HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

Department of Information & Communication Technology

DATABASE PROJECT REPORT

<u>Project</u>: Book Store Management

<u>Lecturer</u>: Dr. Tran Viet Trung

Students:

Trần Hải Sơn – 20176861

Trần Lê Hoàng – 20176764

Hoàng Tuấn Anh Văn – 20170224

I. Introduction:

1. The database design:

- Most significantly this database is designed to manage books in the store, as well as information related to these books such as the total number of books available, authors, genres and publishers.
- Additionally, the database system has the capability to manage books' orders, total bills and the customers' information, together with the information of all the staffs working in the bookstore.

2. Data requirements:

- **Book data**: information related to its title, author, publisher, genre and available inventory quantity
- **Order data**: information related to books, order date, customer, total bill, and cashier
- **Customer data**: information related to the customer's name, address, phone, email
- **Staff data**: information related to the staff's name, hire date, end date, account and position

3. ER Diagram and relation detail:

- Table ***book***
  - *book_id* which is used to identify books, represents primary key
  - *title* is used to represent book's title
  - *price* stores the book's price
  - *inventory_qty* stores the quantity of available books in stock
  - *publisher_id* is foreign key, references to *publisher_id* in *publisher* table
  - 2 indexes are created in this table, one is on *book_id*, the other is on *publisher_id*

- Table ***author***
- *author_id* which is used to identify authors, represents primary key
- *name* stores the author's name
- *country* stores the author' country
- 1 index is created, which is on *author_id*

- Table ***author_detail***
- *author_id* is foreign key, references to *author_id* in *author* table
- *book_id* is foreign key, references to *book_id* in *book* table
- Both *author_id* and *book_id* form primary key in this table
- 2 indexes are created in this table, one is on 2 fields (*book_id, author_id*), the other is on *book_id*

- Table ***genre***
- *book_id* is foreign key, references to *book_id* in *book* table
- *genre* stores the book's genre
- Both *genre* and *book_id* form primary key in this table
- 1 index are created in this table, which is on 2 fields (*book_id, gerne*)

- Table ***publisher***
- *publisher_id* which is used to identify publishers, represents primary key
- *name* stores the publisher's company name
- *address* stores the address of the publisher
- 1 index are created in this table, which is on *publisher_id*

- Table ***orders***
- *order_id* which is used to identify orders, represents primary key

