:** Key (skill_id)=(20) is not present in table "skills".

FOREIGN KEY constraint on the attribute *skill_id* is violated as skill_id = 20 doesn't match any tuple in the table *skills*.

c. (1, 7, 1, 'A+')

**INSERT INTO has_skills VALUES (1, 7, 1, 'A+');**

**ERROR:** new row for relation "has_skills" violates check constraint "check_preference_num"
**DETAIL:** Failing row contains (1, 7, 1, A+).

CHECK constraint on the attribute *preference* (ranking order) is violated as the value of *preference* for robber_id=1 should be 4 as he already has 3 skills.

d. (1, 2, 0, 'A')

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

**ERROR:** new row for relation "has_skills" violates check constraint "check_preference_num"
**DETAIL:** Failing row contains (1, 2, 0, A ).

CHECK constraint on the attribute *preference* is violated as the value of *preference* (ranking order) should be start on with 1.

5. Insert the following tuple into the Robberies table:

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

**INSERT INTO robberies VALUES ('NXP Bank', 'Chicago', '2009-01-08', 1000);**

**ERROR:** duplicate key value violates unique constraint "robberies_pkey"

**DETAIL:** Key (bank_name, city, robbery_date)=(NXP Bank, Chicago, 2009-01-08) already exists.

UNIQUE constraint on the primary key {bank_name, city, robbery_date} is violated as the key already exists in the table.

6. Delete the following tuples from the Banks table:

a. ('PickPocket Bank', 'Evanston', 2000, 'very good')

**DELETE FROM** banks **WHERE** bank_name = 'PickPocket Bank' **AND** city = 'Evanston' **AND** no_accounts = 2000 AND security = 'very good';

**ERROR:** update or delete on table "banks" violates foreign key constraint "robberies_bank_name_fkey" on table "robberies"

**DETAIL:** Key (bank_name, city)=(PickPocket Bank, Evanston) is still referenced from table "robberies".

This tuple cannot be deleted because the references (foreign key) to this primary key is still exists in the table robberies.

b. ('Outside Bank', 'Chicago', 5000, 'good')

**DELETE FROM** banks **WHERE** bank_name = 'Outside Bank' **AND** city = 'Chicago' **AND** no_accounts = 5000 **AND** security = 'good';

DELETE 1

This tuple can be deleted because there is no reference to its primary key.

7. Delete the following tuple from the Robbers table:

a. (1, 'Al Capone', 31, 2).

**DELETE FROM robbers WHERE robber_id = 1 AND nickname = 'Al Capone' AND age = 31 AND no_years = 2;**

**ERROR:** update or delete on table "robbers" violates foreign key constraint "has_accounts_robber_id_fkey" on table "has_accounts"

**DETAIL:** Key (robber_id)=(1) is still referenced from table "has_accounts".

The tuple cannot be deleted. But if CASCADE constraints are added, the tuple and all tuples from other tables that use the tuple as foreign keys reference will be deleted.

8. Delete the following tuple from the Skills table:

a. (1, 'Driving')

**DELETE FROM skills WHERE skill_id = 1 AND description = 'Driving';**

DELETE 0

There is no error thrown but also no tuple in table is deleted, as there is no matching tuple with the values. But if the skill_id = 7 or description = 'Safe-Cracking' then the tuple can be deleted, as the skill_id of 'Driving' is 7 and the description of skill_id = 1 is 'Safe-Cracking'.

