Northeastern University
CS5200 – DBMS
Spring 2025, Derbinsky

HW2

This assignment has two parts. The first focuses on SQL Programming, whereas the second on using SQL to query the Chinook database. To receive credit, submit to Canvas **only** the following files:

- `ChinookApp.java`: the single file in which you added code for part 1
- `p21.sql`: a text file containing **only** your SQL query for part 2, problem 1
- `p22.sql`: a text file containing **only** your SQL query for part 2, problem 2
- `p23.sql`: a text file containing **only** your SQL query for part 2, problem 3
- `p24.sql`: a text file containing **only** your SQL query for part 2, problem 4
- `p25.sql`: a text file containing **only** your SQL query for part 2, problem 5

Each deviation from these instructions will incur a 1 point penalty.

1 SQL Programming

1.1 Preliminaries

You will require some Java development tools for this part of the homework. There is an attempt to be platform agnostic, but you may have to do some searching to solve individual problems that arise.

1.1.1 Java

The Java Standard Edition (SE) Development Kit (or JDK for short) provides the ability to run programs written in Java (including applets in a browser), as well as compile new programs. By contrast, the Java Runtime Environment (JRE) can run programs, but not to make new ones.

The code for this part of the homework requires at least version 8 of the Oracle JDK (you will see JDK8 as well as 1.8, which mean the same thing). If you can run `java` and `javac` at the command prompt/terminal, you are probably fine. Otherwise, the default instructions for Eclipse (below) should also install this component for you.

Java is widely used, and thus it is common for vulnerabilities to be discovered/exploited in the JDK. *You should keep your version up-to-date in order to limit your exposure.*

1.1.2 Eclipse

The provided Java starter code is an Eclipse project. It is recommended, though not required, to use Eclipse to import, build, test, and run the application.

There are several good integrated-development environments (IDEs) for Java (e.g. NetBeans, IntelliJ). Eclipse has a highly extensible plug-in system, supports many languages/platforms (e.g. Android), and is widely used in industry.

If you choose to use Eclipse, and don't already have it, first navigate to the Eclipse Downloads page[1]. By default, consider using the "Eclipse Installer" for your platform – download and run this and install the "Eclipse IDE for Java Developers". You should now be able to run Eclipse (e.g., via Spotlight on Mac, the Desktop/Start Menu on Windows).

1.2 Your Task

You are going to write a Java command-line application that supports five (5) queries on the Chinook database (via SQLite). To get a feel for the system, the following shows the "usage" statement that you would get if you run the completed program from the command line (with no arguments):

```
Usage: java databases.ChinookApp <path to Chinook database> <query #> [parameter value]

1) Count customers in country [parameter value]
2) List all employees (sorted by employee id)
3) Count customers supported by employee id [parameter value]
4) List all customers (sorted by customer id)
5) List all invoices (sorted by invoice id, each line by invoice line id) for customer id [parameter value]
```

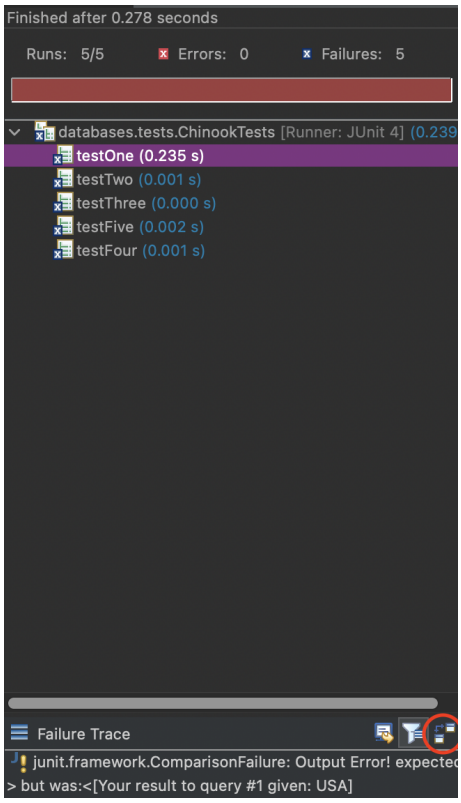
So the first argument will be the path to the SQLite database, the second will be the query number (1 – 5, corresponding to the list above), and the third is a parameter value (only applicable to queries 1, 3, and 5).

[1] `https://www.eclipse.org/downloads/packages/`

You have been provided starter code – unzip and import into Eclipse[2]. There are two important files in this project: `ChinookApp.java`, in which you will write your code and then **submit**, and `ChinookTests.java`, which has automated tests to help you make progress.

1.2.1 Testing with JUnit

To use the automated tests, right-click on `ChinookTests.java`, hover over "Run As", and choose "JUnit Test". A tab will pop-up that runs five tests (one per query) and shows your progress. To begin, you will fail all five tests; as you make progress, you will eventually pass all five. To see the code for a failed test, double click the test name – you will see something about the input and expected output in the editor[3]. To see an input-output comparison, click the icon circled in red in the figure below.



