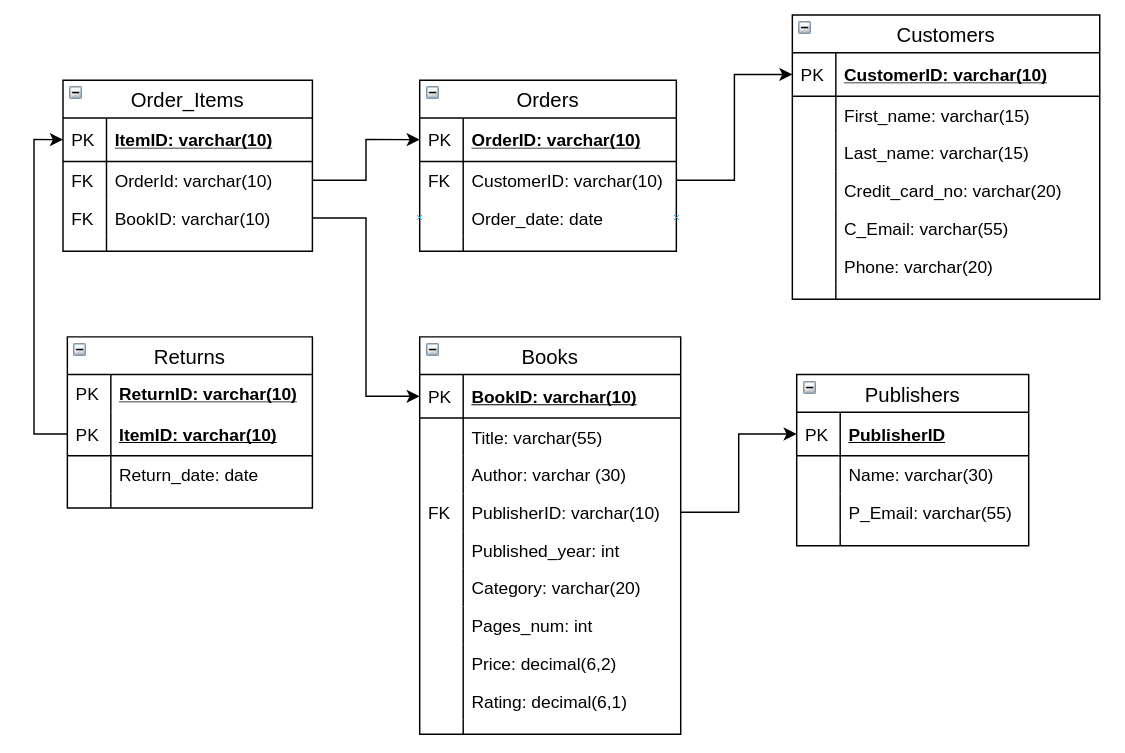
**TABLE OF CONTENT**

I. DATABASE SCHEMA.......................................................................................2

II. QUERIES.........................................................................................................4

1. **DATABASE SCHEMA**

We created a database to manage a small bookstore.



1. **Books**: *information about books that the bookstore sells*
   * BookID: ID of each book
   * Title: Name of the title
   * Author: Name of book’s author
   * *PublisherID*: Foreign key references the ID of book’s publisher
   * Category: Genre of book (Horror, Fiction, etc...)
   * Pages\_num: Number of pages that book contains
   * Price: Book price in dollars
   * Rating: Rating over 5 of book
2. **Publishers**: *infomation about publishers that provide books to the bookstore*
   * PublisherID: ID of each publisher
   * Name: Publisher name
   * P\_Email: Email of publisher
3. **Customers**: *information about customer that buys from the bookstore*
   * CustomerID: ID of the customer who makes the purchase
   * First\_name: first name of customer
   * Last\_name: last name of customer
   * Credit\_card\_no: The credit card number of each customer
   * C\_Email: Customer’s email
   * Phone: Phone number of each customer
4. **Orders**: *Record of the order made in the bookstore*
   * OrderID: ID of each order
   * *CustomerID*: Foreign key references the customer that made the purchase
   * Order\_date: date of the order
5. **Order\_items**: *Record of book items in each order*
   * ItemID: ID of the item purchased
   * *OrderID*: Foreign key references the ID of the order that item belongs to
   * *BookID*: Foreign key reference the ID of the book
6. **Returns**: *Information about book items returned after purchased (the customer’s not satisfy with the quality)*
   * ReturnID: ID of the return
   * *ItemID*: Foreign key references the order item that is returned
   * Return\_date: date of the return
7. **QUERIES**

**-- 1. Get the top 10 best sellers and the number of copies that they sold**

select Title, count(ItemID) No\_books

from Books natural join Order\_Items

where ItemID not in (select ItemID from Returns)

group by BookID

order by No\_books desc

limit 10;