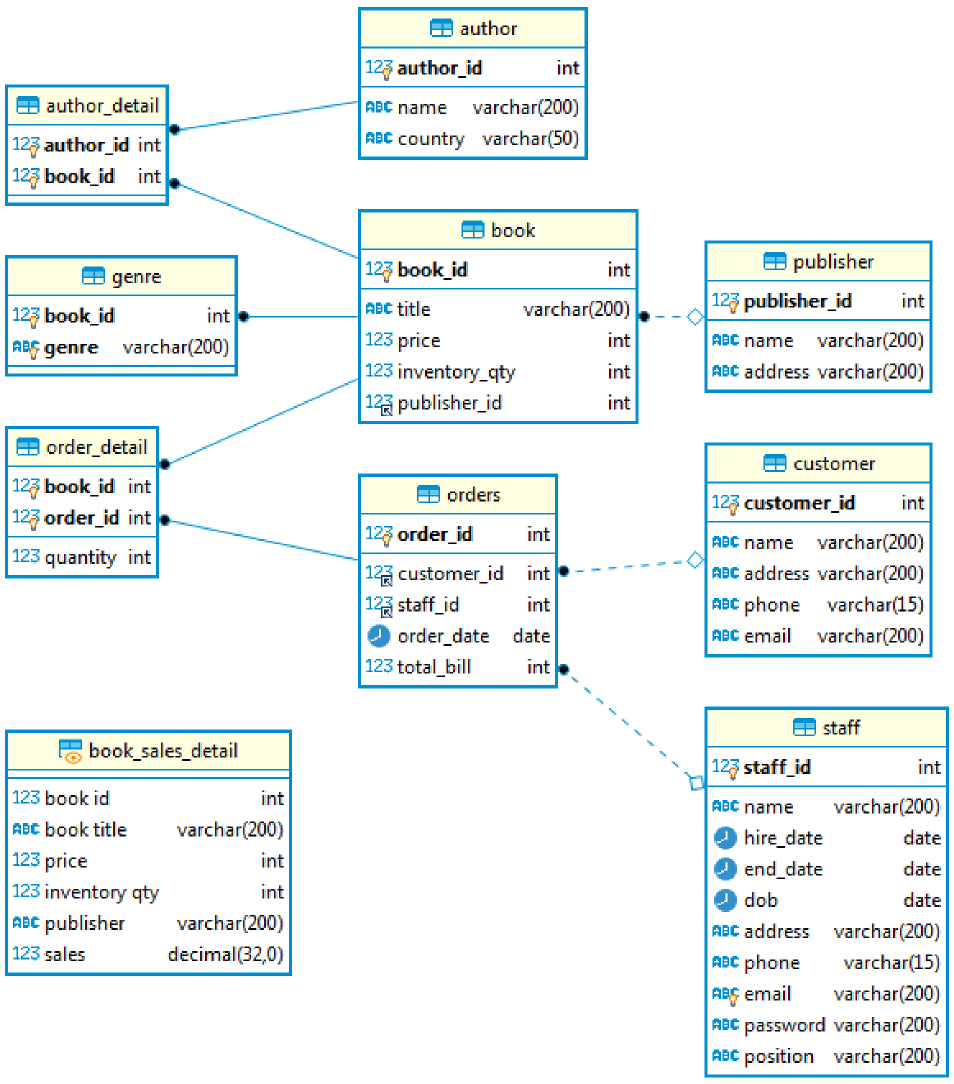
- *customer_id* is foreign key, references to *customer_id* in *<u>customer</u>* table
- *staff_id* is foreign key, references to *staff_id* in *<u>staff</u>* table
- *order_date* stores the date which the order is created
- *total_bill* stores total money that customer have to pay for the order (when *<u>order_detail</u>* is updated, *total_bill* is increased automatically by trigger)
- 3 indexes are created in this table, the first one is on *order_id*, the second one is on *customer_id* and the remaining one is on *staff_id*

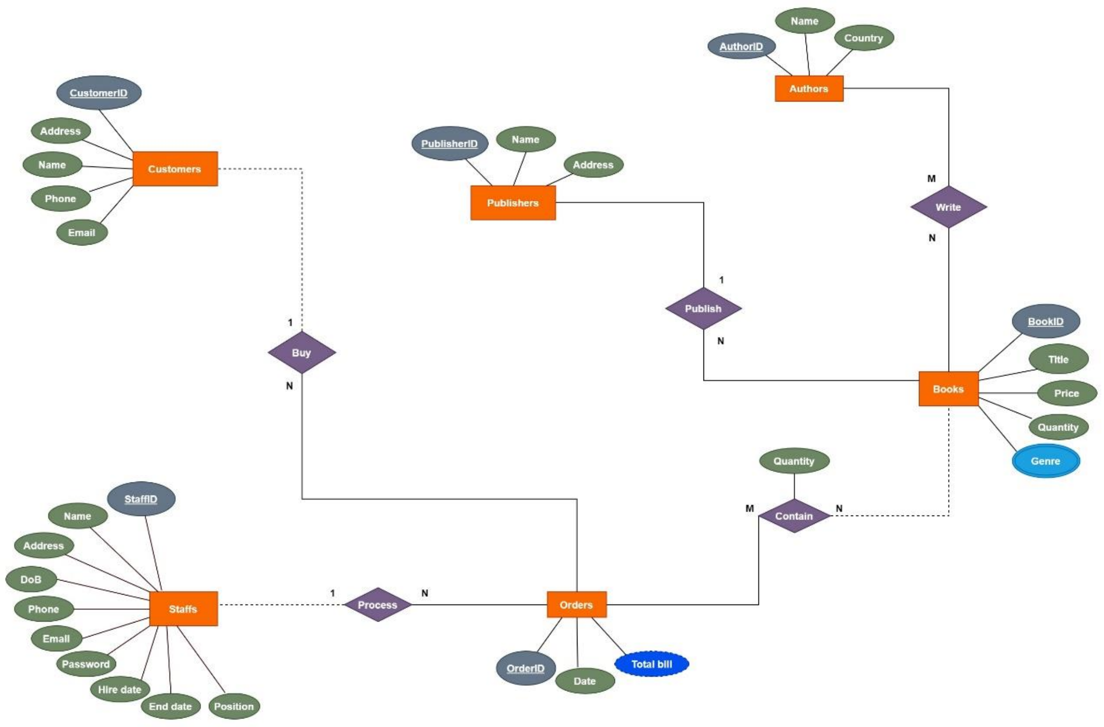
- Table ***<u>order_detail</u>***
- *book*_id is foreign key, references to *book_id* in *<u>book</u>* table
- *order_id* is foreign key, references to *order_id* in *<u>order</u>* table
- *quantity* stores the quantity of selected book in order
- Both *book_id* and *order_id* form primary key in this table
- 2 indexes are created in this table, one is on 2 fields (*book_id*, *order_id*), the other is on *order_id*

