School of Computer Science, McGill University

COMP-421 Database Systems, Winter 2024

Project 3: Writing your Application

Due Date March 20, NOON

This is the last deliverable for your database application project. If you end up adding more data for this project deliverable, that is ok, as long as you do not EXCEED the record limits imposed in project 2.

Your project 3 work should be a continuation of your project 2 work. You are allowed some model adjustments if that is needed to make your application work correctly. All of this work will be still using DB2 as the database.

In this project deliverable you will create a stored procedure, an application program and experiment with advanced features. Please read through the complete description before you start strategizing/working on individual problems.

The Assignment

Some of the solutions to the below questions have to be provided in a document **project3.pdf** whereas others would be independent files of their own (described next to each question).

1. (0 Points) Make sure you are using the DB2 database provided in the winter2024-comp421 server for your project. Not doing this and using another database (including PostgreSQL), using databases installed in your own personal computer, etc., will result in 10 points penalty (even if you had this penalty in the previous deliverable).

2. (0 Points) Include the relational schema that you are using for this phase of the project - even if you have not made any changes from project 1 and 2. Include this in the document **project3.pdf** under the section **Relational Schema**. You need not include any restriction/assumption information along with this. Not doing this will result in 5 point penalty.

3. (15 Points) Write one stored procedure to perform operations on your project database. It should be nontrivial, illustrating a feature or features such as local variables, multiple SQL statements, loops etc. It should also involve a cursor. The stored procedure should use one or more parameters in a significant way. We encourage you to be imaginative. However, here are some sorts of things you might try if you can't think of something more interesting:

   - Compute some aggregate value from a relation and use that value to modify values in that or another relation.
   - Create a new relation and load it with values computed from one or more existing relations.
   - Enforce a constraint by searching your database for violations and fixing them in some way.

   To be turned in:
   Create a section **Stored Procedure** in document **project3.pdf**. Under this section, (a) provide an informal description of what the stored procedure does, (b) provide a listing of the program code of the stored procedure, (c) provide a screenshot of using DB2's command line editor where you execute the stored procedure, (d) show that the stored procedure has the intended effect; for instance, if your stored procedure changes some values in a table extend the screenshot in (c) with SQL statements that query the table before and after the stored procedure was called.

4. (50 Points) Write a user-friendly application program for your database in Java connecting to the DBMS via JDBC. You can base this on the example JDBC program that is given in mycourses as part of the JDBC tutorial. There is no need for a fancy interface. For example, a menu printed via simple console I/O is ok. Your program should consist of a menu with a loop in which:

- A list of at least five alternative tasks is offered to the user. An additional alternative should be quit.
- The loop works as follows
  - The user selects an alternative.
  - The system prompts the user for appropriate input values.
  - The system accesses the database to perform the appropriate queries and/or modifications.
  - Data or an appropriate acknowledgment is returned to the user.
  - The user is returned to the menu after the execution of the task is completed.

Your program should follow the following guidelines.

- Your options should include both queries and modifications.
- Some of your options should contain more than one SQL statement.
- At least one of the tasks should lead to a sub-menu that is created out of a database query.
- Your program must handle errors appropriately. For Java, catch exceptions and print the error messages. Ensure that your program terminates gracefully (after closing any connections) even in the case of a database/SQL exception. If we notice your programs are leaving open hundreds of connections that are piling up at the database end and blocking resources, you might get penalized.

For example, if your project were about skaters and competitions.

1. Look up whether a skater participates in a certain competition by skater name.
2. Unroll a skater S in a competition C. If the rating level is below 3, S cannot enroll in any competition. If it is between 3 and 6, S can enroll in regional competitions only, if its is between 7 and 9, he/she can enroll in regional and national levels, and only with a skating level of 10 can S enroll in all types of competitions. If S is not qualified for the competition C, return a list of alternative competitions for which the S has the minimum rating level and which are close to C in terms of the date.
3. A competition is cancelled: find all skaters participating and replace the participation with a competition close in time to the cancelled competition.
4. Add a new skater.
5. Increase the rating of skaters that were among the first 5 in at least 2 competitions of the highest level they can participate.
6. Quit

With this, the main menu could look like:

```
Skating Main Menu
        1. Look Up Skater in Competition
        2. Enter skater for a competition
        3. Competition Cancellation
        4. Add a new Skater
        5. Quit
Please Enter Your Option:
```

To be turned in:
Hand in all your .java files as separate files. Furthermore, create a section **Application program** in document **project3.pdf**. In this section include the screenshot of the execution of this program. Each of the options should be exercised at least once in your script.

