Database Management Systems – Assignment1

This assignment has five (5) problems worth 100 points total. For each problem, write an SQL query against the Chinook Database v1.4. Each query must run successfully using DB Browser for SQLite. A description of the correct result set for each problem is provided - your query must reproduce this result exactly (including attribute names/order and row order/contents).

To help, you have been provided an `SQLiteDiff` utility to compare the output of your query versus a supplied answer in CSV format. **To use this program, create a text file that contains only your SQL query for a particular problem.** Then run the program, supplying first the path to the supplied CSV file to compare against, then the path to your SQL file:

```
$ java -jar SQLiteDiff.jar p1.csv p1.sql          #This is java
$ python SQLiteDiff.py p1.csv p1.sql              #Python for windows
$ python3 SQLiteDiff.py p1.csv p1.sql             #python for Mac and Linux
```

The program will either report success or indicate the row/column where something differs. Do not include comments (`--` or `/* */`) inside your SQL. This may cause errors.

Note that each question warns against using numeric ids (i.e. internal foreign key values). If your solution uses such identifiers, or tries to "game the system" (i.e. write a query that produces the correct output but does not adhere to the spirit/constraints of the question, such as artistId = 39), you will receive no credit.

Each question also indicates a required sorting of the resulting rows. Because a database management system may produce rows in an arbitrary order, it is always good practice to explicitly indicate sorting in your SQL. Thus, if the result set has more than one row and your SQL does not fully specify row-sorting order (according to the problem) you will lose 50% credit, even if the output happens to match the answer.

To receive credit, submit to Blackboard a single ZIP file that contains only the following files: (I will NOT accept any other compression algorithm, e.g., rar, 7z)

- p1.sql: your SQL query for problem 1
- p2.sql: your SQL query for problem 2
- p3.sql: your SQL query for problem 3
- p4.sql: your SQL query for problem 4
- p5.sql: your SQL query for problem 5

Rubric (per question):
-05: minor mistake (but doesn't affect output)
-05: good attempt (but does not reproduce results)
-10: major mistake (either reproduce or didn't reproduce results)
-10: incorrect submission format
-20: no submission, didn't use JOIN, used soft JOIN

You will earn 0 points if it doesn't work with the grader even though your SQL is correctly written.

1. Problem 1 (20 points). Write a query to produce a discography of the band "Led Zeppelin". The result set should have a single column, titled `album_title`, and should be sorted alphabetically. Your query must not hardcode any numeric ids (e.g. `artist_id=39`).

album_title
BBC Sessions [Disc 1] [Live]
BBC Sessions [Disc 2] [Live]
Coda
Houses Of The Holy
IV
In Through The Out Door
Led Zeppelin I
Led Zeppelin II
Led Zeppelin III
Physical Graffiti [Disc 1]
Physical Graffiti [Disc 2]
Presence
The Song Remains The Same (Disc 1)
The Song Remains The Same (Disc 2)

2. Problem 2 (20 points). Write a query to produce a track list for the two-disc album "Live" by the band "The Black Crowes." The result set should have the following columns: `album_title` (title of the album), `track_id` (numeric track identifier), `track_name` (name of the track), `minutes` (full minutes in the song), and `seconds` (number of seconds, rounded). The rows should be sorted first by the album title (such that all of the disc 1 tracks appear before disc 2), and then by the `track_id` (smallest first). The minutes/seconds should be computed from the milliseconds field: the `ROUND`[1] and `CAST`[2] functions will be useful, and you may want to use **Wolfram Alpha**[3] to sanity check your calculations. Your query must not hardcode any numeric ids (e.g. `artistId, albumId`).

album_title	track_id	track_name	minutes	seconds
Live [Disc 1]	2572	Midnight From The Inside Out	4	47
Live [Disc 1]	2573	Sting Me	4	28
Live [Disc 1]	2574	Thick & Thin	3	43
Live [Disc 1]	2575	Greasy Grass River	3	39
Live [Disc 1]	2576	Sometimes Salvation	6	29
Live [Disc 1]	2577	Cursed Diamonds	6	8
Live [Disc 1]	2578	Miracle To Me	6	13
Live [Disc 1]	2579	Wiser Time	7	40
Live [Disc 1]	2580	Girl From A Pawnshop	6	45
Live [Disc 1]	2581	Cosmic Fiend	5	8
Live [Disc 2]	2582	Black Moon Creeping	5	59
Live [Disc 2]	2583	High Head Blues	6	12
Live [Disc 2]	2584	Title Song	8	26
Live [Disc 2]	2585	She Talks To Angels	6	2
Live [Disc 2]	2586	Twice As Hard	4	36
Live [Disc 2]	2587	Lickin'	5	14
Live [Disc 2]	2588	Soul Singing	3	54
Live [Disc 2]	2589	Hard To Handle	3	27
Live [Disc 2]	2590	Remedy	5	37

