Practical 2: Database management with SQLite

IS5102 – Database Management Systems

Matriculation Id - 220032472

TASK 1

Translation

The given ER model needs to be translated into database schema (collection of relation schemas)

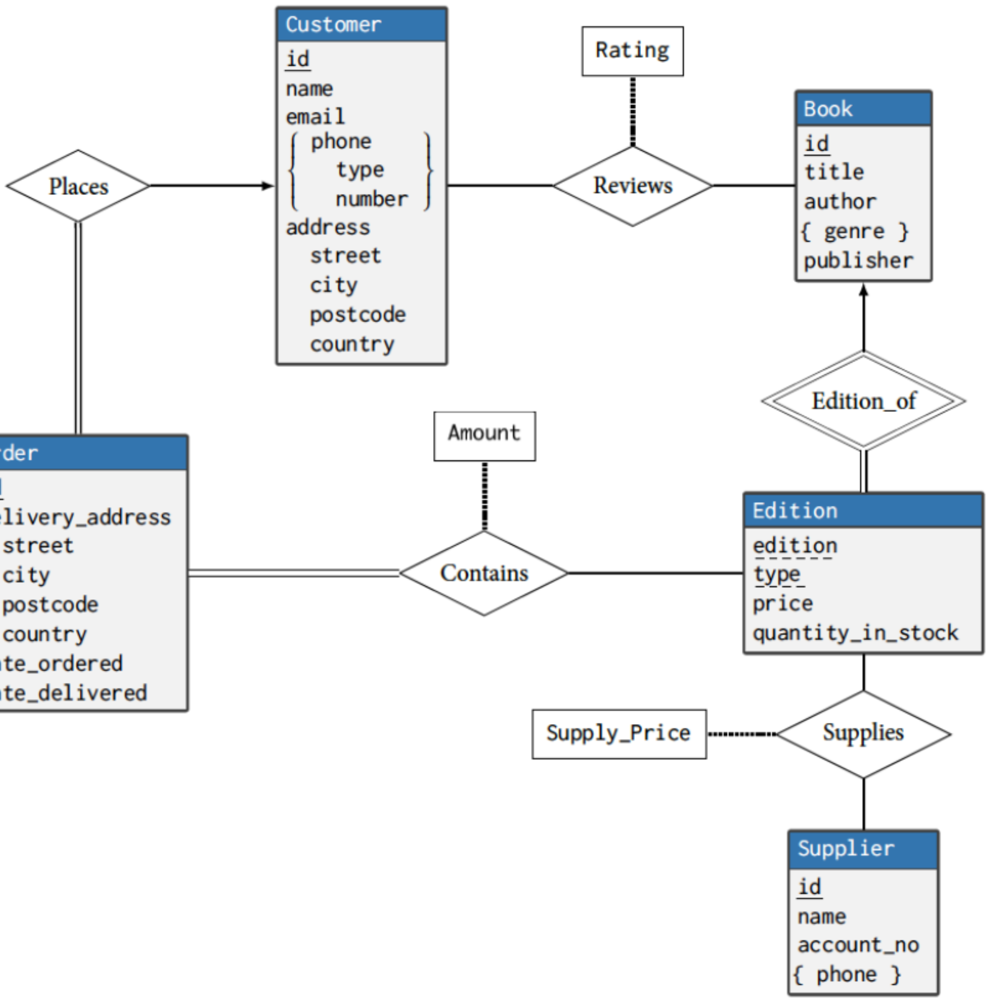


Fig : ER diagram for Bookstore management system

Notation:
- Primary key — underlined
- Discriminator — underlined with dashes
- Defining relationship — doubled diamond
- Cardinality of one — arrow
- Total participation — doubled line
- Multivalued attributes — in curly braces
- Relationship attributes — connected to the relation with a dashed line

The following are the identified entities

1. **customer** (customer_id, name, email, street, city, postcode, country)
   Primary key - customer_id

   The customer table is a strong entity set with fields directly derived from the ER diagram. *Customer_id* is identified as the primary key of this table.

   - **customer_id** – **CHAR(10)** - This is a unique identifier of customer table. It is stored as a CHAR type which can contain letters, numbers, and special characters. The size parameter specifies the column length in characters – in case of customer, it is defined to be as length 10.
   - **name** – **VARCHAR(100) NOT NULL** - The name of the customer is stored in this field. Variable length character length with maximum of 100 characters is used to store customer name. It is assumed to be a field with no null values, which means that all customer needs a name attribute.
   - **email – VARCHAR UNIQUE –** The email attribute of customer is of the type variable length character and is defined to be **UNIQUE** Integrity constraint field, stating that email cannot be repeated and needs to be unique in all tuples and needs to be a mandatory field.
   - **street – VARCHAR(100) –** Street information of customer is stored as variable length character with maximum character length of 100.
   - **city – VARCHAR(100) –** City information of customer is stored as variable length character with maximum character length of 100.
   - **postcode – VARCHAR(10) –** Postcode information of customer is stored as variable length character with maximum character length of 10.
   - **country – VARCHAR(50) –** Country information of customer is stored as variable length character with maximum character length of 50.

2. **customer_phone** (*Customer_Id*, phone_type, phone_number)
   Primary key – *customer_id, phone_number*
   Foreign Key – *customer_id*

   A customer can have multiple phone numbers; therefore, a separate entity needs to be created to store the combination of phone number against customer. Customer to customer_phone is a one-to-many (1:N or N:1) relationship, hence the primary key of the customer i.e., *customer_id* (Foreign key) is added to phone as an attribute in the “many” relationship(*customer_phone)*.

   - **customer_id** – **CHAR(10)** - This is a unique identifier of customer_phone table along with phone_number. It is stored as a CHAR type which can contain letters, numbers, and special characters. The size parameter specifies the column length in characters – in case of customer, it is defined to be as length 10. *customer_id* is a foreign key reference in this table.
   - **phone_type – VARCHAR(10) –** Phone type information such as ‘mobile’, ‘landline’ …, is stored as variable length character with maximum character length of 10.
   - **phone_number – VARCHAR(20) –** Phone number information of customer is stored as variable length character with maximum character length of 20. This allows phone number along with dialling code to be stored. phone_number is a primary key along with *customer_id* , therefore it cannot be nullable.

3. **order_info** (order_id, street, city, postcode, country, date_ordered, date_delivered, *customer_id*)
PRIMARY KEY – *order_id*
FOREIGN KEY – *customer_id*

Since the order is related to customer in a one – to – many manners and is having a total participation in the relationship ‘places’ along with customer, we add the customer id in each order as foreign key reference.

- **order_id** – **CHAR(10)** - This is a unique identifier of order_info table. It is stored as a CHAR type which can contain letters, numbers, and special characters. The size parameter specifies the column length in characters – in case of order, it is defined to be as length 10.
- **customer_id** – **CHAR(10)** - This is a foreign key reference from customer table. It is stored as a CHAR type which can contain letters, numbers, and special characters. The size parameter specifies the column length in characters – in case of customer, it is defined to be as length 10.
- **street – VARCHAR(100) NOT NULL–** Street information of order is stored as variable length character with maximum character length of 100 The NOT NULL integrity constraint prevents a column from having a NULL value, as the order needs to have street value.
- **city – VARCHAR(100) NOT NULL –** City information of order is stored as variable length character with maximum character length of 100. The NOT NULL integrity constraint prevents a column from having a NULL value, as the order needs to have city value.
- **postcode – VARCHAR(10) NOT NULL –** Postcode information of order is stored as variable length character with maximum character length of 10. The NOT NULL integrity constraint prevents a column from having a NULL value, as the order needs to have postcode value.
- **country – VARCHAR(50) NOT NULL –** Country information of order is stored as variable length character with maximum character length of 50. The NOT NULL integrity constraint prevents a column from having a NULL value, as the order needs to have country value.
- **date_ordered – TEXT –** SQLite does not have a storage class set aside for storing dates and/or times. Instead, the built-in Date and Time Functions of SQLite can store dates and times as TEXT as ISO8601 strings ("YYYY-MM-DD HH:MM:SS.SSS").
- **date_delivered – TEXT –** Both ordered and delivered information is stored as ISO8601 text format.

