University of Science and
Technology of Ha Noi

A report on

DATABASE DESIGN
FOR
AN ONLINE BOOK STORE

Teacher: Nguyen Hoang Ha

Students: Dao Viet Long - 22BI13258

Hoang Viet - 22BI13465

Chu Xuan Thang - 22BI13403

Nguyen Tuong Quang - 22BI13385

Group 7

Course: Fundamental of databases

Table of Content

1. Introduction

This comprehensive report delineates a meticulous strategy for logically categorizing the 12 tables within the Online Bookstore Database. The emphasis is placed on functional domains governing their operations, with the overarching goal of constructing a well-organized schema. The objective is to ensure alignment with the business functions of an online bookstore, fostering clarity, maintainability, and efficiency in data management.

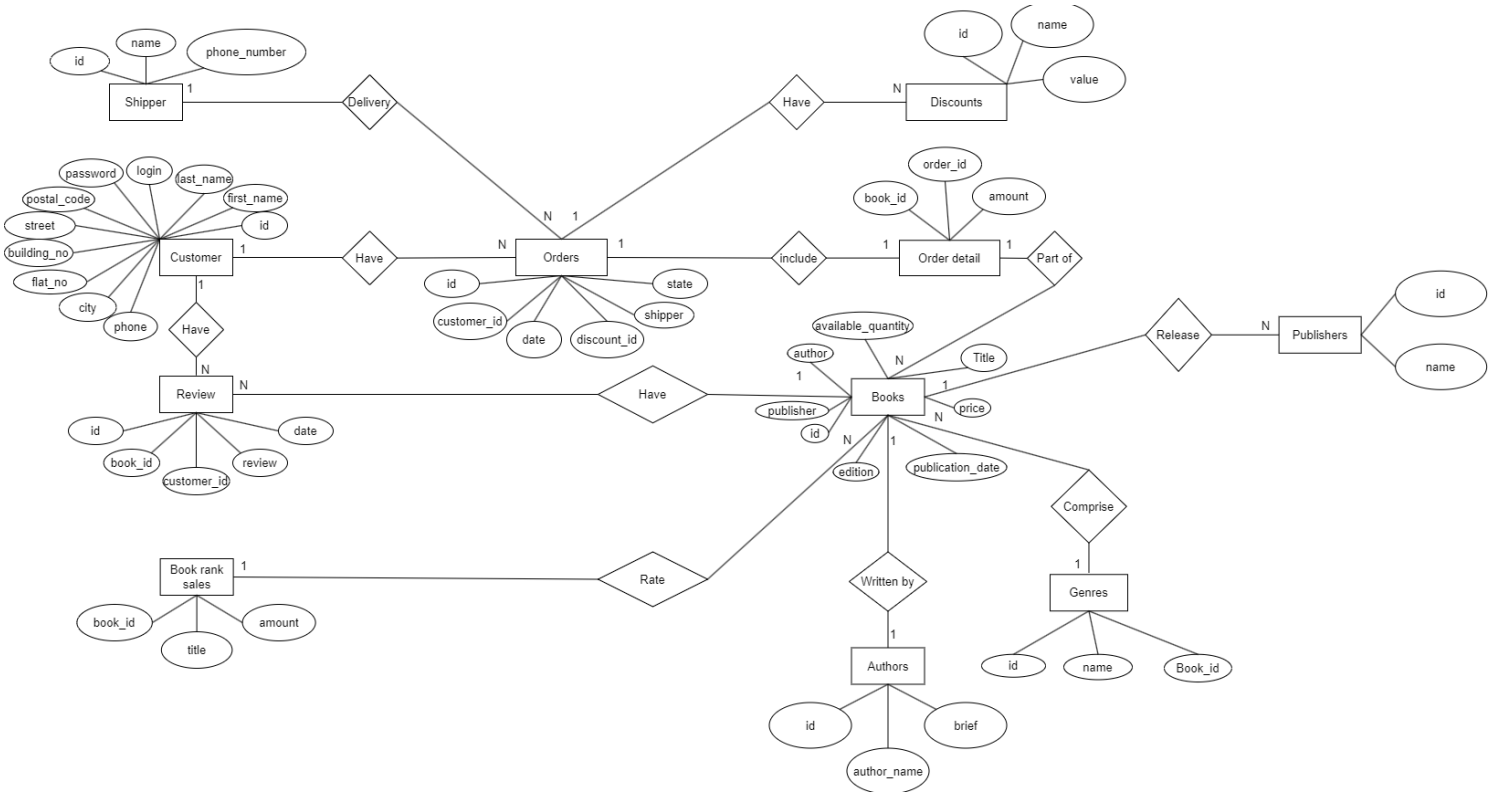
2. Data structure design

2.1. Business processes:

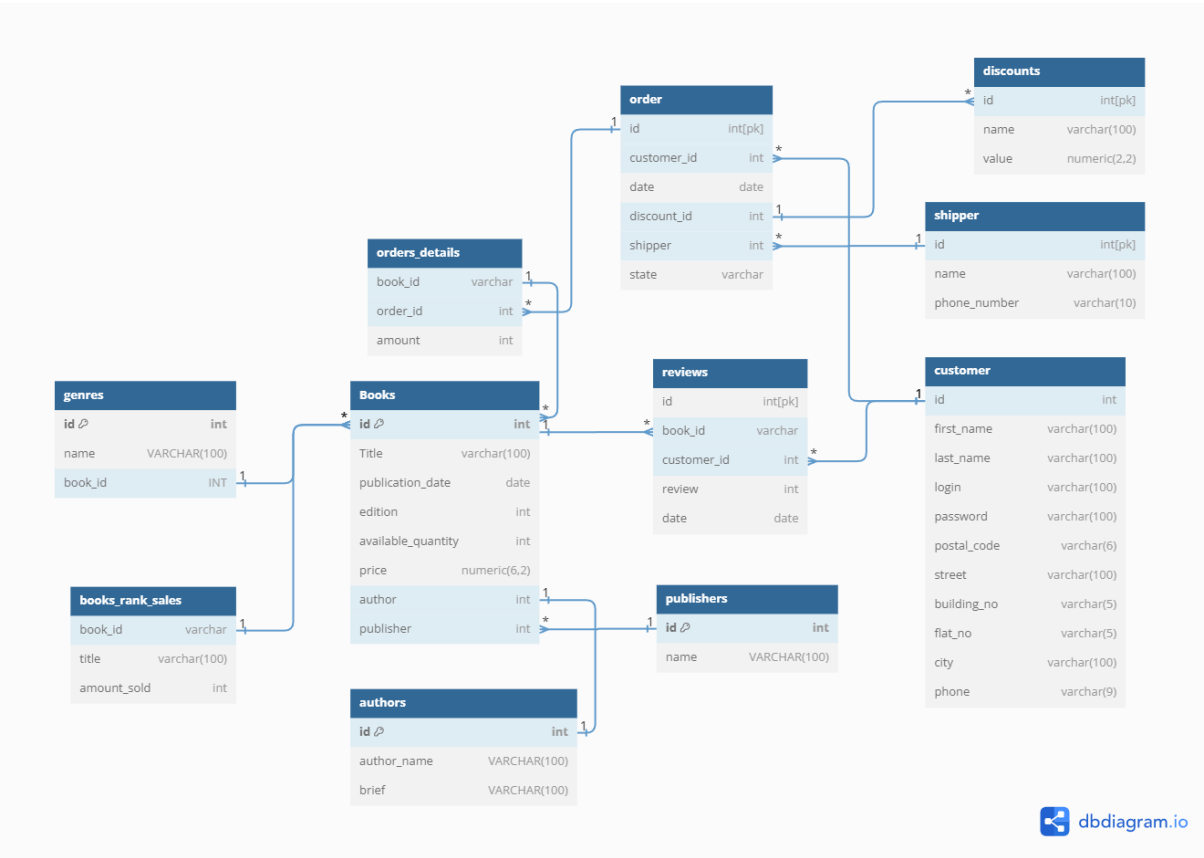
- Customer Registration:
    - Customers can register on the platform by providing their personal information, including names, login credentials, contact details, and address.
- Book Management:
    - The system allows administrators to add, update, or remove books from the inventory. Books are associated with specific authors and publishers.
- Order Placement:
    - Customers can browse books, add them to their cart, and place orders. Discounts can be applied based on promotional offers.
- Shipping and Delivery:
    - Orders are assigned to shippers for delivery. Shippers' information, including their contact details, is stored in the system.
- Customer Reviews:
    - Customers can leave reviews for books.
- Genre Categorization:
    - Books are categorized into genres, allowing customers to explore and filter books based on their interests.
- Ranking sales for books:
    - The amount of books sold is calculated and displayed to the customers.

