## Introduction:

In this practical, we were required to produce a database and a simple GUI for the following scenario: We had to create an Online streaming service for audio books where we could purchase books, keep track of the customers, contributors and reviews made. We were provided by a relational schema which was very useful to complete this task.

# Practical:

## Compilation, Execution & Usage Instruction:

To implement a user friendly interface, We used HTML, CSS, Javascript and Node.js/express. We chose to use this framework in reason of a prior knowledge of Node.js and building responsive Web-apps with these tools. Another reason for choosing a web interface instead of a java app would also be the cons of having an almost automatic connection to the database when using Node.js related frameworks such as mysql and express.

To compile and run the interface, we have to run the command line “node app.js” while being in the project directory. We can now access the Web-app on the localhost running on port 3303. Further usage instructions will be discussed in the GUI implementation part of the report.

## Overview:

To summarise the functionality of my implementation, all basic features presented in the specification sheet are well done and well tested. We also implemented some extra features that can be considered as extensions.

With respect to the requirement listed in the practical specification sheet, We could confidently say that we are at a respectable level of completeness.

## Database Implementation:

First, We had to create a MariaDB database and tables that would translate the relational schema provided. We then had to populate the database with the sample data given. When creating our database, We had four priorities and we had a methodical way of processing the information given to us. Initially, we focused of listing all values and setting the appropriate SQL datatype. This was straight forward as all was given in the relational schema. Then, we defined all the integrity constraints, foreign keys and primary keys. At this point, we are still methodically following the relational schema. However, we now had to deal with uniqueness and nullability, This task was done by following the ideas given in the database scenario and logic. At this point, We are able to create our tables!

To populate our database, We had to clean some values and format a csv file for each of our tables. We then used the “LOAD DATA LOCAL INFILE” to population the database. All the csv file will be in the submission folder. We chose the populate the database that way because it is faster and it provides some kind of error checking and version control on the values that constitute our database.

Creating the queries was a challenge that is about understanding table arithmetic and JOINS. First, Let’s walk through our taught process when creating the first query. We will then generalize to explain the other two. After carefully reading what we were told to create, we start by selecting all the values from the data that is most important in the query. Here, we know that we are going to deal with customers, thus we call the following:
“SELECT * FROM customer”. We now know exactly the data that we are dealing with. We keep calling select all but now using JOINS to get the variety of data that we want. Finally, when the table resulting from the query is composed of all the values that we want and more, we get rid of the junk and select the important values. However, this is not the final step! We have to make sure that

everything is in the right order, get rid of the duplicates if we have to and run other small detailed commands to make our query perfect such as :
“GROUP BY” ; “CONCAT” ; “ORDER BY” etc…
These commands allow us to reorganise, process or sort our data to have the requested query.
Understanding the logic behind query creation, we can now briefly explain the practical queries.
For Q1, the detail lied in using a count and a sum to process the purchases of the customers.
Regarding Q2, the main idea was to use “WHERE ISBN NOT IN” to get the right values.
Q3 was about using “CONCAT” and “group_concat” with a clever combination of AND’s and OR’s in the query conditions. But in summary, all these queries were challenging our use on “LEFT JOINS” and table arithmetic to get the correct values. When out queries were well tested, we were then able to create views q1,q2 and q3 that made everything faster and easier.

We then had to write triggers and procedures to enforce some constraints. To complete this task efficiently, I split the first trigger in 2 triggers that I could test individually. The trick in the second trigger was to calculate the age, we did so using the following:
“SELECT TIMESTAMPDIFF(YEAR,date_of_birth,CURDATE())”
Finally, the last part was writing a procedure with the appropriate values to create either a customer or contributor (or both!) and to prevent direct entry to the person relation. We used 3 procedures for the 3 possible creations and a trigger to prevent access.
A text file Is provided in the submission, It contains all the queries and logs that I went through to result in a nice Database implementation.

## GUI Implementation:

Our GUI implementation revolved around using Node.js and javascript to send an HTML file that would contain all the info that we wanted. To understand what is happening here, we are going to take each file 1 by 1 and explain its purpose and role in the implementation.

First, we have “page.html”, as the name suggests, this is our main html page that is sent and displayed on the navigator. We are using vertical-menu to list the data as I though that it looked clean and better than a table or simple text.
When clicking on “LISTING OF AUDIOBOOKS”, we call an on click function that displays the audiobook titles and authors.
When clicking on “LISTING WITH DETAIL”, we get the same kind of list but with a lot more detail. Plus, when we are in detail mode, if we click on a title, The audiobook’s reviews are displayed.
Finally, we have two  tags with specific id’s that we use to write to the page using inner HTML.
This file has entirely been written by myself with no sources on inspirations takes from outside.

Now app.js, this is where we connect to our SQL database and we set all our constants. This is the core of our Node.js application. Here, we use express and it’s body parser to get the data from the database and be able to display it. Then after the first basic get methods, We have our 2 main getters.
The first one takes care of displaying audiobook’s details. We use a query that is given in the query log to get from the database a JSON file with all the data that we need.
The second one revolves around the review display part. Here, we did a nice job with isbn to set it as a variable on click.
This file has entirely been written by myself with no sources on inspirations takes from outside.

The processing methods take place in the “page.js” file though. We have 3 main methods:

- loadData() is our on-click method that displays the detailed data received through the query called in app.js. The main Idea here is to parse the JSON response Text then loop through the data using forEach(). While looping, we write to the HTML file the data that we choose to show and display.

