BI5631 - Database Systems
Assessed Individual Coursework 2

This assignment must be submitted by 2pm
on Monday 5th December 2016.

Feedback will be provided within ten working days
of the submission deadline.

Learning outcomes assessed

This assignment covers SQL, the Relational Model, and Relational Database Design.

The assessed learning outcomes include: implementing a relational database design using an industrial-strength database management system; and using the data definition, data manipulation, and data control language components of SQL.

Submission Instructions

The coursework is to be submitted via Turnitin, accessible via the course's Moodle page. Log onto the course page and you will find the relevant link for this assignment. Click on the link and follow the instructions. Your submission must be delivered as a ZIP file, containing the three files `createdb.sql`, `inserts.sql`, and `queries.sql`.

NOTE: All the work you submit should be solely your own work. Coursework submissions are routinely checked for this.

Coursework 2, contents

Below is given an example design for a slightly modified version of the database from Coursework 1. The design is given as a list of relational schemas. Your task is to write SQL instructions to create a database implementing such a design, or, if you prefer, a variant of it, then populating it with data and creating SQL statements according to a list of queries.

Reference Design

The example design which I will assume to be used contains the following relations. The primary keys are underlined.

- `Artist(ArtistID, Name, Country, AccountID)`, where AccountID is a reference to Account (with partial participation).
- `User(UserID, Name, Country, AccountID)`, where AccountID is a reference to Account (with total participation).
- `Account(AccountID, Email)`, no foreign keys.
- `Follows(UserID, ArtistID)`, where UserID is a reference to User and ArtistID is a reference to Artist.
- `Song(SongID, Name, Duration)`, no foreign keys.
- `PerformsOnSong(SongID, ArtistID)`, where SongID is a reference to Song and ArtistID is a reference to Artist. Note that this relationship can be many-to-many.
- `Album(AlbumID, Title, MainArtist)`, where MainArtist is a reference to Artist(ArtistID) (with partial participation).
- `AlbumTrack(AlbumID, TrackNo, SongID)`, where AlbumID is a reference to Album and SongID a reference to Song. This relation stores the information about which songs occur on which albums and in which order they occur.
- `Playlist(PlaylistID, Title, UserID)`, where UserID is a reference to User and stores the identity of the user who owns the playlist.
- `PlaylistTrack(PlaylistID, TrackNo, SongID)`, where PlaylistID is a reference to Playlist and SongID a reference to Song. Similarly to AlbumTrack, this relation stores the information about which songs occur on which playlists and in which order they occur.

Alternative designs. If you prefer your own solution from assignment 1, you may use that one instead. But only under the following conditions:

- You announce the change clearly. For example, you may add a comment in the `create.sql` file that explains the differences between the design above and the design you use.
- Your design has the same functionality as the one above. (For example, you may not change relationships from many-to-many into many-to-one or one-to-one.)
- Finally, your design must be well designed, in the sense used in this course: it should have no unnecessary repetitions of data (i.e., it should be in 2NF – although you do not need to prove this fast), and it should not explicitly store derived information (e.g., no field for "number of followers" since this information can be derived from the "follows" table).

  It is likely that a reasonable design derived from an ER diagram already satisfies this requirement. This requirement is largely present to prevent artificial solutions designed only to make the SQL queries easier to implement.

Tasks

Your tasks are the following.

1. In order to create the database objects, you will need to define a set of SQL DDL scripts. These must be included as part of your submission in a file named `createdb.sql`. [20 marks]
2. To populate your tables, create a sequence of INSERT statements. These must be included as part of your submission in a file named `inserts.sql`. Every table should have at least 5 rows, since this is the minimum number of rows expected for testing purposes.

   It is strongly recommended that your data is *not too simplistic* – e.g., if a relation is defined as many-to-many in the design, then you should not populate it with data that makes it "accidentally one-to-one", since such data would be bad for testing. [20 marks]
3. Define queries, in SQL, to obtain the following information over the database you created. Note that these should work with any data that can be present in the model, not just for the test data you used. The queries must be included as part of your submission in a file named `queries.sql`.

   (a) The name and email of the user with ID 1. [10 marks]

(b) A list of all users from France, giving for each the user ID and their name, sorted alphabetically by the name. [10 marks]

(c) A list of all albums, giving for each the title and the main artist's name (if there is one), sorted alphabetically by the artist's name. For full marks, the query should include even albums that don't have a main artist. [10 marks]

(d) A list of artists, giving for each the name, the number of followers, and the number of songs they have performed on. [10 marks]

(e) The name of the song that occurs in the largest number of playlists. [10 marks]

(f) A list of artists, giving for each their name and the number of albums on which they have performed on at least two songs. (Note that "performing on a song that occurs on an album" is a different notion from being the main artist of the album.) [10 marks]

Marking criteria

This coursework is assessed and mandatory and is worth 25% of your total final grade for this course.

In order to obtain full marks for each item requested, you must provide the adequate SQL to implement what is asked: creating the objects, populating them with data, and also for the individual queries. The queries should work on "general" data (i.e., they must work on any data that is legal for the database schema, not only on the test data you have used).

The SQL should use standard features (any may use any standard features from the course), but I kindly request that it be tested against PostgreSQL 9.4, which is the version installed on the Computer Science computer system.