IS5102 Assignment 2 Report
Student ID: 220023823

Task 1: Translation

orderBook(order_id, *customer_id*, street, city, postcode, country, date_ordered, date_delivered)

customer(customer_id, email, street, city, postcode, country)

customer_phone(*customer_id*, phone_type, phone_number)

reviews(*customer_id*, *book_id*, rating)

book(book_id, title, author, publisher)

book_genre(*book_id*, genre)

edition(*book_id*, book_edition, book_type, price, quantity_in_stock)

containsAn(*order_id*, *book_id*, *book_edition,* *book_type*, amount)

supplies(*book_id*, *book_edition,* *book_type,* *supplier_id*, supply_price)

supplier(supplier_id, name, account_no)

supplier_phone(*supplier_id*, *phone*)

Key for relation schema

Underlined: Primary Key

Italicised: Foreign Key

Brief rational for design choices

Every attribute id was converted to a specific attribute id for that table (e.g., attribute id for customer entity is changed to customer_id). Order is written as orderBook. Multivalued attributes are included as a separate table, with the primary key from the entity set they belong included as a primary and foreign key. The multivalued attribute tables are phone (type and number) and genre which are written as customer_phone and book_genre to identify which entity set the attributes belong to. Attributes edition and type from the weak entity set edition are written as book_edition and book_type to avoid confusion.

Task 2: SQL Data Definition

The report of task 2 is separated into two sections. **2.1** describes the rationale behind the table design and any integrity constraints implemented. **2.2** discusses the approach taken to test all the code to ensure the integrity of the data. Testing the data took place after the database had been populated with the test data.

2.1

orderBook

The primary key (PK) order_id must have a distinct value and cannot be null. Order_id is written with two letters and four numbers (e.g., OR1013). For simplicity, however, the numbers are entered to make them minimally recognisable. Customer_id is a foreign key (FK) that refers to the PK in the customer table; the constraint for this attribute also requires the value to be distinct and not a null value. This attribute is written with one letter (C) and four numbers. In an application, when there are many customers and many orders, the order id and customer id would have to be better thought out and consist of two random letters and five random numbers.

The attributes street, city, postcode, and country all have the integrity constraint, not null because every order must have a specific address. Otherwise, the order sends to nowhere. In addition, the date_ordered attribute also has the constraint not null because an order must be placed on a specific day, month, and year. Therefore, these constraints specify that no blank entries are allowed for either of these columns. Date_delivered does not have this integrity constraint; an order is opened when the order is placed and not when the order is delivered. Therefore, the date_delivered attribute may be left empty if the order is still being delivered.

customer

Customer_id is the PK in this table and has unique and not null integrity constraints. There is no FK in this table because the customer table is on the many side of the many-to-one relationship with the orderBook table and has a many-to-many relationship with the Book table.

The attribute email has integrity constraint, not null because every customer must have an email (must be able to communicate with them over email; send them receipt ext.). This attribute also has a unique constraint because an email address must be distinct (if an email already exists in the database, it cannot be entered again). The customer's street, city, postcode, and country

address can have a null value in this table. This database would allow customers to sign up without entering a home address. Whereas every order must have an address because the order must go somewhere, every customer does not need an address. Customers may only want to browse what books are available, order from a third party (such as a friend), or not wish to disclose their home address.

customer_phone

This table is to record the multivalued composite attribute phone (with composite attributes phone_type and phone_number), which belongs to the customer table. The PKs are customer_id, phone_type, and phone_number. Customer_id is also an FK which refers to the PK in the customer table.

reviews

This table records the relationship between customer and book. Customer_id is a PK and an FK, which refers to the primary key in the customer table. Book_id is also a PK and an FK, which refers to the PK in the book table. Both attributes have distinct values and cannot have null values. Rating is a relationship attribute that records a customer's rating of a particular book they order; acceptable values are 1, 2, 3, 4, 5. Rating can have a null value because customers may choose not to give a rating.

book

The PK is book_id, which has distinct values and cannot have null values. Book_id is written with one letter (B) and four numbers. The attribute title cannot have null values, but values do not have to be unique (they may have different types of the same book). The attribute author can have a null value because some books may have an anonymous author, or the author may be unknown. The attribute publisher can also have a null value because it is possible to sell a book without a publisher, perhaps if it is only available in audiobook form or self-published by the author.

book_genre

This table records the multivalued attribute genre which belongs to the book table. Book_id is a PK and FK, which refers to the primary key in the book table. Genre is also a PK and cannot have a null value; every value must be unique.

edition

The edition table is a weak entity set and therefore the PK from the strong entity set it is linked (book_id) makes up a part of the PK in this table and is also an FK which refers to the PK in the Book table. The value of the PK in this table is made up of three columns (book_id, book_edition, book_type).

```
CONSTRAINT PK_edBookType PRIMARY KEY (book_id, book_edition,
book_type),
```

