School of Engineering and Computer Science

SWEN 304 Database System Engineering

Project 1

Due: Monday 21 August, 23:59 pm

INTRODUCTION

This project gives you practice at building and using relational databases in PostgreSQL. The project is worth 10% of your final grade. It will be marked out of 100.

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. **Be carefull: The PostgreSQL manual is HUGE!!! Please do not print out the entire manual!**

The Business Case

The story: You are a database engineer employed by the Chicago police department. For quite some time, the police have been investigating a gang of elusive bank robbers who have been operating in Chicago district. They have collected quite a lot of information about the gang, some from an informant close to one of the gang members. So far, they have been keeping the data in a set of spreadsheets, but they realize that they cannot do many of the queries they want in the spreadsheet and they are also worried that the data entry is introducing errors and inconsistencies that the spreadsheet does 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 generate some standard queries.

You will need to:

- Define the schema of the *RobbersGang* database using PostgreSQL.
- Populate the schema using the data files. This will require some transformation of the data files, and probably making some temporary files or tables.
- 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.
- Specify a set of queries for the database.

The data: You find the data files stored on the Assignment page of the SWEN 304 website. They contain information about the gang and the banks in the cities where they have been operating:

`banks_17.data:` lists all the bank branches in the Chicago district. 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 the security measures installed by the bank.

`robbers_17.data`: contains the name (actually, the nickname), age, and number of years spent in prison of each gang member.

`hasaccounts_17.data`: lists the banks at which the various robbers have accounts.

`hasskills_17.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 (a number where 1 represents first preference), and the grade.

`robberies_17.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_17.data`: lists the robbers that were involved in each robbery and their estimated share of the money.

`plans_17.data`: contains information from the informant about banks that the gang is planning to rob in the future, along with the planed robbery date and number of gang members that would be needed. Note that gang may plan to rob some banks more than once.

Each of these files could be converted directly to a relation in the database. However, this would not be a great design.

The nickname problem: The robbers are currently identified by their nicknames. Although the current list has no duplicates, it is quite 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 whole database schema.

The skills problem: The list of robber skills uses the descriptions of the skills. There should be a finite set of possible skills, and we would like to ensure that data entry does not misspell skills. Misspelled skills constitute a severe concern since queries might then miss out some answers because of the misspelling. One approach is to define a constraint on the skill attribute of the *Has_skills* 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 *Has_skills* 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 that it is not necessary to create an Id for the banks.

QUESTION 1: Defining the Database [16 marks]

You are expected to use SQL as a data definition language. Define a database schema by CREATE TABLE statements for each of the following 8 database relations:

- *Banks*, which stores general information about banks, including the number of accounts and the level of security.
  Attributes: *BankName, City, NoAccounts, Security*
- *Robberies*, which stores information about robberies of banks that the gang has already performed, including how much was stolen.
  Attributes: *BankName, City, Date, Amount*
- *Plans*, which stores information about banks that the gang plans to rob.
  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 particular gang members possess. Each skill of a robber 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*
- *Accomplices*, which stores information about which gang members participated in each bank robbery, and what share of the money they got.
  Attributes: *RobberId, BankName, City, RobberyDate, Share*

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

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

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).

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

Now that you have your relation schemas defined, you can insert data into your database. On the SWEN304 web site you find tab-separated text files for the tables. 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 Bank FROM ~/Pro1/Banks_17.data
```

or