- Table ***<u>customer</u>***
- *customer_id* which is used to identify customers, represents primary key
- *name* stores the customer's name
- *address* stores the customer's address
- *phone* stores the customer's phone
- *email* stores the customer's email
- 1 index are created in this table, which is on *customer_id*

- Table ***<u>staff</u>***
- *staff_id* which is used to identify staffs, represents primary key
- *name* stores the staff's name

- *hire_date* stores the date that the staff was hired
- *end_date* stores the date that the staff ended his/her job
- *dob* stores the staff's date of birth
- *address* stores the staff's address
- *phone* stores the staff's phone
- *email* stores the unique staff's email, which is also the username to login
- *password* stores the staff's account password
- *position* stores information about the staff's position in the bookstore (such as manager, cashier, security, …)
- 2 indexes are created in this table, one is on *staff_id*, the other is on *email*


author
author_id int
name varchar(200)
country varchar(50)
author_detail
author_id int
book_id int
book
book_id int
title varchar(200)
price int
inventory_qty int
publisher_id int
publisher
publisher_id int
name varchar(200)
address varchar(200)
genre
book_id int
genre varchar(200)
order_detail
book_id int
order_id int
quantity int
orders
order_id int
customer_id int
staff_id int
order_date date
total_bill int
customer
customer_id int
name varchar(200)
address varchar(200)
phone varchar(15)
email varchar(200)
book_sales_detail
book id int
book title varchar(200)
price int
inventory qty int
publisher varchar(200)
sales decimal(32,0)
staff
staff_id int
name varchar(200)
hire_date date
end_date date
dob date
address varchar(200)
phone varchar(15)
email varchar(200)
password varchar(200)
position varchar(200)


Name
Country
AuthorID
Authors
CustomerID
Address
Name
Phone
Email
Customers
PublisherID
Name
Address
Publishers
M
Write
N
1
Publish
N
1
Buy
N
BookID
Title
Price
Quantity
Genre
Books
Quantity
M
Contain
N
StaffID
Name
Address
DoB
Phone
Email
Password
Hire date
End date
Position
Staffs
1
Process
N
Orders
OrderID
Date
Total bill

II. Queries:

1. Retrieve total revenue of books by day

```
SELECT order_date date, sum(total_bill) AS `total revenue`
FROM orders
GROUP BY order_date
ORDER BY order_date ASC;
```

2. Retrieve infomation of customers who bought more than 3 Fantasy books

```
SELECT c.*
FROM customer c, orders o, order_detail od, genre g
WHERE c.customer_id = o.customer_id
AND o.order_id = od.order_id AND g.book_id = od.book_id
AND g.genre = 'Fantasy'
GROUP BY c.customer_id
HAVING COUNT(*) > 3;
```

