****

**CASE STUDY: Starlabs Library (A)**

MODULE: CSI\_5\_BDD\_2223

DATE OF SUBMISSION: 09/05/2023

(APPROVED UNCAPPED EC, NO DECREASE IN MARKS)

STUDENT ID: 4008609

**Table of Contents**

[**1. INTRODUCTION** 3](#_Toc134495933)

[**2. DESIGN** 3](#_Toc134495934)

[**2.1. ENTITY RELATIONSHIP DIAGRAM (TASK 1)** 3](#_Toc134495935)

[**2.2. FUNCTIONAL DEPENDENCY STATEMENTS (TASK 2)** 5](#_Toc134495936)

[**3. SQL QUERIES** 7](#_Toc134495937)

[**3.1 Creating Tables (TASK 3)** 7](#_Toc134495938)

[**3.2 Inserting records (TASK 3)** 10](#_Toc134495939)

[**3.2.1 Inserting records in MembershipType + Testing** 10](#_Toc134495940)

[**3.3.2 Inserting records in Members + Testing** 11](#_Toc134495941)

[**3.2.3 Inserting records in Library Card + testing** 12](#_Toc134495942)

[**3.2.4 Inserting records in books + testing** 13](#_Toc134495943)

[**3.2.5 Inserting records in Volumes + testing** 14](#_Toc134495944)

[**3.2.6 Inserting records in PartnerLibrary + Testing** 15](#_Toc134495945)

[**3.2.7 Inserting records in InterLibraryLoans + Testing** 17](#_Toc134495946)

[**3.2.8 Inserting records in Borrow + Testing** 18](#_Toc134495947)

[**3.2.9 Inserting records in Payments + Testing** 19](#_Toc134495948)

[**3.3 MAIN Queries (TASK 4)** 20](#_Toc134495949)

[**3.3.1. TASK 4A** 20](#_Toc134495950)

[**3.3.2. TASK 4B** 21](#_Toc134495951)

[**3.3.3. TASK 4C** 22](#_Toc134495952)

[**3.3.4. TASK 4D** 23](#_Toc134495953)

[**3.3.5. TASK 4E** 24](#_Toc134495954)

[**4. DASHBOARD (TASK 5)** 28](#_Toc134495955)

[**5. VIDEO DEMO (TASK 6)** 29](#_Toc134495956)

[**5. CONCLUSION** 29](#_Toc134495957)

[**6. REFERENCES** 29](#_Toc134495958)

[**‌‌7. APPENDIX (FINAL CODE)** 30](#_Toc134495959)

# **1. INTRODUCTION**

The following report outlines the design, implementation, and testing of a system for the Starlabs library. It also includes a dashboard and a video demonstration for both the execution of queries and the interactive dashboard. Some of the system's features include The ability to keep track of members and their borrowing, loans Starlabs has taken out from partner libraries, and payments and taxes due to incurred loans. It also includes exciting functions such as the expiry date automatically being set to four years past the issue date and the ability to send renewal notices to customers. As per the client's requirements, the system has additional features involving weekly reports, the tracking of overdue books and many more, as set out below in section 3.3.

# **2. DESIGN**

**2.1. ENTITY RELATIONSHIP DIAGRAM (TASK 1)** Diagram, schematic

Description automatically generated

Figure 1: Entity Relationship Diagram

The ER diagram (blueprint for the database design) (see Figure 1) provides the general and abstract structure of all required tables using IE Crow’s Foot notation *(dybka, 2016).* This was possible through the help of The diagram includes all primary keys (indicated with a golden key logo), foreign keys (represented with a column logo and green arrow), data types (and their sizes for memory allocations) and the relationships between tables (using maximum and minimum cardinality).

**NOTE:** The specific content regarding attributes/columns will not be discussed as they can be easily interpreted using the diagram.

The primary keys for the tables produced are as follows:

**Book:** Primary key (ISBN)

**Volume:** Primary key (volume\_id)

**Member:** Primary key (member\_id)

**LibraryCard:** Primary key (card\_number)

**PartnerLibrary:** Primary key (partner\_libraryID)

**InternLibraryLoan:** Primary key (loan\_id)

**Borrow:** Primary key (borrow\_id)

**Membership:** Primary key (membership\_typeID)

The relationships for the tables, along with the thinking behind them, are as follows:

One **book** can have multiple **volumes**(optional).

One **book** (not copy) can be loaned to multiple **libraries**.

