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).

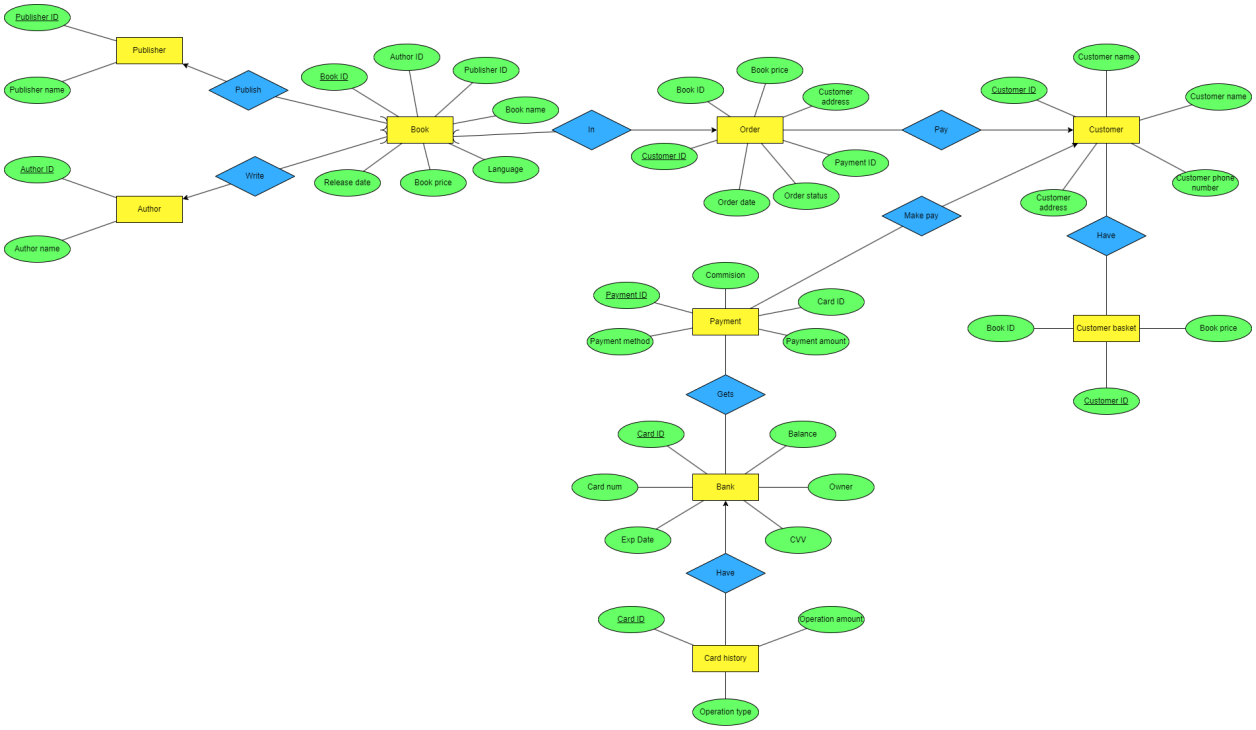
The first normal form (1NF) requires that each table has a primary key and that each column in the table contains atomic values. All the tables in this database have a primary key, and each column contains atomic values.

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


Publisher ID
Publisher
Publisher name
Publish
Author ID
Book ID
Publisher ID
Book
Book name
Release date
Book price
Language
Write
Author ID
Author
Author name
In
Book ID
Book price
Customer address
Order
Customer ID
Payment ID
Order date
Order status
Pay
Customer name
Customer ID
Customer name
Customer
Customer phone number
Customer address
Have
Make pay
Commision
Payment ID
Card ID
Payment
Payment method
Payment amount
Gets
Book ID
Customer basket
Book price
Customer ID
Card ID
Balance
Card num
Bank
Owner
Exp Date
CVV
Have
Card ID
Operation amount
Card history
Operation type