3. Retrieve author(s) that co-operate with only one publisher

```
SELECT DISTINCT a.name 'author', p.name 'publisher'
FROM author a, author_detail ad, book b, publisher p
WHERE a.author_id = ad.author_id AND ad.book_id = b.book_id AND
p.publisher_id = b.publisher_id
AND 1 = (
      SELECT COUNT(DISTINCT b1.publisher_id)
      FROM author a1, author_detail ad1, book b1
      WHERE a1.author_id = ad1.author_id AND ad1.book_id = b1.book_id
      AND a1.author_id = a.author_id
);
```

4. Retrieve unsold books in the past 6 month and its last sold date (null if that book is unsold)

```
SELECT b.title, last_sold.date AS 'last sold', b.book_id
FROM (
      SELECT MAX(o.order_date) 'date', od.book_id
      FROM orders o, order_detail od
      WHERE o.order_id = od.order_id
      GROUP BY od.book_id
) last_sold
RIGHT JOIN book b ON b.book_id = last_sold.book_id
WHERE b.title not in (
      SELECT DISTINCT b1.title
      FROM order_detail od
      JOIN book b1 ON od.book_id = b1.book_id
      JOIN orders o ON o.order_id = od.order_id
      WHERE o.order_date >= DATE_SUB(CURDATE(), INTERVAL 180 DAY)
);
```

5. Retrieve information of customers who spent more than 2000000 in 2020, sort in descending order

```
SELECT c.*, sum(o.total_bill) AS 'total spending'
FROM customer c
JOIN orders o ON o.customer_id = c.customer_id
WHERE YEAR(o.order_date) = 2020
GROUP BY c.customer_id
HAVING SUM(o.total_bill) > 2000000
ORDER BY SUM(o.total_bill) DESC;
```

6. Retrieve all books written by J.K Rowling

```
SELECT b.title
FROM book b
JOIN author_detail ad ON ad.book_id = b.book_id
JOIN author a ON a.author_id = ad.author_id
WHERE a.name = 'J. K. Rowling';
```

7. Retrieve books that have amount in stock smaller than 3 books and were sold more than 10 copies in last 3 month

```
SELECT b.*, SUM(od.quantity) AS 'Copies sold in last 3 month'
FROM book b
JOIN order_detail od ON b.book_id = od.book_id
JOIN orders o ON o.order_id = od.order_id
WHERE b.inventory_qty < 3 AND o.order_date >= DATE_SUB(CURDATE(),
INTERVAL 90 DAY)
GROUP BY b.book_id
HAVING SUM(od.quantity) > 10;
```

8. Retrieve the best seller in top 3 publisher having most books sold

```
SELECT b.title 'book title', p.name 'publisher name',
SUM(od.quantity) 'amount sold'
FROM book b, order_detail od, (
      SELECT publisher.*
      FROM publisher
      JOIN book ON publisher.publisher_id = book.publisher_id
      JOIN order_detail ON order_detail.book_id = book.book_id
      GROUP BY book.publisher_id
      ORDER BY SUM(order_detail.quantity) DESC
      LIMIT 3
) p
WHERE b.book_id = od.book_id AND b.publisher_id = p.publisher_id
GROUP BY b.book_id, b.publisher_id
HAVING SUM(od.quantity) >= ALL (
      SELECT SUM(od1.quantity)
      FROM book b1, order_detail od1
      WHERE b1.book_id = od1.book_id AND b1.publisher_id =
b.publisher_id
      GROUP BY b1.book_id
);
```

9. Retrieve the 2019 quarter revenue in descending order

```
SELECT QUARTER(o.order_date) AS 'quarter', SUM(od.quantity *
b.price) AS 'revenue'
FROM orders o, order_detail od, book b
WHERE o.order_id = od.order_id AND b.book_id = od.book_id AND
YEAR(o.order_date) = 2019
GROUP BY QUARTER(o.order_date)
ORDER BY SUM(od.quantity * b.price) DESC;
```