## QUESTION 4: Simple Database Queries [24 marks]

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

```
SELECT bank_name AS "BankName", security AS "Security"
FROM banks
WHERE city = 'Chicago' AND no_accounts > 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 bank_name AS "BankName"
FROM has_accounts
WHERE robber_id = (SELECT robber_id FROM robbers
                   WHERE nickname = 'Calamity Jane');
```

```
    BankName
-----------------
 Dollar Grabbers
 Bad Bank
 PickPocket Bank
(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 bank_name AS "BankName", city AS "City"
FROM banks
WHERE (SELECT subquery.bank_name
        FROM banks AS "subquery"
        WHERE subquery.city = 'Chicago' AND
        subquery.bank_name = banks.bank_name) IS NULL
ORDER BY no_accounts ASC;
```

```
    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 DISTINCT bank_name AS "BankName", city AS "City"
FROM accomplices
WHERE robbery_date = (SELECT MIN(robbery_date) FROM accomplices);
```

```
    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 robber_id AS "RobberId",
        nickname AS "Nickname",
        SUM(share) AS "Total Earning"
FROM accomplices
  NATURAL INNER JOIN robbers
GROUP BY robber_id, nickname
HAVING SUM(share) > 30000.00
ORDER BY SUM(share) DESC;
```

```
 RobberId |     Nickname       | Total Earning
----------+--------------------+---------------
        5 | Mimmy The Mau Mau  |      70000.00
       15 | Boo Boo Hoff       |      61447.61
       16 | King Solomon       |      59725.80
       17 | Bugsy Siegel       |      52601.10
        3 | Lucky Luchiano     |      42667.00
       10 | Bonnie             |      40085.00
        1 | Al Capone          |      39486.00
        4 | Anastazia          |      39169.62
        8 | Clyde              |      31800.00
(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 robber_id AS "RobberId",
       nickname AS "Nickname",
       description AS "Skill"
FROM has_skills
  NATURAL INNER JOIN robbers
  NATURAL INNER JOIN skills
ORDER BY has_skills.skill_id;
```

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

```
       15 | Boo Boo Hoff      | Planning
       16 | King Solomon      | Planning
        5 | Mimmy The Mau Mau | Planning
       18 | Vito Genovese     | Eating
        6 | Tony Genovese     | Eating
(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 robber_id AS "RobberId",
        nickname AS "Nickname",
        no_years AS "Total prison years"
FROM robbers
WHERE no_years > 3;
```

```
 RobberId |    Nickname    | Total prison years
----------+----------------+--------------------
        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 robber_id AS "RobberId",
        nickname AS "Nickname",
        (age-no_years) AS "Number of year not in prison"
FROM robbers
WHERE no_years > (age/2);
```

```
 RobberId |   Nickname    | Number of year not in prison
----------+---------------+------------------------------
        6 | Tony Genovese |                           12
       16 | King Solomon  |                           31
(2 rows)
```

## QUESTION 5: Complex Database Queries [35 marks]

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. [7 marks]

```
----- Stepwise approach for task 1 -----
CREATE VIEW total_robberies AS
SELECT nickname,
          COUNT(robber_id) AS total,
          SUM(share) AS share
FROM robbers
   NATURAL INNER JOIN accomplices
GROUP BY robber_id
HAVING robbers.no_years = 0
ORDER BY SUM(share) DESC;

CREATE VIEW avg_robberies AS
SELECT ROUND(AVG(total), 0) AS avg_robberies
FROM total_robberies;

CREATE VIEW most_active_robbers AS
SELECT total_robberies.nickname AS "Nickname"
FROM total_robberies
   NATURAL INNER JOIN avg_robberies
WHERE (total > avg_robberies);
```

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

```
----- Single nested SQL query for task 1 -----
SELECT (SELECT nickname
          FROM robbers
          WHERE accomplices.robber_id = robbers.robber_id) AS "Nickname"
FROM accomplices
GROUP BY accomplices.robber_id
HAVING (COUNT(accomplices.robber_id) > (SELECT (COUNT(*)/COUNT(DISTINCT
          accomplices.robber_id)) FROM accomplices)) AND
          (SELECT no_years FROM robbers
          WHERE accomplices.robber_id = robbers.robber_id) = 0
ORDER BY SUM(accomplices.share) 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. [7 marks]

----- Stepwise approach for task 2 -----