Function

```
CREATE OR REPLACE FUNCTION totalOfBasketOfCustomer(id in number)
RETURN number IS
    total number;
BEGIN
    SELECT sum(BOOK_PRICE) into total FROM BASKET where CUSTOMER_ID = id;
    RETURN total;
END;

CREATE OR REPLACE FUNCTION basketCountOfCustomer(id in number)
RETURN number IS
    total number;
BEGIN
    SELECT count(*) into total FROM BASKET where CUSTOMER_ID = id;
    RETURN total;
END;

DECLARE
    s number(3);
    c number(2);
    c_id numeric(10);
BEGIN
    c_id := :id;
    s := totalOfBasketOfCustomer( id: c_id);
    c := basketCountOfCustomer( id: c_id);
    dbms_output.put_line( A: 'Total of basket of ' || c_id || ' is ' || s || ' USD. Item count is ' || c || '.');
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

```
CREATE OR REPLACE TRIGGER BOOK_PRICE_TRIG
BEFORE INSERT ON BOOK
    FOR EACH ROW
DECLARE
    c NUMBER;
BEGIN
    SELECT COUNT(*) INTO c FROM BOOK;
    DBMS_OUTPUT.PUT_LINE( A 'Before insert book table has ' || c || ' values.');
END;

INSERT INTO BOOK (BOOK_ID, BOOK_NAME, RELEASE_DATE, LANGUAGE, BOOK_PRICE, AUTHOR_ID, PUBLISHER_ID)
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

```
CREATE OR REPLACE PROCEDURE countofBooks(x IN number, auth_name OUT varchar, y OUT number) IS
BEGIN
    SELECT count(*) INTO y from book WHERE AUTHOR_ID = x;
    SELECT AUTHOR_NAME INTO auth_name FROM AUTHOR WHERE AUTHOR_ID = x;
END;

DECLARE
    auth_id number;
    auth_name varchar(50);
    s number;
BEGIN
    auth_id := :"AUTHOR ID";
    countofBooks( x: auth_id,  auth_name: auth_name,  y: s);
    dbms_output.put_line( A: ' The author ' || auth_name || ' has ' || s || ' books.');
END;
```

```
CREATE OR REPLACE PROCEDURE countDeliverStatus(x IN varchar, y OUT number) IS
BEGIN
    SELECT count(CUSTOMER_ID) INTO y from CUSTOMER_ORDER WHERE ORDER_STATUS = x GROUP BY ORDER_STATUS;
END;

DECLARE
    dlvr_status varchar(50);
    cstmr_count number;
BEGIN
    dlvr_status := :"STATUS";
    countDeliverStatus( x: dlvr_status,  y: cstmr_count);
    dbms_output.put_line( A: 'We have ' || cstmr_count || ' orders with "' || dlvr_status || '" status');
END;
```

The first procedure, `countofBooks`, takes an input parameter `x` of type number, which represents the author ID, and two output parameters `auth_name` of type varchar and `y` of type number. It then performs two select statements on the `book` and `author` tables, respectively, to retrieve the count of books written by the author with the given ID and the name of the author. The count of books is stored in the `y` output parameter and the author name is stored in the `auth_name` output parameter. This procedure can be used to get the count of books written by a specific author.

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

```
declare
    a_id  number;
    authorInvalid exception;
    tooShortID exception;
    blank BOOK.AUTHOR_ID%TYPE;
begin
    a_id := :"AUTHOR ID";
    if a_id < 9 then
        raise tooShortID;
    end if;

    SELECT AUTHOR_ID INTO BLANK FROM BOOK WHERE AUTHOR_ID = a_id GROUP BY AUTHOR_ID;

    if sql%notfound then
        raise authorInvalid;
    else
        DBMS_OUTPUT.PUT_LINE( A 'The author sells books in our store.');
    end if;

EXCEPTION
    WHEN tooShortID THEN
        raise_application_error(-20227, 'Author ID is too short. Try at least 9 numbers.');
    WHEN authorInvalid THEN
        raise_application_error(-20228, 'Invalid Author ID.');
end;
```

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.