

## Introduction

This is a database schema that contains tables for managing a bookstore's operations, such as storing information on publishers, authors, books, customers, payments, and orders.

The PUBLISHER table contains information about the publishers of the books, with a primary key of PUBLISHER\_ID. The AUTHOR table contains information about the authors of the books, with a primary key of AUTHOR\_ID.

The BOOK table stores information about each book, including its unique identifier, name, release date, language, price, author, and publisher. It has foreign keys referencing the AUTHOR and PUBLISHER tables.

The CUSTOMER table contains information about the bookstore's customers, including their unique identifier, name, address, phone number, and surname.

The BASKET table represents the items in a customer's basket, with the customer's ID and book ID as a composite primary key, and the book's price.

The BANK table stores information about the bank cards, including the card number, expiration date, CVV code, owner's name, and balance.

The CARD\_HISTORY table contains information about the bank card transactions, including the card ID, transaction type, and amount.

The PAYMENT table stores information about each payment made by a customer, including the payment's unique identifier, payment method, card ID, payment amount, and commission.

Finally, the CUSTOMER\_ORDER table stores information about the orders made by customers, including the customer's ID, book ID, book price, address, order date, order status, and payment ID. The table has foreign keys referencing the CUSTOMER, BOOK, and PAYMENT tables.

## Explanation of why the structure follows normal forms

This database schema appears to follow the rules of normalization and is normalized to at least the third normal form (3NF).

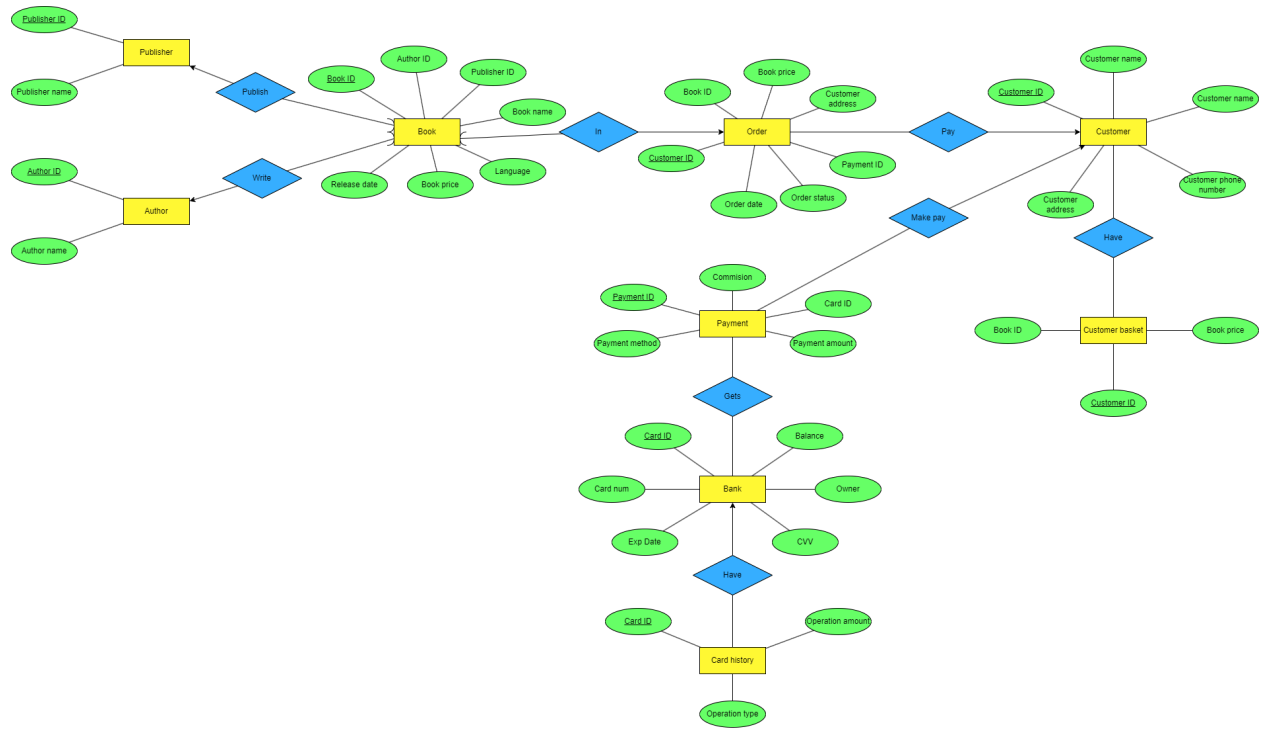
The first normal form (1NF) requires that each table has a primary key and that each column in the table contains atomic values. All the tables in this database have a primary key, and each column contains atomic values.