```
CREATE VIEW merge_banks_info AS
SELECT bank_name, city, COUNT(*) AS total_robberies, SUM(amount) AS total_money
FROM robberies
GROUP BY bank_name, city;
```

```
CREATE VIEW bank_securities AS
SELECT security AS "Security", SUM(total_robberies) AS "Total # of robberies",
          ROUND((SUM(total_money)/SUM(total_robberies)), 2)
          AS "Average amount of money"
FROM merge_banks_info
   NATURAL INNER JOIN banks
GROUP BY security;
```

```
 Security  | Total # of robberies | Average amount of money
-----------+----------------------+-------------------------
 weak      |                    4 |                 2299.50
 excellent |                   12 |                39238.08
 very good |                    3 |                12292.43
 good      |                    2 |                 3980.00
(4 rows)
```

----- Single nested SQL query for task 2 -----

```
SELECT bank_robberies.security AS "Security",
          COUNT(*) AS "Total # of robberies",
          ROUND((SUM(amount)/ COUNT(*)), 2) AS "Average amount of money"
FROM (SELECT banks.bank_name, banks.city, banks.security, robberies.amount
         FROM robberies, banks
         WHERE banks.bank_name = robberies.bank_name AND
                   banks.city = robberies.city) AS bank_robberies
GROUP BY bank_robberies.security;
```

```
 Security  | Total # of robberies | Average amount of money
-----------+----------------------+-------------------------
 weak      |                    4 |                 2299.50
 good      |                    2 |                 3980.00
 excellent |                   12 |                39238.08
 very good |                    3 |                12292.43
(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. [7 marks]

----- Stepwise approach for task 3 -----

```
CREATE VIEW robbers_info AS
SELECT robber_id, nickname, skill_id, bank_name, city
FROM accomplices
   NATURAL INNER JOIN robbers
   NATURAL INNER JOIN has_skills;

CREATE VIEW robbers_skills AS
SELECT robber_id, nickname, description, bank_name, city
FROM robbers_info NATURAL INNER JOIN skills;

CREATE VIEW banks_info AS
SELECT robber_id, nickname, description, security
FROM robbers_skills NATURAL INNER JOIN banks;

CREATE VIEW suspects_list AS
SELECT security AS "Security", description AS "Description", nickname AS "Nickname"
FROM banks_info
GROUP BY security, description, nickname
ORDER BY security;
```

```
 Security  |  Description   |     Nickname
-----------+----------------+------------------
 excellent | Driving        | Bugsy Siegel
 excellent | Driving        | Dutch Schulz
 excellent | Driving        | Longy Zwillman
 excellent | Driving        | Lucky Luchiano
 excellent | Driving        | Mimmy The Mau Mau
```

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

```
 weak      | Guarding       | Lepke Buchalter
 weak      | Lock-Picking   | Clyde
 weak      | Lock-Picking   | Dutch Schulz
 weak      | Lock-Picking   | Greasy Guzik
 weak      | Lock-Picking   | Sonny Genovese
 weak      | Planning       | Al Capone
 weak      | Planning       | Boo Boo Hoff
 weak      | Planning       | Clyde
 weak      | Preaching      | Al Capone
 weak      | Preaching      | Greasy Guzik
 weak      | Safe-Cracking  | Al Capone
 weak      | Safe-Cracking  | Sonny Genovese
 weak      | Scouting       | Clyde
 weak      | Scouting       | Vito Genovese
(65 rows)
```

