School of Engineering and Computer Science

SWEN 435 Database System Engineering

Project 1

Due: Monday 22th April, 11:59 pm

This project gives you practice in developing and using relational databases using PostgreSQL. The project is worth 12**%** of your final grade. It will be marked out of 100.

Submission Instructions

Please submit your project via the submission system:

1. **Your answers to all questions in a pdf file.** For each question please include:
   1) Your SQL code **and**
   2) PostgreSQL's responses to your SQL statements and messages.
   Note: marks will be deducted if responses are not provided.

2. Additionally, for the following questions **submit SQL code in '.sql' files:**
   - Question 1,
   - Questions 4, one for each task, and
   - Question 5 and 6, one for each task (submit only for your nested queries, **but not** for the queries of your stepwise approach).

Note: details about what should be included in your submissions can be found at the end of each question. ***Marks will be deducted if .sql files are not submitted.***

The Database Server

For this project, we will use the PostgreSQL Database Management System. A brief tutorial on using PostgreSQL is given at the end of this handout.

For more detailed information on PostgreSQL, please refer to the online PostgreSQL Manual. The link is given on the SWEN304 home page.

The Business Case

The story: You are a database engineer hired by the Chicago police department. Chicago is the capital of Cook County, including several smaller cities like Burbank, Deerfield, and Evanston. For quite some time, the police have been investigating a gang of elusive bank robbers who have been operating in Cook County. The police have collected quite a lot of information about the gang, some from an informant close to one of the gang members. So far, the police department has been keeping the data in a set of spreadsheets, but they realize that they cannot do many of the queries they want in the spreadsheets, and they are also worried that the data entry is introducing errors and inconsistencies that the spreadsheets do not check for. From the spreadsheets, they have produced a collection of tab-separated files of data. They now want you to convert the data into a well-designed relational database and demonstrate some standard queries to them.

The data: You find the data files stored on the Assignment page of the SWEN304 website. They contain information about the gang and the banks in the cities where they have been operating:

`banks_24.data:` lists all the bank branches in Cook county. The banks are specified by the name of the bank and the city where the branch is located in. The data file also includes the number of accounts held in the bank (an indicator of size) and the level of security measures installed by the bank.

`robbers_24.data:` contains the name (actually, the nickname), age, and the number of years spent in prison of each gang member.

`hasaccounts_24.data:` lists the banks at which the various robbers have accounts.

`hasskills_24.data:` specifies the skills of the robbers. Each robber may have several skills, ranked by preference – what activity the robber prefers to be engaged in. The robbers are also graded on each skill. The file contains a line for each skill of each robber listing the robber's nickname, the skill description, the preference rank (where 1 represents first preference), and the grade.

`robberies_24.data:` contains the banks that have been robbed by the gang so far. For each robbery, it lists the bank branch, the date of the robbery, and the amount that was stolen. Note that some banks may have been robbed more than once.

`accomplices_24.data:` lists the robbers that were involved in each robbery and their estimated share of the money.

`plans_24.data:` contains information from the informant about banks that the gang is planning to rob in the future, along with the planned robbery date and the number of gang members that would be needed. Note that the gang may plan to rob banks more than once.

Each of these files could be converted directly to a relation in the database. However, this would not be a good design. There are a few problems that needs to be overcome in designing the database.

The nickname problem: The robbers are currently identified by their nicknames. Although the current list has no duplicates, it is possible to have two robbers with the same nickname. It would be better to give each robber a unique Id, and to use the Id for identifying the robber in all the tables. This way, adding a new robber with a duplicate nickname would not require redesigning the entire database schema.

The skills problem: The list of robber skills uses the descriptions of the skills to uniquely identify them. There should be a finite set of possible skills, and we would like to ensure that skills are not misspelt during data entry. Misspelt skills are a serious concern as queries could miss out tuples due to the misspellings. One approach is to define a constraint on the skill attribute of the *HasSkills* table that checks that every value is one of the possible skills. However, if we then wanted to add a further kind of skill, we would have to change the database schema. A better design can be achieved if we introduce an additional *Skills* table listing all the possible skills and define a constraint on the *HasSkills* table to ensure that every skill there is also in the new *Skills* table.

Further assumptions: The banks are identified by their name and city rather than by an Id. The business rules set by the local banking authority ensure that the combination of name and city is unique so it is not necessary to create an Id for the banks.

You will need to:

- Define the schema of the *RobbersGang* database using PostgreSQL (Question 1);

- Populate the schema using the data files. This will require some transformation of the data files, and probably making some temporary files or tables (Question 2);

- Check that the database enforces the required consistency checks by submitting a series of data manipulations to the database that should all be rejected by PostgreSQL (Question 3);

- Write a set of queries for the database (Questions 4, 5 and 6).