In the resulting "diff" window, pay close attention to spelling, whitespace, and missing/added lines (right-click anywhere and choose to show these items as necessary).

[2]File, Import, "Existing Projects into Workspace" under "General"; "Browse" button, find the folder extracted, "Finish" button.

[3]If you are seeing strange characters, right-click the file from the Package Explorer, click Properties, and choose `UTF-8` under Text file encoding.

1.2.2 Example Code

The following provides a good starting point for writing safe SQL code in Java:

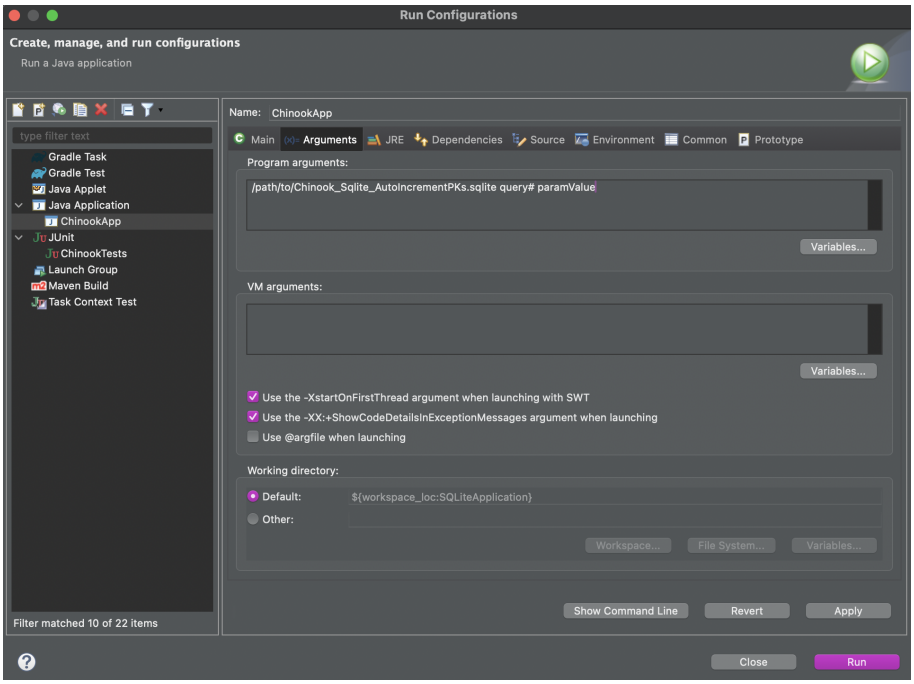
```
final String sql = "SELECT Name AS tName FROM Track t WHERE t.AlbumId=? ORDER BY tName";
try (final PreparedStatement stmt = connection.prepareStatement(sql)) {
    stmt.setInt(1, 10);

    try (final ResultSet res = stmt.executeQuery()) {
        while (res.next()) {
            System.out.printf("%s%n", res.getString("tName"));
        }
    }
}
```

To try this for yourself, type the code within `try` block of `ChinookApp` that follows the "makes a connection to the database" comment. The code creates a prepared statement with a parameterized SQL string. It then fills that in with a value (in this case a constant, 10, but represents how you might use the query parameter supplied by the user). The query is then executed and, while there are results, prints every track name in the album.

1.3 Arguments with Eclipse

To run your own code with command-line arguments, right-click `ChinookApp.java`, hover over "Run As", and choose "Java Application". You should then see the usage statement print to the console – this is expected, as you didn't supply any arguments.

To do so, click the "Run" menu, then "Run Configurations". Find `ChinookApp` in the list, and choose the "Arguments" tab. In the "Program Arguments" box, supply all command-line arguments in order (see usage above). Then click the "Run" button – change this as often as you wish.



1.4 Submission

The starter code has a set of `TODO` comments to guide you in completing all parts.

In order to implement the code, you may want to first start in DB Browser to get the basic SQL working. Then, transfer this to Java (adding parameterization as appropriate).

When you pass all of the JUnit tests, be sure to clean up and comment your code, including placing your name in the placeholder at the top of the file. Importantly, all of your code needs to be within the `ChinookApp.java` file – you are allowed to add methods if you wish (certainly not required), but can't change any of the provided methods (aside from where noted in the comments).

The JUnit tests will be used for grading, but other factors will be evaluated:

- Comments (including your name)
- Use of parameterized [i.e., safe] SQL that adheres to the query descriptions (including sorting)
- Results of additional tests (i.e., make sure your SQL addresses the prompts, fully sorts results, and that you haven't modified the Chinook database)
- Closing resources (the try-with-resource syntax takes care of this)

These other factors aside, each of the five queries will be weighted equally. This part of the assignment is worth 50 points total.