- LoadSmall() uses the exact same logic and doesn't need further explanation.
- This part was actually the most tricky part on the GUI implementation, The way we wanted to Display the reviews was by using the ISBN of the book the user clicks on. Thus, in combination with the getter in app.js, we are able the process ISBN and use it in our query.

This file has entirely been written by myself with no sources on inspirations takes from outside.
Finally, we have the css stylesheet which doesn't need to be explained.
We stayed on the simple side when it came to design.

## Testing:

First, let's test the database implementation. We start by testing the three views that we created.

```
MariaDB [mmd4_P2db]> SELECT * FROM q1;
+-------------+-----------------------+----------------+--------------+-----------------+
| customer_ID | email_address         | full_name      | number_books | total_spending |
+-------------+-----------------------+----------------+--------------+-----------------+
|           7 | jk@rowling.com        | Rowling JK     |            3 |          61.19 |
|           4 | hugh@laurie.com       | Laurie Hugh    |            1 |          38.00 |
|           5 | ruth@letham.com       | Letham Ruth    |            1 |          38.00 |
|           3 | sfry@email.com        | Fry Stephen    |            2 |          29.19 |
|           9 | pippa.smith@email.com | Smith Pippa A  |            1 |          16.00 |
|           2 | bob_snr@bobson.com    | Bobson Bob A B |            0 |           0.00 |
|           1 | bob_jnr@bobson.com    | Bobson Bob B A |            0 |           0.00 |
|          10 | jon@spellbad.com      | Spellbad Jon Q |            0 |           0.00 |
|          65 | oirje@gmail.gt        | pass test      |            0 |           0.00 |
+-------------+-----------------------+----------------+--------------+-----------------+
9 rows in set (0.00 sec)
```

The goal here was to list all the customers, their email address, their full name, the amount of books they bought and how much they spent. The twist here was that we had to order from highest to lowest spend. If two people had spent the same amount of money, they should appear in alphabetical order by surname, forename and middle initials. We can see here that all the conditions above are respected. Q1 success!

Q2 was also a success at listing all the audio books that have not been purchased by anyone.

```
MariaDB [mmd4_P2db]> SELECT * FROM q3;
+---------------+-------------+-------------------------------------------------------------+
| contributor_ID | full_name    | book                                                        |
+---------------+-------------+-------------------------------------------------------------+
|              3 | Fry Stephen  | Harry Potter and the Philosopher's Stone,Moab Is My Washpot |
|              4 | Laurie Hugh  | Gulliver's Travels                                          |
|              7 | Rowling JK   | Harry Potter and the Philosopher's Stone                    |
+---------------+-------------+-------------------------------------------------------------+
3 rows in set (0.01 sec)
```

Finally Q3, we had to list all contributors that bought audiobooks that they either authored or narrated. The twist here was the following :
"Contributor id, full name, comma separated list of the names of the books they have both contributed to and bought."
We can see that the coma separation is respected and that the order is well respected.
Q3 success!

Now we have to test the triggers.

For the first one, we have to test 2 things. Giving a wrong rating input should result in an error and when someone reviews a books that he/she has bought, verified should automatically be set to 1.

Let's try to insert a rating of 7 stars:

```
MariaDB [mmd4_P2db]> INSERT INTO audiobook_reviews VALUES(3,'978-1408855652',7,'hey','testest',0);
ERROR 1644 (45000): Rating is not valid
```

The trigger Passes both tests smoothly. T1 success!
The second trigger should not let a customer that is younger that the age restriction buy the audio book. Let's insert !
We can see that little Bob with ID 1 is only 8 years old ! We can use him for our test. We shall try to make him buy The gun sell which is only for people over 16.

```
MariaDB [mmd4_P2db]> INSERT INTO audiobook_purchases VALUES (1,'978-1611749731','2018-10-23 21:29:48');
ERROR 1644 (45000): TOO YOUNG!
```

He is too young, the trigger is a success!

Finally, let's test the final trigger/procedure. The procedure creation works perfectly after being well tested , but let's make sure we can not access person directly:

```
MariaDB [mmd4_P2db]> INSERT INTO person VALUES (24,'asd','','ff','2009-12-31');
ERROR 1644 (45000): NOT ALLOWED TO ACCESS PERSON, USE PROCEDURE!
```

T3 has the expected behaviour.

Concerning the GUI testing, to avoid crowding the report with screen-shots, We are going to test if the displayed data is the same as the one in the database.

The website outputs the correct 5 different books with all valid details. It switches easily from detail mode to simple mode.

The reviews are the correct ones for each and every one of the given books. Here is an example output:

4 STARS OUT OF 5!
Bobson Bob B A
Fantastic Book
Loved listening to this book before bed.

2 STARS OUT OF 5!
Spellbad Jon Q
Not as good as Harry Potter
Never read the book, seen the movie or listened to the audio book but I can tell you right now - its not as good as harry potter

These are the reviews for the audio book "Fantastic Beasts and Where to Find Them", as we can see, everything looks good.
We can also check and confirm that the change in reviews is effective immediately when switching books.

Any change in the database is reflected on the GUI.

## Evaluation and Conclusion:

This Practical went well despite some important health issues (reason for the late submission). We really try to stay on the simple clean side and avoid any bugs by have a strong and solid logic.
Our coding plan was well set and the practical ran smoothly.

WORDS : ~1900.