----- Single nested SQL query for task 3 -----

```
SELECT DISTINCT banks.security AS "Security",
                    skills.description AS "Description",
                    banks.nickname AS "Nickname"
FROM (SELECT DISTINCT security, nickname, robber_id
          FROM accomplices NATURAL INNER JOIN banks
NATURAL INNER JOIN robbers) AS banks,
        (SELECT DISTINCT skill_id, description, robber_id
         FROM skills NATURAL INNER JOIN has_skills) AS skills
WHERE banks.robber_id = skills.robber_id
 ORDER BY banks.security;
```

```
 Security  |  Description   |     Nickname
-----------+----------------+------------------
 excellent | Driving        | Bugsy Siegel
 excellent | Driving        | Dutch Schulz
 excellent | Driving        | Longy Zwillman
 excellent | Driving        | Lucky Luchiano
 excellent | Driving        | Mimmy The Mau Mau
 excellent | Explosives     | Sonny Genovese
 excellent | Guarding       | Anastazia
 excellent | Guarding       | Bugsy Siegel
 excellent | Gun-Shooting   | Waxey Gordon
 excellent | Lock-Picking   | Clyde
 excellent | Lock-Picking   | Dutch Schulz
 excellent | Lock-Picking   | Greasy Guzik
 excellent | Lock-Picking   | Lucky Luchiano
 excellent | Lock-Picking   | Sonny Genovese
 excellent | Planning       | Al Capone
 excellent | Planning       | Boo Boo Hoff
```

```
excellent | Planning       | Clyde
excellent | Planning       | King Solomon
excellent | Planning       | Mimmy The Mau Mau
excellent | Preaching      | Al Capone
excellent | Preaching      | Bonnie
excellent | Preaching      | Greasy Guzik
excellent | Safe-Cracking  | Al Capone
excellent | Safe-Cracking  | Meyer Lansky
excellent | Safe-Cracking  | Sonny Genovese
excellent | Scouting       | Clyde
good      | Cooking        | Vito Genovese
good      | Eating         | Vito Genovese
good      | Money Counting | Kid Cann
good      | Money Counting | Mickey Cohen
good      | Scouting       | Vito Genovese
very good | Driving        | Lepke Buchalter
very good | Driving        | Longy Zwillman
very good | Explosives     | Bugsy Malone
very good | Explosives     | Sonny Genovese
very good | Guarding       | Anastazia
very good | Guarding       | Lepke Buchalter
very good | Lock-Picking   | Sonny Genovese
very good | Planning       | Al Capone
very good | Planning       | King Solomon
very good | Preaching      | Al Capone
very good | Safe-Cracking  | Al Capone
very good | Safe-Cracking  | Moe Dalitz
very good | Safe-Cracking  | Sonny Genovese
weak      | Cooking        | Vito Genovese
weak      | Driving        | Bugsy Siegel
weak      | Driving        | Dutch Schulz
weak      | Driving        | Lepke Buchalter
weak      | Eating         | Vito Genovese
weak      | Explosives     | Sonny Genovese
weak      | Guarding       | Bugsy Siegel
weak      | Guarding       | Lepke Buchalter
weak      | Lock-Picking   | Clyde
weak      | Lock-Picking   | Dutch Schulz
weak      | Lock-Picking   | Greasy Guzik
weak      | Lock-Picking   | Sonny Genovese
weak      | Planning       | Al Capone
weak      | Planning       | Boo Boo Hoff
weak      | Planning       | Clyde
weak      | Preaching      | Al Capone
weak      | Preaching      | Greasy Guzik
weak      | Safe-Cracking  | Al Capone
```

```
 weak       | Safe-Cracking  | Sonny Genovese
 weak       | Scouting       | Clyde
 weak       | Scouting       | Vito Genovese
(65 rows)
```