QUESTION 1: Defining the Database [15 marks]

A partial EER diagram is designed for the **RobbersGang** database. Note that keys are not yet defined for the types in the diagram.
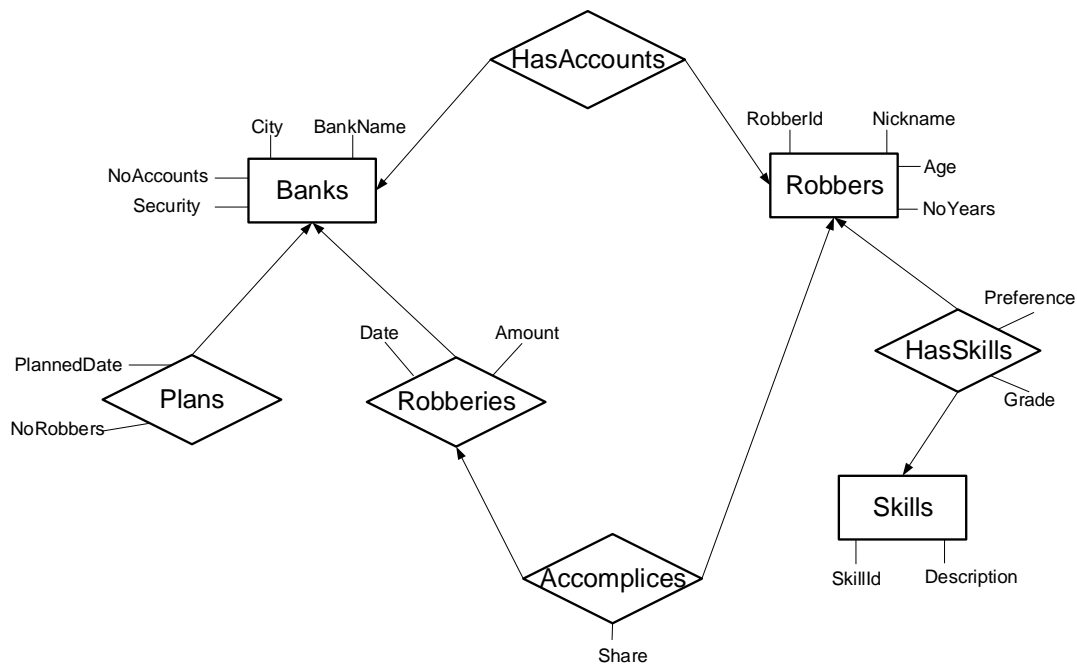
You are expected to use SQL as a data definition language. Define relation schemas by CREATE TABLE statements for each of the following database relations, derived from the EER diagram above:

- **Banks**, which stores information about banks, including the bank name, the city where the bank is located, the number of accounts and the security level of the bank.
    Attributes: *BankName, City, NoAccounts, Security*

- **Robberies**, which stores information about bank robberies that the gang has already performed, including when the robbery took place and how much money was stolen.
    Attributes: *BankName, City, Date, Amount*

- **Plans**, which stores information about the robbery plans of the gang, including the number of gang members needed and when the robbery will take place.
    Attributes: *BankName, City, NoRobbers, PlannedDate*

- **Robbers**, which stores information about gang members. Note that it is not possible to be in prison for more years than you have been alive!
    Attributes: *RobberId, Nickname, Age, NoYears*

- **Skills**, which stores the possible robbery skills.
    Attributes: *SkillId, Description*

- **HasSkills**, which stores information about the skills that specific gang members possess. Each skill of a gang member has a preference rank and a grade.
    Attributes: *RobberId, SkillId, Preference, Grade*

- **HasAccounts**, which stores information about the banks where individual gang members have accounts.
    Attributes: *RobberId, BankName, City*

4

- **Accomplices**, which stores information about which gang members participated in which robbery, and what share of the money they got.

  Attributes: *RobberId, BankName, City, Date, Share*

You are expected to design the database with appropriate choices of:

- *Keys.* Choose appropriate attributes or sets of attributes to be keys and decide on the primary key.

- *Foreign keys.* Determine all foreign keys and decide what should be done if the tuple referred to is deleted or modified.

- *Attribute constraints.* Choose suitable basic data types and additional constraints, such as NOT NULL constraints, CHECK constraints or DEFAULT values.

Your answer to Question 1 should include:

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

2. A list of all your CREATE TABLE statements.

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

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

Remember that there are multiple valid approaches to choosing primary and foreign keys, constraints, and writing SQL statements. It is crucial that you justify your approach to show your understanding – there is no "one right answer".