One **library** can have multiple **loans** from another library

One **volume** can have multiple **borrowers**

One **member** can borrow multiple **books**

One **member** has one **library** **card**

One **library** **card** has one **membership** **type** attached to it (i.e. student or lecturer)

**NOTE:** In this specific use case, the need for ‘mandatory-many’ relationships was redundant. However, this could be represented using the following notation *(Kroenke and Auer, 2016).*

Shape, rectangle

Description automatically generated

Figure 2: Mandatory-many example

It is worth discussing some design decisions made regarding the data types used and their sizes for specific attributes.

Initially, ISBN for books was allocated a variable character length of 13 (including hyphens). However, upon further research *(ISO, 2017)*, it was realised that ISBNs released after 2017 required 17 characters (including hyphens). For this reason, the column was allocated 20 characters to be safe. The description for the book has been allocated a size of 8000 characters. 2000 characters would have been suitable for shorter descriptions. However, it would not be sufficient for all. 8000 characters is a reasonable allocation and not too large as to cause significant performance deterioration.

The membership type column was allocated a type of tiny int (3), this would allow for three different types of users (student, lecturer and staff), an enum type would have also been suitable.

All dates have been allocated a ‘date’ type for obvious reasons including to allow the database to take in to account the fact that different months have a different number of days.

The photo for the library card is of type ‘blob’.

The **foreign key** allocations/references are listed below:

**Members:** library\_card\_id (FK): references the library\_card\_id in the LibraryCard entity.

**Volumes:** book\_id (FK): This foreign key references the book\_id in the Books entity.

**Borrow:** member\_id (FK): This foreign key references the member\_id in the Members entity.

**InterLibraryLoans:** book\_id (FK): This foreign key references the book\_id in the Books entity.

partner\_library\_id (FK): This foreign key references the partner\_library\_id in the PartnerLibrary entity.

**LibraryCard:** member\_id (FK): This foreign key references the member\_id in the Members entity.

These foreign keys help to maintain referential integrity in the database and define relationships between entities.

## **2.2. FUNCTIONAL DEPENDENCY STATEMENTS (TASK 2)**

To verify that all of the attributes within the entities are in at least 3NF/BCNF, we must first identify the functional dependencies for each table *(chapple, 2021).* We can then check for violations and normalize the schema accordingly.

Functional Dependencies:

1. MembershipType

- membership\_type\_id -> type\_name, loan\_period, max\_books, grace\_period

2. Members

- member\_id -> student\_id, first\_name, last\_name, campus\_address, home\_address, phone\_number, membership\_type\_id

3. LibraryCard

- member\_id -> membership\_type\_id, photo, name, issue\_date, renewal\_notice\_sent

4. Books

- ISBN -> title, author, subject\_area, description, is\_lendable, number\_of\_copies

5. Volumes

- volume\_id -> ISBN, status

6. PartnerLibrary

- partner\_library\_id -> library\_name, contact\_info

7. InterLibraryLoans

- interlibrary\_loan\_id -> ISBN, partner\_library\_id, start\_date, end\_date

8. Borrow

- borrow\_id -> member\_id, volume\_id, borrow\_date, due\_date, return\_date

9. Payment

- payment\_id -> interlibrary\_loan\_id, amount

Now, let's analyse each table for 3NF/BCNF compliance:

1. MembershipType: The primary key is membership\_type\_id, and all other attributes depend on it, so it is in BCNF.

2. Members: The primary key is member\_id, and all other attributes depend on it, so it is in BCNF.

3. LibraryCard: The primary key is member\_id, and all other attributes depend on it, so it is in BCNF.

4. Books: The primary key is ISBN, and all other attributes depend on it, so it is in BCNF.

5. Volumes: The primary key is volume\_id, and all other attributes depend on it, so it is in BCNF.

6. PartnerLibrary: The primary key is partner\_library\_id, and all other attributes depend on it, so it is in BCNF.

7. InterLibraryLoans: The primary key is interlibrary\_loan\_id, and all other attributes depend on it, so it is in BCNF.

8. Borrow: The primary key is borrow\_id, and all other attributes depend on it, so it is in BCNF.

9. Payment: The primary key is payment\_id, and all other attributes depend on it, so it is in BCNF.

All the tables are in BCNF, which means they are also in 3NF, as BCNF is a stronger normalization form than 3NF.

