# SWEN304 // Project 01

Philip Chang
300375123

## Question 01: Defining the Database [15 marks]

### Primary & Foreign Keys

1. Banks
    a. Primary Key: (BankName, City)

Each bank branch is uniquely identified by the combination of its name and the city it is located in.

    b. Foreign Key: None

2. Robbers
    a. Primary Key: RobberId

A surrogate key is used to uniquely identify each robber. This avoids ambiguity in case multiple robbers share the same nickname.

    b. Foreign Key: None

3. Skills
    a. Primary Key: SkillId

Each skill has a unique identifier to avoid relying on text descriptions, which may be prone to duplication or typos.

    b. Foreign Key: None

4. HasSkills
    a. Primary Key: (RobberId, SkillId)

This composite key ensures that each robber can only have one record per skill.

    b. Foreign Key: RobberId > Robbers(RobberId)

Ensures the skill belongs to a valid robber.

    c. Foreign Key: SkillId > Skills(SkillId)

Ensures that the skill assigned is a recognized skill from the skills list.

5. HasAccounts
    a. Primary Key: (RobberId, BankName, City)

A robber can have at most one account at a particular bank branch. This combination uniquely identifies these accounts.

    b. Foreign Key: RobberId > Robbers(RobberId)

Ensures only valid robbers are recorded as having accounts.

    c. Foreign Key: (BankName, City) > Banks(BankName, City)

Ensures the referenced bank branch exists.

6. Robberies
    a. Primary Key: (BankName, City, Date)

A robbery is uniquely identified by the bank, city, and the date it occurred.

    b. Foreign Key: (BankName, City) > Banks(BankName, City)

Ensures that the robbery happened at a valid, known bank branch.

7. Plans
    a. Primary Key: (BankName, City, PlannedDate)

A planned robbery is uniquely identified by the bank and the planned date.

    b. Foreign Key: (BankName, City) > Banks(BankName, City)

Ensures the plan refers to an existing bank branch.

8. Accomplices
    a. Primary Key: (RobberId, BankName, City, Date)

A specific robber's involvement in a particular robbery (identified by bank and date) is unique.

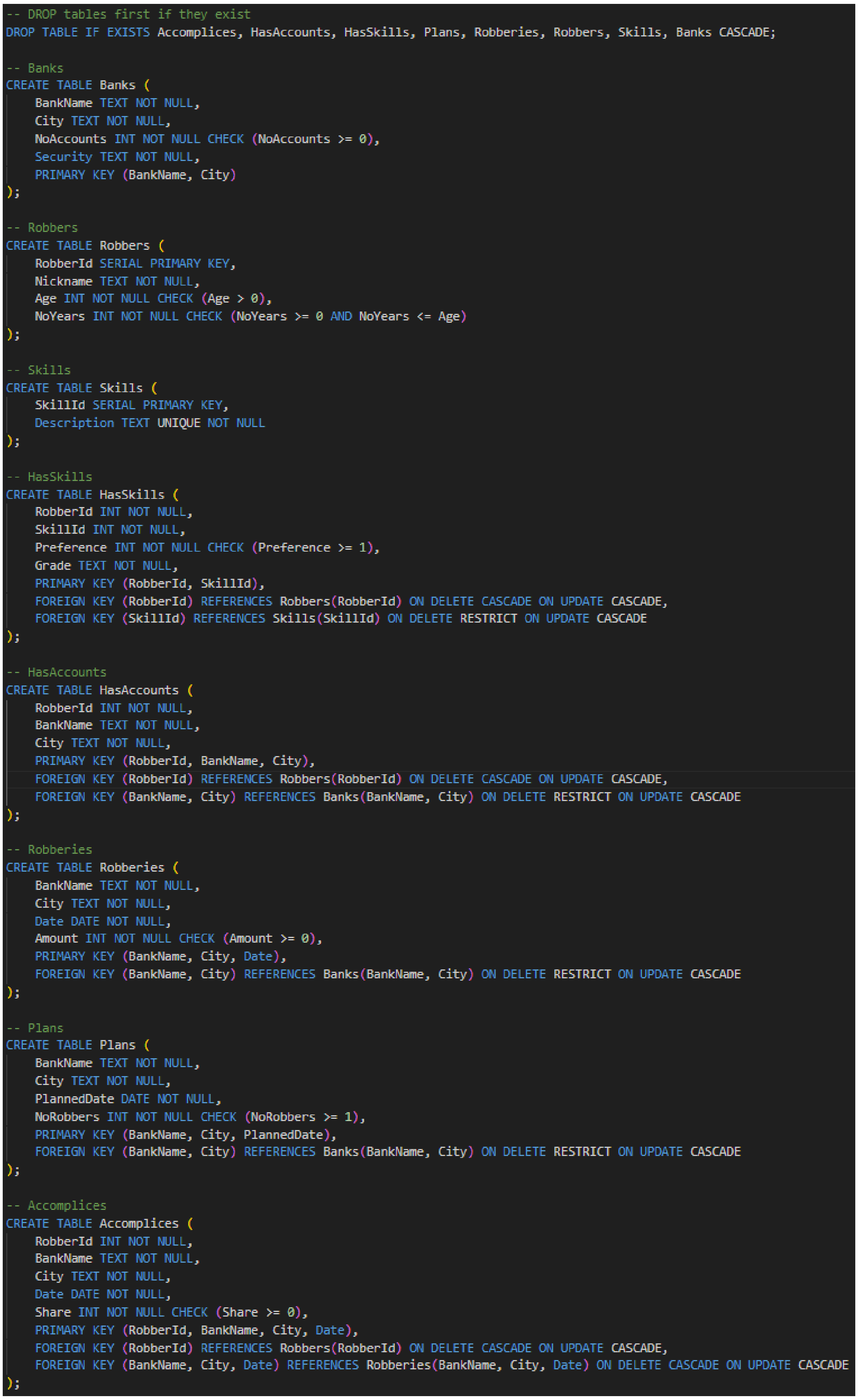
    b. Foreign Key: RobberId > Robbers(RobberId)

Ensures only valid robbers are recorded as accomplices.

    c. Foreign Key: (BankName, City, Date) > Robberies(BankName, City, Date)

Ensures the accomplice is linked to a valid robbery event.

## CREATE TABLE Statements



```
-- DROP tables first if they exist
DROP TABLE IF EXISTS Accomplices, HasAccounts, HasSkills, Plans, Robberies, Robbers, Skills, Banks CASCADE;

-- Banks
CREATE TABLE Banks (
    BankName TEXT NOT NULL,
    City TEXT NOT NULL,
    NoAccounts INT NOT NULL CHECK (NoAccounts >= 0),
    Security TEXT NOT NULL,
    PRIMARY KEY (BankName, City)
);

-- Robbers
CREATE TABLE Robbers (
    RobberId SERIAL PRIMARY KEY,
    Nickname TEXT NOT NULL,
    Age INT NOT NULL CHECK (Age > 0),
    NoYears INT NOT NULL CHECK (NoYears >= 0 AND NoYears <= Age)
);

-- Skills
CREATE TABLE Skills (
    SkillId SERIAL PRIMARY KEY,
    Description TEXT UNIQUE NOT NULL
);

-- HasSkills
CREATE TABLE HasSkills (
    RobberId INT NOT NULL,
    SkillId INT NOT NULL,
    Preference INT NOT NULL CHECK (Preference >= 1),
    Grade TEXT NOT NULL,
    PRIMARY KEY (RobberId, SkillId),
    FOREIGN KEY (RobberId) REFERENCES Robbers(RobberId) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (SkillId) REFERENCES Skills(SkillId) ON DELETE RESTRICT ON UPDATE CASCADE
);

-- HasAccounts
CREATE TABLE HasAccounts (
    RobberId INT NOT NULL,
    BankName TEXT NOT NULL,
    City TEXT NOT NULL,
    PRIMARY KEY (RobberId, BankName, City),
    FOREIGN KEY (RobberId) REFERENCES Robbers(RobberId) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (BankName, City) REFERENCES Banks(BankName, City) ON DELETE RESTRICT ON UPDATE CASCADE
);

-- Robberies
CREATE TABLE Robberies (
    BankName TEXT NOT NULL,
    City TEXT NOT NULL,
    Date DATE NOT NULL,
    Amount INT NOT NULL CHECK (Amount >= 0),
    PRIMARY KEY (BankName, City, Date),
    FOREIGN KEY (BankName, City) REFERENCES Banks(BankName, City) ON DELETE RESTRICT ON UPDATE CASCADE
);

-- Plans
CREATE TABLE Plans (
    BankName TEXT NOT NULL,
    City TEXT NOT NULL,
    PlannedDate DATE NOT NULL,
    NoRobbers INT NOT NULL CHECK (NoRobbers >= 1),
    PRIMARY KEY (BankName, City, PlannedDate),
    FOREIGN KEY (BankName, City) REFERENCES Banks(BankName, City) ON DELETE RESTRICT ON UPDATE CASCADE
);

-- Accomplices
CREATE TABLE Accomplices (
    RobberId INT NOT NULL,
    BankName TEXT NOT NULL,
    City TEXT NOT NULL,
    Date DATE NOT NULL,
    Share INT NOT NULL CHECK (Share >= 0),
    PRIMARY KEY (RobberId, BankName, City, Date),
    FOREIGN KEY (RobberId) REFERENCES Robbers(RobberId) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (BankName, City, Date) REFERENCES Robberies(BankName, City, Date) ON DELETE CASCADE ON UPDATE CASCADE
);
```

**Foreign Key Justifications**

1. HasSkills.RobberId > Robbers
    a. ON DELETE CASCADE

If a robber is deleted, all their associated skills should be removed.

    b. ON UPDATE CASCADE

If a robber’s ID changes, all related skill records should be updated to maintain consistency.

2. HasSkills.SkillId > Skills
    a. ON DELETE RESTRICT

Prevents deleting a skill if it is currently assigned to any robber.

    b. ON UPDATE CASCADE

If a skill’s ID changes, propagate the change to dependent skill assignments.

3. HasAccounts.RobberId > Robbers
    a. ON DELETE CASCADE

If a robber is deleted, their associated bank accounts should also be removed.

    b. ON UPDATE CASCADE

If a robber’s ID changes, update all account records accordingly.

4. HasAccounts.(BankName, City) > Banks
    a. ON DELETE RESTRICT

Prevents deletion of a bank branch if there are existing robber accounts tied to it.

    b. ON UPDATE CASCADE

If a bank’s name or city is updated, propagate changes to account records.

5. Robberies.(BankName, City) > Banks
    a. ON DELETE RESTRICT

Do not allow deletion of a bank branch with recorded robbery events.

    b. ON UPDATE CASCADE

If a bank’s details change, reflect the updates in the robbery records.

6. Plans.(BankName, City) > Banks
    a. ON DELETE RESTRICT

Prevent planning against a non-existent bank.

    b. ON UPDATE CASCADE

Update planned robbery entries if the bank name or city is changed.

7. Accomplices.RobberId > Robbers
    a. ON DELETE CASCADE

If a robber is removed, also delete their accomplice records.

    b. ON UPDATE CASCADE

If a robber ID is changed, propagate to the accomplice relation.

8. Accomplices.(BankName, City, Date) > Robberies
    a. ON DELETE CASCADE

If a robbery is removed, accomplice data tied to that event should be deleted as well.

    b. ON UPDATE CASCADE

If bank or date details of the robbery change, update the referencing accomplice records.

**Attribute Constraint Justifications**

1. NoAccounts
    a. CHECK (NoAccounts >= 0)

A bank cannot have a negative number of accounts.

2. Age
    a. CHECK (Age > 0)

All robbers must have a valid positive age.

3. NoYears
    a. CHECK (NoYears >= 0 AND NoYears <= Age)

A robber cannot spend more years in prison than they are old.

4. Preference
    a. CHECK (Preference >= 1)

Preference must be a positive integer (1 = first preference).

5. Amount, Share
    a. CHECK (>= 0)

Cannot steal or receive a negative amount of money.

6. Most attributes.
    a. NOT NULL

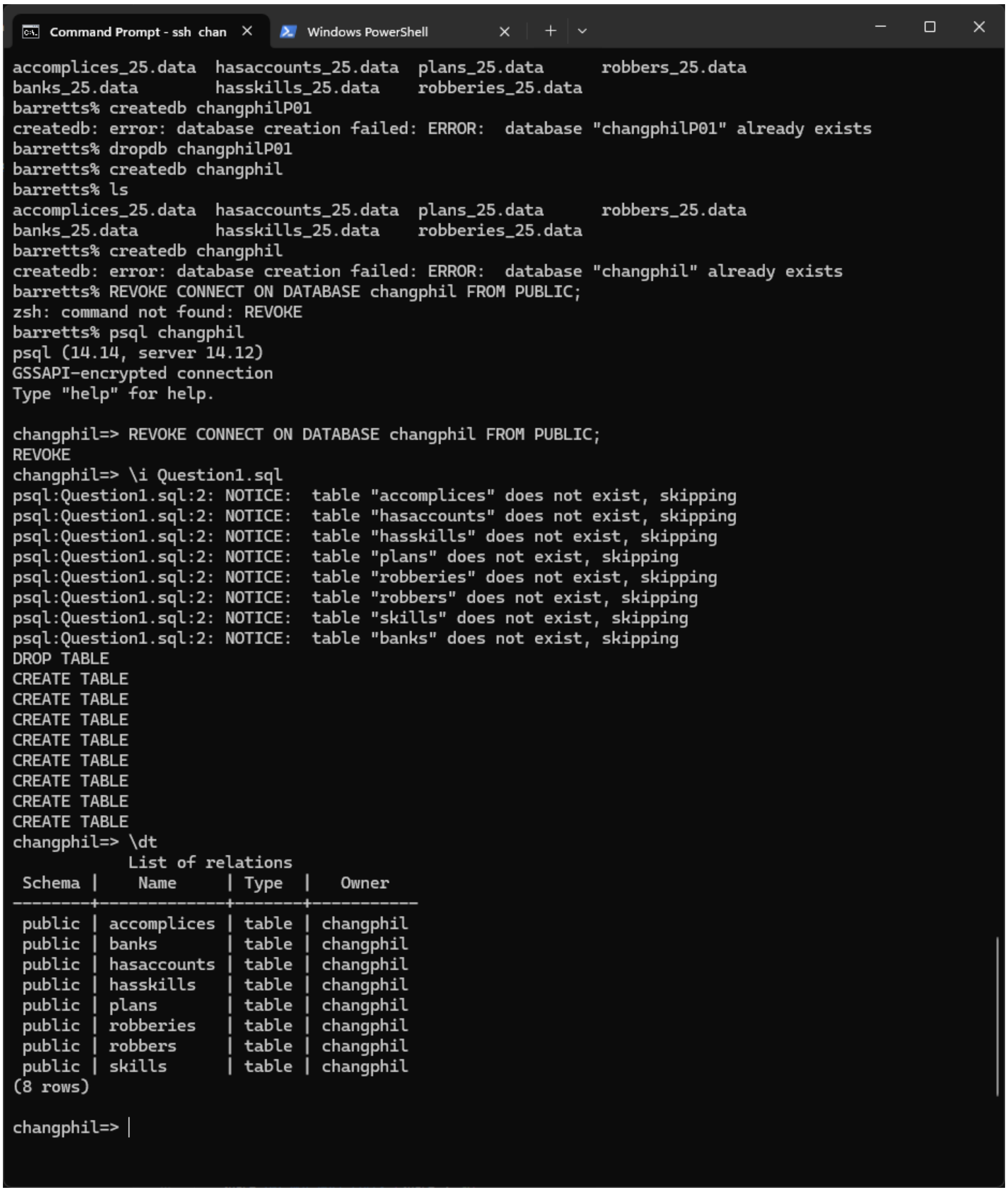
Ensures data completeness and avoids meaningless entries such as null skill descriptions or null robber ages.

