Introduction

A Book Inventory System is essential for managing the operations of libraries, bookstores, and other organizations that handle books. Utilizing a Database Management System (DBMS) enhances the efficiency of tracking book inventory, ensuring accurate and current information on book availability, location, and status.

The main goal of a Book Inventory System is to keep an organized record of all books in an inventory. This includes details such as titles, authors, genres, publication dates, and current status (e.g., available, checked out, reserved). With a DBMS, the system can perform complex queries, updates, and generate reports efficiently, tasks which would be error-prone if done manually.

1.Cataloging: Comprehensive metadata for easy search and retrieval of books

2.Tracking: Real-time status and location tracking of books.

3.User Management: Managing information related to users, such as borrowers or customers.

4.Transactions Management: Recording and managing book loans, returns, and reservations.

5.Reporting: Generating reports on inventory levels, borrowing patterns, and other metrics.

Entities:

1.Book
Attributes: ISBN, Title, Author, Publisher, Publication Date, Genre, Price, Stock Quantity

2.Author:
Attributes: Author ID, Name, Biography, Date of Birth, Nationality

3.Publisher:
Attributes: Publisher ID, Name, Address, Contact Information

4.Customer:
- Attributes: Customer ID, Name, Email, Phone Number, Address, Purchase History

5. Order:
Attributes: Order ID, Customer ID, Order Date, Total Amount, List of Books (with quantities), Order Status

Relationships:

1. Book and Author
Relationship: Many-to-Many

2. Book and Publisher:
Relationship: Many-to-One

3. Customer and Order:
Relationship: One-to-Many

4. Order and Book:
Relationship: Many-to-Many

5. Publisher and Author:

Relationship: Many-to-Many (optional, if tracking)

Relationship Explanation:

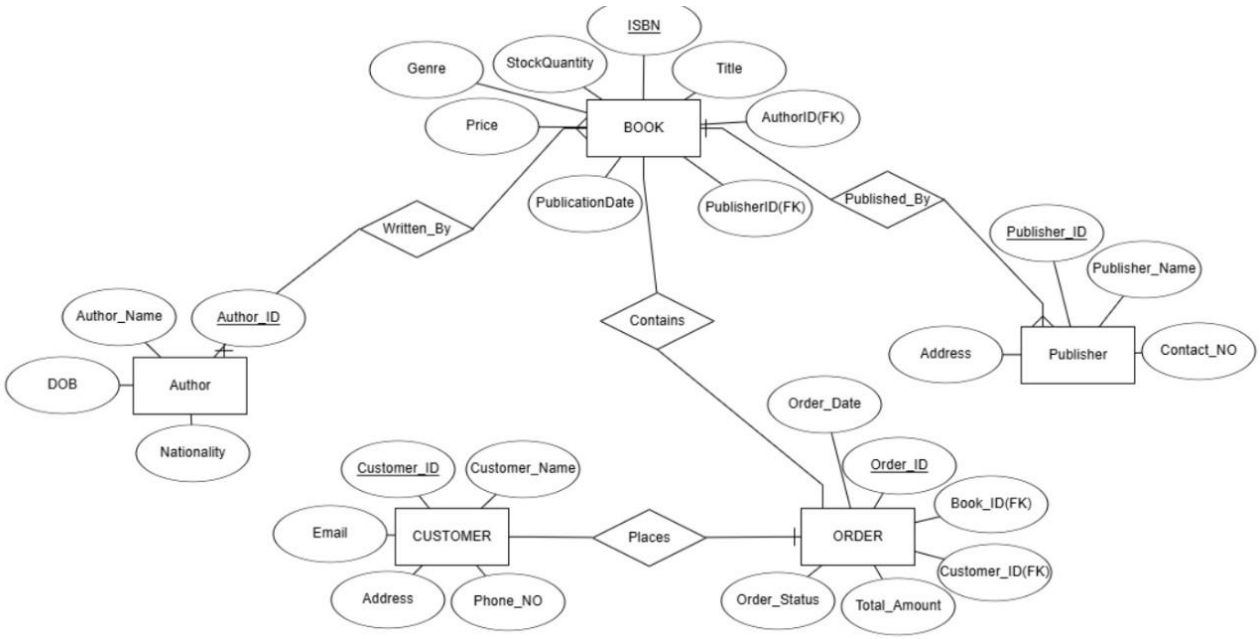
Author has a many-to-many relationship with Book through Book_Author