# **3. SQL QUERIES**

## **3.1 Creating Tables (TASK 3)**

The process of creating SQL tables was achieved through the guidance of lectures carried out by *(Ubakanma, 2023)*

Graphical user interface, text, application

Description automatically generated

Figure 3: Creating Tables 1

The code shown in Figure 3 is responsible for initialising and creating the MemberhipType, Members and LibraryCard tables. Each membership type (student, lecturer and staff) is tied to a fixed value, which is automatically incremented when the user wishes to add more types. As per the user requirements, you can set the specific maximum loan period, the maximum books to be taken out and the grace period based on the type of member.

The members have their type of membership tied to their data, i.e. student.

Initially, I had a different id number for the library card. However, after testing and further reflection, it was much more efficient to have the member id number tied to the library card as a primary key. Per the user requirements, the library card has a photo of type blob (or varchar). The LibraryCard table also automatically sets the expiry date to 4 years past the issue date (as per requirements). The issue date could have also been set automatically based on the current date. However, for testing purposes, it is inputted manually. The card also inherits the foreign key for the membership type and the member id number.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Figure 4: Creating Tables 2

The tables in Figure 4 show the creation of the following tables: Books, Volumes, PartnerLibrary, InterLibraryLoans (loans from other libraries), Borrow and Payment.

As discussed in section 1.1, the book ISBN (primary key) has been allocated 20 characters. The description, however, has been set to type text (instead of varchar) upon further reflection. The book has a binary bit tied to it. This allows the librarian to track whether the book is lendable. It is set to 1 by default (meaning it is lendable). However, it can be switched to 0 with a trigger, for example, when the number of copies hits 0. Ass columns are set to not being lull (meaning they can't be null) apart from the description.

The volumes table had a primary key named Volume\_id, along with a tied ISBN, which is a foreign key from the books table. It also has a status allowing the tracking of whether the volume is available to be lent or not. It is set to available by default.

The PartnerLibrary table is responsible for keeping track of all external libraries our library is partnered with. Their information is held in the table, with the primary key being their id (automatically incremented), library name, and contact information.

The InternLibraryLoans table is created to keep track of all loans the library has taken out, along with specific information such as the book, the partner library’s id and the dates. The ISBN for the book being lent is referenced from the books tables, and the partner library id is referenced from the partner library table.

As per section 1.1, the borrow table has been created to keep track of all books being borrowed by members from the main library. The rows are distinguished with the primary ey borrow\_id. The table also keeps track of member, book and date information. The member id is referenced from the member tables, and the volume id is referenced from the volumes tables. Upon reflection, using the book’s ISBN rather than the volume id would have made more sense.

Due to the future queries required by the client, a payment table had to be created. This allows the library to keep track of payments made to partner libraries when a loan has been taken out. The cost/payment amount is set to two decimal points.

Table

Description automatically generated

Figure 5: Testing table creation

Upon creating the tables, I used the MS SQL SSMS to ensure all tables were created upon executing the queries. They were all successful.

## **3.2 Inserting records (TASK 3)**

### **3.2.1 Inserting records in MembershipType + Testing**

A picture containing text

Description automatically generated

Figure 6: Adding three rows to the membership type table

It made logical sense first to insert the rows for the membership type table. As per the client brief, students can borrow books for 21 days, a maximum of 5 books, with a 7-day grace period.

Lecturers and researchers have more gracious terms where they can borrow a book for 90 days, 5 books maximum (not stated in brief), with a grace period of 2 weeks. All library staff have also been allocated the same conditions

Graphical user interface, application

Description automatically generated

Figure 7: Testing all rows are inserted correctly

Upon executing the insert queries, it was vital to ensure that all rows had been inserted appropriately. This was checked through the SELECT \* query, and ensured they had all been accurate and successful

### **3.3.2 Inserting records in Members + Testing**

Text

Description automatically generated with low confidence

Figure 8: Inserting 20 records into the library card table

It then made logical sense to then insert queries into the members table. 21 Members were inserted with randomised ID numbers. I thought it would be reasonable to have three members who were lecturers, one staff member and the rest students. Their addresses and phone numbers had also been entered into the insert query.

Graphical user interface, application, table

Description automatically generated

Figure 9: Testing all rows are inserted correctly

Upon executing the insert query, I used the select statement to ensure all rows were inserted. This was successful, and using the member\_id numbers, I could see that 20 rows were entered, proving none were missed.

### **3.2.3 Inserting records in Library Card + testing**

Chart, text, scatter chart

Description automatically generated

Figure 10: Inserting 20 records into the Members table

Because I had now specified the members and their associate type, it made sense to now issue their library cards. I had manually inserted all their details (this could have also been done automatically by using the members table) and specified the issue dates. As stated previously, this was done for testing purposes. However, I could have also made it to set the issue date automatically based on the current date. The renewal notice sent bit was set to 0 for all members. Their photo was set to null for the time being.