```
dbname=> \copy Bank(City,BankName,Accounts) FROM ~/Pro1/Banks_17.data
```

The first form assumes that each line of the `Banks_17.data` file contains the right number of attributes for the relation and in exactly 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 a 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 the use of 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 `Has_skills_17.data`, `Has_accounts_17.data`, and `Accomplices_17.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 `Has_skills_17.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 `Has_skills_17.data` file into that relation then extract the values from that.

Please note:

- 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.
- The data in the data files is consistent. We trust (or at least hope) that we have removed all errors from it. In a real situation, there are likely to be errors and inconsistencies in the data, which would make 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 explanation of what enforced a partial order in your implementation of *RobbersGang* database. [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 series of 8 tasks listed below. For each task, record the feedback from PostgreSQL. If all is normal 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 should at least say what the constraint ought to be, even if you cannot implement it.

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

The tasks:

1. 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')
2. Insert the following tuple into the *Skills* table:
    a. (20, 'Guarding')

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

3. Insert the following tuples into the *Robbers* table:
    a. (1, 'Shotgun', 70, 0)
    b. (333, 'Jail Mouse', 25, 35)
4. Insert the following tuples into the *HasSkills* table:
    a. (333, 1, 1, 'B-')
    b. (3, 20, 3, 'B+')
    c. (1, 7, 1, 'A+')
    d. (1, 2, 0, 'A')
5. Insert the following tuple into the *Robberies* table:
    a. ('NXP Bank', 'Chicago', '2009-01-08', 1000)
6. Delete the following tuples from the *Banks* table:
    a. ('PickPocket Bank', 'Evanston', 2000, 'very good')
    b. ('Outside Bank', 'Chicago', 5000, 'good')

In the following two tasks we assume that Al Capone has the robber Id 1. If Al Capone has a different Id in your database then please change the first entry in the following two tuples to be your Id of Al Capone.

7. Delete the following tuple from the *Robbers* table:
    a. (1, 'Al Capone', 31, 2).
8. Delete the following tuple from the *Skills* table:
    a. (1, 'Driving')

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 8 tasks listed below. For each task, record the answer from PostgreSQL.

The tasks:

1. Retrieve *BankName* and *Security* for all banks in Chicago that have more than 9000 accounts. [3 marks]
2. Retrieve *BankName* of all banks where Calamity Jane has an account. The answer should list every bank at most once. [3 marks]
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. [3 marks]
4. Retrieve *BankName* and *City* of the first bank branch that was ever robbed by the gang. [3 marks]
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. [3 marks]
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. [3 marks]
7. Retrieve *RobberId*, *NickName*, and the *Number of Years* in Prison for all robbers who were in prison for more than three years. [3 marks]
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. [3 marks]

Your answer to Question 4 should include your SQL statement for each task, and the answer from PostgreSQL.

Also, submit your SQL queries electronically. Submit each query (just SQL code) as a separate .sql file. Name files in the following way: Query4_TaskX.sql, where X ranges from 1 to 8.

QUESTION 5: Complex Database Queries [35 marks]

You are again expected to use SQL as a query language to retrieve data from the database. Perform the series of 5 tasks listed below. For each task, you must construct SQL queries in two ways: *stepwise*, 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. [7 marks]
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]
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]
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. [7 marks]
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]

Your answer to Question 5 should include:

- A sequence of SQL statements for the basic queries and the views/tables you created, and the output of the final query.
- A single nested SQL query, with its output from PostgreSQL (hopefully the same).

Also, submit your SQL nested queries electronically. Submit each nested query (just SQL code) as a separate .sql file. Name files in the following way: Query5_TaskX.sql, where X ranges from 1 to 5.

****

Submission Instruction

Submit your answers to all questions printed on paper. Don't forget to include:

- Your SQL code and
- PostgreSQL's responses to your SQL statements and messages.

Additionally, **submit electronically SQL code for:**

- Question 1,
- Question 4, Tasks 1 to 8, and
- Question 5, Tasks 1 to 5 (submit only your nested queries, but not the queries of your stepwise approach).

TUTORIAL: Using PostgreSQL on the 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 comp302tools

or

> need postgresql

You may wish to add either "need comp302tools", or the "need postgresql" command to your .cshrc 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 .cshrc 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 cluster used by all the students in the class. You may freely name your database. 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 use multiple lines for a single SQL command (note how the prompt changes on a continuation line). Each command must be ended 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 before you paste them into the interpreter!