**-- 2. Find the cheapest among the top 10 best selling books**

select \*

from (select Books.\*

from Order\_Items natural join Books

where ItemID not in (select ItemID from Returns)

group by BookID

order by count(ItemID) desc

limit 10) best\_sellers

order by Price

limit 1;

**-- 3. Find books for give away events - cheapest among 10 most returned books**

**-- 1 title with lowest price**

select Title, Price

from (select BookID, count(ItemID) No\_of\_returns

from Returns natural join Order\_Items natural join Books

group by BookID

order by No\_of\_returns desc

limit 10) most\_returned natural join Books

order by Price

limit 1;

-- titles with lowest price

delimiter //

create temporary table most\_returned (

BookID varchar(10),

No\_of\_returns int

);

insert into most\_returned

(select BookID, count(ItemID) No\_of\_returns

from Returns natural join Order\_Items

group by BookID

order by No\_of\_returns desc

limit 10);

select Title, price

from Books natural join most\_returned

where price = (select min(price) from most\_returned natural join Books)//

**-- 4. Find the difference between the rating of the highest rated publisher (average book rating) and the lowest rated publisher**

select max(avg\_rating) - min(avg\_rating)

from (select Name, avg(rating) avg\_rating

from Publishers natural join Books

group by PublisherID) p;

**-- 5. Get Publishers who sold the most copies (excluding returns)**

select Name

from Publishers natural join Books natural join Order\_Items

where ItemID not in (select ItemID from Returns)

group by PublisherID

having count(ItemID) >= all (select count(ItemID)

from Publishers natural join Books natural join Order\_Items

where ItemID not in (select ItemID from Returns)

group by PublisherID);

**-- 6. Find the publishers that have the most books rated 4.0 or above**

select Name

from Publishers natural join Books

where rating > 4.0

group by PublisherID

having count(BookID) >= all (select count(BookID)

from Books

where rating > 4.0

group by PublisherID);

**-- 7. Return the percentage of titles with rating > 4.0 of each publisher**

select Name, round((select count(b2.BookID)

from Books b2

where b2.PublisherID = b1.PublisherID

and b2.rating > 4)

/

(select count(BookID))

\* 100, 1) Percentage

from Books b1 natural join Publishers

group by PublisherID

order by Percentage desc;

**-- 8. Find the differences between the number of books released after and before the year 2010**

select abs(b1.no\_books - b2.no\_books)

from (select count(BookId) no\_books

from Books

where Published\_year >= 2010) b1,

(select count(BookID) no\_books

from Books

where Published\_year < 2010) b2;

**-- 9. Find the difference in the number of books published each year to the number of books published in the busiest year**

delimiter //

set @SoB = (select count(BookID) SoB

from Books

group by Published\_year

order by SoB desc

limit 1);

select Published\_year, @SoB - count(BookID) 'Difference'

from Books

group by Published\_year//

**-- 10. Return the size of books (<200, 200-300, 300-400, >400) and their corresponding popularity (successful orders)**

delimiter //

set @under\_200 = (select count(ItemID)

from Order\_Items natural join Books

where ItemID not in (select ItemID from Returns)

and pages\_num < 200);

set @200\_300 = (select count(ItemID)

from Order\_Items natural join Books

where ItemID not in (select ItemID from Returns)

and pages\_num between 200 and 300);

set @300\_400 = (select count(ItemID)

from Order\_Items natural join Books

where ItemID not in (select ItemID from Returns)

and pages\_num between 301 and 400);

set @over\_400 = (select count(ItemID)

from Order\_Items natural join Books

where ItemID not in (select ItemID from Returns)

and pages\_num > 400);

create temporary table size\_popularity(Size varchar(10),

No\_of\_orders int);

insert into size\_popularity values('under 200', @under\_200), ('200 to 300', @200\_300), ('300 to 400', @300\_400), ('over 400', @over\_400);

select \* from size\_popularity//

**-- 11. Find books that get returned the quickest**

select Title, datediff(return\_date, order\_date) days

from Books natural join Order\_Items natural join Orders natural join Returns

group by BookID

having days <= all (select datediff(return\_date, order\_date)

from Orders natural join Order\_Items natural join Returns

group by BookID);

**-- 12. Get info of customers who return their books within the a month after they bought them**

select c.\*

from Customers c natural join Orders natural join Order\_Items natural join Returns

where datediff(return\_date, order\_date) <= 31;

**-- 13. Find the success percentage (number of successful purchases / total titles) of each category**

select category, round((select count(ItemID)

from Order\_Items natural join Books b2

where ItemID not in (select ItemID from Returns)

and b2.category = b1.category)

/

(select count(ItemID)

from Order\_Items natural join Books b3

where b3.category = b1.category) \* 100, 1) success\_percentage

from Books b1

group by category

order by success\_percentage desc;