4. **order_contains** (order_Id, book_id, edition, edition_type, amount)
PRIMARY KEY – *order_id,book_id,edition,edition_type*
FOREIGN KEY – *order_id,book_id,edition,edition_type*

order_contains is a relationship between order_info and edition, a many to many relationships with one total participation from *order_info* table. Therefore, a separate table is to be created with all the primary keys of the connecting table as part of the attributes.

- **order_id** – **CHAR(10)** - This is a unique identifier of order_info table used a foreign key in this entity. It is stored as a CHAR type which can contain letters, numbers, and special characters. The size parameter specifies the column length in characters – in case of order, it is defined to be as length 10.
- **book_id** – **CHAR(10) –** This is a foreign key reference to **book** table referencing to primary key of that table. It is stored as a CHAR type which can contain letters, numbers, and special characters.
- **edition – INTEGER –** book edition is stored in order_contains as INTEGER type which indicates the book edition.
- **edition_type – VARCHAR –** Indicates the type of book such as *‘paperback’, ‘audiobook’* or *‘hardcover’.*

- **amount – INTEGER CHECK (amount >=0) –** Indicates the number of books of the given edition type ,that belongs to the given order. Attribute domain constraint CHECK condition amount >=0 checks if the entered amount value is positive.

5. **book** (book_id, title, author, publisher)
   PRIMARY KEY - *book_Id*

   - **book_id** – **CHAR(10) –** This is a primary key reference to **book**. It is stored as a CHAR type which can contain letters, numbers, and special characters and has a length of 10.
   - **title – VARCHAR(50) –** Book title is stored as variable length character with maximum character length of 50.
   - **author – VARCHAR –** Author information of book is stored as variable length character with maximum character length provided by VARCHAR.
   - **publisher – TEXT –** Publisher information of book is stored as TEXT.

6. **book_genre** (book_id,genre)
   PRIMARY KEY – *book_id*
   Since genre is a multi-valued attribute, it needs to be in a separate relation with books, with book id as a foreign key reference.
   - **book_id** – **CHAR(10) –** This is a foreign key reference to **book**. It is stored as a CHAR type which can contain letters, numbers, and special characters and has a length of 10.
   - **genre – VARCHAR(255) –** Book genre is stored as variable length character with maximum character length of 255.

7. **edition** (edition , edition_type, book_id, price, quantity_in_stock)
   PRIMARY KEY – edition, edition_type, book_id
   FOREIGN KEY – *book_id*

   Edition is a weak entity. Weak Entity table does not have a primary key but the primary key of a weak entity set (**edition**) is formed by the primary key of the strong entity on which the weak entity set existence dependent (in this case **book** table that it is related to), plus the weak entity set's discriminator set (*edition,edition_type)* and needs to have the corresponding entity's primary key as part of the discriminators.

   - **book_id** – **CHAR(10) –** This is a foreign key reference to **book**. It is stored as a CHAR type which can contain letters, numbers, and special characters and has a length of 10.
   - **edition – INTEGER CHECK (edition >=0) –** book edition is stored as INTEGER type. Attribute domain constraint CHECK condition edition >=0 checks if the entered amount value is positive.
   - **edition_type – VARCHAR –** Indicates what type of book it is. Attribute domain constraint limits the edition_type to have only these following values –
     'paperback', 'audiobook' or *'hardcover'.*
   - **price – NUMERIC (8,2) CHECK (price >= 0) –** The cost at which a particular edition of book is to be sold at in the bookstore. It is stored as a numeric data type defined in terms of its precision (total number of digits) which is 8 (assuming the maximum cost of book doesn't exceed this limit) and scale (number of digits to the right of the decimal point) which is 2 – indicating that only 2 values after decimal point can be stored.
   - **quantity_in_stock – INTEGER –** The total quantity of the books of edition in the bookstore is stored in this attribute.

8. **supplier** (supplier_id, name, account_number)
PRIMARY KEY - supplier_id

This is a strong entity that stores supplier related information.

- **supplier_id** – **CHAR(10) –** This is a primary key of the entity supplier. It is stored as a CHAR type which can contain letters, numbers, and special characters and has a length of 10.
- **name – VARCHAR –** Indicates the name of the supplier that provides book of different edition to bookstore, it is stored as variable length characters type.
- **account_number – VARCHAR(50) NOT NULL UNIQUE –** Stores the account information of a supplier. It is of the type variable length characters of maximum length of 50 characters. The NOT NULL integrity constraint prevents a column from having a NULL value, as the supplier needs to have account number to conduct transactions. Attribute domain constraint limits the account_number to be UNIQUE and not repeatable for each supplier.

9. **supplier_phone** (supplier_id, phone)
PRIMARY KEY – *supplier_id, phone* (composite)

Since supplier phone is a multi-valued attribute, it needs to be in a separate relation with supplier, with supplier_id as a foreign key reference.

- **supplier_id** – **CHAR(10) –** This is a foreign key reference to **supplier**. It is stored as a CHAR type which can contain letters, numbers, and special characters and has a length of 10.
- **Phone – VARCHAR(20) –** Phone number information of supplier is stored as variable length character with maximum character length of 20. This allows phone number along with dialling code to be stored. *phone* is a primary key along with *supplier_id*, therefore it needs to be unique as a combination with supplier_id.

10. **supply** (supplier_id, edition , edition_type, book_id, price)
PRIMARY KEY – *supplier_id, edition, edition_type, book_id*

supply is a relationship between supplier and edition, a many-to-many relationships with no total participation between connecting entities. Therefore, a separate table is to be created with all the primary keys of the connecting table as part of the attributes. This entity represents the result of a natural join on the relations corresponding to edition and supplier.

- **supplier_id** – **CHAR(10) –** This is a foreign key reference to **supplier**. It is stored as a CHAR type which can contain letters, numbers, and special characters and has a length of 10.
- **book_id** – **CHAR(10) –** This is a foreign key reference to **edition** table referencing to primary key of that table. It is stored as a CHAR type which can contain letters, numbers, and special characters with length of 10. It is a foreign key reference to **edition** table.
- **edition – INTEGER –** book edition is stored in **supply** as INTEGER type which indicates the book edition. It is a foreign key reference to **edition** table.
- **edition_type – VARCHAR –** Indicates the type of book such as *'paperback', 'audiobook'* or *'hardcover'*. It is a foreign key reference to **edition** table.
- **price – REAL (price >= 0) –** used to store the supply price of the edition provided by the supplier. Attribute domain constraint limits the price to be a real positive value.

11. **review** (book_id, customer_id, rating)
PRIMARY KEY – *book_id, customer_id*
FOREIGN KEY – *book_id, customer_id*

This is a many to many relationships but without total participation, therefore it also needs a separate entity named review. Composite key is present – book_id, customer_id.

The *book_id* and *customer_id* of the connected strong entities is used as a composite key in this entity.