2.2. Database design

2.2.1. ER-Diagram:



2.2.2. Convert to DB Schema:



2.3.3. Relation:

Shipper to Orders (single to multiple): 1 shipper can delivery 1 or many orders
Orders to Discounts (single to multiple): 1 order can have 1 or many discounts
Customer to Orders (single to multiple): 1 customer can have 1 or many orders
Customer to Review (single to multiple): 1 customer can have 1 or many reviews
Orders to Order detail (single to single): 1 order include only 1 order detail
Order detail to Books (single to multiple): 1 order detail can contain 1 or many books
Review to Books (multiple to single): 1 book can have 1 or many reviews
Publishers to Books (single to multiple): 1 publisher can publish 1 or many books
Books to Book rank sales (multiple to single): One or many books is calculated to get the numbers sold to calculate the rank
Books to Authors (single to single): 1 book can only written by 1 author
Books to Genres (multiple to single): 1 genre can comprise 1 or many books

2.3. Grouping tables

2.3.1. Group 1: Customer Management

a. Table: customer
    i. Id: a unique number series to specify a person, which helps the business track customer interactions and provide personalized services. (type: INT)
    ii. First name and last name: full name of a customer. (type: VARCHAR)
    iii. Login: username of the customer's account. (type: VARCHAR)
    iv. Password: created by user to access the bookstore. (type: VARCHAR)

v. Postal code: series of letters, digits, or both, appended to a postal address to assist in the sorting and delivery of mail. (type: VARCHAR)

vi. Address detail: the place for customers to receive their orders. (type: VARCHAR)

vii. Phone number: for the shippers to contact their customers. (type: VARCHAR)

```sql
CREATE TABLE if not exists `customer` (
  `id` INT PRIMARY KEY,
  `first_name` VARCHAR(100),
  `last_name` VARCHAR(100),
  `login` VARCHAR(100),
  `password` VARCHAR(100),
  `postal_code` VARCHAR(6),
  `street` VARCHAR(100),
  `building_no` VARCHAR(5),
  `flat_no` VARCHAR(5),
  `city` VARCHAR(100),
  `phone` VARCHAR(9)
);
```

- Serves as the core repository for managing customer information, encompassing details such as names, contact information, and addresses. This table is pivotal for customer interactions, order processing, and personalized services.

2.3.2. Group 2: Content Management

a. Table: authors

i. id: a unique identifier to distinguish an author from others. (type: INT)

ii. name: full name of the authors. (type: VARCHAR)

iii. Brief: basic information about the author. (type: VARCHAR):

```sql
CREATE TABLE if not exists `authors` (
  `id` INT PRIMARY KEY,
  `first_name` VARCHAR(100),
  `last_name` VARCHAR(100),
  `brief` VARCHAR(100)
);
```

b. Table: publishers

i. id: refers to a unique identifier assigned to a publisher. (type: INT)

ii. name: refers to the individual or entity responsible for producing, printing, and distributing the books. (type: VARCHAR)

```sql
CREATE TABLE if not exists `publishers` (
  `id` INT PRIMARY KEY,
  `full_name` VARCHAR(100) UNIQUE
);
```

c. Table: genres

   i. id: unique identifier for each genre. (type: INT)
   ii. full_name: full name or description of the genre. (type: VARCHAR)
   iii. book_id: foreign key referencing the books table

```sql
CREATE TABLE IF NOT EXISTS `genres` (
  `id` INT PRIMARY KEY,
  `full_name` VARCHAR(100) UNIQUE,
  `book_id` INT,
  FOREIGN KEY (`book_id`) REFERENCES `Books` (`id`)
);
```

