DATABASE SYSTEMS

Coursework 2 (rev. 1)

Dr Paolo Guagliardo

Due date: Fri 23 November 2018 at 16:00

Before you begin, please read carefully the policy on Late coursework and Academic misconduct and related links, in particular the rules for the publication of solutions to coursework (which is not allowed unless you have explicitly obtained my permission in writing).

Database schema. The schema we use in this assignment models a simplified university scenario consisting of the tables below.

COUNTRIES have a unique code (ISO alpha-3, a string of exactly 3 characters) and a name (string).

ARTISTS have a unique name (string), (string), a type (string) that is either `PERSON` or `BAND`, and a reference to the code of the country they come from.

ALBUMS have a title (string), a reference to an artist's name (string), a year (positive integer), a type (string) that is either `STUDIO`, `LIVE` or `COMPILATION`, a rating (an integer between 1 and 5), a number of tracks (positive integer). An album is uniquely identified by its title and artist (so there may be albums with the same title but by different artists).

TRACKS have a reference to an album (i.e., a pair of strings consisting of album's title and artist), a number (positive integer), a title (string), and a length expressed in seconds (as a positive integer). A track is uniquely identified by its album's title and artist, and its number.

The detailed SQL schema can be found in the following AFS folder:

`/afs/inf.ed.ac.uk/group/teaching/dbs/2018/cw2/`

as a file called "`schema.sql`". **Important:** in addition to the constraints specified by the schema, you can assume that the number of each track will not be greater than the total number of tracks of the corresponding album reported in the ALBUMS table.

Test instance. A test instance to work on the assignment will be made available during the next few days in the same AFS folder as the schema. But you can start working on your queries using only the information provided by the schema.

Please note that you cannot make assumptions based on the test data: your queries will be executed on different instances (of the same schema) on which such assumptions may not hold. The only constraints that are guaranteed to hold on every instance are those declared in the schema, and the additional assumption indicated above concerning the track's numbers (which is enforced by the data generator, not by the schema, even though it could be coded with a trigger).

Assignment. Write the following queries in SQL.

- The order in which the rows appear in the answer to your queries is irrelevant for this assignment (no ordering is enforced on the answers, so the DBMS will output rows in an arbitrary order). The names of the columns in the answers are also irrelevant. What is **important** is the number of columns, their types, and the order in which they appear in the output.
- All queries can and **must** be written using only the SQL constructs we have seen in class, which now also include outer joins (`LEFT`, `RIGHT`, `FULL`), `COALESCE`, and `WITH`. If in doubt, please check with me. Please observe that none of the queries requires recursion.

(01) **Report on artists within each country**

For each country, output its code, and the following information (in the given order) about its artists: number of bands, number of artists who are not bands, total number of artists, percentage of bands, percentage of artists who are not bands. Countries without artists do no appear in the output. Format percentages as decimal numbers with 0 decimal places.

The output row for a country with code `XYZ` will look as follows:

```
XYZ | 6 | 8 | 14 | 43 | 57
```

(02) **Number of studio, live and compilation albums per country**

Consider only countries for which there are artists from that country who have released some albums. For each such country, output its code, and the total number of studio, live and compilation albums, respectively, released by artists from that country.

The output table will have 4 columns, in the following order: country code, number of studio albums, number of live albums, number of compilation albums.

(03) **Number of studio, live and compilation albums by artist, average per country**

Compute the number of studio, live and compilation albums, respectively, released by each artist; then, for each country, output its code and the average of each of these totals across all the artists from that country. Consider only countries for which there are artists from that country who have released some albums. Format the averages as decimal numbers with 1 decimal place (you can safely assume that no artist will have more than 99 albums).

The output table will have 4 columns, in the following order: country code, average number of studio albums, average number of live albums, average number of compilation albums.

For example, suppose only artists A and B from country `XYZ` have released some albums. If A released a total of 5 studio albums, 6 live albums, and 3 compilations, while B released 4 studio albums, 2 live albums, and 0 compilations, then the output row for `XYZ` will be:

```
XYZ | 4.5 | 4.0 | 1.5
```

(04) **Bands from USA whose debut studio album was rated 5 stars**

A *debut* studio album is one for which there is no other studio album released by the same artist in a previous year. USA bands who released more than one studio album on their debut year are returned if at least one of those albums is rated 5 stars.

The output table will have two columns: the artist's name, followed by their debut year; there will be one and only one row for each artist that satisfies the requirements.

(05) **Artists who have never released two consecutive studio albums more than 4 years apart**

Two albums A and B, of the same type and by the same artist, are *consecutive* if A was released before B, and no album of the same type and by the same artist was released after A and before B. Note that the granularity of albums' releases is in years, thus A and B (of the same type and by the same artist) are trivially consecutive if B was released precisely one year after A.