- **book_id** – **CHAR(10) –** This is a foreign key reference to **book**. It is stored as a CHAR type which can contain letters, numbers, and special characters and has a length of 10.
- **customer_id** – **CHAR(10)** - This is a foreign key reference from **customer** table. It is stored as a CHAR type which can contain letters, numbers, and special characters. The size parameter specifies the column length in characters – in case of customer, it is defined to be as length 10.
- **rating – INTEGER CHECK (rating BETWEEN 1 AND 5) –** Refers to the rating provided by a customer for a book. Attribute domain constraint CHECK (rating BETWEEN 1 AND 5) limits the rating to be within the range of 1 to 5. Hence other values cannot be stored in that attribute.

Task 2: SQL Data Definition

The Integrity constraint for the following tables that have foreign key references is defined as follows

- customer_phone (customer_Id, phone_type, phone_number)

  ON DELETE CASCADE and ON UPDATE CASCADE constraint is used to delete or update the rows from the customer_phone table automatically, when the tuple from the customer table is deleted or updated, as there cannot exists a phone number without a candidate information.

- book_genre (book_id,genre)
  Integrity constraints on book_id foreign key

  ON DELETE CASCADE and ON UPDATE CASCADE constraint is used to delete or update the rows from the book_genre table automatically, when the tuple from the book table is deleted or updated, as there cannot exists a genre without a book information.

- edition (edition , edition_type, book_id, price, quantity_in_stock)
  Integrity constraints on book_id foreign key

  ON DELETE CASCADE and ON UPDATE CASCADE constraint is used to delete or update the rows from the edition table automatically, when the tuple from the book table is deleted or updated, as there cannot exists an edition without a book information.

- order_info (order_id, street, city, postcode, country, date_ordered, date_delivered, customer_id)
  Integrity constraints on customer_id foreign key

  - ON DELETE SET NULL constraint is set to prevent the tuple in order getting deleted when a customer information is deleted. Instead, the corresponding tuple in order_info gets updated with NULL in customer_id attribute for deleted customer.
  - ON UPDATE CASCADE constraint is used to update the rows from the order_info table automatically when the tuple from the customer table is updated.

- order_contains (order_Id, book_id, edition, edition_type, amount)

  ON DELETE CASCADE constraint is used for all the foreign key references, to delete the rows from the order contain table automatically, when the rows from the parent table are deleted. For example, when an order is deleted in the order_info table, the corresponding order contain tuple will not make sense, hence it gets deleted. Similarly ON UPDATE CASCADE is used to update the tuple in order_contain whenever an update is made in the referenced tables attributes.

- supply (supplier_id, edition , edition_type, book_id, price)

  ON DELETE CASCADE constraint is used for all the foreign key references, to delete the rows from the supply table automatically, when the rows from the parent table are deleted. For example, when a supplier information is deleted in the supplier table, the corresponding supply containing tuple will not make sense, hence it gets deleted. Similarly ON UPDATE CASCADE is used to update the tuple in supply whenever an update is made in the referenced tables attributes.

- supplier_phone (supplier_id, phone)

  ON DELETE CASCADE and ON UPDATE CASCADE constraint is used to delete or update the rows from the supplier_phone table automatically, when the tuple from the supplier table is deleted or updated, as there cannot exists an phone number without a supplier information.

- review (book_id, customer_id, rating)

  ON DELETE CASCADE constraint is used for all the foreign key references, to delete the rows from the review table automatically, when the rows from the parent table are deleted. For example, when a book information is deleted in the **book** table, the corresponding review table containing tuple will not make sense, hence it gets deleted. Similarly ON UPDATE CASCADE is used to update the tuple in review whenever an update is made in the referenced tables attributes.

TASK 3: SQL Data Manipulation

The bookstore.sql script is extended to return response for the following queries

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

```
SELECT b.book_id ,b.title ,b.author ,b.publisher, bg.genre FROM book b
JOIN book_genre bg on bg.book_id = b.book_id
         WHERE bg.genre = "Science Fiction" AND b.publisher = 'Ultimate Books';
```

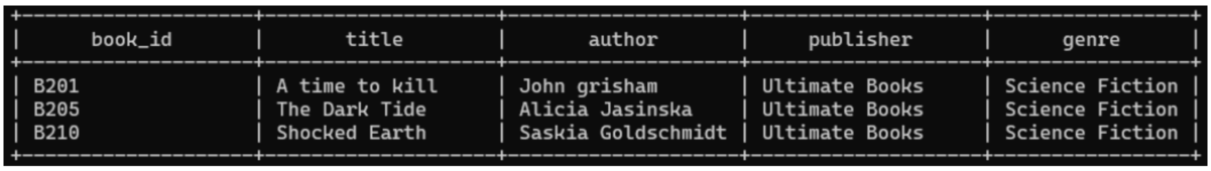
Consider the above query lists the book information such as *Book Id, Book Title, Author, publisher* and *Genre* of the books.

- The **SELECT** clause is used to list the attributes desired in the result of a query from the table **book**.
- The **FROM** clause is a list of the relations between **book** and **book_genre** to be accessed in the evaluation of the query. The from clause defines a Cartesian product of the relations and contains all the attributes of **book** and **book_genre.**
- The **WHERE** clause is a predicate involving attributes of the relation in the from clause and defines the conditions as per the given query requirement, in the case of this query the result must contain only books that belong to genre – ‘*Science Fiction’* from the publisher
  ‘Ultimate books’.
- The **JOIN** operation is used to relate **book** and **book_genre** tables by matching ‘*book_id’* . The Join only considers those pairs of tuples that are present from **book** and the tuple from **book_genre** that have the same ‘*book_id’* value as the common attribute.

The above query can be simplified using **NATURAL JOIN** to avoid the danger of equating attributes erroneously, we can use the SQL feature Natural Join construct that allows us to specify exactly which columns should be equated.
This query can be written more concisely using the natural-join operation in as:

```
SELECT b.book_id ,b.title ,b.author ,b.publisher, bg.genre FROM book b
NATURAL JOIN book_genre bg
         WHERE bg.genre = "Science Fiction" AND b.publisher = 'Ultimate Books';
```

The result of the above queries for the inserted values is :



```
+-------------------+-------------------+-------------------+-------------------+-----------------+
|      book_id      |       title       |      author       |     publisher     |      genre      |
+-------------------+-------------------+-------------------+-------------------+-----------------+
| B201              | A time to kill    | John grisham      | Ultimate Books    | Science Fiction |
| B205              | The Dark Tide     | Alicia Jasinska   | Ultimate Books    | Science Fiction |
| B210              | Shocked Earth     | Saskia Goldschmidt| Ultimate Books    | Science Fiction |
+-------------------+-------------------+-------------------+-------------------+-----------------+
```

The result of the join allows us to select the column *‘genre’* from book_genre table and display it along with the attributes of the books.

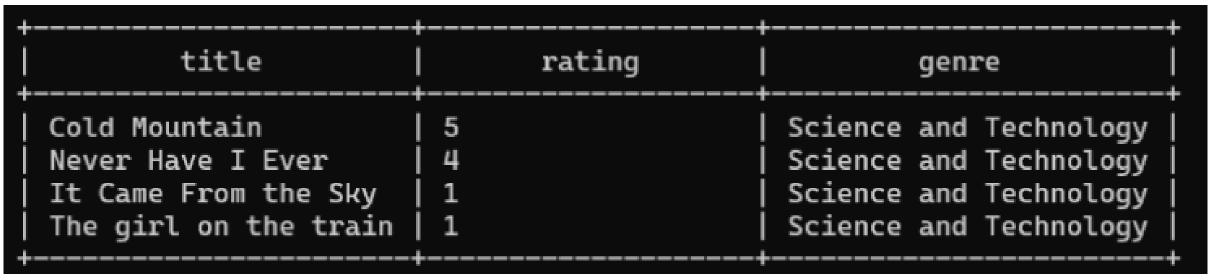
QUERY 2 : List titles and ratings of all books in the "Science and Technology" genre, ordered first by rating (Top rated first), and then by the title.

```
SELECT b.title,r.rating,bg.genre FROM book b
NATURAL JOIN review r
NATURAL JOIN book_genre bg
       WHERE bg.genre  =  'Science and Technology'
             ORDER BY rating DESC,title;
```

The above query lists the book information such as *Book Title* and *Rating* of the books.