- These tables collaboratively manage information about authors, publishers, and book genres, forming the foundational framework for content organization.

d. Table: Books

   i. id: unique identifier for each book. (type: INT)
   ii. Title: title of the book. (type: VARCHAR)
   iii. publication_date: date when the book was published. (type: DATE)
   iv. edition: edition number of the book. (type: INT)
   v. available_quantity: number of copies available for sale. (type: INT)
   vi. price: price of the book. (type: NUMERIC(6,2))
   vii. author: foreign key referencing the authors table. (type: INT)
   viii. publisher: foreign key referencing the publishers table. (type: INT)

```sql
CREATE TABLE if not exists  `Books` (
  `id` INT PRIMARY KEY,
  `Title` VARCHAR(100),
  `publication_date` DATE,
  `edition` INT,
  `available_quantity` INT,
  `price` NUMERIC(6,2),
  `author` INT,
  `publisher` INT,
  FOREIGN KEY (`author`) REFERENCES authors(id),
  FOREIGN KEY (`publisher`) REFERENCES publishers(id)
);
```

- The table above is the central to content management, this table houses details about individual books, including titles, publication dates, editions, quantities, and prices.

e. Table: books_rank

   i. book_id: foreign key referencing the Books table. (type: INT)
   ii. title: title of the book. (type: VARCHAR)
   iii. amount_sold: number of books sold. (type: INT)

```sql
CREATE TABLE IF NOT EXISTS `books_rank_sales` (
  `book_id` INT,
  `title` VARCHAR(100),
  `amount_sold` INT,
  FOREIGN KEY (`book_id`) REFERENCES `Books` (`id`)
```

f. Table: reviews

i. id: unique identifier for each review. (type: INT)

ii. book_id: foreign key referencing the Books table. (type: INT)

iii. customer_id: foreign key referencing the customer table. (type: INT)

```sql
CREATE TABLE if not exists `reviews` (
  `id` INT PRIMARY KEY,
  `book_id` INT,
  `customer_id` INT,
  `review` VARCHAR(255),
  `date` DATE,
  FOREIGN KEY (`customer_id`)
  REFERENCES `customer` (`id`),
  FOREIGN KEY (`book_id`)
  REFERENCES `Books` (`id`)
);
```

iv. review: textual content of the review. (type: VARCHAR)

v. date: date when the review was posted. (type: DATE)

- These tables respectively establish relationships between books and genres, capture ranking data for books, and manage customer reviews, providing valuable insights into customer sentiments and feedback.

2.3.3. Group 3: Order and Transaction Management

a. Table: discounts

i. id: unique identifier for each discount. (type: INT)

ii. name: name or description of the discount. (type: VARCHAR)

iii. value: value or percentage of the discount. (type: NUMERIC(2,2))

```sql
CREATE TABLE if not exists `discounts` (
  `id` INT PRIMARY KEY,
  `name` VARCHAR(100),
  `value` NUMERIC(2,2)
);
```

b. Table: shipper

i. id: unique identifier for each shipper. (type: INT)

ii. name: name of the shipper. (type: VARCHAR)

iii. phone_number: phone number of the shipper. (type: VARCHAR)

```sql
CREATE TABLE if not exists `shipper` (
  `id` INT PRIMARY KEY,
  `name` VARCHAR(100),
  `phone_number` VARCHAR(10)
);
```

- The discounts table manages discount information, contributing to pricing strategies and promotions
- The shipper table stores data related to shipping companies, facilitating efficient order fulfillment.