10. Create procedure to retrieve books in a price range

```
DELIMITER $$
CREATE PROCEDURE book_in_price_range(IN low int, IN high int)
BEGIN
      SELECT book.*
      FROM book
      WHERE price >= low AND price <= HIGH;
END; $$
DELIMITER ;
CALL book_in_price_range(100000, 300000);
```

11. Create view which retrieves the list of all books, its publisher and the amount of each book sold, sorts by the amount of sold books in descending order

```
CREATE VIEW book_sales_detail AS
SELECT b.book_id 'book id', b.title 'book title', b.price,
b.inventory_qty 'inventory qty', p.name 'publisher',
COALESCE(sum(od.quantity), 0) 'sales'
FROM book b LEFT JOIN order_detail od ON b.book_id = od.book_id,
publisher p
WHERE p.publisher_id = b.publisher_id
GROUP BY b.book_id
ORDER BY sales;
```

12. In the top 3 most favorite genres, retrieve top 4 books which were bought the most of each genre

```
SELECT * FROM (
      SELECT genre_detail_sales.book_id,
      genre_detail_sales.title, genre_detail_sales.genre,
      genre_detail_sales.sales,
      DENSE_RANK() OVER ( PARTITION BY genre_detail_sales.genre
            ORDER BY genre_detail_sales.sales DESC) AS 'rank'
      FROM (
            SELECT b.book_id, b.title, g.genre, sum(od.quantity) 'sales'
            FROM genre g, book b, order_detail od
            WHERE g.book_id = b.book_id AND od.book_id = b.book_id
            AND g.genre IN (
                  SELECT genre_sales_rank.genre FROM (
                        SELECT genre_sales.*, DENSE_RANK() OVER
                              (ORDER BY sales DESC) sales_rank
                        FROM (
                              SELECT g.genre, sum(od.quantity) 'sales'
                              FROM genre g, book b, order_detail od
                              WHERE g.book_id = b.book_id AND
                                    od.book_id = b.book_id
                              GROUP BY g.genre
                              ORDER BY sales DESC
                        ) genre_sales
                  ) genre_sales_rank
                  WHERE sales_rank <= 3
            )
            GROUP BY b.book_id, g.genre
            ORDER BY g.genre, sales DESC
      ) genre_detail_sales ) temp
WHERE temp.`rank` <= 4;
```

13. Retrieve the average revenue per months in 2019

```
SELECT MONTH(o.order_date) 'month', round(avg(o.total_bill))
'average revenue'
FROM orders o
```

```
WHERE YEAR(o.order_date) = 2019
GROUP BY MONTH(o.order_date)
ORDER BY MONTH(o.order_date);
```

14. Retrieve information of the customer who bought the most number of books and that amount of books

```
SELECT c.*, sum(od.quantity) 'book amount'
FROM orders o, customer c, order_detail od
WHERE o.order_id = od.order_id AND c.customer_id = o.customer_id
GROUP BY c.customer_id
HAVING `book amount` >= ALL (
      SELECT sum(od1.quantity)
      FROM orders o1, customer c1, order_detail od1
      WHERE o1.order_id = od1.order_id
      AND c1.customer_id = o1.customer_id
      GROUP BY c1.customer_id
);
```

15. Retrieve books information and authors of the top 7 most favorite books

```
SELECT * FROM (
      SELECT salesinfo.*, dense_rank() over (order by sales
            desc) sales_rank
      FROM (
            SELECT b.title, b.price, b.inventory_qty, a.name,
            a.country, sum(od.quantity) 'sales'
            FROM author a, author_detail ad, book b, order_detail od
            WHERE a.author_id = ad.author_id AND
            ad.book_id = b.book_id AND od.book_id = b.book_id
            GROUP BY b.book_id, a.author_id
            ORDER BY sales DESC
      ) salesinfo
) salesinfo_withrank
WHERE salesinfo_withrank.sales_rank <= 7;
```