- The **SELECT** clause is used to list the attributes desired in the result of a query from the table **book**, **review** and **book_genre.** Aliasing has been used to assign 'b' to refer to books, 'r' to refer to the attributes of review and 'bg' refers to the table book_genre. This makes it convenient to indicate the table from which the columns need to be picked in SELECT statement.
- The **FROM** clause is a list of the relations between **book** , **review** and **book_genre** to be accessed in the evaluation of the query. The from clause defines a Cartesian product of the relations and contains all the attributes of **book, review** and **book_genre** after the given conditions has been satisfied**.**
- The **WHERE** clause is a predicate involving attributes of the relation in the from clause and defines the conditions as per the given query requirement, in the case of this query the result must contain only books that belong to genre – '*Science and Technology'* .
- The **NATURAL JOIN** operation is used to relate **book** with **reivew** and **book_genre** tables by matching '*book_id'* . The first Natural Join only considers those pairs of tuples that are present from **book** and the tuple from **review** and in the second Natural Join statement only considers those pairs of tuples that are present from **book** and the tuple from **book_genre** that have the same '*book_id'* value as the common attribute. The matching column need not be specified as Natural join performs that match without explicit definition.
- The **ORDER BY** clause causes the tuples in the result to appear in a sorted order as specified in the query. The ordering needs to be performed on **rating** attribute where the top-rated books needs to be displayed on the top of the result and the successive ones in a descending]order. We use **DESC** to indicate that the ordering needs to be performed in a descending manner. Furthermore, Order By also accept multiple attributes. In the case of this query, we need to sort the tuples in the result as per the title. Hence the second attribute given to this clause is *title*.

The result of the above queries for the inserted values is :



```
+-----------------------+--------------------+------------------------+
|         title         |       rating       |         genre          |
+-----------------------+--------------------+------------------------+
| Cold Mountain         | 5                  | Science and Technology |
| Never Have I Ever     | 4                  | Science and Technology |
| It Came From the Sky  | 1                  | Science and Technology |
| The girl on the train | 1                  | Science and Technology |
+-----------------------+--------------------+------------------------+
```

The result of the join allows us to select the column *'rating'* from **review** and *'genre'* from **book_genre** table and display it along with the attribute – *'title'* of the table books.

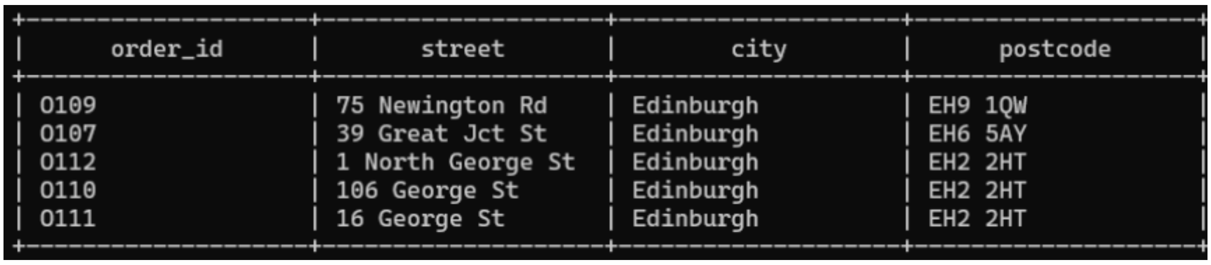
QUERY 3: List all orders placed by customers with customer address in the city of Edinburgh, since 2020, in chronological order, latest first.

```
SELECT * FROM order_info oi
      WHERE oi.city = "Edinburgh" AND oi.date_ordered >= '2020'
            ORDER BY oi.date_ordered DESC;
```

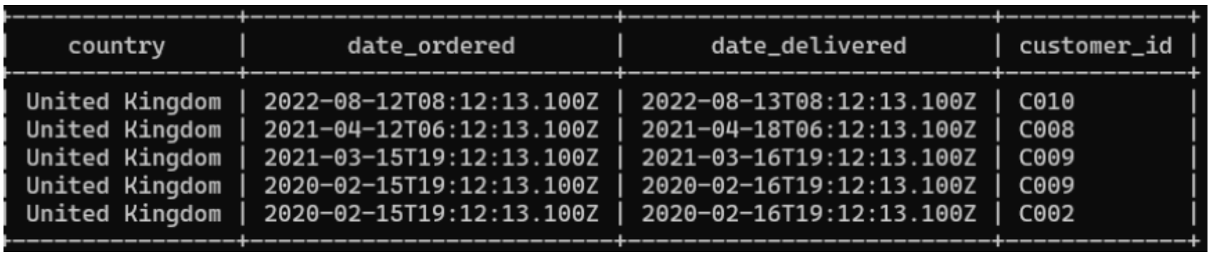
The above query lists the order information such as *order_id, street, city, postcode, country , date_ordered, date_delievered,* and *customer_id* for the order.

- The **SELECT** clause is used to list the attributes desired in the result of a query from the table **order_info.** Aliasing has been used to assign 'oi' to refer to order_info. This makes it convenient to indicate attributes in table from which the columns need to be picked in SELECT statement.
- The **FROM** clause is a list of the tuples from table **order_info** that needs to be accessed in the evaluation of the query.
- The **WHERE** clause is a predicate involving attributes of the relation in the from clause and defines the conditions as per the given query requirement, in the case of this query the result must contain only orders that has been delivered to the city *'Edinburgh'* and the date of delivery must be since the year 2020 i.e *date_delivered >= '2020'.*
- The **ORDER BY** clause causes the tuples in the result to appear in a sorted order as specified in the query. The ordering needs to be performed on **rating** attribute where the top-rated books needs to be displayed on the top of the result and the successive ones in a descending]order. We use **DESC** to indicate that the ordering needs to be performed in a descending manner. Furthermore, Order By also accept multiple attributes. In the case of this query, we need to sort the tuples in the result as per the title. Hence the second attribute given to this clause is *title*.

The result of the above queries for the inserted values is :



```
+--------------------+--------------------+--------------------+--------------------+
|      order_id      |       street       |        city        |      postcode      |
+--------------------+--------------------+--------------------+--------------------+
| 0109               | 75 Newington Rd    | Edinburgh          | EH9 1QW            |
| 0107               | 39 Great Jct St    | Edinburgh          | EH6 5AY            |
| 0112               | 1 North George St  | Edinburgh          | EH2 2HT            |
| 0110               | 106 George St      | Edinburgh          | EH2 2HT            |
| 0111               | 16 George St       | Edinburgh          | EH2 2HT            |
+--------------------+--------------------+--------------------+--------------------+
```



```
----------------+--------------------------+--------------------------+-------------+
     country    |       date_ordered       |      date_delivered      | customer_id |
----------------+--------------------------+--------------------------+-------------+
 United Kingdom | 2022-08-12T08:12:13.100Z | 2022-08-13T08:12:13.100Z | C010        |
 United Kingdom | 2021-04-12T06:12:13.100Z | 2021-04-18T06:12:13.100Z | C008        |
 United Kingdom | 2021-03-15T19:12:13.100Z | 2021-03-16T19:12:13.100Z | C009        |
 United Kingdom | 2020-02-15T19:12:13.100Z | 2020-02-16T19:12:13.100Z | C009        |
 United Kingdom | 2020-02-15T19:12:13.100Z | 2020-02-16T19:12:13.100Z | C002        |
----------------+--------------------------+--------------------------+-------------+
```

*Both the tables represent one continuous result as a series of column displayed one next to another and not broken into two separate results.