c. Table: orders

i. id: unique identifier for each order. (type: INT)

ii. customer_id: foreign key referencing the customer table. (type: INT)

iii. date: date when the order was placed. (type: DATE)

iv. discount_id: foreign key referencing the discounts table. (type: INT)

v. shipper: foreign key referencing the shipper table. (type: INT)

vi. state: state or status of the order. (type: VARCHAR)

```sql
CREATE TABLE if not exists `orders` (
  `id` INT PRIMARY KEY,
  `customer_id` INT,
  `date` DATE,
  `discount_id` INT,
  `shipper` INT,
  `state` VARCHAR(255),
  FOREIGN KEY (`customer_id`) REFERENCES `customer` (`id`),
  FOREIGN KEY (`discount_id`) REFERENCES `discounts` (`id`),
  FOREIGN KEY (`shipper`) REFERENCES `shipper` (`id`)
);
```

d. Table: orders_details

i. book_id: foreign key referencing the Books table. (type: INT)

ii. order_id: foreign key referencing the orders table. (type: INT)

iii. amount: quantity of books in the order. (type: INT)

```sql
CREATE TABLE if not exists `orders_details` (
  `book_id` INT,
  `order_id` INT,
  `amount` INT,
  FOREIGN KEY (`order_id`) REFERENCES `orders` (`id`),
  FOREIGN KEY (`book_id`) REFERENCES `Books` (`id`)
);
```

- The orders table captures overall order details, including customer information, dates, discounts, shipper information, and order status.
- The orders_details table provides a detailed breakdown of items within each order, including book IDs and quantities.

3. Implementation and evaluation

3.1. Testing data:

3.1.1. Insert Sample Authors:

3.1.2. Insert Sample Genres:

3.1.3. Insert Sample Publishers:

3.1.4. Insert Sample Books:

3.1.5. Insert Sample Customers:

3.1.6. Insert Sample Discounts:

3.1.7. Insert Sample Shippers:

3.1.8. Insert Sample Orders:

3.1.9. Insert Sample Orders Details:

3.1.10. Insert Sample Reviews:

3.2. Some sample business cases:

3.2.1. Views:

Book Adder View:

- The book_adder view consolidates information about books, including details about authors and publishers. It simplifies the process of displaying comprehensive book information.

```sql
CREATE OR REPLACE VIEW book_adder AS (
  SELECT
    Books.id,
    Books.title,
    Books.publication_date,
    Books.edition,
    Books.available_quantity,
    Books.price,
    authors.first_name,
    authors.last_name,
    authors.brief,  -- Include other columns as needed
    publishers.full_name AS publisher
  FROM Books
    JOIN authors ON Books.author = authors.id
    JOIN publishers ON Books.publisher = publishers.id);
  SELECT * FROM `customer`;
  SELECT * FROM book_adder ORDER BY id ASC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	title	publication_date	edition	available_quantity	price	first_name	last_name	brief	publisher
1	The Mystery of the Lost Key	2022-01-15	1	100	19.99	John	Doe	Bestselling author	HarperCollins
2	Shadow of the Silent Assassin	2022-02-20	2	75	24.99	Jane	Smith	Award-winning novelist	Penguin Random House
3	Galactic Odyssey	2022-03-10	1	120	29.99	Michael	Johnson	Historical fiction writer	Simon & Schuster
4	Love in the Moonlight	2022-04-05	1	90	17.99	Emily	Brown	Mystery and thriller author	Macmillan Publishers
5	The Elven Kingdom	2022-05-18	3	80	39.99	Christopher	Lee	Science fiction writer	Hachette Book Group
6	The Hidden Conspiracy	2023-02-10	2	110	21.99	John	Doe	Bestselling author	Scholastic Corporation
7	Eternal Love	2023-03-15	1	95	18.99	Jane	Smith	Award-winning novelist	Wiley
8	Beyond the Stars	2023-04-20	1	85	27.99	Michael	Johnson	Historical fiction writer	Oxford University Press
9	Whispers in the Shadows	2023-05-25	2	105	23.99	Emily	Brown	Mystery and thriller author	Pearson
10	The Time Traveler's Dilemma	2023-06-30	1	88	22.99	Christopher	Lee	Science fiction writer	Cambridge University Press
11	Midnight Whispers	2022-06-15	1	95	21.99	Amanda	Williams	Romance novelist	HarperCollins
12	Realm of Dreams	2022-07-20	2	80	26.99	Daniel	Miller	Fantasy storyteller	Penguin Random House

3.2.2. Getting top 5 most expensive book:

```sql
use company;
SELECT * FROM books
ORDER BY price DESC
LIMIT 5;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell C

id	Title	publication_date	edition	available_quantity	price	author	publisher
25	Shadows of Reality	2023-03-18	3	80	40.99	10	5
5	The Elven Kingdom	2022-05-18	3	80	39.99	5	5
35	Realms of Enchantment	2024-01-18	3	85	35.99	10	5
15	Enchanting Realms	2022-10-18	3	85	34.99	10	5
45	Eternal Realms	2024-11-18	3	85	34.99	10	5

3.2.3. Getting books based on the name of author:

```sql
SELECT
    Books.id,
    Books.Title,
    Books.publication_date,
    Books.edition,
    Books.available_quantity,
    Books.price,
    authors.first_name AS author_first_name,
    authors.last_name AS author_last_name,
    publishers.full_name AS publisher
FROM
    Books
JOIN
    authors ON Books.author = authors.id
JOIN
    publishers ON Books.publisher = publishers.id
WHERE
    authors.first_name = 'Robert' and authors.last_name = 'White';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	Title	publication_date	edition	available_quantity	price	author_first_name	author_last_name	publisher
14	Hearts Alight	2022-09-05	1	75	19.99	Robert	White	Macmillan Publishers
24	Echoes of Destiny	2023-02-05	1	90	18.99	Robert	White	Macmillan Publishers
34	Whispers of the Past	2023-12-05	1	75	21.99	Robert	White	Macmillan Publishers
44	Hearts Entwined	2024-10-05	1	75	20.99	Robert	White	Macmillan Publishers

3.2.4. Getting details of orders:

```sql
use company;
SELECT
    o.id AS order_id,
    o.date,
    o.state,
    od.amount,
    b.price AS book_price
FROM
    orders o
JOIN
    orders_details od ON o.id = od.order_id
JOIN
    Books b ON od.book_id = b.id
WHERE
    o.state = 'sent' OR o.state = 'paid';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_id	date	state	amount	book_price
1	2023-01-01	Sent	2	19.99
1	2023-01-01	Sent	1	29.99
1	2023-01-01	Sent	3	39.99
2	2023-01-02	Paid	1	18.99
2	2023-01-02	Paid	2	23.99
4	2023-01-04	Paid	3	34.99
4	2023-01-04	Paid	2	23.99
6	2023-01-06	Paid	2	30.99
6	2023-01-06	Paid	1	40.99
8	2023-01-08	Paid	1	24.99
8	2023-01-08	Paid	2	17.99
10	2023-01-10	Paid	3	22.99

3.2.5. Calculating the current income:

```
use company;
DELIMITER //
DROP FUNCTION IF EXISTS calculateOrderIncome;
CREATE FUNCTION calculateOrderIncome(order_id INT) RETURNS DECIMAL(10, 2) DETERMINISTIC
BEGIN
    DECLARE totalIncome DECIMAL(10, 2);
    SELECT COALESCE(SUM(Books.price * orders_details.amount)) INTO totalIncome
    FROM orders_details
    JOIN Books ON orders_details.book_id = Books.id
    WHERE orders_details.order_id = order_id;
    RETURN totalIncome;
END //
DELIMITER ;
SELECT
    id AS order_id,
    calculateOrderIncome(id) AS order_income
FROM
    orders
WHERE
    state = 'sent' OR 'paid';
SELECT
    SUM(calculateOrderIncome(id)) AS total_income
FROM
    orders
WHERE
    state = 'sent' OR state = 'paid';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

total_income
1407.49

From the code in section 3.2.3, we improve it to calculate total income from the orders in the state: “Sent” or “Paid”

- Stored Function (calculateOrderIncome):
    + It takes an order_id as an argument.
    + Initializes a variable totalIncome to store the calculated income.
    + Uses a SELECT query to join orders_details and Books tables, calculating the income for the specified order by summing the product of book prices and order quantities.
    + Returns the calculated totalIncome.
- First SELECT Query:
    + Retrieves order IDs (id) and their corresponding incomes calculated using the calculateOrderIncome function.
    + Filters orders with the state 'sent' or 'paid'.
- Second SELECT Query:
    + Computes the total income for all orders with states 'sent' or 'paid'

3.2.6. Getting the price for any order:

This function calculate the price for a specific order, the order can be choose by replacing the number ‘n’ in the line:
SELECT calculateprice(n) AS Total_price