7. Skills.Description
    a. UNIQUE

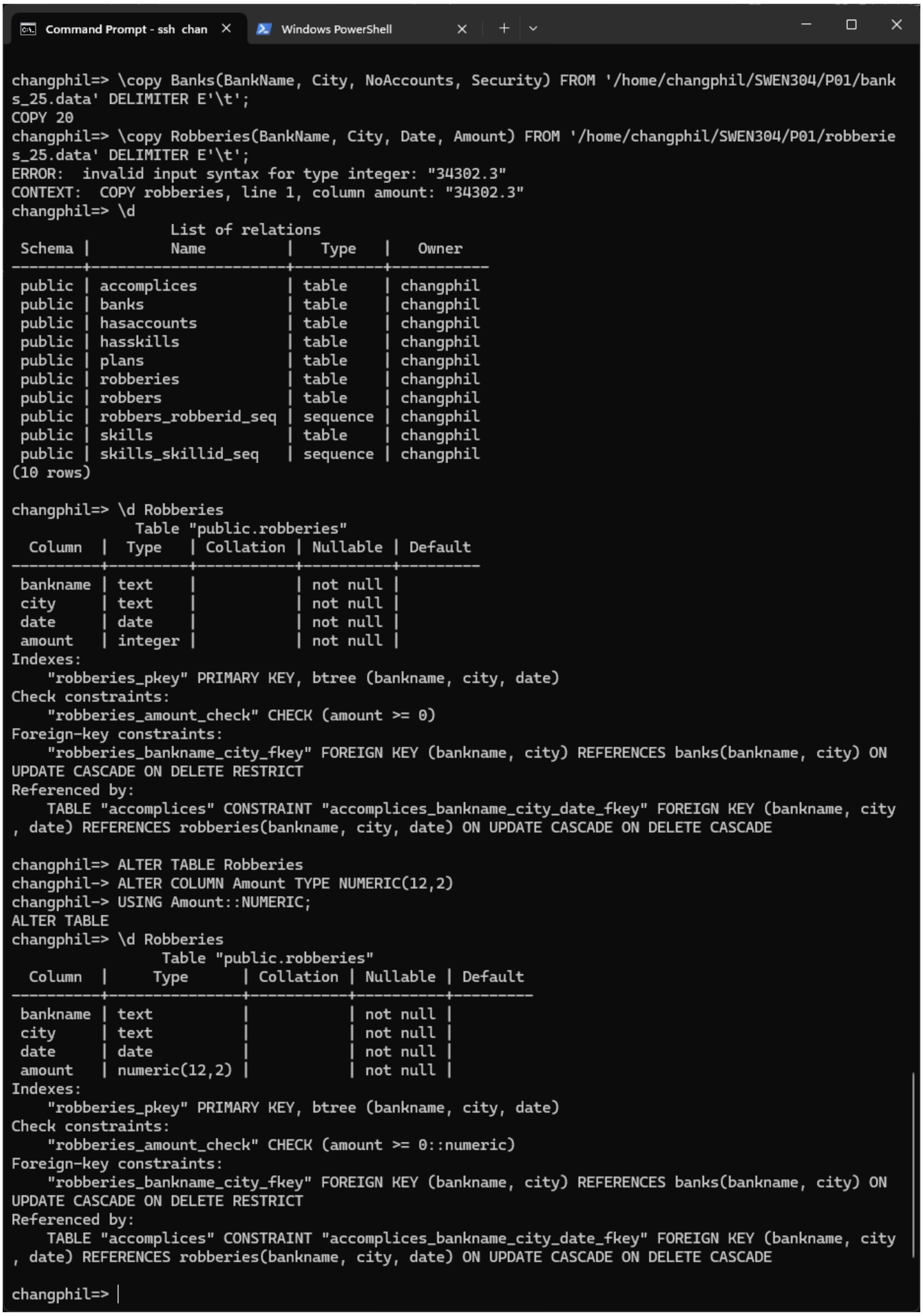
Ensures distinct skill names.

## Question 02: Populating your Database with Data [15 marks]

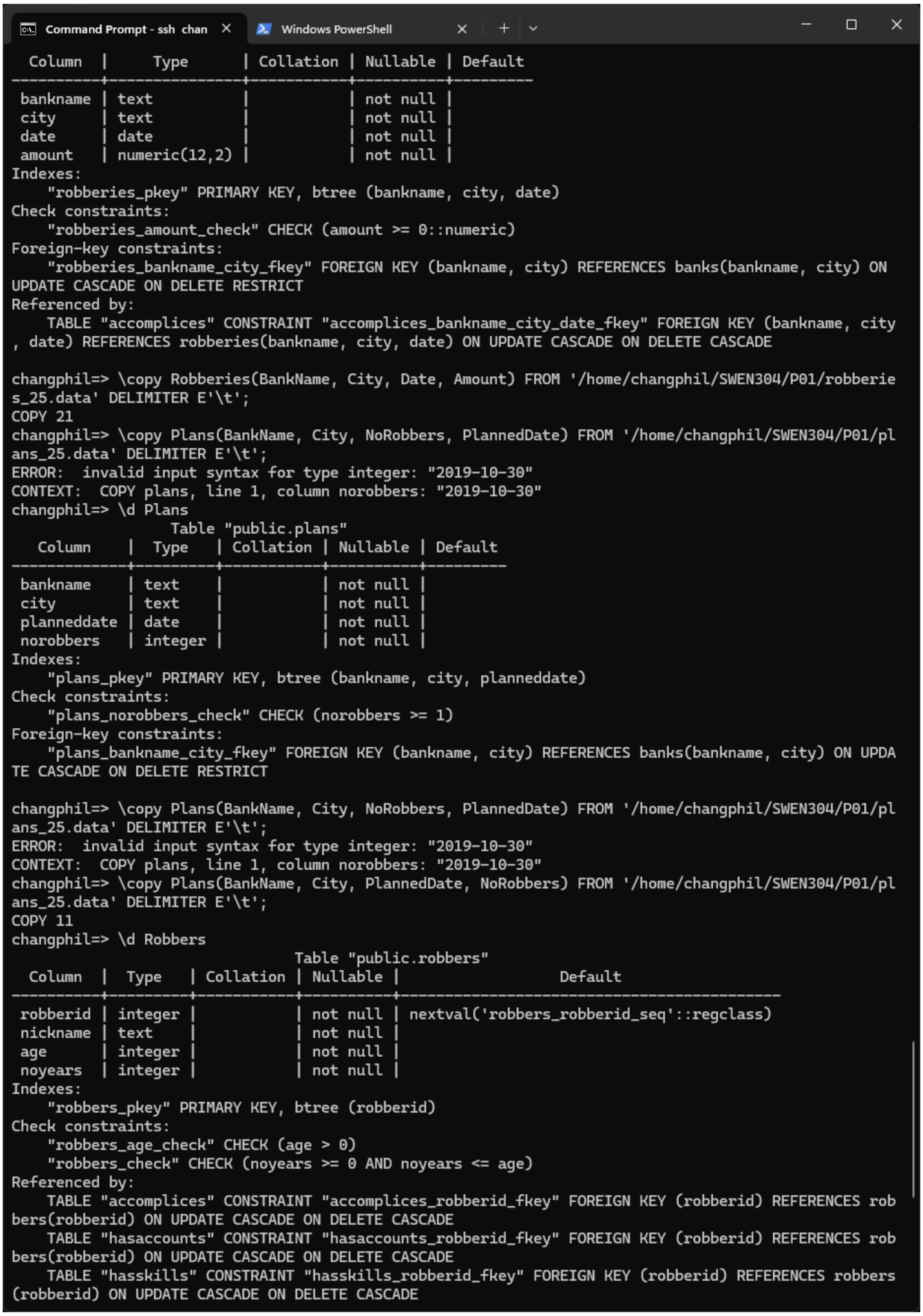
### Data Conversion Performance



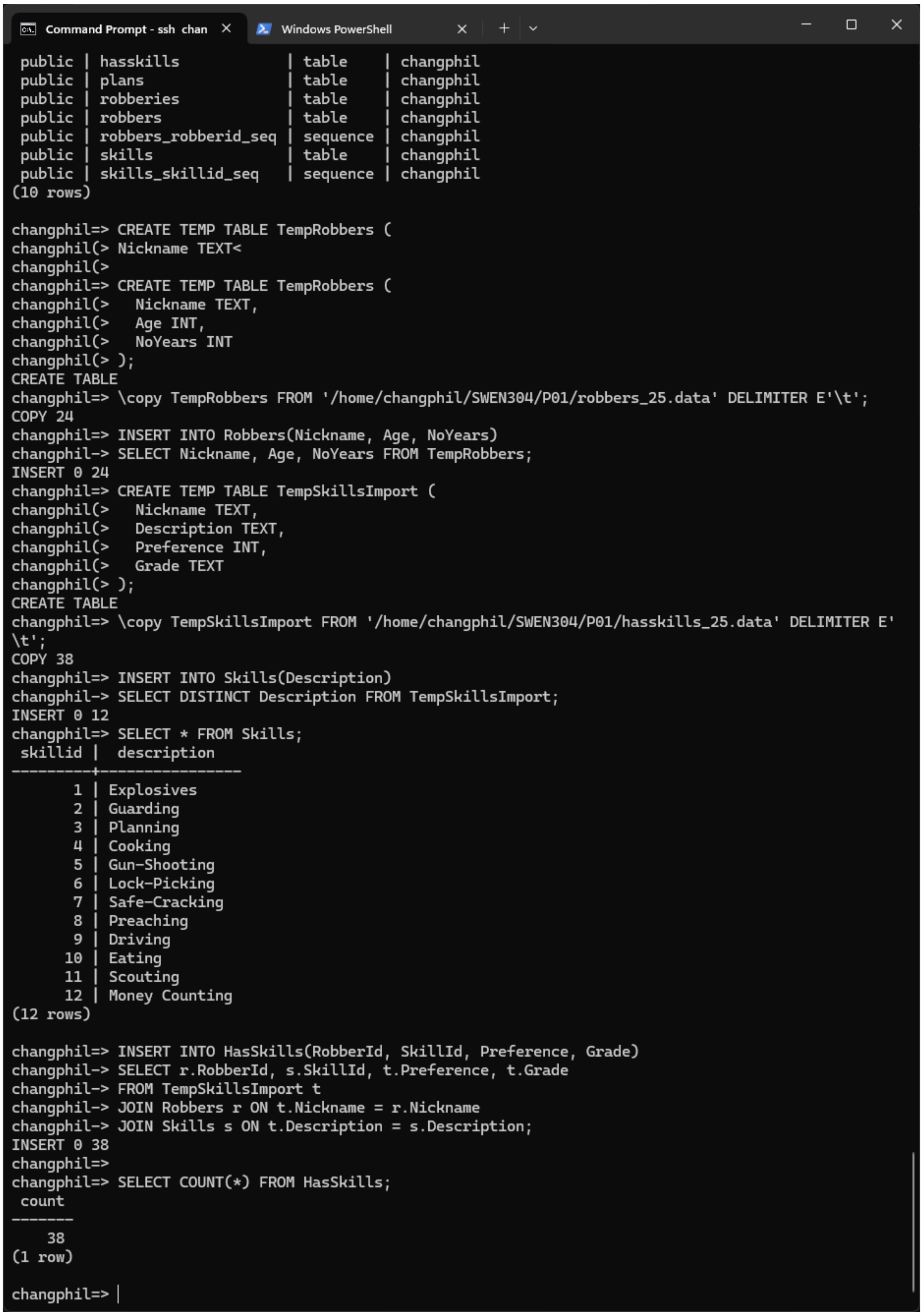
```
accomplices_25.data  hasaccounts_25.data  plans_25.data      robbers_25.data
banks_25.data        hasskills_25.data    robberies_25.data
barretts% createdb changphilP01
createdb: error: database creation failed: ERROR:  database "changphilP01" already exists
barretts% dropdb changphilP01
barretts% createdb changphil
barretts% ls
accomplices_25.data  hasaccounts_25.data  plans_25.data      robbers_25.data
banks_25.data        hasskills_25.data    robberies_25.data
barretts% createdb changphil
createdb: error: database creation failed: ERROR:  database "changphil" already exists
barretts% REVOKE CONNECT ON DATABASE changphil FROM PUBLIC;
zsh: command not found: REVOKE
barretts% psql changphil
psql (14.14, server 14.12)
GSSAPI-encrypted connection
Type "help" for help.

changphil=> REVOKE CONNECT ON DATABASE changphil FROM PUBLIC;
REVOKE
changphil=> \i Question1.sql
psql:Question1.sql:2: NOTICE:  table "accomplices" does not exist, skipping
psql:Question1.sql:2: NOTICE:  table "hasaccounts" does not exist, skipping
psql:Question1.sql:2: NOTICE:  table "hasskills" does not exist, skipping
psql:Question1.sql:2: NOTICE:  table "plans" does not exist, skipping
psql:Question1.sql:2: NOTICE:  table "robberies" does not exist, skipping
psql:Question1.sql:2: NOTICE:  table "robbers" does not exist, skipping
psql:Question1.sql:2: NOTICE:  table "skills" does not exist, skipping
psql:Question1.sql:2: NOTICE:  table "banks" does not exist, skipping
DROP TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
changphil=> \dt
            List of relations
 Schema |    Name     | Type  |   Owner
--------+-------------+-------+-----------
 public | accomplices | table | changphil
 public | banks       | table | changphil
 public | hasaccounts | table | changphil
 public | hasskills   | table | changphil
 public | plans       | table | changphil
 public | robberies   | table | changphil
 public | robbers     | table | changphil
 public | skills      | table | changphil
(8 rows)

changphil=> |
```