The result of the join allows us to select the column *'rating'* from **review** and *'genre'* from **book_genre** table and display it along with the attribute – *'title'* of the table books.

QUERY 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.

```
SELECT e.*,b.title, MIN(s1.price) AS minimum_supplier_price FROM edition e
      LEFT JOIN supply s1 ON s1.edition = e.edition
      AND s1.edition_type = e.edition_type AND s1.book_id = e.book_id
      NATURAL JOIN supplier s2
      NATURAL JOIN book b -- This is just used to show books instead of just id
            WHERE e.quantity_in_stock < 5
            GROUP BY e.edition,e.edition_type,e.book_id;
```

The above query lists the book edition information such as *edition, edition_type, price, quantity_in_stock, and book_id* from **edition** table, along with matching *title* from table **book,** and supplier information such as *supplier_name, supplier_account_number and minimum supply price* for matching edition from table **supplier.**

- The **SELECT** clause is used to list the attributes desired in the result of a query from the table **edition, book, supply** and **supplier.** Aliasing has been used to assign 'e to refer to edition, 'b' refers to book, 's1' refers to supply and 's2' refers to supplier table. This makes it convenient to indicate attributes in table from which the columns need to be picked in SELECT statement.

  The book table has been joined to get the book title and to be more informative about on the response from the query. In case of simplicity this might not be required and can be safely discarded to reduce the time taken to execute the query.

- The **FROM** clause is a list of the tuples from table **edition, book, supply** and **supplier** that needs to be accessed in the evaluation of the query.
- The **WHERE** clause is a predicate involving attributes of the relation in the from clause and defines the conditions as per the given query requirement, in the case of this query the result must contain only editions that are less than 5 in number in stock.
- The **MIN** aggregation function is used to get the minimum supply price for a particular edition. The aggregate function is applied to a group of sets of tuples which is specified in SQL using the **GROUP BY** clause. The *edition, edition_type,book_id* attributes in the group by clause are used to form groups. Tuples with same value on all attributes in the group by are place in one group along with the minimum aggregation function used to get the lowest supply price in that group.

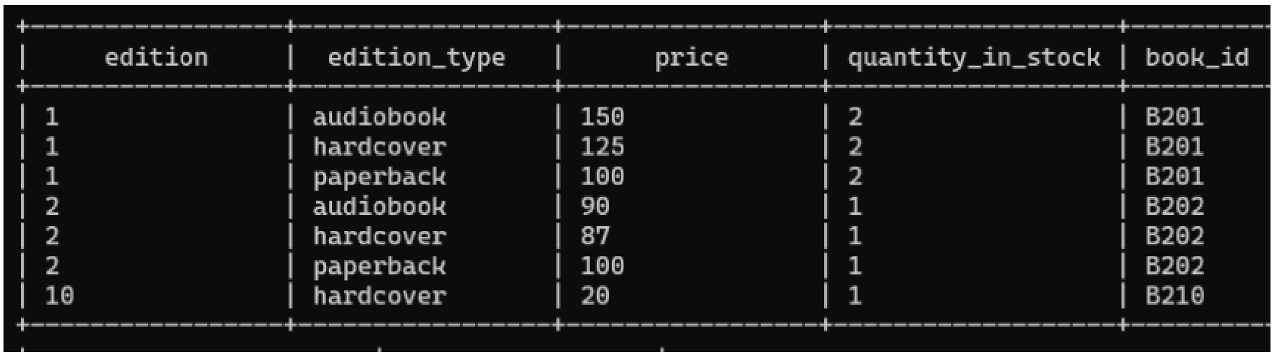
- The **NATURAL** and **LEFT JOIN** operation is used to relate **edition** with **supply, supplier and book** tables by matching '*edition', 'edition_type', 'book_id'* across these tables .

  The first statement - LEFT JOIN between **edition** and **supply** returns all records from the left table (edition), and the matching records from the right table (supply) as book edition is our primary focus for the query, LEFT JOIN suits the needs to include all edition and matching values from the supply table. If there is no match then the result is 0 records from the right side.

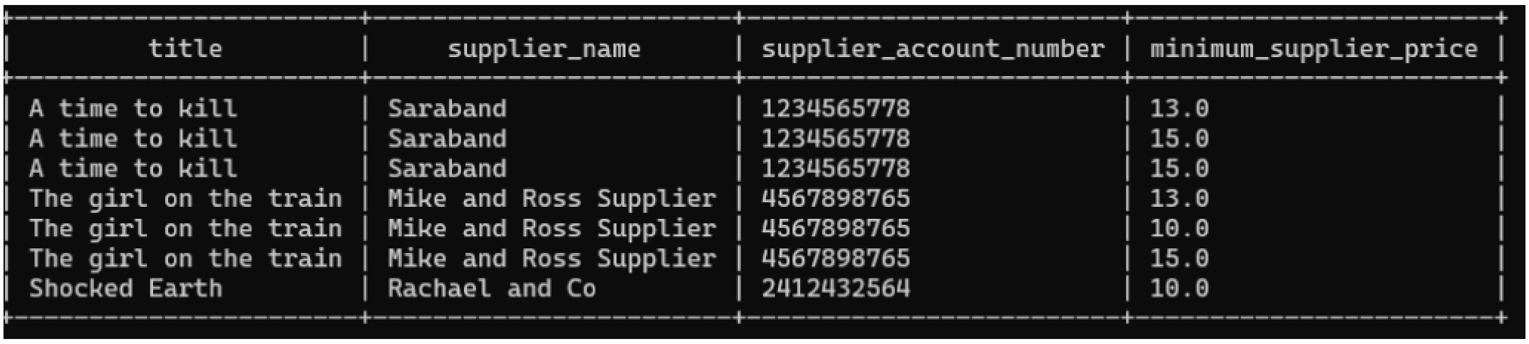
  The second statement - NATURAL JOIN joins the edition, supply and supplier with matching supplier_id from supply and considers those pairs of tuples that are present from **edition, supply** and the tuple from **supplier.**

  The third statement - NATURAL JOIN joins the **edition** and **book** and considers those pairs of tuples that are present from **edition and book.** This was added to get the title of the books and be more informative about the result rather than displaying book_id.

The result of the above queries for the inserted values is :



```
+-----------------+-----------------+-----------------+------------------+----------
|     edition     |  edition_type   |      price      | quantity_in_stock | book_id
+-----------------+-----------------+-----------------+------------------+----------
| 1               | audiobook       | 150             | 2                | B201
| 1               | hardcover       | 125             | 2                | B201
| 1               | paperback       | 100             | 2                | B201
| 2               | audiobook       | 90              | 1                | B202
| 2               | hardcover       | 87              | 1                | B202
| 2               | paperback       | 100             | 1                | B202
| 10              | hardcover       | 20              | 1                | B210
+-----------------+-----------------+-----------------+------------------+----------
```



```
+----------------------+-----------------------+-------------------------+-------------------------+
|        title         |     supplier_name     | supplier_account_number | minimum_supplier_price |
+----------------------+-----------------------+-------------------------+-------------------------+
| A time to kill       | Saraband              | 1234565778              | 13.0                    |
| A time to kill       | Saraband              | 1234565778              | 15.0                    |
| A time to kill       | Saraband              | 1234565778              | 15.0                    |
| The girl on the train | Mike and Ross Supplier | 4567898765             | 13.0                    |
| The girl on the train | Mike and Ross Supplier | 4567898765             | 10.0                    |
| The girl on the train | Mike and Ross Supplier | 4567898765             | 15.0                    |
| Shocked Earth        | Rachael and Co        | 2412432564              | 10.0                    |
+----------------------+-----------------------+-------------------------+-------------------------+
```

*Both the tables represent one continuous result as a series of column displayed one next to another and not broken into two separate results.