Table

Description automatically generated

Figure 11: Testing all rows are inserted correctly

Through testing, I ensured that the expiry date set was four years past the issue date (as per requirements). This was expected based on the conditions set when initially creating the table.

### **3.2.4 Inserting records in books + testing**

A picture containing text

Description automatically generated

Figure 12: Inserting 21 records into the books table with a variety of subjects (cropped)

I could insert rows of around 20 books. I decided to have a variety of types of books for testing purposes. Ten books were based on computer science, two psychology, three on cooking and six on physics. The ISBN and number of copies were relatively random. The image is cropped for easier readability.

Graphical user interface, text, application, email

Description automatically generated

Figure 13: Testing all rows are inserted correctly

After executing the insert query, I tested by using a select statement to list the table. All rows were inserted as expected, and the lendable bit was set to 1 by default.

### **3.2.5 Inserting records in Volumes + testing**

A picture containing logo

Description automatically generated

Figure 14: Copying records from books to Volumes

I then inserted the ISBN column from the books table into the volumes table. I also set the status for all volumes to be available.

Table

Description automatically generated

Figure 15: Testing all rows are inserted correctly

Upon testing, it could be seen that the isbn rows were imported as expected.

### **3.2.6 Inserting records in PartnerLibrary + Testing**

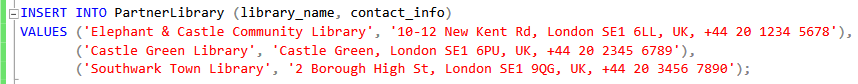


Figure 16: Inserting three partner libraries

I then inserted three rows of partner libraries from which our library can borrow books. The columns contain the library’s name and contact information

Graphical user interface, text, application

Description automatically generated

Figure 16: Testing all rows are inserted correctly

I was then able to test that the rows were inserted using the select statement and could observe it was successful.

### **3.2.7 Inserting records in InterLibraryLoans + Testing**

Table

Description automatically generated with medium confidence

Figure 17: Inserting entries into InterLibraryLoans Table

After adding the list of partner libraries, I inserted 20 entries for loans taken out for books from one of three partner libraries. Again, the start date could be set in the create table query to be automatically based on the current date. I did it manually for testing purposes

Table

Description automatically generated

Figure 18: Testing all rows are inserted correctly

I then tested using the select statements, and all rows were inserted correctly.

### **3.2.8 Inserting records in Borrow + Testing**

Table

Description automatically generated with medium confidence

Figure 19: Inserting entries in to Borrow table

I then inserted some rows into the borrow table (members borrowing from the main library). The rows are distinguished by the borrow id, which is incremented automatically. Again I set the dates manually for testing purposes. Furthermore, as stated above, using the ISBN rather than volume id would have been more efficient and straightforward.

Table

Description automatically generated

Figure 20: Testing all rows are inserted correctly

There are more rows here than in the insert statement above due to testing.

### **3.2.9 Inserting records in Payments + Testing**

Table

Description automatically generated

Figure 21: Inserting entries into Payment Table

I then inserted rows into the payment table for loans taken out from partner libraries. This table will be used for tasks d and e.

Table

Description automatically generated with medium confidence

Figure 22: Testing all rows are inserted correctly

All rows were inserted correctly, with each entry's primary key payment\_id automatically incremented.

## **3.3 MAIN Queries (TASK 4)**

### **3.3.1. TASK 4A**

Table

Description automatically generated

Figure 23: Query to find books overdue by at least three days from the due date (task 4A)

This query retrieves a list of overdue books by at least three days, along with their details and associated borrow information. The query selects data from the "Books" table, assigning it an alias 'b,' and combines it with the "Volumes" table using an INNER JOIN. The "Volumes" table is assigned an alias 'v,' and the JOIN condition specifies that the rows from both tables should be matched based on their ISBN values (Date, 2013). Additionally, the query performs another INNER JOIN with the "Borrow" table, which is assigned an alias 'bo.' This join matches rows based on their volume\_id values. The WHERE clause filters the results to include only records where the return\_date is NULL, indicating that the book has not been returned and the due\_date is more than three days from the current date.