Book_edition is expressed as an integer. Book_type can be either audiobook, hardcover, or paperback. To test ensure this integrity constraint is upheld, the following condition is included in the table code.

```
CONSTRAINT chk_book_type CHECK (book_type IN ('paperback',
'hardcover', 'audiobook')),
```

This requires that any value entered in the book_type column must conform to one of these three values. In addition, book_type cannot contain a null value – every book must conform to some type of book. Quantity_in_stock is also an integer, with the default value set to 0. Furthermore, because this table is a weak entity set cascading actions are included to ensure referential integrity,

```
ON DELETE CASCADE
ON UPDATE CASCADE);
```

containsAn

In this relationship table, order_id is a PK, and FK (that refers to the PK in the table orderBook). Book_id is a PK and FK, which refers to the PK in the table book. Book_edition and book_type are PKs and FKs, which refer to the PKs in the edition table. Amount is a relationship attribute and records the number of a specific type of book in an order as an integer.

supplies

In this relationship table, Book_id is a PK, and FK (that refers to the PK in the table book). Book_edition and book_type are PKs and FKs, which refer to the PKs in the edition table. Supplier_id is a PK and FK, which refers to the PK in the table supplier. Supply_price records

the price charged by each supplier to this bookstore. This attribute has a not null integrity constraint because every book sold to the bookstore must be associated with a specific price.

supplier

supplier_id is the PK. The attributes name and account_no have integrity constraints, not null, because every supplier must have a name and an account number. The account number must also be unique to identify the three suppliers.

2.2

An overview of the testing process of the SQL code to ensure the integrity of the data will now be discussed. First, it is important to test that the table constraints make it impossible for a user of the database to input a field value which should not be allowed. To test that the PK constraint is upheld (i.e., that every PK is unique) , the same PK tried to be inserted twice. For example, the following test was run,

```
INSERT INTO orderBook
VALUES ('OR1000', 'C1000', 'Daisy street', 'London', 'LO
3187', 'England', '2021-04-16', '2021-04-18');
```

which returns the following error message,

```
SQL Error [19]: [SQLITE_CONSTRAINT_PRIMARYKEY] A PRIMARY KEY
constraint failed (UNIQUE constraint failed:
orderBook.order_id).
```

To test not null integrity constraints, the INSERT INTO code was utilized to try to insert null values where they should not be allowed. For example, to test that order_id cannot obtain a null value the following test was run,

```
INSERT INTO orderbook
VALUES (NULL, 'C7530', 'Daisy street', 'London', 'LO 3187',
'England', '2018-04-16', '2021-06-28');
```

which returns the following error message,

```
SQL Error [19]: [SQLITE_CONSTRAINT_NOTNULL] A NOT NULL
constraint failed (NOT NULL constraint failed:
orderBook.order_id)
```

Therefore, the integrity constraint NOT NULL for order_id is verified because when you try to enter a null value it returns error. This method of testing was employed through all null value constraints written in the code.

To test the unique integrity constraint, the same value for an attribute was added twice. For example, phone_number is required to be unique. Customer C1000 already has the phone number '714921236', so when we try to add this same number for customer C7530 it should not be allowed. An example of how this was tested,

```
INSERT INTO customer_phone
VALUES ('C7530', '+44', '714921236');
```

which returns the following error message,

```
SQL Error [19]: [SQLITE_CONSTRAINT_UNIQUE] A UNIQUE constraint
failed (UNIQUE constraint failed: customer_phone.phone_number)
```

To test that the values entered in book_type conform to one of the three values ‘hardcover’, ‘paperback’, ‘audiobook’, the following code was run,

```
INSERT INTO edition
VALUES ('B0001', 1, 'eBook', 40.00, 2);
```

which returns the following error message

```
SQL Error [19]: [SQLITE_CONSTRAINT_CHECK] A CHECK constraint
failed (CHECK constraint failed: chk_book_type).
```

Task 3: SQL Data Manipulation

3.1 Provided Queries

1. List all books published by “Ultimate Books” which are in the “Science Fiction” genre.

Code:

```
SELECT book_id, title, author
FROM book NATURAL JOIN
      (SELECT book_id
       FROM book_genre
       WHERE genre = 'Science Fiction')
WHERE publisher = 'Ultimate Books';
```

Output:

```
|book_id|title                              |author       |
|-------|-----------------------------------|-------------|
|B0001  |A Hitchhikers Guide to the Galaxy|Douglas Adams|
|B0005  |Dune                               |Frank Herbert|
```