The second normal form (2NF) requires that each non-key column in a table is fully dependent on the primary key. In the BOOK table, for example, the non-key columns RELEASE\_DATE, LANGUAGE, BOOK\_PRICE are fully dependent on the primary key BOOK\_ID. Similarly, the non-key columns in other tables depend on the respective primary keys.

The third normal form (3NF) requires that there are no transitive dependencies between non-key columns. In other words, if a non-key column depends on another non-key column in the same table, the table should be split into two. The schema appears to satisfy this condition as there are no transitive dependencies in any of the tables.

Therefore, this database schema appears to follow the rules of normalization, and the tables are designed to be efficient and effective for storing and retrieving information in a well-organized and structured manner.

# ERD



## Function

```
1 CREATE OR REPLACE FUNCTION totalOfBasketOfCustomer(id in number)
2 RETURN number IS
3     total number;
4 BEGIN
5     SELECT sum(BOOK_PRICE) into total FROM BASKET where CUSTOMER_ID = id;
6     RETURN total;
7 END;
8
9 CREATE OR REPLACE FUNCTION basketCountOfCustomer(id in number)
10 RETURN number IS
11     total number;
12 BEGIN
13     SELECT count(*) into total FROM BASKET where CUSTOMER_ID = id;
14     RETURN total;
15 END;
16
17 DECLARE
18     s number(3);
19     c number(2);
20     c_id numeric(10);
21 BEGIN
22     c_id := :id;
23     s := totalOfBasketOfCustomer(id: c_id);
24     c := basketCountOfCustomer(id: c_id);
25     dbms_output.put_line('A: Total of basket of ' || c_id || ' is ' || s || ' USD. Item count is ' || c || '.');
26 END;
```

The first function `totalOfBasketOfCustomer(id in number)` takes a customer ID as an argument and returns the total price of all the books in the customer's basket. It does this by selecting the sum of the `BOOK_PRICE` column from the `BASKET` table where the `CUSTOMER_ID` matches the input parameter. The result is then returned as a number.

The second function `basketCountOfCustomer(id in number)` takes a customer ID as an argument and returns the number of items in the customer's basket. It does this by selecting the count of all the rows from the `BASKET` table where the `CUSTOMER_ID` matches the input parameter. The result is then returned as a number.

The block of code declares three variables: `s` to hold the total price, `c` to hold the item count, and `c_id` to hold the customer ID. It then sets `c_id` to the input parameter and calls the two functions to calculate the total price and the item count for the given customer ID. The results are then printed to the console using the `dbms_output.put_line()` function.

Overall, these functions and code blocks can be used to retrieve information about the contents of a customer's basket, which could be useful for generating reports or for other business purposes.

# Trigger

```
1 CREATE OR REPLACE TRIGGER BOOK_PRICE_TRIG
2 BEFORE INSERT ON BOOK
3 FOR EACH ROW
4 DECLARE
5     c NUMBER;
6 BEGIN
7     SELECT COUNT(*) INTO c FROM BOOK;
8     DBMS_OUTPUT.PUT_LINE('Before insert book table has ' || c || ' values.');
```

```
9 END;
10
11 INSERT INTO BOOK (BOOK_ID, BOOK_NAME, RELEASE_DATE, LANGUAGE, BOOK_PRICE, AUTHOR_ID, PUBLISHER_ID)
12 VALUES (1, 'TEST', TO_DATE('2022/04/01 16:09:08', 'yyyy/mm/dd hh24:mi:ss'), 'KZ', 150, 573422699, 1534335471);
```

This is a trigger in the Oracle database that is designed to execute automatically before an insert operation is performed on the **BOOK** table. The trigger is defined to execute for each row that is inserted into the **BOOK** table.

The trigger starts with a declaration section that defines a variable **c** of type **NUMBER**. Then, inside the **BEGIN** and **END** block of the trigger, a SQL statement is used to select the count of rows in the **BOOK** table and assign it to the variable **c**. Finally, the **DBMS\_OUTPUT.PUT\_LINE** statement is used to display the number of rows in the **BOOK** table before a new row is inserted.

When a new row is inserted into the **BOOK** table, this trigger will fire and display the count of rows in the **BOOK** table before the insertion is performed. In this case, the **INSERT** statement inserts a new row into the **BOOK** table and sets the values for the columns **BOOK\_ID**, **BOOK\_NAME**, **RELEASE\_DATE**, **LANGUAGE**, **BOOK\_PRICE**, **AUTHOR\_ID**, and **PUBLISHER\_ID**.