### **3.3.2. TASK 4B**

Table

Description automatically generated

Figure 24: Query to find customers who took less than five loans in 3 months (task 4B)

This query retrieves the details of members who have borrowed less than five books within the last three months. The query begins by selecting the required columns from the "Members" and "Borrow" tables, including member\_id, first\_name, last\_name, borrow\_id, borrow\_date, due\_date, and return\_date. It also calculates each member's total number of loans using the COUNT function.

The "Members" table is joined with the "Borrow" table using an INNER JOIN, matching the rows based on their member\_id values. The WHERE clause filters the results to show only those records where the borrow\_date is within the last three months.

The query then groups the results by member\_id, first\_name, last\_name, borrow\_id, borrow\_date, due\_date, and return\_date. The HAVING clause further filters the grouped results to include only those members who have borrowed less than five books within the specified period.

### **3.3.3. TASK 4C**

Table

Description automatically generated

Figure 25: Weekly lending report (task 4C)

First, I executed an insert query for testing purposes so that the main query would output some rows.

This query is designed to retrieve information on the loans that took place within the last seven days. It gathers data on the loan item, book description, start and due dates of the loan, actual return date (if applicable), and member details such as first name, last name, phone number, campus address, and home address.

The query starts by selecting the required columns from the "Borrow," "Books," and "Members" tables. It then performs INNER JOIN operations to combine the "Borrow" table with the "Volumes" and "Books" tables based on the volume\_id and ISBN values, respectively. Another INNER JOIN is carried out to link the "Members" table using member\_id.

The WHERE clause filters the results to show only those records where the borrow\_date falls within the range of the last seven days, including today. This way, the query provides information on all loans within the specified time frame.

### **3.3.4. TASK 4D**

The structure of the stored procedure was created through the guidence from *(W3Schools, 2019).*

Text

Description automatically generated

Figure 26: Current Inter Loan stored procedure (task 4D)

The stored procedure generates a report on current interlibrary loans by displaying all active inter-library loan items, the partner library that loaned the item to Starlabs, and the item details, along with the current loan duration in days. The query achieves this by joining the InterLibraryLoans, Books, and PartnerLibrary tables based on their common attributes, such as ISBN and partner\_library\_id. This allows the procedure to gather information on the book's title, author, ISBN, and the partner library's name.

The loan start and end dates are obtained directly from the InterLibraryLoans table. The loan duration in days is calculated using the DATEDIFF function, which measures the difference between the loan's start\_date and the current date. The WHERE clause then filters the results to display only those loans that are either currently ongoing or have no specified end date, effectively showing only active interlibrary loans in the report.

Graphical user interface, application

Description automatically generated

Figure 27: Testing Current Inter Loan stored procedure

To test that the stored procedure was working as intended, I first modified the InterLibraryLoans table to ensure it met the conditions set out in Figure 26 by setting the end date to null for the last three loans in the table. After, I could use the EXEC command was used to execute the stored procedure. It executed as expected, as shown in Figure 27.

### **3.3.5. TASK 4E**

Text

Description automatically generated

Figure 28: Stored procedure for dynamic tax report

The stored procedure called "MonthlyItemizedLoansReport" is designed to generate a monthly itemized statement of loans for each lender, allowing for dynamic searches by week, month, or quarter (3 months). The report includes payment, tax (at a 20% VAT rate), and total amounts for each loan.

The procedure accepts a single input parameter, @IntervalType, which determines the time period for the report (week, month, or quarter). It begins by setting NOCOUNT to ON, which suppresses the display of the number of rows affected by the query. It then declares two local variables, @StartDate and @EndDate, and initializes them with sample date ranges for demonstration purposes.

The procedure uses conditional statements to set the end date based on the given @IntervalType. If the value is "Week," it adds one week to the start date. If it's "Month," it adds one month to the start date. For "Quarter," it adds three months to the start date. If none of these conditions are met, the procedure returns without generating a report.

Next, the procedure queries the InterLibraryLoans, Books, PartnerLibrary, and Payment tables by joining them on common attributes such as ISBN, partner\_library\_id, and interlibrary\_loan\_id. This allows it to gather information about the lender, book title, author, ISBN, loan start and end dates, payment, tax, and total amounts for each loan.

The WHERE clause filters the results to include only loans with start dates within the specified range between @StartDate and @EndDate. Finally, the query orders the results by lender and loan start date, providing a neatly organized report for the requested time period.

Table

Description automatically generated

Figure 29: Updating InterLibraryLoans table for testing

Before testing the stored procedure for calculating tax, I first had to update the InterLibraryLoans table to ensure that the start and end dates were compliant with the conditions set in the stored procedure

Graphical user interface, application

Description automatically generated

Figure 30: Executing stored procedure for the week

I then executed the stored procedure with the week parameter. It can be observed that as per the update entries shown above, it only outputted one row, successfully calculating the tax and adding it to the total.