16. Retrieve information of customer(s) who bought more than 3 times, the total number of books they bought, and the minimum time interval between their 2 consecutive orders

```sql
SELECT tt.customer_id, tt.name, sum(tt.books) 'total books',
min(datediff(tt.order_date, tt.prev_order)) 'min interval (days)'
FROM (
      SELECT c.customer_id, c.name, o.order_id, sum(od.quantity) 'books',
      o.order_date, lag(order_date, 1) OVER (PARTITION BY customer_id
      ORDER BY customer_id, order_date) AS 'prev_order'
      FROM customer c, order_detail od, orders o
      WHERE o.customer_id = c.customer_id AND o.order_id = od.order_id
      AND 3 < (SELECT count(*) FROM orders o1
               WHERE o1.customer_id = c.customer_id)
               GROUP BY customer_id, order_id
) tt
GROUP BY tt.customer_id;
```

17. Retrieve information of the order which has the highest total bill in 2019 and the cashier that was in charge of that order

```sql
SELECT o.order_id, o.order_date, o.total_bill, s.name, s.dob,
s.phone, s.email
FROM orders o, staff s
WHERE o.staff_id = s.staff_id
AND YEAR(o.order_date) = 2019
AND o.total_bill >= ALL (
      SELECT o.total_bill
      FROM orders o, staff s
      WHERE o.staff_id = s.staff_id
      AND YEAR(o.order_date) = 2019
);
```

18. Retrieve information of customers who bought both "Adventure", "Mystery" and "Action" books

```
SELECT DISTINCT c.*
FROM genre g, book b, order_detail od, orders o, customer c
WHERE g.book_id = b.book_id AND b.book_id = od.book_id
AND od.order_id = o.order_id AND o.customer_id = c.customer_id
AND g.genre = 'Adventure'
AND c.customer_id IN (
	SELECT c1.customer_id
	FROM genre g1, book b1, order_detail od1, orders o1, customer c1
	WHERE g1.book_id = b1.book_id AND b1.book_id = od1.book_id
	AND od1.order_id = o1.order_id AND o1.customer_id = c1.customer_id
	AND g1.genre = 'Action'
)
AND c.customer_id IN (
	SELECT c2.customer_id
	FROM genre g2, book b2, order_detail od2, orders o2, customer c2
	WHERE g2.book_id = b2.book_id AND b2.book_id = od2.book_id
	AND od2.order_id = o2.order_id AND o2.customer_id = c2.customer_id
	AND g2.genre = 'Mystery'
);
```

19. Retrieve information of the publisher(s) selling all books written by 'George R. R. Martin'

```
SELECT p.publisher_id, p.name 'Publisher', p.address
FROM publisher p, author a, book b, author_detail ad
WHERE p.publisher_id = b.publisher_id AND b.book_id = ad.book_id AND
ad.author_id = a.author_id
AND a.name = 'George R. R. Martin'
GROUP BY p.publisher_id
HAVING count(*) = (
	SELECT count(*)
	FROM book b1, author a1, author_detail ad1
	WHERE b1.book_id = ad1.book_id AND ad1.author_id = a1.author_id
	AND a1.name = 'George R. R. Martin'
);
```

20. During the Covid-19 pandemic, in order to maintain the business, the director of the bookstore decided to fire some non-manager staffs which were newly hired in 2020. Update and retrieve the list of these staffs who were fired.

```
UPDATE staff s
SET s.end_date = now()
WHERE YEAR(s.hire_date) = 2020 AND s.`position` <> 'manager';
```

21. Create trigger to update the quantity of books left after an order is made

```
CREATE TRIGGER update_book_quantity BEFORE INSERT ON
order_detail
FOR EACH ROW
      UPDATE book
      SET inventory_qty = inventory_qty - NEW.quantity
      WHERE book_id = NEW.book_id;
```