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

Now that you have your relation schemas defined, you need to insert data into your database. On the SWEN304 website, you will find tab-separated text files for the tables of your database. To begin with, copy these files to a folder (say Pro1) in your private directory. If the data in the text file matches the relation directly (which should be true for some of the files), you can insert the data using the `\copy` command inside the PostgreSQL interpreter:

```
dbname=> \copy Banks FROM ~/Pro1/Banks_24.data
```

or

```
dbname=> \copy Banks(bankname,city,noaccounts,security) FROM
~/Pro1/Banks_24.data
```

The first form assumes that each line of the `Banks_24.data` file contains the right number of attributes for the relation in the same order as they were specified in the CREATE TABLE statement. The second form allows you to specify which attributes are present in the file and in what order. If not all attributes of the relation are specified, the other attributes will be assigned a default value or null.

For other files, you will need to do more work to convert the data. Although you could (for this little database) convert the text files by hand, this would no longer be feasible for large amounts of data. Therefore, we want you to practice using PostgreSQL to do the conversion.

Dealing with the Robber Ids is a little trickier. You are expected to generate these Ids. You can use PostgreSQL to generate Ids with the help of the *Serial* data type. Please consult Section II (8. Data Types) of the online PostgreSQL manual. You will also need to convert the data in `Hasskills_24.data`, `Hasaccounts_24.data`, and `Accomplices_24.data` to use the Robber Ids instead of the nicknames. You will probably need to make temporary relations and do various joins. You may wish to use the INSERT statement in the form:

```
dbname=> INSERT INTO <table_name> (<attribute_list>) SELECT …
```

Moreover, you are expected to construct the *Skills* table based on the data in the `Hasskills_24.data` file. To do this, you will need to load the data into a table, then extract the *Description* column, and put it into the *Skills* table. Note that you should not be able to use the *HasSkills* table to do this because of the foreign key in the *HasSkills* table that depends on the *Skills* table. Rather you will need to construct a temporary relation, copy the `Hasskills_24.data` file into that relation then extract the values from that.

Please note:

1. You need to keep a record of the steps that you went through during the data conversion. This can be just the sequence of PostgreSQL statements you performed.

2. The data in the data files is consistent. We trust (or at least *hope*) that we have removed all errors. In a real situation, there are likely to be errors and inconsistencies in the data, making the data conversion process a lot trickier.

Your answer to Question 2 should include:

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

2. A brief description of the order in which you have implemented the tables of the *RobbersGang* database. Justify your answer.   [3 points]

QUESTION 3: Checking your Database [10 marks]

You are now expected to check that your database design enforces all the mentioned consistency checks. Use SQL as a data manipulation language to perform the tasks listed below.

For each task, record the feedback from PostgreSQL. If your database is created correctly, you should receive error messages from PostgreSQL.

For each task, briefly state which kind of constraint it violates. If no error message is returned, then your database is probably not yet correct. You will get partial marks for correctly identifying what the constraint should be, even if you did not implement it correctly.

Please note: If you give names to your constraints, the error messages are more informative.

1. Insert the following tuple into the *Skills* table:

    a. (21, 'Driving')

2. Insert the following tuples into the *Banks* table:

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

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

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

3. Insert the following tuple into the *Robberies* table:

    a. ('NXP Bank', 'Chicago', '2019-01-08', 1000)

4. Delete the following tuple from the *Skills* table:

    a. (1, 'Driving')

5. Delete the following tuples from the *Banks* table:

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

6. Delete the following tuple from the *Robberies* table:

    a. ('Loanshark Bank', 'Chicago', '', '')

In the following two tasks, we assume that there is a robber with Id 3, but no robber with Id 999.

7. Insert the following tuples into the *Robbers* table:

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

    b. (999, 'Jail Mouse', 25, 35)

8. Insert the following tuples into the *HasSkills* table:

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

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

    c. (999, 1, 1, 'B-')

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

In the following task, we assume Al Capone has robber Id 1. If Al Capone has a different Id in your database, then please change the first entry in the following tuple to your Id of Al Capone.

9. Delete the following tuple from the *Robbers* table:

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

Your answer to Question 3 should include:

Your SQL statements for each task, the feedback from PostgreSQL, and the constraint that has been violated in case of an error message.

QUESTION 4: Simple Database Queries   [24 marks]

You are now expected to use SQL as a query language to retrieve data from the database. Perform the series of tasks listed below.

For each task, record the answer from PostgreSQL.

The tasks:

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

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

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

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

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.   [4 marks]

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.

[4 marks]

Your answer to Question 4 should include:

- Your SQL statement for each task, and the answer from PostgreSQL.
- Also, submit your SQL queries, with each query (just SQL code) as a separate .sql file. Name files in the following way: Question4_TaskX.sql, where X stands for the task number 1, 2, ...