Graphical user interface, application

Description automatically generated

Figure 31: Executing stored procedure for the month

I then executed the same stored procedure, but with the month parameter, and as expected based on the update query, it only displayed two rows

Graphical user interface, application

Description automatically generated

Figure 32: Executing stored procedure for the quarter

Finally, I executed the monthly tax report, which successfully displayed all six loans taken out by the library and the calculated tax, which is then added to the total.

# **4. DASHBOARD (TASK 5)**

Chart

Description automatically generated

Figure 33: Dashboard for storytelling

The dashboard shown above visualises some critical data regarding Starlabs.

The design decision are in line with guidance of a video from (YouTube, 2016) provided by *(Ubakanma, 2023).*

It can be observed that 50% of the books are related to computer science, 25% to Physics, 15% to cooking and 10% to psychology. This suggests the business may want to increase the range of types of topics for books they provide. Furthermore, they could also increase their psychology book range to entice more customers.

The gauge located on the top right displays the total number of members. I have set the maximum capacity to 25, considering only one staff member exists. This suggests that once the library surpasses 23 members, they may want to consider hiring more staff/librarians.

The gauge on the bottom left displays the average price/payment paid to partner libraries. I thought it would be reasonable to set the maximum to £30. This gauge can be used to measure KPIs and encourage staff to keep payments low to increase profit margins.

The bar chart displayed on the bottom right shows the distribution of the number of students, lecturers and staff members.

Though the use of colour is high, which may be undesired for colour-blind stakeholders, this can be overlooked due to the interactiveness of the dashboard, along with the fact that tags are shown once the user hovers over the specific data, for example, purple on the pie chart.

# **5. VIDEO DEMO (TASK 6)**

<https://www.youtube.com/watch?v=mlgsWv1gkzc>

# **5. CONCLUSION**

Overall, the creation of the StarLabs library system was successful, with all required queries being successfully implemented and tested. The design decisions made in section 2.1 using the ER diagram were also well thought out, considering that only minor changes had to be made with the implementation.

It is worth noting that creating a volumes table caused the design of the tables/system to be a little complicated. This was, however, well managed through effective relationship management between tables.

Through data visualisation and analysis, we could incur the following:

1. The library may benefit from having a larger variety of books.
2. The library has high payments to partner libraries for computer science books.
3. The member capacity is reaching near its maximum threshold. The library may want to hire another staff member.
4. The amount of tax paid is relatively high. The library might want to employ an accountant to manage tax right offs.

# **6. REFERENCES**

[1] Dybka, P. (2016). *Crow’s Foot Notation*. [online] Vertabelo Data Modeler. Available at: <https://vertabelo.com/blog/crow-s-foot-notation/>. [Accessed 20/05/2023]

[2] Kroenke, D.M. and Auer, D.J. (2016). *Database processing : fundamentals, design, and implementation*. Boston: Pearson.

‌[3] Chapple M.C (2016). *Protect Your Data by Normalizing It to at Least 2NF*. [online] Lifewire. Available at: <https://www.lifewire.com/full-functional-dependency-1019753>. [Accessed 20/05/2023]

[4] Ubakanma, G. (2022) SQL Database Construction [Handout from Lecture]. London south bank university, week 3.

[5] www.youtube.com. (n.d.). Tableau in Two Minutes - Tableau Basics for Beginners. [online] Available at: https://youtu.be/jEgVto5QME8 [Accessed 4 May 2023].

‌[6] W3Schools (2019). SQL Stored Procedures. [online] W3schools.com. Available at: <https://www.w3schools.com/sql/sql_stored_procedures.asp>. [Accessed 04 May 2023]

[7] 14:00-17:00 (n.d.). ISO 2108:2017. [online] ISO. Available at: https://www.iso.org/standard/65483.html [Accessed 9 May 2023].

‌

# **‌‌7. APPENDIX (FINAL CODE)**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generatedGraphical user interface, text, application, email

Description automatically generatedText

Description automatically generatedChart, text, scatter chart

Description automatically generatedA picture containing background pattern

Description automatically generatedA picture containing table

Description automatically generatedTable

Description automatically generatedText, application, email

Description automatically generatedGraphical user interface, text, application, email

Description automatically generatedGraphical user interface, application, table

Description automatically generated