2. List titles and ratings of all books in the “Science and Technology” genre, ordered first by rating (top rated first), and then by the title

Code:

```
SELECT book_id, title, AVG(rating)
FROM book NATURAL JOIN reviews
NATURAL JOIN (
      SELECT book_id
      FROM book_genre
      WHERE genre = 'Science and Technology')
GROUP BY book_id ORDER BY AVG(rating) DESC, title;
```

Output:

```
|book_id|title                                  |AVG(rating)|
|-------|---------------------------------------|-----------|
|B0008  |The Structure of Scientific Revolutions|5          |
|B0003  |A Brief History of Time                |4          |
|B0009  |Sapians: A Brief History of Humankind  |3          |
```

3. List all orders placed by customers with customer address in the city of Edinburgh, since 2020, in chronological order, latest first

Code:

```
SELECT order_id, customer_id, date_ordered
FROM orderBook NATURAL JOIN (
      SELECT customer_id
      FROM customer
      WHERE city = 'Edinburgh')
WHERE date_ordered >= '2020-01-01'
ORDER BY date_ordered DESC;
```

Output:

```
|order_id|customer_id|date_ordered|
|--------|-----------|------------|
|OR6000  |C5000      |2022-01-01  |
|OR4000  |C3000      |2021-06-13  |
```

4. List all book editions which have less than 5 items in stock, together with the name, account number and supply price of the minimum priced supplier for that edition.

Code:

```
SELECT book_id, book_edition, name, account_no,
MIN(supply_price) AS "Min Pr. Supplier"
FROM edition NATURAL JOIN supplies NATURAL JOIN book NATURAL
JOIN supplier
WHERE quantity_in_stock < 5
GROUP BY book_id, book_edition;
```

Output:

```
|book_id|book_edition|name           |account_no|Min Pr. Supplier|
|-------|------------|---------------|----------|----------------|
|B0001  |1           |Macro Books    |S1002     |25              |
|B0001  |2           |Exclusive Books|S1001     |45              |
|B0002  |1           |Exclusive Books|S1001     |75              |
|B0003  |1           |Macro Books    |S1002     |40              |
|B0003  |2           |AudioRus       |S1003     |48              |
|B0003  |3           |Macro Books    |S1002     |65              |
|B0005  |1           |Exclusive Books|S1001     |100             |
|B0007  |1           |Macro Books    |S1002     |76              |
|B0008  |1           |Exclusive Books|S1001     |25              |
|B0008  |3           |Exclusive Books|S1001     |30              |
|B0010  |1           |Macro Books    |S1002     |82              |
```

Note: book_id was included because some books have multiple editions and so it will be sensible to view which specific book that edition is associated with.

5. Calculate the total value of all audiobook sales since 2020 for each publisher.

Code:

Note: First I wanted to check the values manually, in case one publisher has sold different audio books.

```
SELECT publisher,  book_id, order_id, price, amount,
price*amount AS "Total Sales"
FROM book NATURAL JOIN edition NATURAL JOIN containsAn NATURAL
JOIN orderBook
WHERE book_type ='audiobook' AND date_delivered >= '2020-01-
01';
```

Output:

```
|publisher      |book_id|order_id|price|amount|Total Sales|
|---------------|-------|--------|-----|------|-----------|
|Ultimate Books |B0001  |OR1000  |50   |1     |50         |
|Ultimate Books |B0008  |OR2000  |45   |5     |225        |
|Penguin        |B0004  |OR4000  |35   |3     |105        |
|Dvir House Ltd.|B0009  |OR5000  |42   |1     |42         |
```

Note: Then I implemented the final code

Code:

```
SELECT publisher, SUM(price*amount) AS "Total Sales"
FROM book NATURAL JOIN edition NATURAL JOIN containsAn NATURAL
JOIN orderBook
WHERE book_type ='audiobook' AND date_delivered >= '2020-01-
01'
GROUP BY publisher;
```

Output:

```
|publisher      |Total Sales|
|---------------|-----------|
|Dvir House Ltd.|42         |
|Penguin        |105        |
|Ultimate Books |275        |
```

3.2 Three new queries

1. List title, edition, type, author, price, and quantity in stock of book/s with a 5-star rating.

Code:

```
SELECT title, book_edition AS "Edition", book_type, price,
quantity_in_stock AS "Quantity"
FROM book NATURAL JOIN edition
NATURAL JOIN (
     SELECT book_id
     FROM reviews
     WHERE rating = 5);
```

Output:

title	Edition	book_type	price	Quantity
A Hitchhikers Guide to the Galaxy	1	paperback	40	2
A Hitchhikers Guide to the Galaxy	2	paperback	50	3
A Brief History of Time	1	paperback	50	3
A Brief History of Time	2	paperback	60	3
A Brief History of Time	3	hardcover	80	2
The Structure of Scientific Revolutions	1	paperback	35	4
The Structure of Scientific Revolutions	3	audiobook	45	7
The Structure of Scientific Revolutions	3	paperback	45	2
RISE	1	hardcover	92	1