5. (10 Points) In class we discuss indexes that help to speed up queries. You can create and drop an index using: commands:
   `CREATE INDEX <IndexName> ON <RelName>(<Attribute List>);`
   `e.g., CREATE INDEX skatersname ON Skaters(sname);`
   `DROP INDEX <IndexName>;`

   Statements for more sophisticated indexes (unique, clustered etc.) can be found in the lecture notes and also in the DBS manuals.

Create at least two useful indexes for your project database. Do not build indexes on primary keys and unique constraints. Database systems usually create indexes on these attributes automatically as they need them to check the uniqueness property.

To be turned in:
Create a section **Indexing** in document **project3.pdf** with subsections **Index 1** and **Index 2**. For each of the subsections, (a) show a script executing the create index statement in DB2 and the result the database returns, (b) describe which application relevant queries you expect to run faster with this index and why.

6. (15 Points) In the first lecture we mentioned about the role data visualization plays in analyzing data and decision making. Although we will not cover this in the course, nevertheless we will take a small peek into this aspect.
For this purpose, you will produce two charts that visualizes some important aspect of your application from the data. Some examples are:

   (a) Overall sales per day/month,..etc.
   (b) Number of user posts per day,..etc.
   (c) Average money spent by a user per month for the top 10 spending users in your store.

   Make sure that the two charts do not portray similar data. For example, if one of the charts is sales of a particular product per day, the other should not be sales of another product per day or sales of a product per month, etc...

   Choose a chart so that it does not create a "clutter". If you cannot read it, there is no point in visualizing it. For example, if you have sales data for an entire year, and decide to produce a chart for sales of each day, you will end up with a clutter on the x axis, and will be unable to read anything. Instead you could chose to just show sales for the last one week, or sales per month.

   For this work, you can export your data into excel/google spread sheets. You can export the required data from DB2 tables using the db2 command line utility (not from IDE) using its `EXPORT TO` syntax. You may also export the data in any other way that is convenient to you (but the data must be extracted using a SQL query and from the database). The following should work. If it doesn't, look at online information for the proper syntax.

   ```
   EXPORT TO result.csv OF DEL MODIFIED BY NOCHARDEL SELECT * FROM skaters
   ```

   You will find the links to two short tutorials in mycourses (under database information / simple data visualization) on how to create pivot charts in Excel and Google spread sheets.

   To be turned in:
   Create a section **Visualization** in document **project3.pdf** with subsections **Vis 1** and **Vis 2**. For each visualization, turn in the (i) SQL used to generate the data, (ii) a JPG/PNG image of the chart (readable resolution). Furthermore, hand in the Excel / Google spreadsheet you did the work on as a separate file.

7. (10 Points) For creativity points, you may explore any one of these (some are topics not covered in class).

   - Triggers - create a trigger that does something interesting for your application logic - give a brief description of its purpose.
   - A sophisticated GUI (instead of the console based UI for question 2, but does the same operations)
   - An extra stored procedure (it should do something totally different from the one required in this deliverable)

   To be turned in:
   Create a section **Creativity** in document **project3.pdf** with the following content: (a) describe in informal English what you have done, (b) if you create triggers or stored procedure, follow the guidelines as shown under Q3. If you have a GUI explain in this section the tools used and provide some information about your program structure.

8. (0) Points. Indicate in one paragraph how you worked together (how many meetings) and how each of the team members contributed to the project deliverable. If you do not describe this, you will get 10 penalty points.

Some other notes

- **Possibility of demo:** We might offer a limited number of slots for life demos for those who decide to create a GUI. To determine whether this is feasible, please email the latest by March 11 whether your group is intending to implement a GUI outlining what kind of GUI (tool etc.) you will be using. If we decide to go with life demos, then those who implement a GUI and will give a life demo, do not need to provide screenshots in their project3.pdf"
- Some comments on the **application program design**
  - When implementing the "details" of each functionality, you are free to follow a programming approach that intuitively works with the intent of each functionality you want to achieve. However, the application should not probe the user for any information that it can already compute from the data stored in the database or is supposed to generate by itself. For example, the user should not have to enter information that might be automatically retrieved by the system or which the application should have "remembered" when it navigates through the various menu options.
  - You do not need to handle errors related to data formatting (i.e. user entered date where number was expected, etc.) or worry about a user deliberately trying to break your system. You can assume that once a user selects an option they will provide all the necessary information required for the application to act up on that option if the application prompts the user to do so.
  - No errors should result in the application crashing and terminating.
  - You are graded primarily by the functionality and not how "pretty" the application looks. Do not spend time on cosmetics before you get the functionality under control (but it should be "readable").
  - It is assumed that any additional information/data that you need to support the application are already present in the database. They need not be added/maintained through your java application. You can insert them separately (reuse the template scripts from project 2 to make it easy). For the skating example, you can enter for some competitions sensible information with dates, skaters that are already registered, etc.