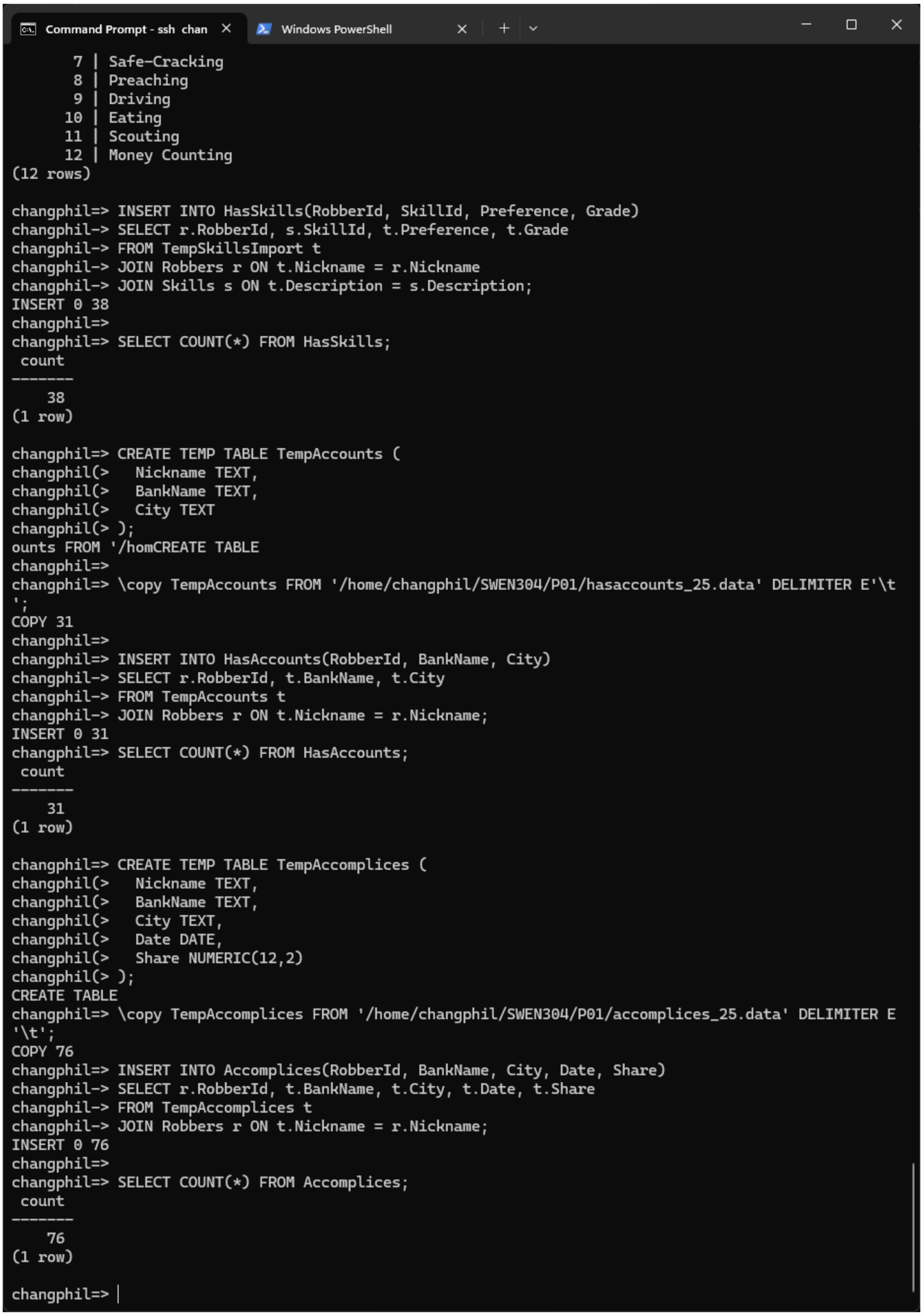
```
changphil=> \copy Banks(BankName, City, NoAccounts, Security) FROM '/home/changphil/SWEN304/P01/bank
s_25.data' DELIMITER E'\t';
COPY 20
changphil=> \copy Robberies(BankName, City, Date, Amount) FROM '/home/changphil/SWEN304/P01/robberie
s_25.data' DELIMITER E'\t';
ERROR:  invalid input syntax for type integer: "34302.3"
CONTEXT:  COPY robberies, line 1, column amount: "34302.3"
changphil=> \d
                 List of relations
 Schema |         Name          |   Type   |   Owner
--------+-----------------------+----------+-----------
 public | accomplices           | table    | changphil
 public | banks                 | table    | changphil
 public | hasaccounts           | table    | changphil
 public | hasskills             | table    | changphil
 public | plans                 | table    | changphil
 public | robberies             | table    | changphil
 public | robbers               | table    | changphil
 public | robbers_robberid_seq  | sequence | changphil
 public | skills                | table    | changphil
 public | skills_skillid_seq    | sequence | changphil
(10 rows)

changphil=> \d Robberies
              Table "public.robberies"
  Column  |  Type   | Collation | Nullable | Default
----------+---------+-----------+----------+---------
 bankname | text    |           | not null |
 city     | text    |           | not null |
 date     | date    |           | not null |
 amount   | integer |           | not null |
Indexes:
    "robberies_pkey" PRIMARY KEY, btree (bankname, city, date)
Check constraints:
    "robberies_amount_check" CHECK (amount >= 0)
Foreign-key constraints:
    "robberies_bankname_city_fkey" FOREIGN KEY (bankname, city) REFERENCES banks(bankname, city) ON
UPDATE CASCADE ON DELETE RESTRICT
Referenced by:
    TABLE "accomplices" CONSTRAINT "accomplices_bankname_city_date_fkey" FOREIGN KEY (bankname, city
, date) REFERENCES robberies(bankname, city, date) ON UPDATE CASCADE ON DELETE CASCADE

changphil=> ALTER TABLE Robberies
changphil-> ALTER COLUMN Amount TYPE NUMERIC(12,2)
changphil-> USING Amount::NUMERIC;
ALTER TABLE
changphil=> \d Robberies
                 Table "public.robberies"
  Column  |     Type      | Collation | Nullable | Default
----------+---------------+-----------+----------+---------
 bankname | text          |           | not null |
 city     | text          |           | not null |
 date     | date          |           | not null |
 amount   | numeric(12,2) |           | not null |
Indexes:
    "robberies_pkey" PRIMARY KEY, btree (bankname, city, date)
Check constraints:
    "robberies_amount_check" CHECK (amount >= 0::numeric)
Foreign-key constraints:
    "robberies_bankname_city_fkey" FOREIGN KEY (bankname, city) REFERENCES banks(bankname, city) ON
UPDATE CASCADE ON DELETE RESTRICT
Referenced by:
    TABLE "accomplices" CONSTRAINT "accomplices_bankname_city_date_fkey" FOREIGN KEY (bankname, city
, date) REFERENCES robberies(bankname, city, date) ON UPDATE CASCADE ON DELETE CASCADE

changphil=> |
```



```
  Column  |     Type      | Collation | Nullable | Default
----------+---------------+-----------+----------+---------
 bankname | text          |           | not null |
 city     | text          |           | not null |
 date     | date          |           | not null |
 amount   | numeric(12,2) |           | not null |
Indexes:
    "robberies_pkey" PRIMARY KEY, btree (bankname, city, date)
Check constraints:
    "robberies_amount_check" CHECK (amount >= 0::numeric)
Foreign-key constraints:
    "robberies_bankname_city_fkey" FOREIGN KEY (bankname, city) REFERENCES banks(bankname, city) ON
UPDATE CASCADE ON DELETE RESTRICT
Referenced by:
    TABLE "accomplices" CONSTRAINT "accomplices_bankname_city_date_fkey" FOREIGN KEY (bankname, city
, date) REFERENCES robberies(bankname, city, date) ON UPDATE CASCADE ON DELETE CASCADE

changphil=> \copy Robberies(BankName, City, Date, Amount) FROM '/home/changphil/SWEN304/P01/robberie
s_25.data' DELIMITER E'\t';
COPY 21
changphil=> \copy Plans(BankName, City, NoRobbers, PlannedDate) FROM '/home/changphil/SWEN304/P01/pl
ans_25.data' DELIMITER E'\t';
ERROR:  invalid input syntax for type integer: "2019-10-30"
CONTEXT:  COPY plans, line 1, column norobbers: "2019-10-30"
changphil=> \d Plans
                  Table "public.plans"
   Column    |  Type   | Collation | Nullable | Default
-------------+---------+-----------+----------+---------
 bankname    | text    |           | not null |
 city        | text    |           | not null |
 planneddate | date    |           | not null |
 norobbers   | integer |           | not null |
Indexes:
    "plans_pkey" PRIMARY KEY, btree (bankname, city, planneddate)
Check constraints:
    "plans_norobbers_check" CHECK (norobbers >= 1)
Foreign-key constraints:
    "plans_bankname_city_fkey" FOREIGN KEY (bankname, city) REFERENCES banks(bankname, city) ON UPDA
TE CASCADE ON DELETE RESTRICT

changphil=> \copy Plans(BankName, City, NoRobbers, PlannedDate) FROM '/home/changphil/SWEN304/P01/pl
ans_25.data' DELIMITER E'\t';
ERROR:  invalid input syntax for type integer: "2019-10-30"
CONTEXT:  COPY plans, line 1, column norobbers: "2019-10-30"
changphil=> \copy Plans(BankName, City, PlannedDate, NoRobbers) FROM '/home/changphil/SWEN304/P01/pl
ans_25.data' DELIMITER E'\t';
COPY 11
changphil=> \d Robbers
                              Table "public.robbers"
  Column  |  Type   | Collation | Nullable |                  Default
----------+---------+-----------+----------+-------------------------------------------
 robberid | integer |           | not null | nextval('robbers_robberid_seq'::regclass)
 nickname | text    |           | not null |
 age      | integer |           | not null |
 noyears  | integer |           | not null |
Indexes:
    "robbers_pkey" PRIMARY KEY, btree (robberid)
Check constraints:
    "robbers_age_check" CHECK (age > 0)
    "robbers_check" CHECK (noyears >= 0 AND noyears <= age)
Referenced by:
    TABLE "accomplices" CONSTRAINT "accomplices_robberid_fkey" FOREIGN KEY (robberid) REFERENCES rob
bers(robberid) ON UPDATE CASCADE ON DELETE CASCADE
    TABLE "hasaccounts" CONSTRAINT "hasaccounts_robberid_fkey" FOREIGN KEY (robberid) REFERENCES rob
bers(robberid) ON UPDATE CASCADE ON DELETE CASCADE
    TABLE "hasskills" CONSTRAINT "hasskills_robberid_fkey" FOREIGN KEY (robberid) REFERENCES robbers
(robberid) ON UPDATE CASCADE ON DELETE CASCADE
```



```
 public | hasskills            | table    | changphil
 public | plans                | table    | changphil
 public | robberies            | table    | changphil
 public | robbers              | table    | changphil
 public | robbers_robberid_seq | sequence | changphil
 public | skills               | table    | changphil
 public | skills_skillid_seq   | sequence | changphil
(10 rows)

changphil=> CREATE TEMP TABLE TempRobbers (
changphil(> Nickname TEXT<
changphil(>
changphil=> CREATE TEMP TABLE TempRobbers (
changphil(>   Nickname TEXT,
changphil(>   Age INT,
changphil(>   NoYears INT
changphil(> );
CREATE TABLE
changphil=> \copy TempRobbers FROM '/home/changphil/SWEN304/P01/robbers_25.data' DELIMITER E'\t';
COPY 24
changphil=> INSERT INTO Robbers(Nickname, Age, NoYears)
changphil-> SELECT Nickname, Age, NoYears FROM TempRobbers;
INSERT 0 24
changphil=> CREATE TEMP TABLE TempSkillsImport (
changphil(>   Nickname TEXT,
changphil(>   Description TEXT,
changphil(>   Preference INT,
changphil(>   Grade TEXT
changphil(> );
CREATE TABLE
changphil=> \copy TempSkillsImport FROM '/home/changphil/SWEN304/P01/hasskills_25.data' DELIMITER E'
\t';
COPY 38
changphil=> INSERT INTO Skills(Description)
changphil-> SELECT DISTINCT Description FROM TempSkillsImport;
INSERT 0 12
changphil=> SELECT * FROM Skills;
 skillid |  description
---------+----------------
       1 | Explosives
       2 | Guarding
       3 | Planning
       4 | Cooking
       5 | Gun-Shooting
       6 | Lock-Picking
       7 | Safe-Cracking
       8 | Preaching
       9 | Driving
      10 | Eating
      11 | Scouting
      12 | Money Counting
(12 rows)

changphil=> INSERT INTO HasSkills(RobberId, SkillId, Preference, Grade)
changphil-> SELECT r.RobberId, s.SkillId, t.Preference, t.Grade
changphil-> FROM TempSkillsImport t
changphil-> JOIN Robbers r ON t.Nickname = r.Nickname
changphil-> JOIN Skills s ON t.Description = s.Description;
INSERT 0 38
changphil=>
changphil=> SELECT COUNT(*) FROM HasSkills;
 count
-------
    38
(1 row)

changphil=> |
```



Command Prompt - ssh chan | Windows PowerShell

```
       7 | Safe-Cracking
       8 | Preaching
       9 | Driving
      10 | Eating
      11 | Scouting
      12 | Money Counting
(12 rows)

changphil=> INSERT INTO HasSkills(RobberId, SkillId, Preference, Grade)
changphil-> SELECT r.RobberId, s.SkillId, t.Preference, t.Grade
changphil-> FROM TempSkillsImport t
changphil-> JOIN Robbers r ON t.Nickname = r.Nickname
changphil-> JOIN Skills s ON t.Description = s.Description;
INSERT 0 38
changphil=>
changphil=> SELECT COUNT(*) FROM HasSkills;
 count
-------
    38
(1 row)

changphil=> CREATE TEMP TABLE TempAccounts (
changphil(>   Nickname TEXT,
changphil(>   BankName TEXT,
changphil(>   City TEXT
changphil(> );
ounts FROM '/homCREATE TABLE
changphil=>
changphil=> \copy TempAccounts FROM '/home/changphil/SWEN304/P01/hasaccounts_25.data' DELIMITER E'\t
';
COPY 31
changphil=>
changphil=> INSERT INTO HasAccounts(RobberId, BankName, City)
changphil-> SELECT r.RobberId, t.BankName, t.City
changphil-> FROM TempAccounts t
changphil-> JOIN Robbers r ON t.Nickname = r.Nickname;
INSERT 0 31
changphil=> SELECT COUNT(*) FROM HasAccounts;
 count
-------
    31
(1 row)

changphil=> CREATE TEMP TABLE TempAccomplices (
changphil(>   Nickname TEXT,
changphil(>   BankName TEXT,
changphil(>   City TEXT,
changphil(>   Date DATE,
changphil(>   Share NUMERIC(12,2)
changphil(> );
CREATE TABLE
changphil=> \copy TempAccomplices FROM '/home/changphil/SWEN304/P01/accomplices_25.data' DELIMITER E
'\t';
COPY 76
changphil=> INSERT INTO Accomplices(RobberId, BankName, City, Date, Share)
changphil-> SELECT r.RobberId, t.BankName, t.City, t.Date, t.Share
changphil-> FROM TempAccomplices t
changphil-> JOIN Robbers r ON t.Nickname = r.Nickname;
INSERT 0 76
changphil=>
changphil=> SELECT COUNT(*) FROM Accomplices;
 count
-------
    76
(1 row)

changphil=> |
```

## 1. Description of data conversion performance.

I populated the database using a mix of direct file imports and SQL transformations with temporary tables and joins. Below is a sequence of steps that I performed:

First, I imported Banks, Robberies, and Plans via direct imports using the \copy command. Before importing Robberies, I altered the Amount column to NUMERIC(12,2) to accommodate for decimal values.

1. \copy Banks(BankName, City, NoAccounts, Security) FROM '/home/changphil/SWEN304/P01/banks_25.data' DELIMITER E'\t';
2. \copy Robberies(BankName, City, Date, Amount) FROM '/home/changphil/SWEN304/P01/robberies_25.data' DELIMITER E'\t';
3. \copy Plans(BankName, City, PlannedDate, NoRobbers) FROM '/home/changphil/SWEN304/P01/plans_25.data' DELIMITER E'\t';

Next, I used a temporary table to load robbers and inserted into the final Robbers table:

- CREATE TEMP TABLE TempRobbers (Nickname TEXT, Age INT, NoYears INT);
- \copy TempRobbers FROM '/home/changphil/SWEN304/P01/robbers_25.data' DELIMITER E'\t';
- INSERT INTO Robbers(Nickname, Age, NoYears) SELECT Nickname, Age, NoYears FROM TempRobbers;

After that, I loaded the raw skill data into a temporary table called TempSkillsImport. From here, I extracted all distinct skill descriptions and inserted them into the Skills table. Then, I populated HasSkills by joining the temporary skill data with both Robbers and Skills.

- CREATE TEMP TABLE TempSkillsImport (Nickname TEXT, Description TEXT, Preference INT, Grade TEXT);
- \copy TempSkillsImport FROM '/home/changphil/SWEN304/P01/hasskills_25.data' DELIMITER E'\t';
-
- INSERT INTO Skills(Description) SELECT DISTINCT Description FROM TempSkillsImport;
-
- INSERT INTO HasSkills(RobberId, SkillId, Preference, Grade)
- SELECT r.RobberId, s.SkillId, t.Preference, t.Grade
- FROM TempSkillsImport t
- JOIN Robbers r ON t.Nickname = r.Nickname
- JOIN Skills s ON t.Description = s.Description;

The data in hasaccounts_25.data referenced robbers by nickname, so I created a temporary table to hold the raw data. Using a join with the Robbers table, I inserted the corresponding RobberId values into the final HasAccounts table alongside the bank name and city.