The result of the join allows us to select the column *'edition', 'edition_type','book_id','price', 'qunatity_in_stock'* from **edition** , *'title'* from **book,** and '*supplier_name', 'supplier_account_number', and 'minimum_supplier_price'* from supplier table.

_Bo

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

```
SELECT DISTINCT publisher,MAX(total_value) FROM (

SELECT DISTINCT b.publisher, SUM(amount*price) AS total_value FROM order_info oi
      NATURAL JOIN order_contain oc
      NATURAL JOIN edition e
      NATURAL JOIN book b
            WHERE oi.date_delivered >= '2020' AND e.edition_type = 'audiobook'
            GROUP BY b.publisher

UNION

SELECT DISTINCT publisher, 0 AS total_value FROM book)
      GROUP BY publisher;
```

The above query lists the total sales of all audiobooks for each publisher in the book table.

The query can be broken down into 3 steps:

1. Query to get all audiobook sales from year 2020

```
SELECT DISTINCT b.publisher, SUM(amount*price) AS total_value FROM order_info oi
      NATURAL JOIN order_contain oc
      NATURAL JOIN edition e
      NATURAL JOIN book b
            WHERE oi.date_delivered >= '2020' AND e.edition_type 'audiobook'
            GROUP BY b.publisher
```

- The **SELECT** clause is used to list the attributes desired in the result of a query from the table **order_info.** Aliasing has been used to assign 'oi' to refer to order_info, 'e' refers to edition, 'b' refers to book. This makes it convenient to indicate attributes in table from which the columns need to be picked in SELECT statement.

  The total sales values for all audiobooks are calculated by adding all the sale cost i.e. amount * price The number of books (*amount*) that a particular order contains is taken from **order_contain** table and the price of each edition is taken from **edition** table, which are then multiplied to get the sale price.

- The **FROM** clause is a list of the tuples from table **order_info** that needs to be accessed in the evaluation of the query.
- The **WHERE** clause is a predicate involving attributes of the relation in the from clause and defines the conditions as per the given query requirement, in the case of this query the result must contain only orders that has been delivered since the year 2020 i.e oi.*date_delivered >= '2020'.* and the book edition must be of the type '*audiobook*' i.e e.edition_type 'audiobook.
- The **NATURAL JOIN** operation is used to relate **order_info** with **order_contain, edition and book** tables by matching '*edition*', '*edition_type*', '*book_id*' across these tables.
- The **GROUP BY** clause causes the tuples in the result to be grouped based on book publisher.

2. Query to get all publisher names from book table

```
SELECT DISTINCT publisher, 0 AS total_value FROM book
```

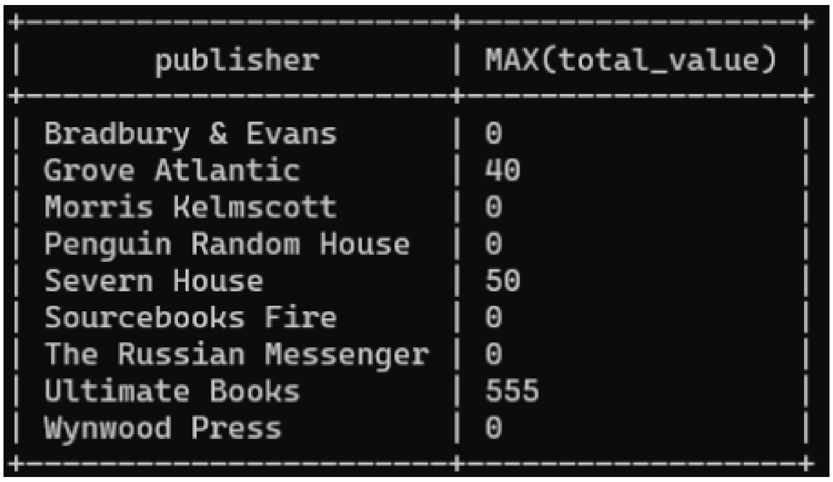
The above SELECT query returns all the publishers that are available on the **book** table.

DISTINCT keyword is used prevent duplicate publisher value to be returned from this query. A new column called *'total_value'* is selected and populated with 0 as default value.

3. Query to get required result from above 2 queries (Outer query
```
SELECT DISTINCT publisher,MAX(total_value) FROM (SELECT T1 UNION SELECT T2)
      GROUP BY publisher;
```

The outer **SELECT** query is used to merge the above 2 queries using **UNION.** The **DISTINCT** clause eliminates duplicates in the *'Publisher'* column. Paired along with **MAX** aggregation gives a value in the *'total_value'* column.

The result of the above queries for the inserted values is :



```
+------------------------+-----------------+
|        publisher       | MAX(total_value) |
+------------------------+-----------------+
| Bradbury & Evans       | 0               |
| Grove Atlantic         | 40              |
| Morris Kelmscott       | 0               |
| Penguin Random House   | 0               |
| Severn House           | 50              |
| Sourcebooks Fire       | 0               |
| The Russian Messenger  | 0               |
| Ultimate Books         | 555             |
| Wynwood Press          | 0               |
+------------------------+-----------------+
```

TASK 3.1

Formulate at least 3 new queries, stating them as precisely as you can in plain English.
Implement them in SQL, and include the SQL code for all the queries, and their output, in the report.

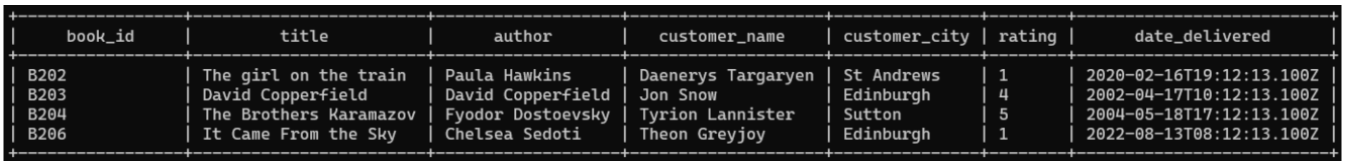
Query 1: Find the customer info (name and city) who has provided rating for a book. List the information only if a customer has previously purchased that book.

```
SELECT b.book_id,b.title,b.author,c.name AS customer_name,c.city as
customer_city,r.rating, oi.date_delivered  FROM review r
       LEFT JOIN book b ON b.book_id = r.book_id
       LEFT JOIN customer c ON r.customer_id = c.customer_id
       LEFT JOIN order_contain oc ON oc.book_id = r.book_id
       LEFT JOIN order_info oi ON (oi.customer_id = r.customer_id)
              WHERE oi.date_delivered NOT NULL
                     GROUP BY c.customer_id;
```

The above query lists the rating of books by the customers who had previously purchased a book and provided a review for it.

- The **SELECT** clause is used to list the attributes desired in the result of a query from the table **customer, order_info, order_contain, review and book.** Aliasing has been used to assign 'oi' to refer to order_info, 'c' refers to customer, 'b' refers to book, 'r' refers to review and 'oc' refers to order_contain. This makes it convenient to indicate attributes in table from which the columns need to be picked in SELECT statement.
- Multiple tables have been joined using LEFT JOIN keeping rating as the primary table for reference and discarding the tuples for successive tables if no match is found.
- The **WHERE** clause is a predicate involving attributes of the relation in the from clause and defines the conditions as per the given query requirement, in the case of this query the result must contain only orders that has been delivered.
- The **GROUP BY** clause causes the tuples in the result to be grouped based on customer_id.