[1] See https://www.sqlite.org/lang_corefunc.html

[2] See https://www.sqlite.org/lang_expr.html#castexpr

[3] Example: http://www.wolframalpha.com/input/?i=286981+milliseconds+to+seconds

3. Problem 3 (20 points). There is one invoice that totals more than $25. For that order, write a query to produce all information necessary for an invoice report: for each purchased track, include the `invoice_line_id` (invoice line), `track_id` (track identifier), `album_title` (title of the corresponding album), `artist_name` (name of the corresponding artist), `track_name` (name of the track), `media_type` (type of media), `unit_price` (unit price of the track), and `qty` (number purchased of the track). To format the price you may find it useful to use either string concatenation (see lecture) or the **`PRINTF`**[4] function. The information result should be sorted by the `album_title` (alphabetically), then the `artist_name` (alphabetically), then the `track_name` (alphabetically). Your query must not hardcode any numeric ids (e.g. `invoice_id`, `album_id`).

invoice_line_id	track_id	album_title	artist_name	track_name	media_type	unit_price	qty
2201	2931	Achtung Baby	U2	So Cruel	MPEG audio file	$0.99	1
2189	2823	Battlestar Galactica, Season 3	Battlestar Galactica	Collaborators	Protected MPEG-4 video file	$1.99	1
2190	2832	Battlestar Galactica, Season 3	Battlestar Galactica	The Woman King	Protected MPEG-4 video file	$1.99	1
2191	2841	Heroes, Season 1	Heroes	One Giant Leap	Protected MPEG-4 video file	$1.99	1
2192	2850	Heroes, Season 1	Heroes	The Fix	Protected MPEG-4 video file	$1.99	1
2199	2913	Lost, Season 1	Lost	Deus Ex Machina	Protected MPEG-4 video file	$1.99	1
2198	2904	Lost, Season 1	Lost	Outlaws	Protected MPEG-4 video file	$1.99	1
2195	2877	Lost, Season 1	Lost	The Moth	Protected MPEG-4 video file	$1.99	1
2194	2868	Lost, Season 1	Lost	Walkabout	Protected MPEG-4 video file	$1.99	1
2200	2922	Lost, Season 2	Lost	Live Together, Die Alone, Pt. 1	Protected MPEG-4 video file	$1.99	1
2193	2859	Lost, Season 2	Lost	Man of Science, Man of Faith (Premiere)	Protected MPEG-4 video file	$1.99	1
2197	2895	Lost, Season 3	Lost	Par Avion	Protected MPEG-4 video file	$1.99	1
2196	2886	Lost, Season 3	Lost	Stranger In a Strange Land	Protected MPEG-4 video file	$1.99	1
2188	2814	Volume Dois	Titãs	Insensível	MPEG audio file	$0.99	1

4. Problem 4 (20 points). There are 26 tracks in the “Heavy Metal Classic" playlist - write a query to find the set of 9 artists that are represented by these tracks. The result set should have a single column (`artist_name`) and should be sorted alphabetically. Your query must not hardcode any numeric ids (e.g. `playlistId`).

artist_name
AC/DC
Accept
Black Sabbath
Iron Maiden
Metallica
Motörhead
Mötley Crüe
Ozzy Osbourne
Scorpions

5. Problem 5 (20 points). There is a single album that has any “Bossa Nova” tracks - write a query to find it (Bossa Nova is genre). The result set should have a single column (`album_title`). Your query must not hardcode any numeric ids (e.g. `genreId`, `albumId`). You do not need to indicate any sorting for this query.

album_title
Vinícius De Moraes - Sem Limite

[4] See https://www.sqlite.org/lang_corefunc.html