```
USE company;
DROP FUNCTION IF EXISTS calculateprice;
DELIMITER //
CREATE FUNCTION calculateprice(orderID INT) RETURNS DECIMAL(10, 2) DETERMINISTIC
BEGIN
    DECLARE Total_price DECIMAL(10, 2);

    -- Handle NULL values in case of no matching orders
    SELECT COALESCE(SUM(Books.price * orders_details.amount), 0) INTO Total_price
    FROM Books
    JOIN orders_details ON Books.id = orders_details.book_id
    WHERE orders_details.order_id = orderID;

    RETURN Total_price;
END //
DELIMITER ;

-- Calculate expected income for a specific order (replace '1' with the desired order ID)
select * from orders_details;
SELECT calculateprice(1) AS Total_price;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Total_price
189.94

This section is used to calculate the price for a specific order:

- Stored Function (calculateExpectedIncome):
    + Takes an orderID as an argument.
    + Initializes totalExpectedIncome to store the calculated income.
    + Uses a SELECT query to join the Books and orders_details tables, calculating the expected income for the specified order by summing the product of book prices and order quantities.
    + Returns the calculated totalExpectedIncome.
- Example Usage:
    + Outputs all rows from the orders_details table.
    + Executes a SELECT query to demonstrate the usage of calculateExpectedIncome for a specific order (order ID 3 in this example).

3.2.7. Book ranking table:

This function calculate the amount of books sold and arrange them by descending order:

```sql
-- Calculate total sales for each book and update books_rank_sales
INSERT INTO books_rank_sales (book_id, title, amount_sold)
SELECT
  Books.id AS book_id,
  Books.Title AS title,
  COALESCE(SUM(orders_details.amount), 0) AS total_sales
FROM
  Books
LEFT JOIN
  orders_details ON Books.id = orders_details.book_id
GROUP BY
  Books.id, Books.Title
ON DUPLICATE KEY UPDATE
  amount_sold = VALUES(amount_sold);
  select * from books_rank_sales order by amount_sold desc;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

book_id	title	amount_sold
13	The Lost Child	6
17	Infinite Love	6
5	The Elven Kingdom	6
13	The Lost Child	6
17	Infinite Love	6
5	The Elven Kingdom	6

- Explaining:

- INSERT INTO Statement:
  + Uses a SELECT query to calculate the total sales for each book.
  + Books.id AS book_id: Selects the book ID.
  + Books.Title AS title: Selects the book title.
  + FROM Books LEFT JOIN orders_details ON Books.id = orders_details.book_id: Joins the Books and orders_details tables, linking books to their corresponding sales details.
  + GROUP BY Books.id, Books.Title: Groups the results by book ID and title.
  + ON DUPLICATE KEY UPDATE amount_sold = VALUES(amount_sold): If there's a duplicate key (combination of book_id and title) in the

books_rank_sales table, update the amount_sold column with the calculated total sales.

- SELECT Statement:

  + Retrieves all rows from the books_rank_sales table after the update.
  + Orders the results by amount_sold in descending order.

4. Conclusion

In conclusion, the systematic grouping of tables by functional domain in the Online Bookstore Database ensures an organization that harmonizes with the core business processes of the online bookstore. This meticulous approach enhances data management practices, promotes ease of maintenance, and offers a lucid representation of the database's functional components. The proposed groupings lay the groundwork for a resilient and efficient database system precisely tailored to the distinctive needs of an online bookstore.