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 approach for task 4 -----

```
CREATE VIEW never_been_robbed AS
SELECT DISTINCT banks.bank_name, banks.city, banks.security, banks.no_accounts
FROM banks
  FULL JOIN robberies ON robberies.bank_name = banks.bank_name AND
                robberies.city = banks.city AND
                NOT (robberies.robbery_date BETWEEN CURRENT_DATE AND
                      CURRENT_DATE - INTERVAL '1 year')
GROUP BY banks.bank_name, banks.city;

CREATE VIEW planned_robberies AS
SELECT DISTINCT bank_name, city, security, no_accounts
FROM banks NATURAL JOIN plans
WHERE (planned_date BETWEEN CURRENT_DATE AND CURRENT_DATE +
           INTERVAL '1 year')
GROUP BY bank_name, city
ORDER BY no_accounts DESC;

CREATE VIEW target_banks AS
SELECT bank_name AS "BankName", city AS "City", security AS "Security"
FROM planned_robberies NATURAL INNER JOIN never_been_robbed
ORDER BY planned_robberies.no_accounts DESC;
```

```
   BankName      |   City   | Security
-----------------+----------+-----------
Loanshark Bank   | Deerfield | very good
PickPocket Bank | Deerfield | excellent
Bad Bank         | Chicago   | weak
(3 rows)
```

----- Single nested SQL query for task 4 -----

```
SELECT target_banks.bank_name AS "BankName",
            target_banks.city AS "City",
            target_banks.security AS "Security"
FROM (SELECT DISTINCT plans.bank_name, plans.city, banks.security,
                                banks.no_accounts
          FROM plans NATURAL JOIN banks
          WHERE plans.planned_date BETWEEN CURRENT_DATE AND
                    CURRENT_DATE + INTERVAL '1 year' AND
                    NOT EXISTS (SELECT * FROM robberies
                                    WHERE robberies.robbery_date BETWEEN
                                            CURRENT_DATE AND CURRENT_DATE -
                                            INTERVAL '1 year')) AS target_banks
ORDER BY target_banks.no_accounts DESC;
```

```
   BankName      |   City    | Security
-----------------+-----------+-----------
Loanshark Bank  | Deerfield | very good
PickPocket Bank | Deerfield | excellent
Bad Bank         | Chicago   | weak
(3 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. [7 marks]

----- Stepwise approach for task 5 -----

```
CREATE VIEW chicago_avg AS
SELECT ROUND((SUM(share)/COUNT(robber_id)), 2) AS avg
FROM accomplices
WHERE city = 'Chicago';

CREATE VIEW others_avg AS
SELECT ROUND((SUM(share)/COUNT(robber_id)), 2) AS avg
FROM accomplices
WHERE NOT(city = 'Chicago');

CREATE VIEW avg_share AS
SELECT chicago_avg.avg AS "Average share in Chicago",
            others_avg.avg AS "Average share in other cities"
FROM chicago_avg, others_avg;
```

```
 Average share in Chicago | Average share in other cities
--------------------------+--------------------------------
                   4221.41 |                         8255.16
(1 row)
```

----- Single nested SQL query for task 5 -----

```
SELECT chicago.average AS "Average share in Chicago",
          others.average AS "Average share in other cities"
FROM (SELECT ROUND((SUM(share)/COUNT(robber_id)), 2) AS "average"
         FROM accomplices
         WHERE accomplices.city = 'Chicago') AS chicago,
        (SELECT ROUND((SUM(share)/COUNT(robber_id)), 2) AS "average"
         FROM accomplices
         WHERE NOT(accomplices.city = 'Chicago')) AS others;
```

```
 Average share in Chicago | Average share in other cities
--------------------------+--------------------------------
                   4221.41 |                         8255.16
(1 row)
```

----- EXTRAS: Other approach for task 5 -----

```
SELECT
   ROUND(SUM(CASE WHEN city = 'Chicago' THEN share ELSE 0 END)/COUNT(CASE
   WHEN city = 'chicago' THEN 1 ELSE 0 END), 2) AS "Average share in Chicago",
   ROUND(SUM(CASE WHEN NOT (city = 'Chicago') THEN share ELSE 0 END)/
   COUNT(CASE WHEN NOT(city = 'chicago') THEN 1 ELSE 0 END), 2)
   AS "Average share in other cities"
FROM accomplices
GROUP BY city;
```

```
 Average share in Chicago | Average share in other cities
--------------------------+--------------------------------
                      0.00 |                         8255.16
                   4221.41 |                            0.00
(2 rows)
```