- CREATE TEMP TABLE TempAccounts (Nickname TEXT, BankName TEXT, City TEXT);
- \copy TempAccounts FROM '/home/changphil/SWEN304/P01/hasaccounts_25.data' DELIMITER E'\t';
-
- INSERT INTO HasAccounts(RobberId, BankName, City)
- SELECT r.RobberId, t.BankName, t.City
- FROM TempAccounts t
- JOIN Robbers r ON t.Nickname = r.Nickname;

Finally, as with the other nickname-based data, I created a temporary table to load the raw accomplice records. I made sure the Share column used NUMERIC(12,2) to allow decimal values. Then I joined the temporary table with Robbers to retrieve each RobberId and inserted the final data into the Accomplices table.

- CREATE TEMP TABLE TempAccomplices (Nickname TEXT, BankName TEXT, City TEXT, Date DATE, Share NUMERIC(12,2));
- \copy TempAccomplices FROM '/home/changphil/SWEN304/P01/accomplices_25.data' DELIMITER E'\t';
-
- INSERT INTO Accomplices(RobberId, BankName, City, Date, Share)
- SELECT r.RobberId, t.BankName, t.City, t.Date, t.Share
- FROM TempAccomplices t
- JOIN Robbers r ON t.Nickname = r.Nickname;

## 2. Table Implementation Order and Justification

The order that I implemented the tables of the database was chosen to satisfy foreign key dependencies and minimize errors during the insertion.

First, I started with the Banks, as it is an independent table, required by the Robberies, Plans, and HasAccounts tables.

Second, I loaded Robberies (which failed at first due to the decimal value) after fixing the Amount type as it is required before loading Accomplices.

Third, I loaded Plans (original column order caused failure) after correcting column order as it refers to Banks, but is not referenced by any other table, meaning it was safe to load early.

Fourth, I loaded Robbers as many of the following tables depend on Robbers (HasSkills, HasAccounts, Accomplices).

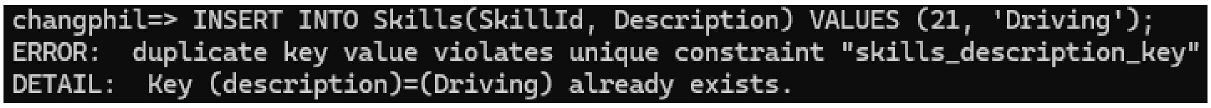
Fifth, I loaded Skills and then HasSkills as they are related, and HasSkills depends on both Robbers and Skills.

Sixth, I loaded HasAccounts using a join on Robbers. I loaded it here as it depends on both Robbers and Banks.

Lastly, I loaded Accomplices here at the end. No real reason it was last, but it was loaded towards the end as it relies on both Robbers and Robberies and includes references to Robberies.

## Question 03: Checking your Database [10 marks]

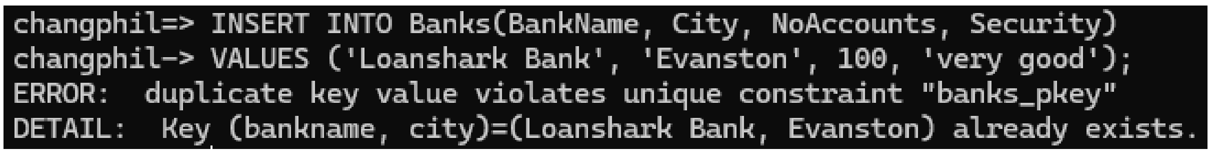
1. Insert the following tuple into the Skills table:
    a. (21, ‘Driving’)



```
changphil=> INSERT INTO Skills(SkillId, Description) VALUES (21, 'Driving');
ERROR:  duplicate key value violates unique constraint "skills_description_key"
DETAIL:  Key (description)=(Driving) already exists.
```

Here, we can see that this violates the UNIQUE constraint on the Description column of Skills.

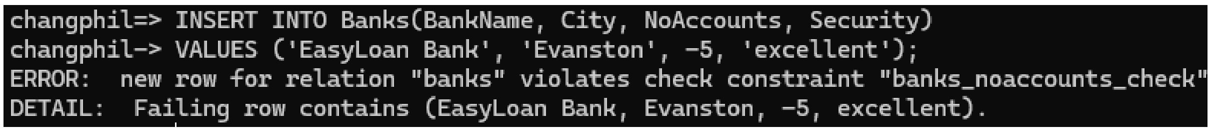
2. Insert the following tuples into the Banks table:
    a. (‘Loanshark Bank’, ‘Evanston’, 100, ‘very good’)



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

Here, we can see that it violated the primary key constraint on (BankName, City) in Banks.

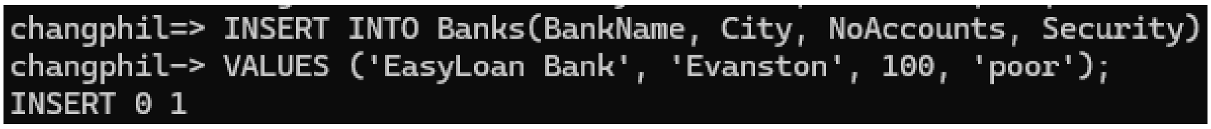
    b. (‘EasyLoan Bank’, ‘Evanston’, -5, ‘excellent’)



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

Here, it violates the CHECK constraint on NoAccounts (NoAccounts >= 0).

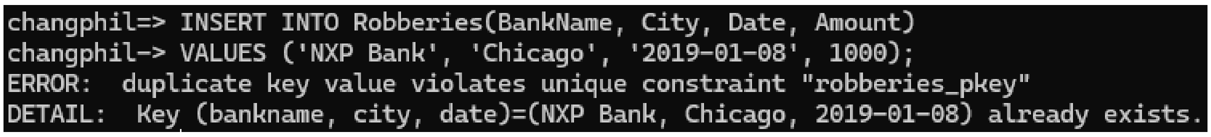
    c. (‘EasyLoan Bank’, ‘Evanston’, 100, ‘poor’)



```
changphil=> INSERT INTO Banks(BankName, City, NoAccounts, Security)
changphil-> VALUES ('EasyLoan Bank', 'Evanston', 100, 'poor');
INSERT 0 1
```

Here, it successfully inserted meaning that I did not correctly implement a CHECK constraint on the Security column to only allow values like ‘low’, ‘medium’, or ‘high’.

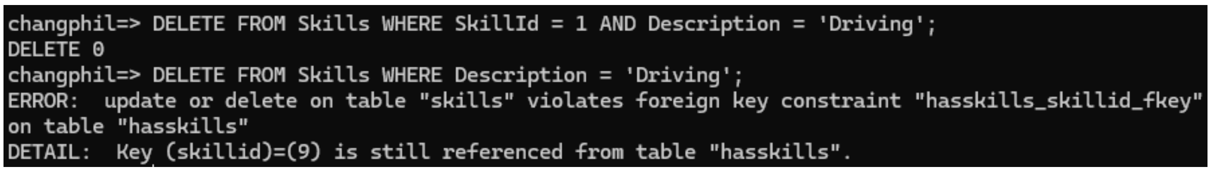
3. Insert the following tuple into the Robberies table:
    a. (‘NXP Bank’, ‘Chicago’, ‘2019-01-08’, 1000)



```
changphil=> INSERT INTO Robberies(BankName, City, Date, Amount)
changphil-> VALUES ('NXP Bank', 'Chicago', '2019-01-08', 1000);
ERROR:  duplicate key value violates unique constraint "robberies_pkey"
DETAIL:  Key (bankname, city, date)=(NXP Bank, Chicago, 2019-01-08) already exists.
```

Here, it violates the unique primary key constraint on (BankName, City, Date) in Robberies.

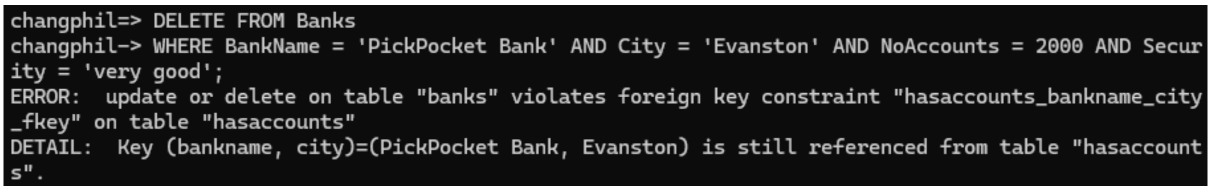
4. Delete the following tuple from the Skills table:
    a. (1, ‘Driving’)



```
changphil=> DELETE FROM Skills WHERE SkillId = 1 AND Description = 'Driving';
DELETE 0
changphil=> DELETE FROM Skills WHERE Description = 'Driving';
ERROR:  update or delete on table "skills" violates foreign key constraint "hasskills_skillid_fkey"
on table "hasskills"
DETAIL:  Key (skillid)=(9) is still referenced from table "hasskills".
```

Here, we can see that it violated the foreign key constraint, meaning it could not be deleted.

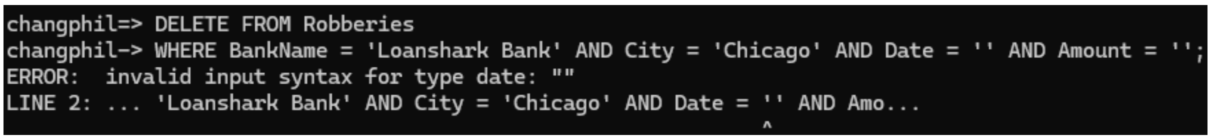
5. Delete the following tuples from the Banks table:
    a. (‘PickPocket Bank’, ‘Evanston’, 2000, ‘very good’)



```
changphil=> DELETE FROM Banks
changphil-> WHERE BankName = 'PickPocket Bank' AND City = 'Evanston' AND NoAccounts = 2000 AND Secur
ity = 'very good';
ERROR:  update or delete on table "banks" violates foreign key constraint "hasaccounts_bankname_city
_fkey" on table "hasaccounts"
DETAIL:  Key (bankname, city)=(PickPocket Bank, Evanston) is still referenced from table "hasaccount
s".
```

Here it violates the foreign key constraint.

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

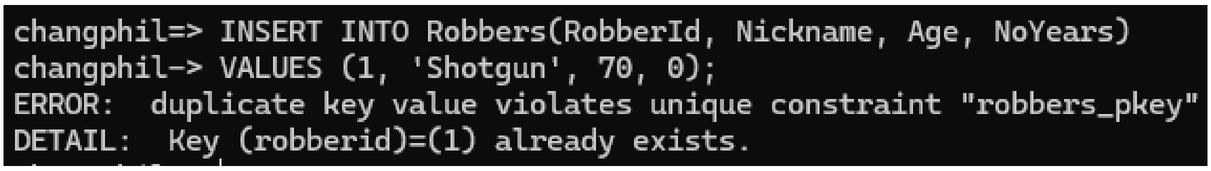


```
changphil=> DELETE FROM Robberies
changphil-> WHERE BankName = 'Loanshark Bank' AND City = 'Chicago' AND Date = '' AND Amount = '';
ERROR:  invalid input syntax for type date: ""
LINE 2: ... 'Loanshark Bank' AND City = 'Chicago' AND Date = '' AND Amo...
                                                             ^
```

Here, it does not technically violate a constraint but a type mismatch error. PostgreSQL cannot cast an empty string to a DATE type, causing it to fail.

In the following two tasks, we assume that there is a robber with Id 3, but no robber with Id 333.

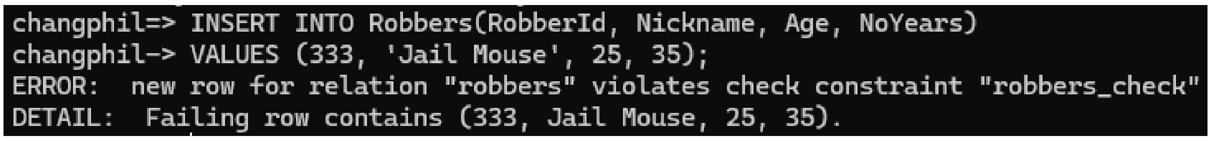
7. Insert the following tuples into the Robbers table:
   a. (1, ‘Shotgun’, 70, 0)



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

Here, it violates the primary key constraint on Robbers.RobberId.

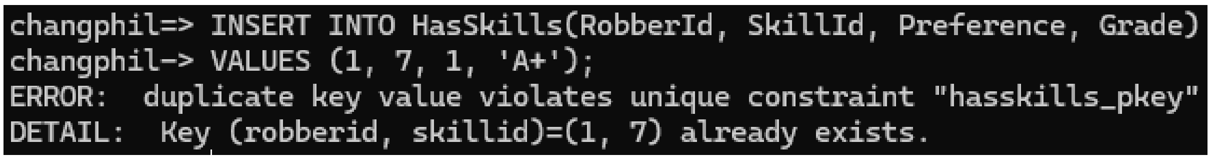
   b. (333, ‘Jail Mouse’, 25, 35)



```
changphil=> INSERT INTO Robbers(RobberId, Nickname, Age, NoYears)
changphil-> VALUES (333, 'Jail Mouse', 25, 35);
ERROR:  new row for relation "robbers" violates check constraint "robbers_check"
DETAIL:  Failing row contains (333, Jail Mouse, 25, 35).
```

Here, it violates the CHECK constraint: NoYears <= Age which is a part of robbers_check.

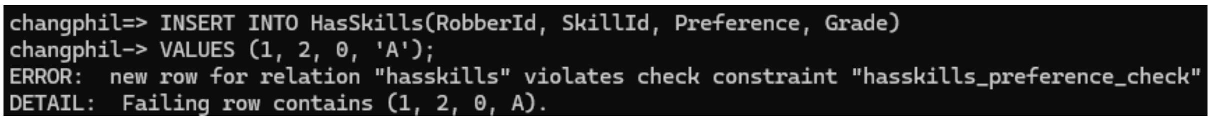
8. Insert the following tuples into the HasSkills table:
    a. (1, 7, 1, ‘A+’)



```
changphil=> INSERT INTO HasSkills(RobberId, SkillId, Preference, Grade)
changphil-> VALUES (1, 7, 1, 'A+');
ERROR:  duplicate key value violates unique constraint "hasskills_pkey"
DETAIL:  Key (robberid, skillid)=(1, 7) already exists.
```

Here, it violates the primary key constraint on (RobberId, SkillId) in HasSkills.

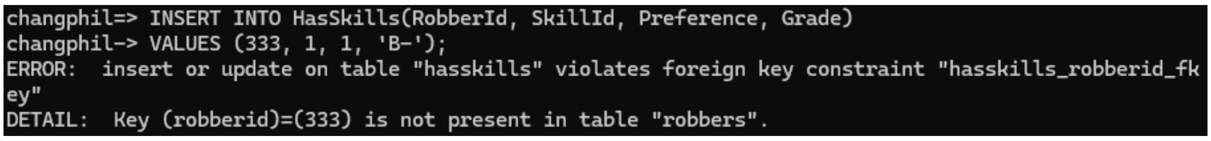
    b. (1, 2, 0, ‘A’)



```
changphil=> INSERT INTO HasSkills(RobberId, SkillId, Preference, Grade)
changphil-> VALUES (1, 2, 0, 'A');
ERROR:  new row for relation "hasskills" violates check constraint "hasskills_preference_check"
DETAIL:  Failing row contains (1, 2, 0, A).
```

Here it violates the CHECK constraint on Preference (likely Preference >= 1).