The result of the above queries for the inserted values is :



```
+---------------+------------------------+-------------------+--------------------+---------------+--------+--------------------------+
|    book_id    |         title          |      author       |   customer_name    | customer_city | rating |      date_delivered      |
+---------------+------------------------+-------------------+--------------------+---------------+--------+--------------------------+
| B202          | The girl on the train  | Paula Hawkins     | Daenerys Targaryen | St Andrews    | 1      | 2020-02-16T19:12:13.100Z |
| B203          | David Copperfield      | David Copperfield | Jon Snow           | Edinburgh     | 4      | 2002-04-17T10:12:13.100Z |
| B204          | The Brothers Karamazov | Fyodor Dostoevsky | Tyrion Lannister   | Sutton        | 5      | 2004-05-18T17:12:13.100Z |
| B206          | It Came From the Sky   | Chelsea Sedoti    | Theon Greyjoy      | Edinburgh     | 1      | 2022-08-13T08:12:13.100Z |
+---------------+------------------------+-------------------+--------------------+---------------+--------+--------------------------+
```

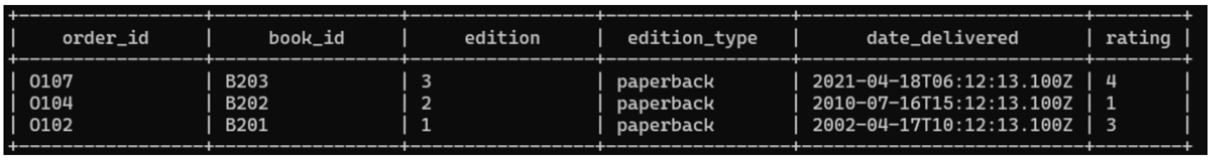
Query 2: Find the total paperback book that was sold from the bookstore in chronological order (LATEST FIRST) and displays its review if available from the customer

```
SELECT o FROM order_contain oc
NATURAL JOIN order_info oi
LEFT JOIN review r ON oc.book_id = r.book_id
       WHERE oc.edition_type = 'paperback'
       GROUP BY oc.book_id
       ORDER BY oi.date_delivered DESC;
```

The above query lists the all the books of the type ‘paperback’ sold from the bookstore along with its rating of books by the customers who had previously purchased a book.

- The **SELECT** clause is used to list the attributes desired in the result of a query from the table **order_info, order_contain, review .** Aliasing has been used to assign ‘oi’ to refer to order_info, ‘r’ refers to review and ‘oc’ refers to order_contain. This makes it convenient to indicate attributes in table from which the columns need to be picked in SELECT statement.
- Multiple tables have been joined using NATURAL and LEFT JOIN keeping order_contain as the primary table for reference and discarding the tuples for successive tables if no match is found.
- The **WHERE** clause is a predicate involving attributes of the relation in the from clause and defines the conditions as per the given query requirement, in the case of this query the result must contain only orders containing the book type ‘paperback’.
- The **ORDER BY date_delivered DESC** clause causes the tuples in the result to be displayed in chronological order. I.e., latest delivered orders come first in the list.

The result of the above queries for the inserted values is :



```
+----------------+----------------+----------------+----------------+--------------------------+--------+
|    order_id    |    book_id     |    edition     |  edition_type  |      date_delivered      | rating |
+----------------+----------------+----------------+----------------+--------------------------+--------+
| O107           | B203           | 3              | paperback      | 2021-04-18T06:12:13.100Z | 4      |
| O104           | B202           | 2              | paperback      | 2010-07-16T15:12:13.100Z | 1      |
| O102           | B201           | 1              | paperback      | 2002-04-17T10:12:13.100Z | 3      |
+----------------+----------------+----------------+----------------+--------------------------+--------+
```

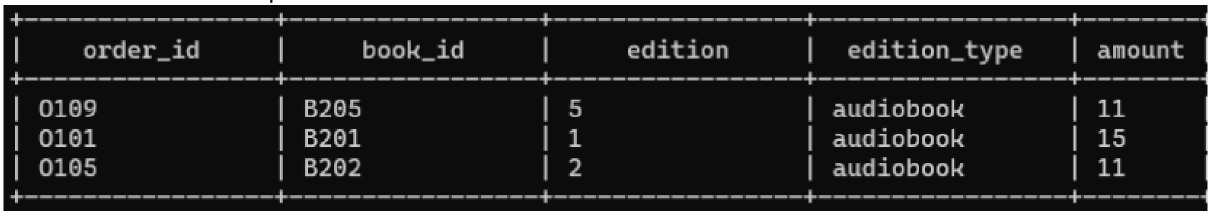
Query 3 : Find the audiobooks book that was delivered within 5 days’ time period , in chronological order (latest first) and the supplier phone number information for these books

```
SELECT oc.*,oi.date_ordered,oi.date_delivered, sp.phone as supplier_phone_number
FROM order_contain oc
NATURAL JOIN order_info oi
NATURAL JOIN supply s
NATURAL JOIN supplier_phone sp
      WHERE oi.date_delivered - oi.date_ordered < 5 AND oc.edition_type =
      'audiobook'
            GROUP BY oc.book_id
            ORDER BY oi.date_delivered DESC;
```

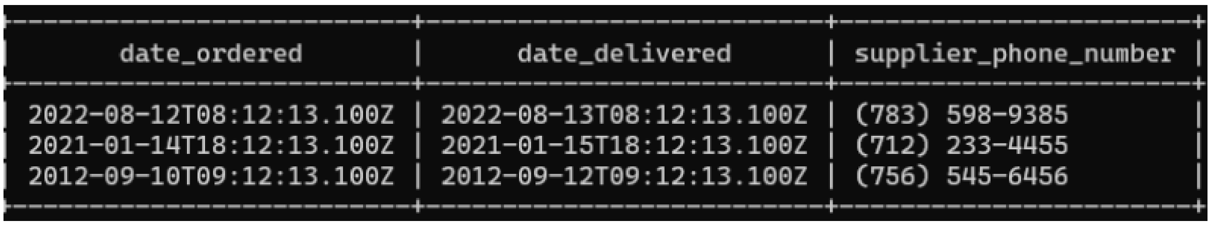
The above query lists the all the audiobooks books that were delivered within a time of 5 days from the date of order and provides the suppliers for those book editions.

- The **SELECT** clause is used to list the attributes desired in the result of a query from the table **order_info, order_contain, supplier and supplier_phone .** Aliasing has been used to assign ‘oi’ to refer to order_info, ‘sp’ refers to supplier phone, ‘s’ refers to supplier and ‘oc’ refers to order_contain. This makes it convenient to indicate attributes in table from which the columns need to be picked in SELECT statement.
- Multiple tables have been joined using NATURAL JOIN keeping order_contain as the primary table for reference and discarding the tuples for successive tables if no match is found.
- The **WHERE** clause is a predicate involving attributes of the relation in the from clause and defines the conditions as per the given query requirement, in the case of this query the result must contain only orders containing the book type ‘audiobook’ and has been delivered within 5 days of ordering.
- The **GROUP BY book_id** lets the resultant table, group all matching books into a single tuple.
- The **ORDER BY date_delivered DESC** clause causes the tuples in the result to be displayed in chronological order. I.e., latest delivered orders come first in the list.