2 Querying Chinook

This part of the assignment has five (5) problems worth 25 points total. For each problem, write an SQL query against the Chinook Database v1.4.5. Each query **must** run successfully using DB Browser. 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 p21.csv p21.sql
```

The program will either report success, or describe an issue. **If your query does not produce the correct output via this utility, you can earn at most 2 points for each problem.**

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), 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 1 point, even if the output *happens* to match the answer**.

While an RDBMS will try to optimize your query, avoid queries that are wildly large/inefficient, such as those that unjustifiably include unneccessary joins or correlated subqueries

Problem 1 (5 points). Write a query to produce the list of artists that have more than 80 tracks. For each artist, include the name and number of tracks. You should sort the results from most tracks to least, breaking any ties by arist name (alphabetically). Your query must not hardcode any numeric ids (e.g., `ArtistId`, `TrackId`), nor identifying information about the artists other than the criteria listed above.

artistName	trackCount
Iron Maiden	213
U2	135
Led Zeppelin	114
Metallica	112
Deep Purple	92
Lost	92

Problem 2 (5 points). There are "lazy" artists in Chinook – artists that are not associated with any albums. Write a query to produce a list of these artists, sorted by artist name (alphabetically). Note: the figure below provides the first 10 rows of the result set; there are 71 in total. Your query must not hardcode any numeric ids (e.g., `ArtistId`, `AlbumId`), nor identifying information about the artists other than the criteria listed above.

lazyArtist
A Cor Do Som
Academy of St. Martin in the Fields, Sir Neville Marriner & William Bennett
Aerosmith & Sierra Leone's Refugee Allstars
Avril Lavigne
Azymuth
Baby Consuelo
Banda Black Rio
Barão Vermelho
Bebel Gilberto
Ben Harper

Problem 3 (5 points). Write a query that reports on the number of invoices and total revenue per customer state (only including those states that have produced billing revenue greater than $60). The rows should be sorted by revenue (largest first). Your query must not hardcode any numeric ids (e.g., `CustomerId`, `InvoiceId`), nor identify customer states/countries by criteria other than that listed above. To format revenue, you may find it useful to use the `PRINTF`[4] function.

customerState	customerCountry	numInvoices	revenue
CA	USA	21	$115.86
SP	Brazil	21	$114.86
ON	Canada	14	$75.24

[4] `https://www.sqlite.org/lang_corefunc.html` (and here's a pretty good primer if you've never used `printf` in another language: `https://www.codingunit.com/printf-format-specifiers-format-conversions-and-formatted-output`)

Problem 4 (5 points). Write a query to identify the tracks that have been sold twice, both billed in the same country. For each such track, include the track's name, album title, artist name, genre, and country to which both invoices were billed. You should sort the results by genre, then artist name, then track name (all alphabetically). Your query must not hardcode any numeric ids (e.g. `InvoiceId`, `TrackId`), nor identifying information about the tracks other than the criteria listed above.

trackName	albumTitle	artistName	genreName	billingCountry
Train In Vain	The Singles	The Clash	Alternative & Punk	USA
Meditação	Prenda Minha	Caetano Veloso	Latin	USA
Música No Ar	Serie Sem Limite (Disc 1)	Tim Maia	Latin	USA
Turbo Lover	Living After Midnight	Judas Priest	Metal	Canada
Rehab	Back to Black	Amy Winehouse	R&B/Soul	USA
Plaster Caster	Unplugged [Live]	Kiss	Rock	Canada
Speed Of Light	Beyond Good And Evil	The Cult	Rock	USA
Sun King	Pure Cult: The Best Of The Cult (For Rockers, Ravers, Lovers & Sinners) [UK]	The Cult	Rock	USA
Peace On Earth	All That You Can't Leave Behind	U2	Rock	USA
Gay Witch Hunt	The Office, Season 3	The Office	TV Shows	USA

Problem 5 (5 points). Write a query to generate a ranked list of employees based upon the amount of money brought in via supported customer invoices. The result should include, for all employees (even those that did not support any customers), id, first name, last name, title, number of supported invoices, and sum total of the cost of all supported invoices. The rows should be sorted by the average total-per-invoice (greatest first), then by last name (alphabetically), then first name (alphabetically). Your query must not hardcode any numeric ids (e.g. `EmployeeId`, `InvoiceId`). The invoice total should be preceded by a dollar sign ($) and have two digits after the decimal point (rounded, as appropriate); in the case of employees without any invoices, you should output a `$0.00`, not `NULL`. You may find it useful to look at the `IFNULL` and `PRINTF` functions of SQLite.

empId	empFirst	empLast	empTitle	numInvoices	invoiceTotal
5	Steve	Johnson	Sales Support Agent	126	$720.16
3	Jane	Peacock	Sales Support Agent	146	$833.04
4	Margaret	Park	Sales Support Agent	140	$775.40
1	Andrew	Adams	General Manager	0	$0.00
8	Laura	Callahan	IT Staff	0	$0.00
2	Nancy	Edwards	Sales Manager	0	$0.00
7	Robert	King	IT Staff	0	$0.00
6	Michael	Mitchell	IT Manager	0	$0.00