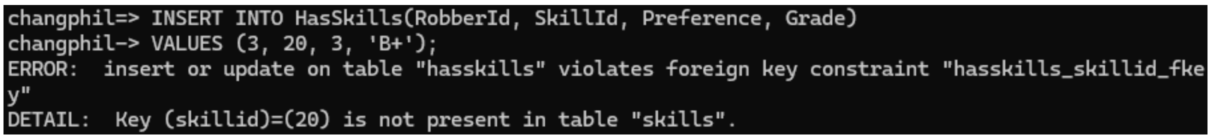
    c. (333, 1, 1, ‘B-‘)



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

Here, it violates the foreign key constraint from HasSkills.RobberId > Robbers.RobberId

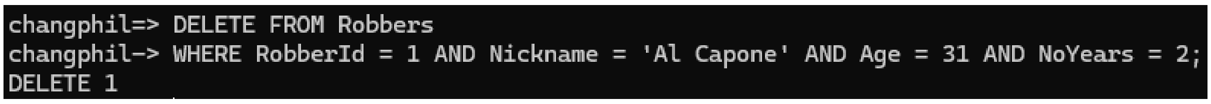
    d. (3, 20, 3, ‘B+’)



```
changphil=> INSERT INTO HasSkills(RobberId, SkillId, Preference, Grade)
changphil-> VALUES (3, 20, 3, 'B+');
ERROR:  insert or update on table "hasskills" violates foreign key constraint "hasskills_skillid_fke
y"
DETAIL:  Key (skillid)=(20) is not present in table "skills".
```

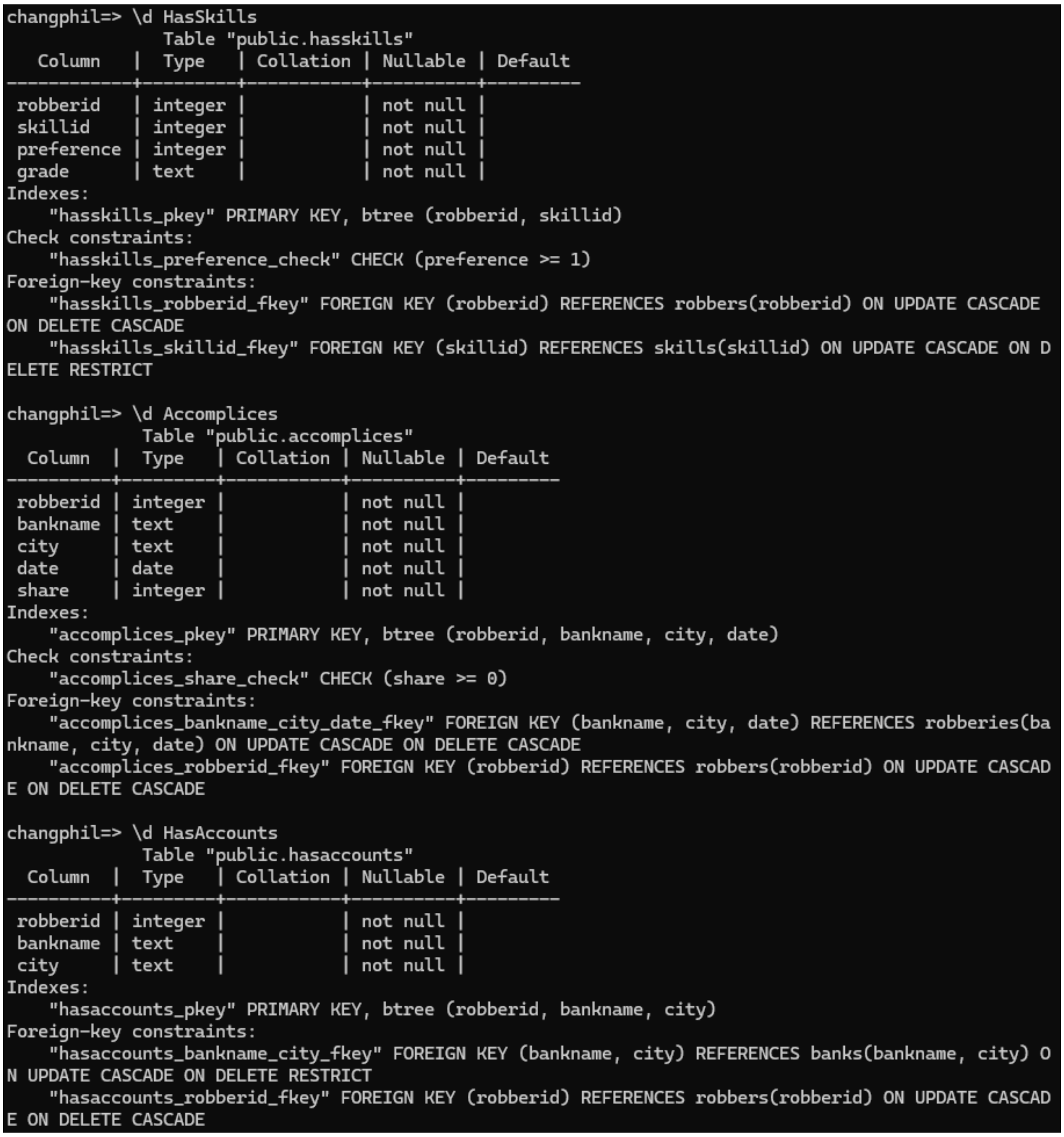
Here, it violates the foreign key constraint from HasSkills.SkillId > Skills.SkillId

9. Delete the following tuple from the Robbers table:
    a. (1, ‘Al Capone’, 31, 2)



```
changphil=> DELETE FROM Robbers
changphil-> WHERE RobberId = 1 AND Nickname = 'Al Capone' AND Age = 31 AND NoYears = 2;
DELETE 1
```

Here, no violation occurred, which means either there we no referencing rows in HasSkills, HasAccounts, or Accomplices, OR my foreign keys were defined with ON DELETE CASCADE, allowing the deletion to succeed cleanly.



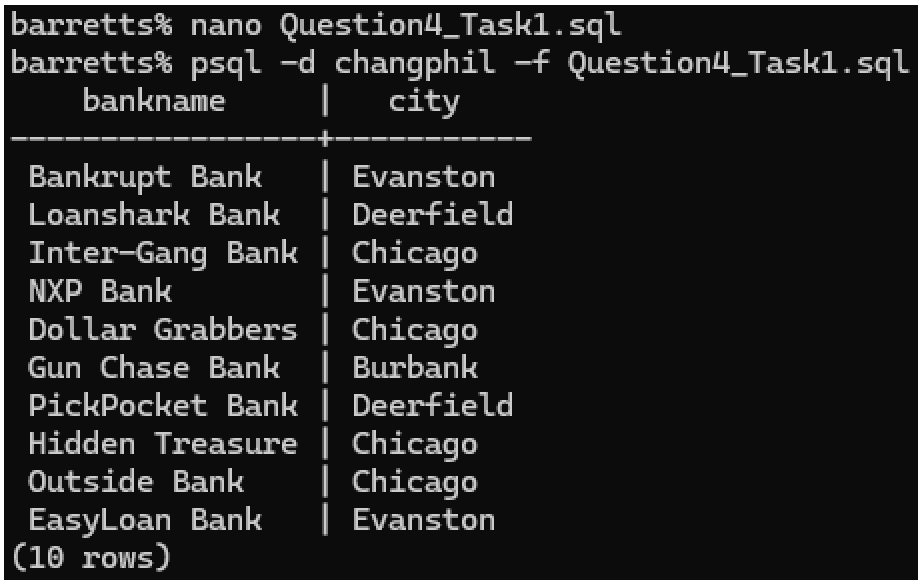
```
changphil=> \d HasSkills
               Table "public.hasskills"
   Column   |  Type   | Collation | Nullable | Default
------------+---------+-----------+----------+---------
 robberid   | integer |           | not null |
 skillid    | integer |           | not null |
 preference | integer |           | not null |
 grade      | text    |           | not null |
Indexes:
    "hasskills_pkey" PRIMARY KEY, btree (robberid, skillid)
Check constraints:
    "hasskills_preference_check" CHECK (preference >= 1)
Foreign-key constraints:
    "hasskills_robberid_fkey" FOREIGN KEY (robberid) REFERENCES robbers(robberid) ON UPDATE CASCADE
ON DELETE CASCADE
    "hasskills_skillid_fkey" FOREIGN KEY (skillid) REFERENCES skills(skillid) ON UPDATE CASCADE ON D
ELETE RESTRICT

changphil=> \d Accomplices
             Table "public.accomplices"
  Column  |  Type   | Collation | Nullable | Default
----------+---------+-----------+----------+---------
 robberid | integer |           | not null |
 bankname | text    |           | not null |
 city     | text    |           | not null |
 date     | date    |           | not null |
 share    | integer |           | not null |
Indexes:
    "accomplices_pkey" PRIMARY KEY, btree (robberid, bankname, city, date)
Check constraints:
    "accomplices_share_check" CHECK (share >= 0)
Foreign-key constraints:
    "accomplices_bankname_city_date_fkey" FOREIGN KEY (bankname, city, date) REFERENCES robberies(ba
nkname, city, date) ON UPDATE CASCADE ON DELETE CASCADE
    "accomplices_robberid_fkey" FOREIGN KEY (robberid) REFERENCES robbers(robberid) ON UPDATE CASCAD
E ON DELETE CASCADE

changphil=> \d HasAccounts
             Table "public.hasaccounts"
  Column  |  Type   | Collation | Nullable | Default
----------+---------+-----------+----------+---------
 robberid | integer |           | not null |
 bankname | text    |           | not null |
 city     | text    |           | not null |
Indexes:
    "hasaccounts_pkey" PRIMARY KEY, btree (robberid, bankname, city)
Foreign-key constraints:
    "hasaccounts_bankname_city_fkey" FOREIGN KEY (bankname, city) REFERENCES banks(bankname, city) O
N UPDATE CASCADE ON DELETE RESTRICT
    "hasaccounts_robberid_fkey" FOREIGN KEY (robberid) REFERENCES robbers(robberid) ON UPDATE CASCAD
E ON DELETE CASCADE
```

To stop this from happening, I could have used ON DELETE RESTRICT instead of ON DELETE CASCADE. This would hopefully result in a message like below rather than deletion of the tuple:

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

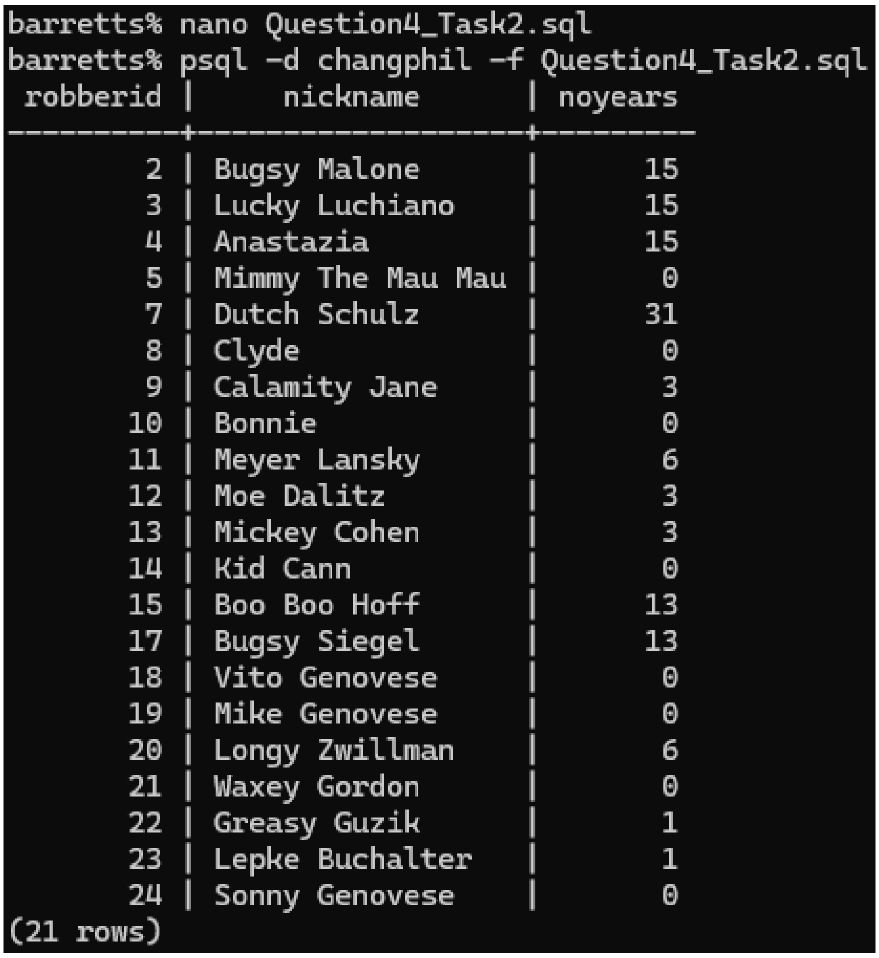
**Question 04: Simple Database Queries [24 marks]**

1. Retrieve BankName and City of all banks that have never been robbed.
   - SELECT BankName, City
   - FROM Banks
   - WHERE NOT EXISTS (
   -  SELECT 1
   -  FROM Robberies
   -  WHERE Robberies.BankName = Banks.BankName
   -   AND Robberies.City = Banks.City
   - );



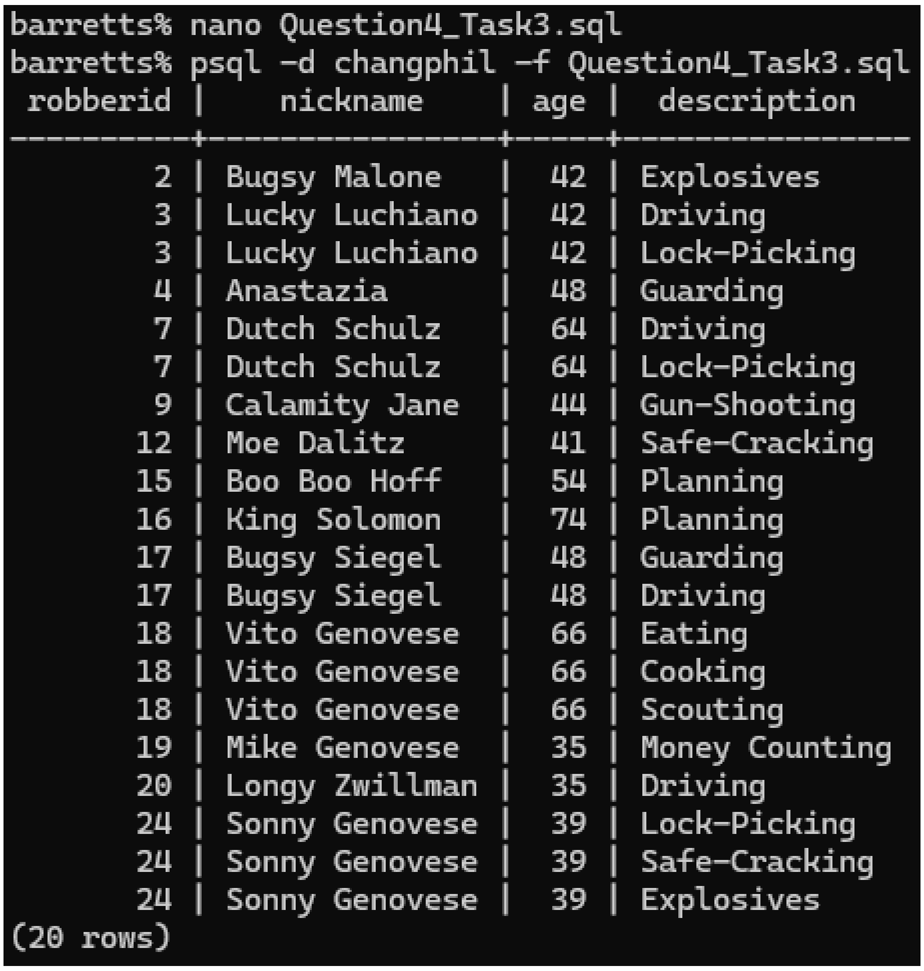
```
barretts% nano Question4_Task1.sql
barretts% psql -d changphil -f Question4_Task1.sql
    bankname     |   city
-----------------+-----------
 Bankrupt Bank   | Evanston
 Loanshark Bank  | Deerfield
 Inter-Gang Bank | Chicago
 NXP Bank        | Evanston
 Dollar Grabbers | Chicago
 Gun Chase Bank  | Burbank
 PickPocket Bank | Deerfield
 Hidden Treasure | Chicago
 Outside Bank    | Chicago
 EasyLoan Bank   | Evanston
(10 rows)
```