As a result, before the new row is inserted into the **BOOK** table, the trigger will execute and display the number of rows in the **BOOK** table at that time.

## Procedure

```
1  create or replace procedure lastprice(id in numeric, price in number, row out numeric)
2  is
3  begin
4      update book set BOOK_PRICE = BOOK_PRICE + price where PUBLISHER_ID = id;
5      if sql%found then
6          row := sql%rowcount ;
7      else
8          row := 0;
9      end if;
10 end;
11
12 declare
13     p_id number;
14     b_price number;
15     rowcount number;
16     begin
17         p_id := : "PUBLISHER ID";
18         b_price := : "CHANGE PRICE BY";
19
20         lastprice(id: p_id, price: b_price, row: rowcount);
21
22         DBMS_OUTPUT.PUT_LINE('Changed price of ' || rowcount || ' books. ');
23     end;
```

```
1  CREATE OR REPLACE PROCEDURE countDeliverStatus(x IN varchar, y OUT number) IS
2  BEGIN
3      SELECT count(CUSTOMER_ID) INTO y from CUSTOMER_ORDER WHERE ORDER_STATUS = x GROUP BY ORDER_STATUS;
4  END;
5
6  DECLARE
7      dlvr_status varchar(50);
8      cstmr_count number;
9  BEGIN
10     dlvr_status := : "STATUS";
11     countDeliverStatus(x: dlvr_status, y: cstmr_count);
12     dbms_output.put_line('A: ' || 'We have ' || cstmr_count || ' orders with "' || dlvr_status || '" status');
13 END;
```

The first procedure is named `lastprice`. The procedure takes three parameters: "id" (numeric data type), "price" (number data type), and "row" (numeric data type as an output parameter).

The procedure updates the "BOOK\_PRICE" column in the "book" table, increasing the price by the value of the "price" parameter for all books that have a "PUBLISHER\_ID" equal to the "id" parameter. The "UPDATE" statement is followed by an "IF" statement that checks whether any rows were affected by the update. If rows were updated, the number of affected rows is stored in the "row" parameter. Otherwise, the "row" parameter is set to zero.

In summary, this procedure updates the price of books from a particular publisher and returns the number of books whose price was updated.

The second procedure, `countDeliverStatus`, takes an input parameter `x` of type `varchar`, which represents the delivery status, and an output parameter `y` of type `number`. It then performs a select statement on the `customer_order` table to retrieve the count of customer orders that have the given delivery status. The count is stored in the `y` output parameter. This procedure can be used to get the count of customer orders with a specific delivery status.

# Exception

```
1 declare
2     a_id number;
3     authorInvalid exception;
4     tooShortID exception;
5     blank BOOK.AUTHOR_ID%TYPE;
6 begin
7     a_id := "AUTHOR ID";
8     if a_id < 9 then
9         raise tooShortID;
10    end if;
11
12    SELECT AUTHOR_ID INTO BLANK FROM BOOK WHERE AUTHOR_ID = a_id GROUP BY AUTHOR_ID;
13
14    if sql%notfound then
15        raise authorInvalid;
16    else
17        DBMS_OUTPUT.PUT_LINE('The author sells books in our store.');
```

This is a PL/SQL block that checks if an author ID exists in the BOOK table of the database. Here is a step-by-step explanation of how it works:

1. Declare variables: The variables declared at the beginning of the block are **a\_id**, **authorInvalid**, **tooShortID**, and **blank**. **a\_id** is the input parameter that holds the author ID that needs to be checked. **authorInvalid** and **tooShortID** are user-defined exceptions that can be raised when there is an error. **blank** is an empty variable that is used to check if the author ID exists in the database.
2. Check author ID length: The block checks if the **a\_id** variable is less than 9. If it is less than 9, it raises a user-defined exception called **tooShortID**.
3. Check author ID in the database: The block tries to select **AUTHOR\_ID** from the **BOOK** table, where **AUTHOR\_ID** equals the **a\_id** variable. If there is no row returned by the query, the **sql%notfound** attribute returns **TRUE**, and the block raises a user-defined exception called **authorInvalid**.
4. Handle exceptions: If any of the exceptions are raised, the block handles them and raises an application error with a specific error message. If no exceptions are raised, the block prints a message to the console indicating that the author sells books in their store.

In summary, this PL/SQL block validates an author ID by checking if it exists in the **BOOK** table and has a minimum length of 9. If the ID is invalid or does not exist, it raises an exception and returns an error message.