The result of the above queries for the inserted values is :



```
+----------------+----------------+----------------+----------------+--------
|    order_id    |    book_id     |    edition     |  edition_type  | amount |
+----------------+----------------+----------------+----------------+--------
| O109           | B205           | 5              | audiobook      | 11
| O101           | B201           | 1              | audiobook      | 15
| O105           | B202           | 2              | audiobook      | 11
+----------------+----------------+----------------+----------------+--------
```



```
+--------------------------+--------------------------+------------------------+
|       date_ordered       |      date_delivered      | supplier_phone_number |
+--------------------------+--------------------------+------------------------+
| 2022-08-12T08:12:13.100Z | 2022-08-13T08:12:13.100Z | (783) 598-9385         |
| 2021-01-14T18:12:13.100Z | 2021-01-15T18:12:13.100Z | (712) 233-4455         |
| 2012-09-10T09:12:13.100Z | 2012-09-12T09:12:13.100Z | (756) 545-6456         |
+--------------------------+--------------------------+------------------------+
```

TASK 3.2

Create at least two appropriate views, stating what the view represents (in plain English) and including the SQL code used to create the views, and their output, in the report.

VIEW 1: Create a view that holds the information of all audiobooks whose price is greater than 10

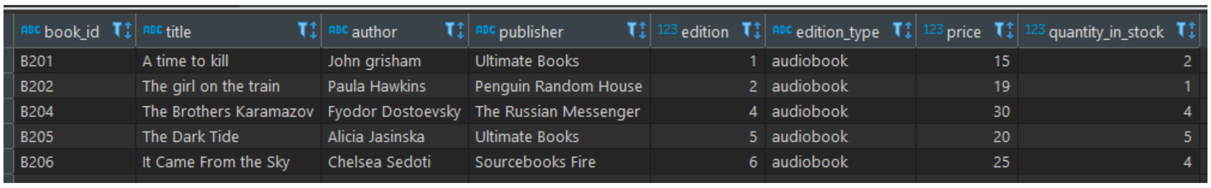
```
CREATE VIEW audiobook_view AS
SELECT * FROM book AS b
NATURAL JOIN edition e
       WHERE e.edition_type = 'audiobook' AND e.price > 10;
```

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. In this case the attributes from tables book and edition whose edition type is 'audiobook' and the prices above 10 is held by the view.

SQL statements to a view and present the data as if the data were coming from one single table.

```
SELECT * FROM audiobook_view;
```

The result of the above select query for the view audiobook_view:



book_id	title	author	publisher	edition	edition_type	price	quantity_in_stock
B201	A time to kill	John grisham	Ultimate Books	1	audiobook	15	2
B202	The girl on the train	Paula Hawkins	Penguin Random House	2	audiobook	19	1
B204	The Brothers Karamazov	Fyodor Dostoevsky	The Russian Messenger	4	audiobook	30	4
B205	The Dark Tide	Alicia Jasinska	Ultimate Books	5	audiobook	20	5
B206	It Came From the Sky	Chelsea Sedoti	Sourcebooks Fire	6	audiobook	25	4

VIEW 2: Create a view that holds all books information and editions for all supplier ordered by the supply price Lowest first

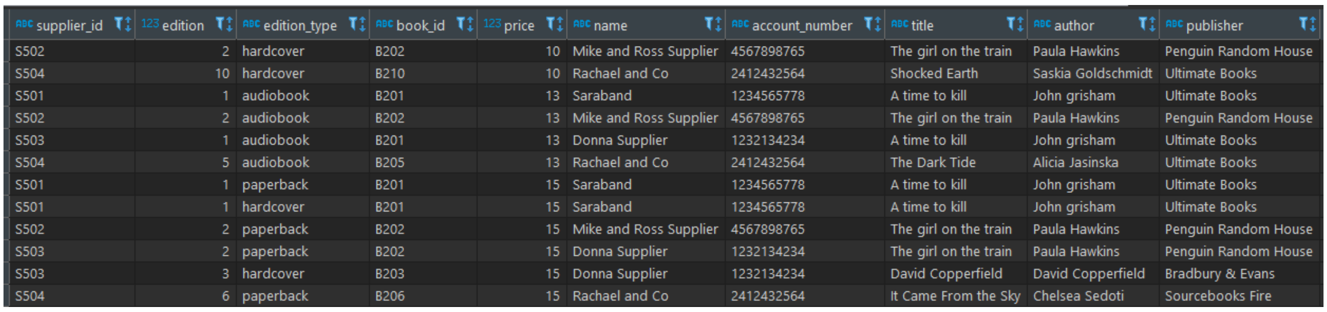
```
CREATE VIEW supplier_book_view AS
SELECT * FROM supply as s1
NATURAL JOIN supplier as s2
NATURAL JOIN book as b
       ORDER BY s1.price;
```

In this case the attributes from tables book, supply and supplier are joined together and ordered by price in ascending order – lowest first.

SQL statements to a view and present the data as if the data were coming from one single table.

```
SELECT * FROM supplier_book_view;
```

The result of the above select query for the view supplier_book_view



supplier_id	edition	edition_type	book_id	price	name	account_number	title	author	publisher
S502	2	hardcover	B202	10	Mike and Ross Supplier	4567898765	The girl on the train	Paula Hawkins	Penguin Random House
S504	10	hardcover	B210	10	Rachael and Co	2412432564	Shocked Earth	Saskia Goldschmidt	Ultimate Books
S501	1	audiobook	B201	13	Saraband	1234565778	A time to kill	John grisham	Ultimate Books
S502	2	audiobook	B202	13	Mike and Ross Supplier	4567898765	The girl on the train	Paula Hawkins	Penguin Random House
S503	1	audiobook	B201	13	Donna Supplier	1232134234	A time to kill	John grisham	Ultimate Books
S504	5	audiobook	B205	13	Rachael and Co	2412432564	The Dark Tide	Alicia Jasinska	Ultimate Books
S501	1	paperback	B201	15	Saraband	1234565778	A time to kill	John grisham	Ultimate Books
S501	1	hardcover	B201	15	Saraband	1234565778	A time to kill	John grisham	Ultimate Books
S502	2	paperback	B202	15	Mike and Ross Supplier	4567898765	The girl on the train	Paula Hawkins	Penguin Random House
S503	2	paperback	B202	15	Donna Supplier	1232134234	The girl on the train	Paula Hawkins	Penguin Random House
S503	3	hardcover	B203	15	Donna Supplier	1232134234	David Copperfield	David Copperfield	Bradbury & Evans
S504	6	paperback	B206	15	Rachael and Co	2412432564	It Came From the Sky	Chelsea Sedoti	Sourcebooks Fire

Short reflective session

• The designing process made me understand the complexities that arise while converting the ER model to relational schema. I was able to come up with some queries that helps to understand the system flow better.

• Without a good set of queries, a clean schema cannot be built. As lot of refactoring was based out of these queries that could be formed to link the tables. The mapping, standardization/generalization concepts were possible only when referencing to the queries that were formulated before the modelling process.

• The complexities that arise due to improper understanding of the basic DBMS concepts related to ER to relational schema needs to be avoided by preparing a well-structured note for each concept before diving into the conversion process.

• The reference books mentioned in the lecture slides were of a good help when forming the constraints and types of joins to be used as a single source for most part of the coursework.

References

[1] Database System Concepts by Abraham Silberschatz (Yale University), Henry F. Korth (Lehigh University), S. Sudarshan (Indian Institute of Technology, Bombay), sixth and seventh edition, Published by McGraw-Hill Education

[2] IS5102 – Database Management Systems, Lecture Notes, University of St Andrews, 2022 - 23.

[3] Datatypes in SQLite, https://www.sqlite.org/datatype3.html