**-- 14. Find average value (Dollars per number of pages) of each publisher**

select Name, (select avg(round(price / pages\_num, 2))

from Books b2

where b2.PublisherID = b1.PublisherID) average\_value

from Publishers natural join Books b1

group by PublisherID

order by average\_value desc;

**-- 15. Customers who make the most orders in the most recent year**

select Customers.\*

from Customers natural join Orders o1

where year(order\_date) = (select max(year(order\_date))

from Orders)

group by CustomerID

having count(OrderID) >= all (select count(OrderID)

from Orders

where year(order\_date) = (select max(year(order\_date))

from Orders)

group by CustomerID);

**-- 16. Get customers who bought the most copies of book**

select concat(first\_name, ' ', last\_name) Name

from Customers natural join Orders natural join Order\_Items

where ItemID not in (select ItemID from Returns)

group by CustomerID

having count(ItemID) >= all (select count(ItemID)

from Orders natural join Order\_Items natural join Customers

where ItemID not in (select ItemID from Returns)

group by CustomerID);

**-- 17. Get the favourite publisher of each customer**

select concat(first\_name, ' ', last\_name) Name, p.Name "Favourite publisher"

from Customers c1 natural join Orders natural join Order\_Items natural join Books natural join Publishers p

where PublisherID = (select PublisherID

from Books natural join Order\_Items natural join Orders natural join Customers c2

where c1.CustomerID = c2.CustomerID

group by PublisherID

order by count(ItemID) desc

limit 1)

-- group by CustomerID;

**-- 18. Get the favourite genre/category of each customer**

select concat(first\_name, ' ', last\_name) Name, category "Favourite category"

from Customers c1 natural join Orders natural join Order\_Items natural join Books

where category = (select category

from Books natural join Order\_Items natural join Orders natural join Customers c2

where c1.CustomerID = c2.CustomerID

group by category

order by count(ItemID) desc

limit 1)

group by CustomerID;

**-- 19. Get info of customers who bought more than 2 copies of any book**

select c1.\*

from Customers c1 natural join Orders natural join Order\_Items b1

where (select count(distinct ItemID)

from Order\_Items b2 natural join Orders natural join Customers c2

where c1.CustomerID = c2.CustomerID

and b1.BookID = b2.BookID

and ItemID not in (select ItemID from Returns)) > 2

group by CustomerID;

**-- 20. Return the customers with valid credit card numbers according to Luhn’s algorithm:**

**-- 1. Multiply every other digit by 2, starting with the number’s second-to-last digit, and then add those products’ digits together.**

**-- 2. Add the sum to the sum of the digits that weren’t multiplied by 2.**

**-- 3. If the total’s last digit is 0 (or, put more formally, if the total modulo 10 is congruent to 0), the number is valid!**

delimiter //

create function check\_validity(Credit\_card\_no varchar(20)) returns boolean

begin

set @card\_no = trim(replace(Credit\_card\_no, '-', ''));

set @total = 0;

set @len = 16;

set @ptr = 0;

while @ptr < @len do

set @add\_value = convert(substr(@card\_no, @ptr + 1, 1), int);

if mod(@ptr, 2) = 0 then

set @add\_value = @add\_value \* 2;

end if;

set @total = @total + @add\_value;

set @ptr = @ptr + 1;

end while;

if mod(@total, 10) = 0 then

return true;

else

return false;

end if;

end;

select concat(first\_name, ' ', last\_name) Customer, Credit\_card\_no, check\_validity(Credit\_card\_no) as validity

from Customers

order by validity desc//

-- BONUS

**-- 21. Find the publishers who got the most copies returned**

select Name

from Publishers natural join Books natural join Order\_Items natural join Returns

group by PublisherID

having count(ItemID) >= all (select count(ItemID)

from Publishers natural join Books natural join Order\_Items natural join Returns

group by PublisherID);

**-- 22. Return the percentage of orders that get returned within a month of each genre/category**

select category, round((select count(ItemID)

from Orders natural join Order\_Items natural join Returns natural join Books b2

where b2.category = b1.category

and datediff(return\_date, order\_date) <= 31)

/

(select count(ItemID)

from Order\_Items natural join Books b3

where b3.category = b1.category) \* 100, 1) Percentage

from Books b1

group by category;

**-- 23. Find the ID and price of the most expensive order**

select OrderID, sum(Price) Gia\_Tien

from Order\_Items natural join Books

group by OrderID

having Gia\_Tien >= all (select sum(Price)

from Order\_Items natural join Books

group by OrderID);

**-- 24. Find the average rating of each year**

select Published\_year, round(avg(rating), 1) 'Average rating'

from Books

group by Published\_year ;

**-- 25. Find the publishers whose all books are rated 3.0 or above**

select Name

from Publishers

where PublisherID not in (select PublisherID

from Books

where rating < 3.0);