QUESTION 5: Complex Database Queries [20 marks]

You are again expected to use SQL as a query language to retrieve data from the database to perform the tasks listed below. **For each task, you are asked to construct SQL queries in two ways**: using the **stepwise approach**, and as a **single nested** SQL query.

The **stepwise approach** of computing complex queries consists of a sequence of basic (not nested) SQL queries. The results of each query must be put into a virtual or a materialised view (with the CREATE VIEW … AS SELECT … command, or the CREATE TABLE … AS SELECT … command). The output of the last query should be the requested result. The first query in the sequence uses the base relations as input. Each subsequent query in the sequence may use the base relations and/or the intermediate results of the preceding views as input.

1. Retrieve *BankName* and *City* of all banks that were robbed by all robbers. [5 marks]

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

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

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

Your answer to Question 5 should include:

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

- **A single nested SQL query**, with its output from PostgreSQL.

- Also, submit your SQL queries, with each query (just SQL code) as a separate sql file. Name files in the following way: Question5_TaskX.sql, where X stands for the task number 1, 2, …

QUESTION 6: Even More Database Queries   [16 marks]

You are again expected to use SQL as a query language to retrieve data from the database to perform the tasks listed below. **For each task, you are asked to construct SQL queries in two ways**: using the **stepwise approach**, and as a **single nested** SQL query.

The **stepwise approach** of computing complex queries consists of a sequence of basic (not nested) SQL queries. The results of each query must be put into a virtual or a materialised view (with the CREATE VIEW … AS SELECT … command, or the CREATE TABLE … AS SELECT … command). The output of the last query should be the requested result. The first query in the sequence uses the base relations as input. Each subsequent query in the sequence may use the base relations and/or the intermediate results of the preceding views as input.

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.   [8 marks]

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

   Construct a view *containing* the *Security* level, the total *Number* of robberies that occurred in bank branches of that security level, and the average *Amount* of money that was stolen during these robberies.   [8 marks]

Your answer to Question 6 should include:

- **A sequence of SQL statements** for the basic queries and the views/tables you created, and the output of the final query.
- **A single nested SQL query**, with its output from PostgreSQL (hopefully the same).
- Also, submit your SQL nested queries, with each nested query (just SQL code) as a separate sql file. Name files in the following way: Question6_TaskX.sql, where X stands for the task number 1 or 2.

Using PostgreSQL on the school workstations

We have a command line interface to PostgreSQL server, so you need to run it from a Unix prompt in a shell window. To enable the various applications required, first type either

> need postgresql

You may wish to add either "need comp302tools", or the "need postgresql" command to your .zshrc file so that it is run automatically. Add this command after the command need SYSfirst, which has to be the first need command in your .zshrc file.

There are several commands you can type at the Unix prompt:

> createdb ⟨db name⟩

Creates an empty database. The database is stored in the same PostgreSQL server used by all the students in the class. You may freely name your database (e.g. based on your username). But to ensure security, you must issue the following command as soon as you log-in into your database for the first time:

REVOKE CONNECT ON DATABASE <database_name> FROM PUBLIC;

You only need to do this once (unless you get rid of your database to start again). **Note**, your markers may check whether you have issued this command and if they find you didn't, you may be **penalized**.

> psql [**–d** ⟨db name⟩]

Starts an interactive SQL session with PostgreSQL to create, update, and query tables in the database. The db name is optional (unless you have multiple databases)

> dropdb ⟨db name⟩

Gets rid of a database. (In order to start again, you will need to create a database again)

> pg_dump -i ⟨db name⟩ > ⟨file name⟩

Dumps your database into a file in a form consisting of a set of SQL commands that would reconstruct the database if you loaded that file.

> **psql –d** <database_name> **-f** <file_name>

Copies the file <file_name> into your database <database_name>.

Inside an interactive SQL session, you can type SQL commands. You can type the command on multiple lines (note how the prompt changes on a continuation line). End commands with a ';'

There are also many single-line PostgreSQL commands starting with '\' . No ';' is required. The most useful are

\? to list the commands,

\i ⟨file name⟩
loads the commands from a file (eg, a file of your table definitions or the file of data we provide).

\dt to list your tables.

\d ⟨table name⟩ to describe a table.

\q   to quit the interpreter

\copy <table_name> **to** <file_name>
    Copy your table_name data into the file file_name.

\copy <table_name> **from** <file_name>
    Copy data from the file file_name into your table table_name.

Note also that the PostgreSQL interpreter has some line editing facilities, including up and down arrow to repeat previous commands.
For longer commands, it is safer (and faster) to type your commands in an editor, then paste them into the interpreter!

If you want to work from home, you can securely remote into the ECS systems (e.g. using ssh) by following the instructions here: https://ecs.wgtn.ac.nz/Support/TechNoteWorkingFromHome

****