Output the name of each artist that satisfies the requirements. Artists who have released less than two studio albums (which includes artists who have not released any albums at all) **will** be in the output. The output table will have a single column, consisting of artists' names, and there will be one and only one row for each artist that satisfies the requirements.

(06) **Longest number of years without releasing an album**

For each artist who has released at least 2 albums (of any type), compute the longest number of years between any two *successive* albums (of any type) released by that artist. Two **different** albums A and B (each one of any type, but both by the same artist) are considered *successive* if

- A and B were released in the same year,

or

- B was released after A, **and**
- there is no other album (of any type) by the same artist released after A and before B.

For each artist that satisfies the requirements, output their name and longest number of years during which they did not release any albums. Artists with less than 2 albums **will not** be in the output.

The output table will have two columns: first the artist's name, followed by the number of years. There will be one and only one row for each artist that satisfies the requirements.

(07) **Live albums by UK artists with higher rating than the average in the same year**

More precisely, find all live albums that were released by artists from `GBR` and that have a higher rating than the average rating of all albums (of any type, by any artist) released in the same year. For each album satisfying the requirements, output its title, artist's name, and year (in this order).

(08) **Albums with a lower rating than every subsequent album by the same artist**

This is quite straightforward, but with one twist: the last album, or albums, of an artist (that is, those released in a year after which there are no other albums by the same artist) are included in the output, *unless they are rated 5 stars*. Output the album's title, artist, year, and rating (in this order).

(09) **Total running time of albums without missing tracks**

An album A has *missing tracks* if the Tracks table contains less rows for A than the number of tracks for A reported in the Albums table. For each album without missing tracks, find its total running time in hours, minutes, and seconds. To this end, use SQL's modulo operator `%` that returns the remainder of integer division: e.g., `5 % 3` is `2`.

The output table will have 5 columns, in the following order: album's title, album's artist, hours (int), minutes (int), and seconds (int). There will be one and only one row for each album that satisfies the requirements.

(10) **Stats about tracks and running times of compilation albums**

For each compilation album, output its title, artist, and the following information (in the given order):

- number of tracks, as indicated in the Albums table;
- number of tracks actually present in the Tracks table;
- percentage of missing tracks, formatted as a decimal number with 1 decimal place;

- total running time in seconds (int), calculated as the average length of the album's tracks listed in the TRACKS table, multiplied by the number of tracks reported in the ALBUMS table. Observe that for albums without missing tracks this will be the same as the sum of all track lengths.

The output table will have 6 columns, and precisely one row for each compilation album in the ALBUMS table. Compilation albums without tracks in the TRACKS table will be included in the output: they will have 100% missing tracks and `NULL` running time.

Submission instructions.

- Each query must be written in a text file named `<xx>.sql` where `<xx>` is the two-digit number in the list of queries above. For example, the first query will be written in file `01.sql` and the last one in file `10.sql`. **The character encoding of the file must be ASCII; if you use UNICODE your queries will fail.**
- Each file consists of a single SQL statement, terminated by semicolon (`;`). Submitted files that do not contain *exactly one semicolon* will be discarded when the submission is processed and consequently they will not be assessed (as if they were not submitted). **Please pay attention to this; even if it looks like a trivial detail, it is not.**
- If a file contains more than one statement, or a statement that is not a query (such as `CREATE VIEW` or `UPDATE`), it will fail. Do not use comments either, as they may cause a correct query to fail if they contain semicolons.
- Submission is via the `submit` command on DICE.[1] You can submit all your files at once as follows:

  ```
  submit dbs cw2 01.sql 02.sql 03.sql 04.sql 05.sql 06.sql 07.sql 08.sql 09.sql 10.sql
  ```

  You can also submit your files one by one, or in smaller groups, and in any order. For example:

  ```
  submit dbs cw2 03.sql
  submit dbs cw2 01.sql
  submit dbs cw2 04.sql 02.sql
  ```

Assessment.

- Your queries will be executed on 5 randomly generated database instances that comply with the given schema (including the constraints). Each query is worth 10 marks, allocated as follows:
  - If the query returns a *single wrong row* on *any* of the test instances, 0 marks are given; otherwise
  - 1 mark is given for each test instance on which the query returns *at least 50%* of the correct rows, but not all of them; and
  - 2 marks are given for each test instance on which the query returns *all and only* the correct rows.
- The queries will not be assessed for their performance, even though each query should not take more than a few seconds to run. Your writing style will not be assessed either: you could write a query on a single line and in NonSenSicAL-CAse. Obviously, this is not recommended for your own sake.

[1] See how to submit a practical exercise.