2. List the average price charged by each supplier for each book type

Code:

```
SELECT supplier_id, name, book_type, round(AVG(supply_price),
2) AS "Avg. Supply Price"
FROM supplies NATURAL JOIN supplier NATURAL JOIN book
GROUP BY supplier_id, book_type;
```

Output:

supplier_id	name	book_type	Avg. Supply Price
S1	Exclusive Books	audiobook	45
S1	Exclusive Books	hardcover	87,67
S1	Exclusive Books	paperback	33,75
S2	Macro Books	hardcover	74,33
S2	Macro Books	paperback	40,5
S3	AudioRus	audiobook	33
S3	AudioRus	paperback	48

3. List customer ID and emails of customers who purchased more than one hardcover book.

Code:

```
SELECT order_id, customer_id, email
```

```
FROM containsAn NATURAL JOIN orderBook NATURAL JOIN customer
WHERE book_type = 'hardcover' AND amount > 1;
```
Output:

```
|order_id|customer_id|email               |
|--------|-----------|--------------------|
|OR2000  |C1000      |bongi@gmail.com     |
|OR7000  |C6000      |hendrikse@gmail.com|
|OR1012  |C1010      |cheslin@gmail.com   |
```

3.3 Views

1. Create a view of profit margin made (mark-up) on each book.

Code:

```
CREATE VIEW mark_up AS
SELECT book_id, price, supply_price, price - supply_price AS
"Profit Margin"
FROM edition NATURAL JOIN supplies;

SELECT * FROM mark_up
```

Output:

```
|book_id|price|supply_price|Profit Margin|
|-------|-----|------------|-------------|
|B0001  |40   |35          |5            |
|B0001  |40   |25          |15           |
|B0001  |50   |45          |5            |
|B0002  |80   |75          |5            |
|B0003  |50   |40          |10           |
|B0003  |80   |65          |15           |
|B0003  |50   |45          |5            |
|B0003  |60   |48          |12           |
|B0004  |35   |32          |3            |
|B0005  |110  |100         |10           |
|B0005  |72   |62          |10           |
|B0006  |45   |35          |10           |
|B0007  |86   |76          |10           |
|B0008  |35   |25          |10           |
|B0008  |45   |30          |15           |
|B0008  |45   |30          |15           |
```

B0009	42	37	5
B0010	92	82	10
B0010	92	88	4

3.2 Create a view of orders without customer address

Code:

```
CREATE VIEW order_overview AS
SELECT order_id, customer_id, date_ordered, date_delivered
FROM orderBook;

SELECT * FROM order_overview
```

Output:

order_id	customer_id	date_ordered	date_delivered
OR1000	C1000	2021-04-16	2021-04-18
OR2000	C1000	2022-03-12	2022-03-14
OR3000	C2000	2022-03-20	2022-03-22
OR4000	C3000	2021-06-13	2021-06-15
OR5000	C4000	2021-05-26	2021-05-28
OR6000	C5000	2022-01-01	2021-01-03
OR7000	C6000	2020-04-26	2020-04-28
OR8000	C6000	2021-11-26	2021-11-28
OR9000	C7000	2019-11-02	2019-11-08
OR1010	C8000	2022-05-20	2022-05-23
OR1011	C9000	2018-12-03	2018-12-11
OR1012	C1010	2022-02-19	2022-02-24
OR1013	C1010	2022-02-25	2022-02-29

Task 4: Reflection

The overall experience was positive and felt more evident than the first assignment. The E-R diagram was a huge help; all the data and its links were nearly all set out for you. If we had to design an E-R diagram and then follow the steps in this assignment, it would become evident where the mistakes in the design process are taking place. I implemented the integrity constraints in a manner that was thoughtful and realistic. I also kept the design process and test data simple and easily readable.

Moreover, I could implement the given queries accurately and provide views that made practical sense. Initially, I found answering queries 4 and 5 very challenging. I needed help understanding how to get the minimum amount for the price with the group supplier. The same is true for finding the product of price times amount for total sales and grouping it correctly. The biggest problem I encountered was that I left the PRAGMA foreign_keys = TRUE; integrity constraint out. I only realised this when running my code for the final time. This issue took me a long time to resolve as I had multiple FK constraint failures and mismatches. The way I resolved this issue was to go over each table again and inputs of test data to see where the mistakes were occurring. If I had more time, I would have tried to think of more advanced queries. I would also consider more views to ensure better customer and supplier data privacy.