2. Retrieve RobberId, Nickname and the Number of years not spent in prison for all robbers who spent more than half of their life in prison.
   - SELECT RobberId, Nickname, NoYears
   - FROM Robbers
   - WHERE (Age - NoYears) > (Age / 2);



```
barretts% nano Question4_Task2.sql
barretts% psql -d changphil -f Question4_Task2.sql
 robberid |      nickname      | noyears
----------+--------------------+---------
        2 | Bugsy Malone       |      15
        3 | Lucky Luchiano     |      15
        4 | Anastazia          |      15
        5 | Mimmy The Mau Mau  |       0
        7 | Dutch Schulz       |      31
        8 | Clyde              |       0
        9 | Calamity Jane      |       3
       10 | Bonnie             |       0
       11 | Meyer Lansky       |       6
       12 | Moe Dalitz         |       3
       13 | Mickey Cohen       |       3
       14 | Kid Cann           |       0
       15 | Boo Boo Hoff       |      13
       17 | Bugsy Siegel       |      13
       18 | Vito Genovese      |       0
       19 | Mike Genovese      |       0
       20 | Longy Zwillman     |       6
       21 | Waxey Gordon       |       0
       22 | Greasy Guzik       |       1
       23 | Lepke Buchalter    |       1
       24 | Sonny Genovese     |       0
(21 rows)
```

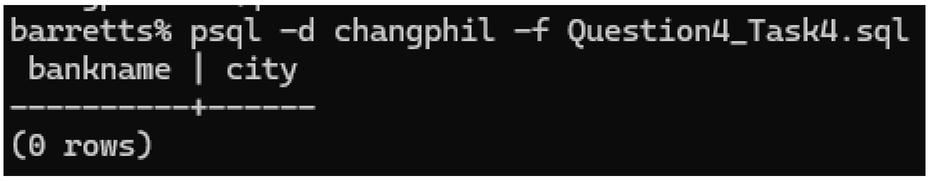
3. Retrieve RobberId, Nickname, Age, and all skill descriptions of all robbers who are not younger than 35 years.
   - SELECT r.RobberId, r.Nickname, r.Age, s.Description
   - FROM Robbers r
   - JOIN HasSkills hs ON r.RobberId = hs.RobberId
   - JOIN Skills s ON hs.SkillId = s.SkillId
   - WHERE r.Age >= 35;



```
barretts% nano Question4_Task3.sql
barretts% psql -d changphil -f Question4_Task3.sql
 robberid |    nickname     | age |  description
----------+-----------------+-----+----------------
        2 | Bugsy Malone    |  42 | Explosives
        3 | Lucky Luchiano  |  42 | Driving
        3 | Lucky Luchiano  |  42 | Lock-Picking
        4 | Anastazia       |  48 | Guarding
        7 | Dutch Schulz    |  64 | Driving
        7 | Dutch Schulz    |  64 | Lock-Picking
        9 | Calamity Jane   |  44 | Gun-Shooting
       12 | Moe Dalitz      |  41 | Safe-Cracking
       15 | Boo Boo Hoff    |  54 | Planning
       16 | King Solomon    |  74 | Planning
       17 | Bugsy Siegel    |  48 | Guarding
       17 | Bugsy Siegel    |  48 | Driving
       18 | Vito Genovese   |  66 | Eating
       18 | Vito Genovese   |  66 | Cooking
       18 | Vito Genovese   |  66 | Scouting
       19 | Mike Genovese   |  35 | Money Counting
       20 | Longy Zwillman  |  35 | Driving
       24 | Sonny Genovese  |  39 | Lock-Picking
       24 | Sonny Genovese  |  39 | Safe-Cracking
       24 | Sonny Genovese  |  39 | Explosives
(20 rows)
```

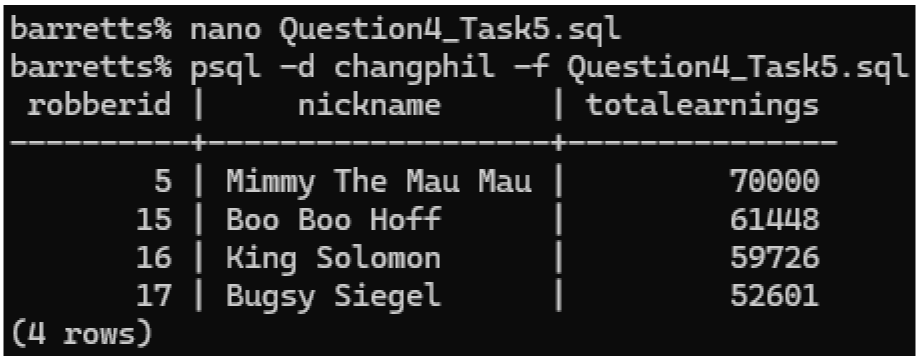
4. Retrieve BankName and city of all banks where Al Capone has an account. The answer should list every bank at most once.

**(I'm not sure if the output will be as expected, as during a previous question I deleted Al Capone and another value from the database when it should not have been removed. I overwrote the files with fresh ones from the datafiles.zip file so I hope it worked.)**



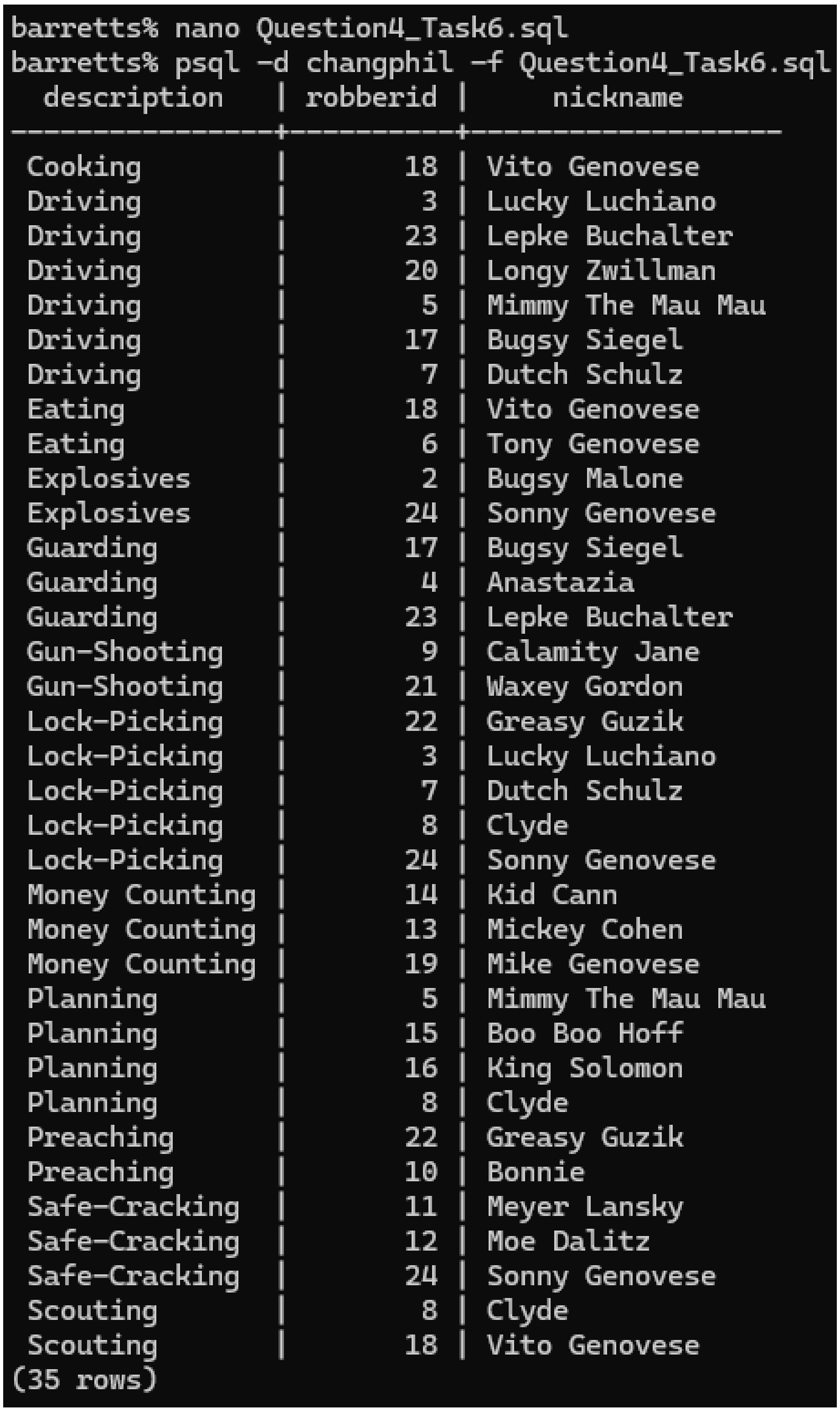
```
barretts% psql -d changphil -f Question4_Task4.sql
 bankname | city
----------+------
(0 rows)
```

5. Retrieve RobberId, Nickname and individual total "earnings" of those robbers who have earned at least $50,000 by robbing banks. The answer should be sorted in decreasing order of the total earnings.



```
barretts% nano Question4_Task5.sql
barretts% psql -d changphil -f Question4_Task5.sql
 robberid |     nickname      | totalearnings
----------+-------------------+---------------
        5 | Mimmy The Mau Mau |         70000
       15 | Boo Boo Hoff      |         61448
       16 | King Solomon      |         59726
       17 | Bugsy Siegel      |         52601
(4 rows)
```

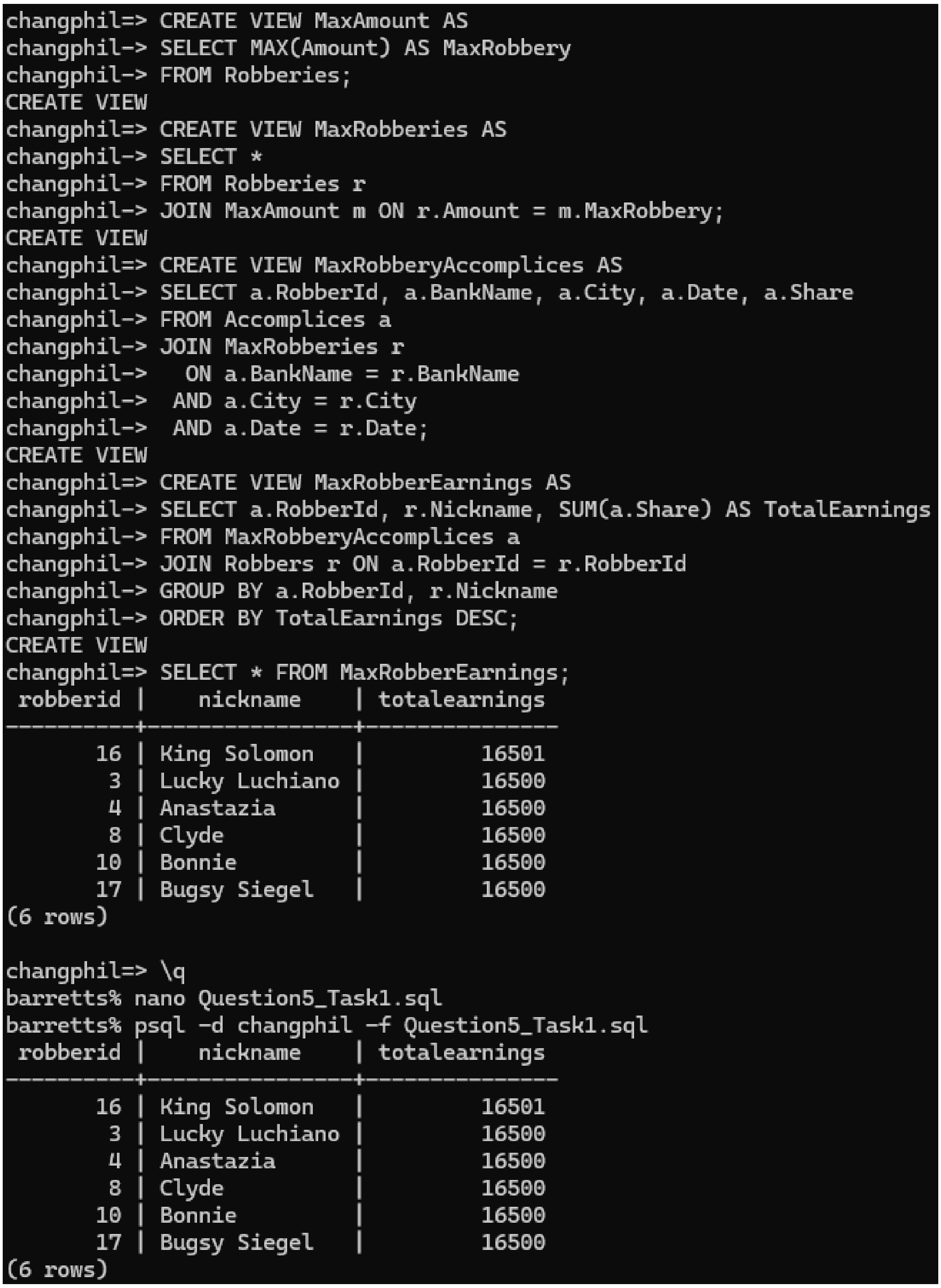
6. Retrieve the Description of all skills together with RobberId and NickName of all robbers who possess this skill. The answer should be ordered by skill description.



```
barretts% nano Question4_Task6.sql
barretts% psql -d changphil -f Question4_Task6.sql
  description   | robberid |     nickname
----------------+----------+-------------------
 Cooking        |       18 | Vito Genovese
 Driving        |        3 | Lucky Luchiano
 Driving        |       23 | Lepke Buchalter
 Driving        |       20 | Longy Zwillman
 Driving        |        5 | Mimmy The Mau Mau
 Driving        |       17 | Bugsy Siegel
 Driving        |        7 | Dutch Schulz
 Eating         |       18 | Vito Genovese
 Eating         |        6 | Tony Genovese
 Explosives     |        2 | Bugsy Malone
 Explosives     |       24 | Sonny Genovese
 Guarding       |       17 | Bugsy Siegel
 Guarding       |        4 | Anastazia
 Guarding       |       23 | Lepke Buchalter
 Gun-Shooting   |        9 | Calamity Jane
 Gun-Shooting   |       21 | Waxey Gordon
 Lock-Picking   |       22 | Greasy Guzik
 Lock-Picking   |        3 | Lucky Luchiano
 Lock-Picking   |        7 | Dutch Schulz
 Lock-Picking   |        8 | Clyde
 Lock-Picking   |       24 | Sonny Genovese
 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       |       16 | King Solomon
 Planning       |        8 | Clyde
 Preaching      |       22 | Greasy Guzik
 Preaching      |       10 | Bonnie
 Safe-Cracking  |       11 | Meyer Lansky
 Safe-Cracking  |       12 | Moe Dalitz
 Safe-Cracking  |       24 | Sonny Genovese
 Scouting       |        8 | Clyde
 Scouting       |       18 | Vito Genovese
(35 rows)
```