22. Create trigger to get the total bill of an order when books are added into order

```
CREATE TRIGGER order_total_bill BEFORE INSERT ON order_detail
FOR EACH ROW
      UPDATE orders
      SET total_bill = total_bill + (
            SELECT (price * NEW.quantity)
            FROM book
            WHERE book_id = NEW.book_id
      )
      WHERE order_id = NEW.order_id;
```

23. Give the name of exactly 2 publishers publishing the most number of books

```sql
SELECT p.name, count(b.book_id) AS '# published books'
FROM book b, publisher p
WHERE b.publisher_id = p.publisher_id
GROUP BY p.publisher_id
ORDER BY count(b.book_id) DESC
LIMIT 2;
```

24. Give the information of all the authors whose books published by publisher 'Lao Động'

```sql
SELECT a.name, a.country
FROM author AS a, author_detail AS ad, book AS b, publisher AS p
WHERE b.publisher_id = p.publisher_id
AND b.book_id = ad.book_id
AND ad.author_id = a.author_id
AND p.name LIKE N'%Lao Động%';
```

25. Give the name, hire date and the number of books sold by staffs who have been working less than 1 year

```sql
SELECT s.name, s.hire_date, SUM(od.quantity) AS book_sold
FROM staff s, orders o, order_detail od
WHERE s.staff_id = o.staff_id AND o.order_id = od.order_id
AND (DATEDIFF(now(), s.hire_date)/365) < 1
GROUP BY s.staff_id
ORDER BY book_sold DESC;
```

26. Retrieve the titles, authors of books written by American or English authors, whose books were published by more than 3 publishers

```sql
SELECT b.title 'Books', a.name
FROM book b, author_detail ad, author a
WHERE b.book_id = ad.book_id AND ad.author_id = a.author_id
AND ad.author_id IN (
      SELECT a1.author_id
```

```
        FROM author AS a1, author_detail AS ad1, book AS b1
        WHERE a1.author_id = ad1.author_id AND b1.book_id = ad1.book_id
        AND a1.country IN ('USA', 'England')
        GROUP BY a1.author_id
        HAVING COUNT(DISTINCT b1.publisher_id) >= 3
);
```

27. Function to get the number of books published by 1 publisher

```
DELIMITER $$
CREATE FUNCTION book_count(publisherID INT) RETURNS INT
LANGUAGE SQL DETERMINISTIC
BEGIN
        DECLARE number_of_book INT;
        SELECT COUNT(book_id)
        INTO number_of_book
        FROM book
        WHERE publisher_id = publisherID
        GROUP BY publisher_id;
        RETURN number_of_book;
END $$
DELIMITER ;
```

28. Give the name of customers who buy books in only 1 genre

```
SELECT c.name
FROM customer AS c, orders AS o, order_detail AS od, book AS b,
genre AS g
WHERE c.customer_id = o.customer_id
AND o.order_id = od.order_id
AND od.book_id = b.book_id
AND b.book_id = g.book_id
GROUP BY c.customer_id
HAVING COUNT(DISTINCT g.genre) = 1;
```

29. Retrieve the title and price of books whose price is greater than the sum of 5 books that are the cheapest

```
SELECT b.title, b.price
FROM book AS b
WHERE b.price > (
      SELECT SUM(cheapest.price)
      FROM (
            SELECT price
            FROM book
            ORDER BY price
            LIMIT 5) AS cheapest
);
```

30. Give the name and phone number of customers who used to buy at least 1 book written by author 'J. K. Rowling' and the total number of books he/she's already bought till now is less than 5

```
SELECT c.name, c.phone
FROM customer AS c, orders AS o, order_detail AS od
WHERE c.customer_id = o.customer_id
AND o.order_id = od.order_id
AND od.book_id IN (
      SELECT b.book_id
      FROM book AS b, author_detail AS ad, author AS a
      WHERE b.book_id = ad.book_id
      AND a.author_id = ad.author_id
      AND a.name = 'J. K. Rowling'
)
GROUP BY c.customer_id
HAVING SUM(od.quantity) < 5;
```