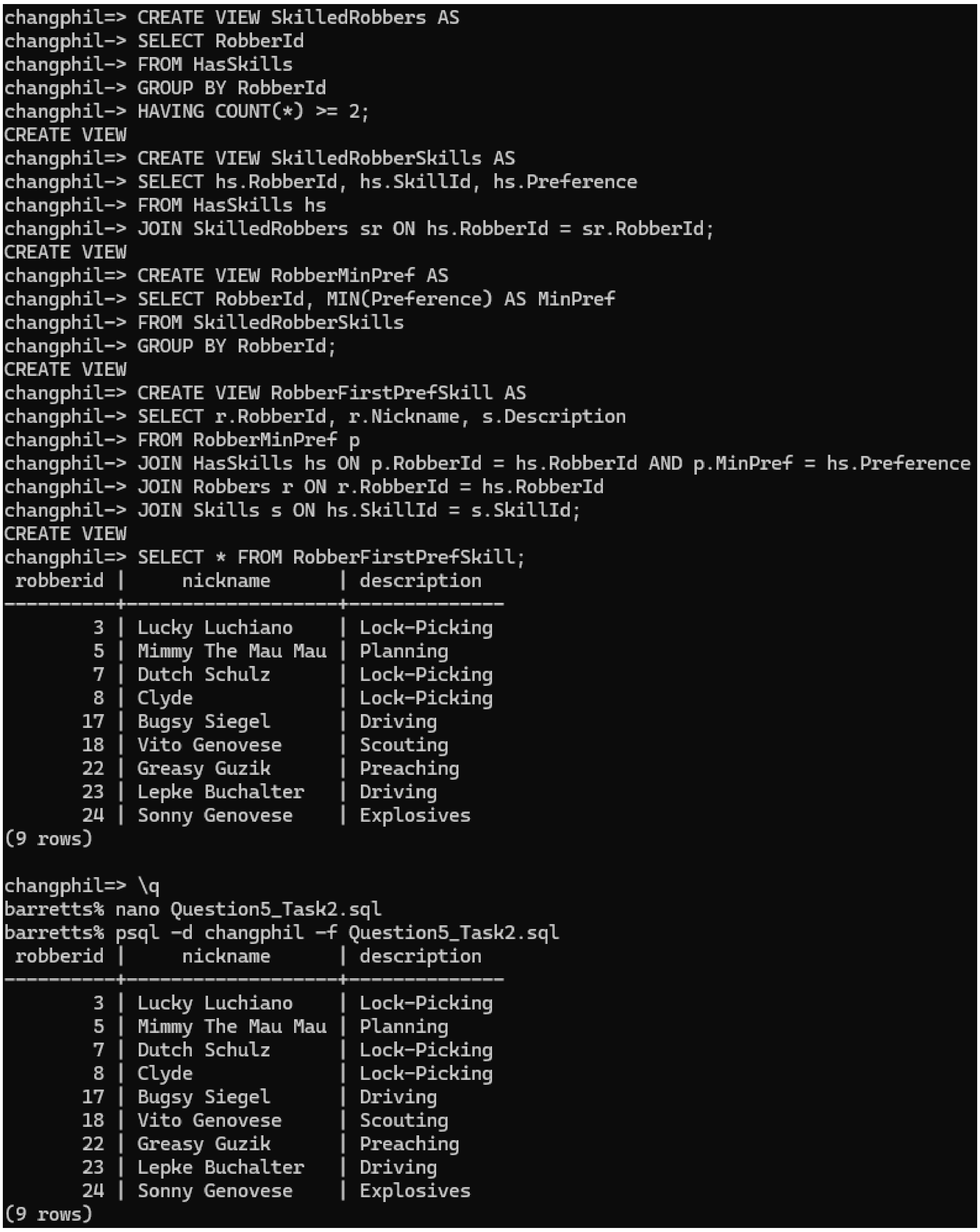
**Question 05: Complex Data Queries [20 marks]**

1. Retrieve RobberId, Nickname and individual total "earnings" of those robbers who participated in the robbery with the highest amount. The answer should be sorted in decreasing order of the total earnings.



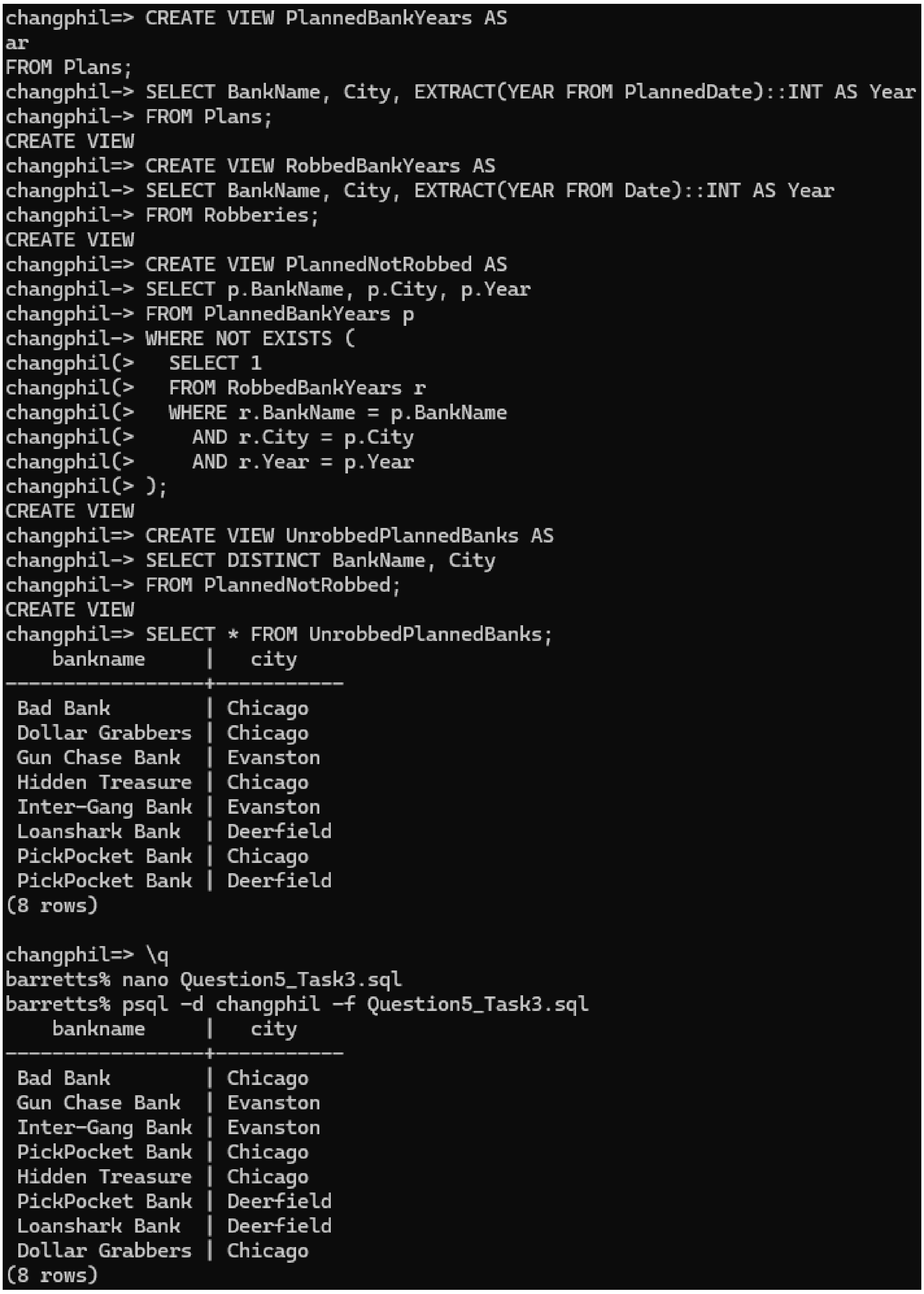
```
changphil=> CREATE VIEW MaxAmount AS
changphil-> SELECT MAX(Amount) AS MaxRobbery
changphil-> FROM Robberies;
CREATE VIEW
changphil=> CREATE VIEW MaxRobberies AS
changphil-> SELECT *
changphil-> FROM Robberies r
changphil-> JOIN MaxAmount m ON r.Amount = m.MaxRobbery;
CREATE VIEW
changphil=> CREATE VIEW MaxRobberyAccomplices AS
changphil-> SELECT a.RobberId, a.BankName, a.City, a.Date, a.Share
changphil-> FROM Accomplices a
changphil-> JOIN MaxRobberies r
changphil->   ON a.BankName = r.BankName
changphil->  AND a.City = r.City
changphil->  AND a.Date = r.Date;
CREATE VIEW
changphil=> CREATE VIEW MaxRobberEarnings AS
changphil-> SELECT a.RobberId, r.Nickname, SUM(a.Share) AS TotalEarnings
changphil-> FROM MaxRobberyAccomplices a
changphil-> JOIN Robbers r ON a.RobberId = r.RobberId
changphil-> GROUP BY a.RobberId, r.Nickname
changphil-> ORDER BY TotalEarnings DESC;
CREATE VIEW
changphil=> SELECT * FROM MaxRobberEarnings;
 robberid |    nickname    | totalearnings
----------+----------------+---------------
       16 | King Solomon   |         16501
        3 | Lucky Luchiano |         16500
        4 | Anastazia      |         16500
        8 | Clyde          |         16500
       10 | Bonnie         |         16500
       17 | Bugsy Siegel   |         16500
(6 rows)

changphil=> \q
barretts% nano Question5_Task1.sql
barretts% psql -d changphil -f Question5_Task1.sql
 robberid |    nickname    | totalearnings
----------+----------------+---------------
       16 | King Solomon   |         16501
        3 | Lucky Luchiano |         16500
        4 | Anastazia      |         16500
        8 | Clyde          |         16500
       10 | Bonnie         |         16500
       17 | Bugsy Siegel   |         16500
(6 rows)
```

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



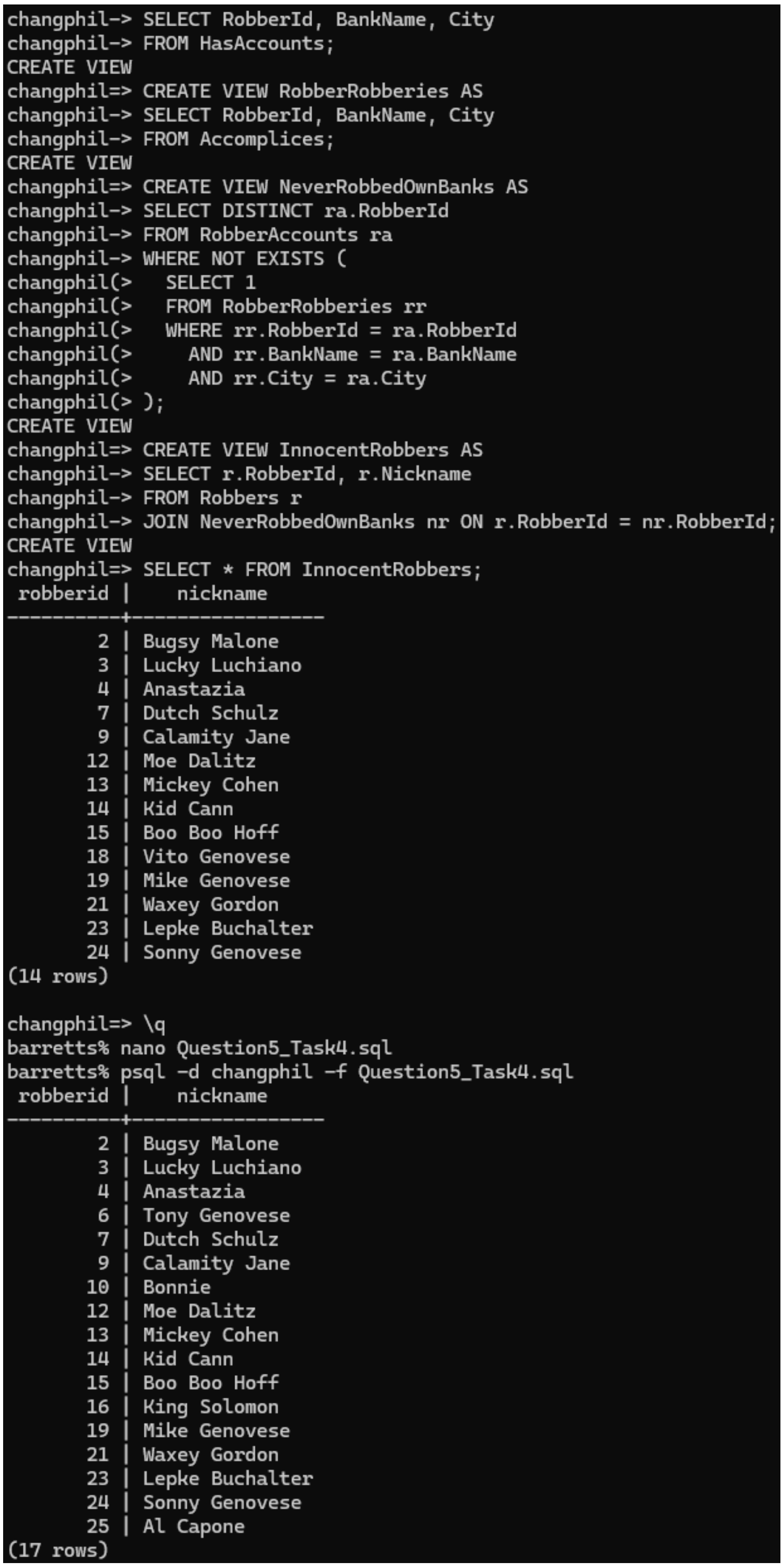
```
changphil=> CREATE VIEW SkilledRobbers AS
changphil-> SELECT RobberId
changphil-> FROM HasSkills
changphil-> GROUP BY RobberId
changphil-> HAVING COUNT(*) >= 2;
CREATE VIEW
changphil=> CREATE VIEW SkilledRobberSkills AS
changphil-> SELECT hs.RobberId, hs.SkillId, hs.Preference
changphil-> FROM HasSkills hs
changphil-> JOIN SkilledRobbers sr ON hs.RobberId = sr.RobberId;
CREATE VIEW
changphil=> CREATE VIEW RobberMinPref AS
changphil-> SELECT RobberId, MIN(Preference) AS MinPref
changphil-> FROM SkilledRobberSkills
changphil-> GROUP BY RobberId;
CREATE VIEW
changphil=> CREATE VIEW RobberFirstPrefSkill AS
changphil-> SELECT r.RobberId, r.Nickname, s.Description
changphil-> FROM RobberMinPref p
changphil-> JOIN HasSkills hs ON p.RobberId = hs.RobberId AND p.MinPref = hs.Preference
changphil-> JOIN Robbers r ON r.RobberId = hs.RobberId
changphil-> JOIN Skills s ON hs.SkillId = s.SkillId;
CREATE VIEW
changphil=> SELECT * FROM RobberFirstPrefSkill;
 robberid |     nickname      | description
----------+-------------------+--------------
        3 | Lucky Luchiano    | Lock-Picking
        5 | Mimmy The Mau Mau | Planning
        7 | Dutch Schulz      | Lock-Picking
        8 | Clyde             | Lock-Picking
       17 | Bugsy Siegel      | Driving
       18 | Vito Genovese     | Scouting
       22 | Greasy Guzik      | Preaching
       23 | Lepke Buchalter   | Driving
       24 | Sonny Genovese    | Explosives
(9 rows)

changphil=> \q
barretts% nano Question5_Task2.sql
barretts% psql -d changphil -f Question5_Task2.sql
 robberid |     nickname      | description
----------+-------------------+--------------
        3 | Lucky Luchiano    | Lock-Picking
        5 | Mimmy The Mau Mau | Planning
        7 | Dutch Schulz      | Lock-Picking
        8 | Clyde             | Lock-Picking
       17 | Bugsy Siegel      | Driving
       18 | Vito Genovese     | Scouting
       22 | Greasy Guzik      | Preaching
       23 | Lepke Buchalter   | Driving
       24 | Sonny Genovese    | Explosives
(9 rows)
```

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



```
changphil=> CREATE VIEW PlannedBankYears AS
ar
FROM Plans;
changphil-> SELECT BankName, City, EXTRACT(YEAR FROM PlannedDate)::INT AS Year
changphil-> FROM Plans;
CREATE VIEW
changphil=> CREATE VIEW RobbedBankYears AS
changphil-> SELECT BankName, City, EXTRACT(YEAR FROM Date)::INT AS Year
changphil-> FROM Robberies;
CREATE VIEW
changphil=> CREATE VIEW PlannedNotRobbed AS
changphil-> SELECT p.BankName, p.City, p.Year
changphil-> FROM PlannedBankYears p
changphil-> WHERE NOT EXISTS (
changphil(>   SELECT 1
changphil(>   FROM RobbedBankYears r
changphil(>   WHERE r.BankName = p.BankName
changphil(>     AND r.City = p.City
changphil(>     AND r.Year = p.Year
changphil(> );
CREATE VIEW
changphil=> CREATE VIEW UnrobbedPlannedBanks AS
changphil-> SELECT DISTINCT BankName, City
changphil-> FROM PlannedNotRobbed;
CREATE VIEW
changphil=> SELECT * FROM UnrobbedPlannedBanks;
    bankname     |   city
-----------------+-----------
 Bad Bank        | Chicago
 Dollar Grabbers | Chicago
 Gun Chase Bank  | Evanston
 Hidden Treasure | Chicago
 Inter-Gang Bank | Evanston
 Loanshark Bank  | Deerfield
 PickPocket Bank | Chicago
 PickPocket Bank | Deerfield
(8 rows)

changphil=> \q
barretts% nano Question5_Task3.sql
barretts% psql -d changphil -f Question5_Task3.sql
    bankname     |   city
-----------------+-----------
 Bad Bank        | Chicago
 Gun Chase Bank  | Evanston
 Inter-Gang Bank | Evanston
 PickPocket Bank | Chicago
 Hidden Treasure | Chicago
 PickPocket Bank | Deerfield
 Loanshark Bank  | Deerfield
 Dollar Grabbers | Chicago
(8 rows)
```

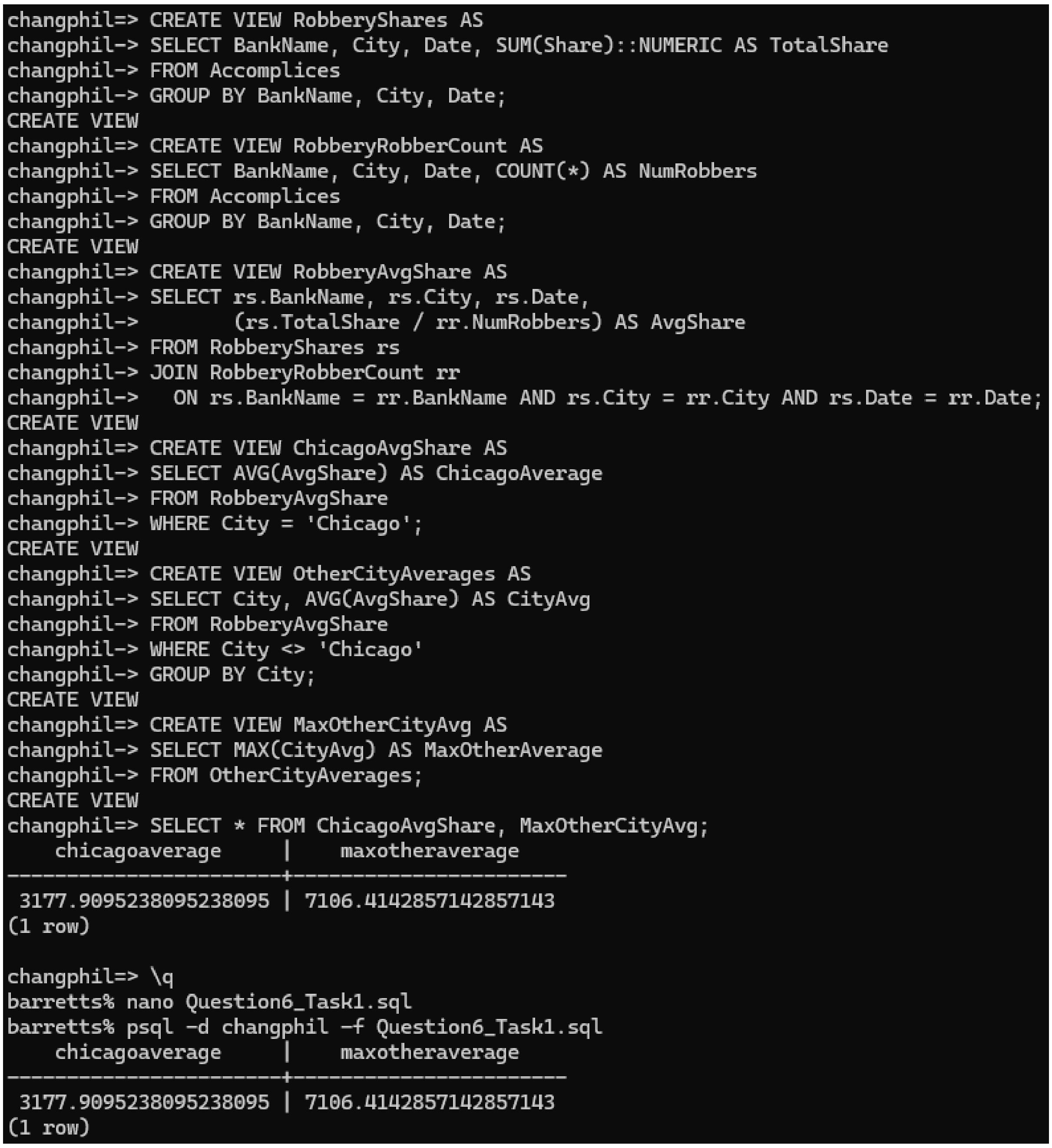
4. Retrieve RobberId and Nickname of all robbers who never robbed the banks at which they have an account.



```
changphil-> SELECT RobberId, BankName, City
changphil-> FROM HasAccounts;
CREATE VIEW
changphil=> CREATE VIEW RobberRobberies AS
changphil-> SELECT RobberId, BankName, City
changphil-> FROM Accomplices;
CREATE VIEW
changphil=> CREATE VIEW NeverRobbedOwnBanks AS
changphil-> SELECT DISTINCT ra.RobberId
changphil-> FROM RobberAccounts ra
changphil-> WHERE NOT EXISTS (
changphil(>   SELECT 1
changphil(>   FROM RobberRobberies rr
changphil(>   WHERE rr.RobberId = ra.RobberId
changphil(>     AND rr.BankName = ra.BankName
changphil(>     AND rr.City = ra.City
changphil(> );
CREATE VIEW
changphil=> CREATE VIEW InnocentRobbers AS
changphil-> SELECT r.RobberId, r.Nickname
changphil-> FROM Robbers r
changphil-> JOIN NeverRobbedOwnBanks nr ON r.RobberId = nr.RobberId;
CREATE VIEW
changphil=> SELECT * FROM InnocentRobbers;
 robberid |    nickname
----------+-----------------
        2 | Bugsy Malone
        3 | Lucky Luchiano
        4 | Anastazia
        7 | Dutch Schulz
        9 | Calamity Jane
       12 | Moe Dalitz
       13 | Mickey Cohen
       14 | Kid Cann
       15 | Boo Boo Hoff
       18 | Vito Genovese
       19 | Mike Genovese
       21 | Waxey Gordon
       23 | Lepke Buchalter
       24 | Sonny Genovese
(14 rows)

changphil=> \q
barretts% nano Question5_Task4.sql
barretts% psql -d changphil -f Question5_Task4.sql
 robberid |    nickname
----------+-----------------
        2 | Bugsy Malone
        3 | Lucky Luchiano
        4 | Anastazia
        6 | Tony Genovese
        7 | Dutch Schulz
        9 | Calamity Jane
       10 | Bonnie
       12 | Moe Dalitz
       13 | Mickey Cohen
       14 | Kid Cann
       15 | Boo Boo Hoff
       16 | King Solomon
       19 | Mike Genovese
       21 | Waxey Gordon
       23 | Lepke Buchalter
       24 | Sonny Genovese
       25 | Al Capone
(17 rows)
```

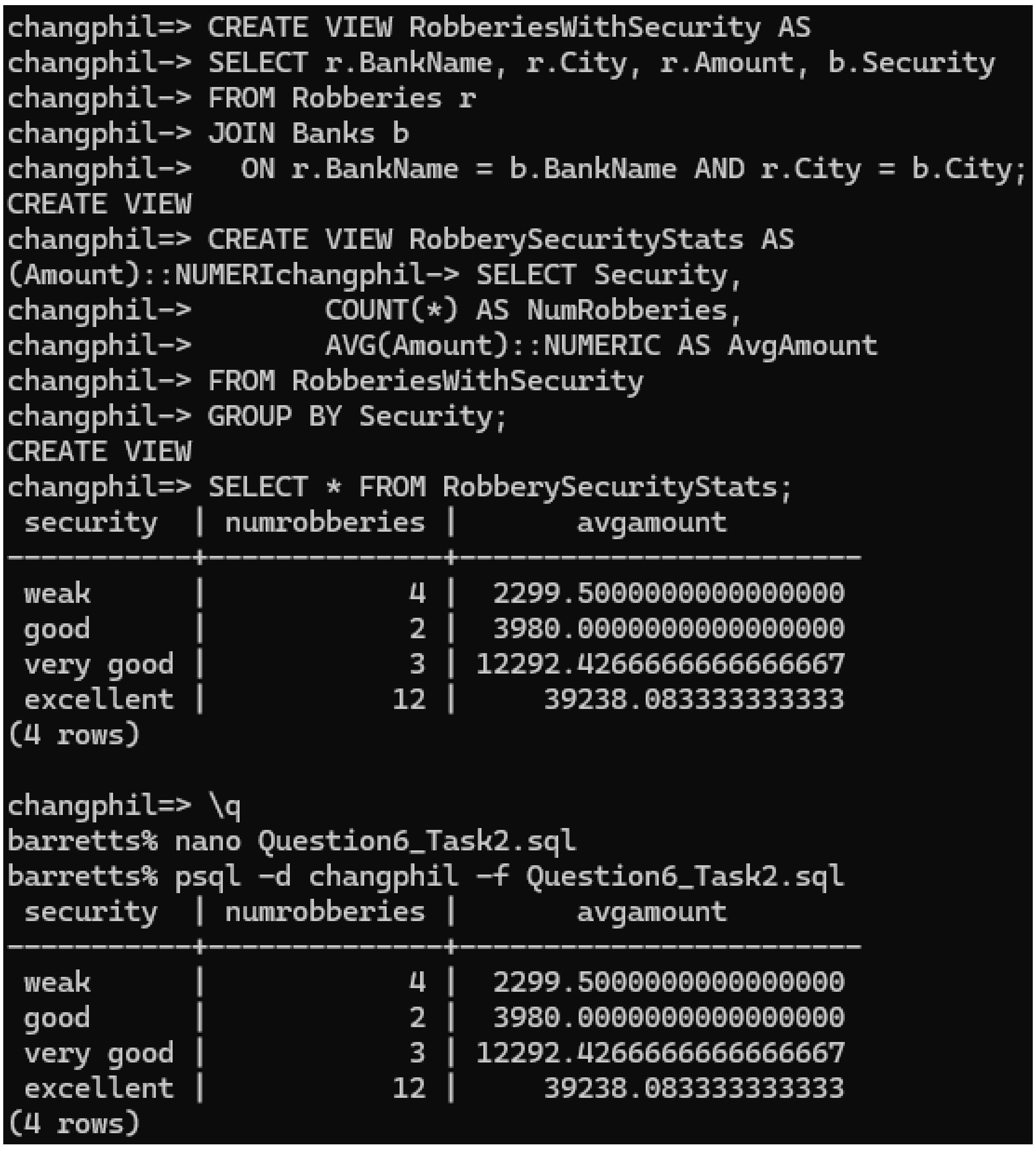
**Question 06: Even More Database Queries [16 marks]**

1. To support the police, you are asked to write a query that finds the average share of all robberies in Chicago, and also the average share of all robberies in the other city (i.e., not Chicago) with the largest average share. Note that the average share of a bank robbery can be determined based on the number of participating robbers.



```
changphil=> CREATE VIEW RobberyShares AS
changphil-> SELECT BankName, City, Date, SUM(Share)::NUMERIC AS TotalShare
changphil-> FROM Accomplices
changphil-> GROUP BY BankName, City, Date;
CREATE VIEW
changphil=> CREATE VIEW RobberyRobberCount AS
changphil-> SELECT BankName, City, Date, COUNT(*) AS NumRobbers
changphil-> FROM Accomplices
changphil-> GROUP BY BankName, City, Date;
CREATE VIEW
changphil=> CREATE VIEW RobberyAvgShare AS
changphil-> SELECT rs.BankName, rs.City, rs.Date,
changphil->        (rs.TotalShare / rr.NumRobbers) AS AvgShare
changphil-> FROM RobberyShares rs
changphil-> JOIN RobberyRobberCount rr
changphil->   ON rs.BankName = rr.BankName AND rs.City = rr.City AND rs.Date = rr.Date;
CREATE VIEW
changphil=> CREATE VIEW ChicagoAvgShare AS
changphil-> SELECT AVG(AvgShare) AS ChicagoAverage
changphil-> FROM RobberyAvgShare
changphil-> WHERE City = 'Chicago';
CREATE VIEW
changphil=> CREATE VIEW OtherCityAverages AS
changphil-> SELECT City, AVG(AvgShare) AS CityAvg
changphil-> FROM RobberyAvgShare
changphil-> WHERE City <> 'Chicago'
changphil-> GROUP BY City;
CREATE VIEW
changphil=> CREATE VIEW MaxOtherCityAvg AS
changphil-> SELECT MAX(CityAvg) AS MaxOtherAverage
changphil-> FROM OtherCityAverages;
CREATE VIEW
changphil=> SELECT * FROM ChicagoAvgShare, MaxOtherCityAvg;
    chicagoaverage     |    maxotheraverage
-----------------------+-----------------------
 3177.9095238095238095 | 7106.4142857142857143
(1 row)

changphil=> \q
barretts% nano Question6_Task1.sql
barretts% psql -d changphil -f Question6_Task1.sql
    chicagoaverage     |    maxotheraverage
-----------------------+-----------------------
 3177.9095238095238095 | 7106.4142857142857143
(1 row)
```

2. To support the police, you are asked to write a query to retrieve 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.



```
changphil=> CREATE VIEW RobberiesWithSecurity AS
changphil-> SELECT r.BankName, r.City, r.Amount, b.Security
changphil-> FROM Robberies r
changphil-> JOIN Banks b
changphil->   ON r.BankName = b.BankName AND r.City = b.City;
CREATE VIEW
changphil=> CREATE VIEW RobberySecurityStats AS
(Amount)::NUMERIchangphil-> SELECT Security,
changphil->        COUNT(*) AS NumRobberies,
changphil->        AVG(Amount)::NUMERIC AS AvgAmount
changphil-> FROM RobberiesWithSecurity
changphil-> GROUP BY Security;
CREATE VIEW
changphil=> SELECT * FROM RobberySecurityStats;
 security  | numrobberies |       avgamount
-----------+--------------+-----------------------
 weak      |            4 |  2299.5000000000000000
 good      |            2 |  3980.0000000000000000
 very good |            3 | 12292.4266666666666667
 excellent |           12 |     39238.083333333333
(4 rows)

changphil=> \q
barretts% nano Question6_Task2.sql
barretts% psql -d changphil -f Question6_Task2.sql
 security  | numrobberies |       avgamount
-----------+--------------+-----------------------
 weak      |            4 |  2299.5000000000000000
 good      |            2 |  3980.0000000000000000
 very good |            3 | 12292.4266666666666667
 excellent |           12 |     39238.083333333333
(4 rows)
```