Book has a many-to-one relationship with Publisher

Order has a many-to-many relationship with Book through Order_Book

Customer has a one-to-many relationship with Order

Entity-Relationship Diagram:



Relational Schema Diagram

B_ID	TITLE	AUTHOR_ID	PUBLISHER_ID	PUBLICATION_DATE	PRICE	GENRE	STOCK_QNTY

Author

AUTHOR_ID	AUTHOR_NAME	DOB	NATIONALITY

Publisher

PUBLISHER_ID	PUBLISHER_NAME	CONTACT_NO	ADDRESS

Order

CUSTOMER_ID	CUSTOMER_NAME	EMAIL	ADDRESS	PHONE_NO

Customer

ORDER_ID	ORDER_DATE	BOOK_ID	CUSTOMER_ID	TOTAL_AMOUNT	ORDER_STATUS

Logical Structure:

1. Query for creating table Author :

```
CREATE TABLE Author (AuthorID INT generated always as identity,
Name VARCHAR(255) NOT NULL,
DateOfBirth DATE,Nationality VARCHAR(100),
primary key(AuthorID
));
```

Output :

```
Table created.

SQL> desc author;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 AUTHORID                                  NOT NULL NUMBER(38)
 NAME                                      NOT NULL VARCHAR2(255)
 DATEOFBIRTH                                        DATE
 NATIONALITY                                        VARCHAR2(100)
```

2. Query for creating table Publisher :

```
CREATE TABLE Publisher (PublisherID INT generated always as identity,
Name VARCHAR(255) NOT NULL,
Address varchar(30),
ContactInformation VARCHAR(255),
primary key(publisherid
));
```

Output :

```
SQL> desc publisher;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PUBLISHERID                               NOT NULL NUMBER(38)
 NAME                                      NOT NULL VARCHAR2(15)
 ADDRESS                                            VARCHAR2(25)
 CONTACTINFORMATION                                 VARCHAR2(15)
```

3. Query for creating table Book :

```
CREATE TABLE Book1 (BookID INT generated always as identity,
 ISBN VARCHAR(20) UNIQUE NOT NULL,
 Title VARCHAR(255) NOT  NULL,PublisherID INT,
 PublicationDate DATE,
 Genre VARCHAR(100),
 Price DECIMAL(10, 2),StockQuantity INT,FOREIGN KEY (PublisherID)
REFERENCES Publisher(PublisherID),
 primary key(bookid)
);
```

```
SQL> desc book1;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 BOOKID                                    NOT NULL NUMBER(38)
 ISBN                                      NOT NULL VARCHAR2(20)
 TITLE                                     NOT NULL VARCHAR2(255)
 PUBLISHERID                                        NUMBER(38)
 PUBLICATIONDATE                                    DATE
 GENRE                                              VARCHAR2(100)
 PRICE                                              NUMBER(10,2)
 STOCKQUANTITY                                      NUMBER(38)
```

4. Query for creating table Customer :

CREATE TABLE Customer (CustomerID INT generated always as identity,Name VARCHAR(255) NOT NULL,
Email VARCHAR(255) UNIQUE NOT NULL,
PhoneNumber VARCHAR(20),
Address varchar(10),
primary key(customerid)
);
Output:

```
SQL> desc customer;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUSTOMERID                                NOT NULL NUMBER(38)
 NAME                                      NOT NULL VARCHAR2(255)
 EMAIL                                     NOT NULL VARCHAR2(255)
 PHONENUMBER                                        VARCHAR2(20)
 ADDRESS                                            VARCHAR2(10)
```

5. Query for creating table Order :

CREATE TABLE Order2 (OrderID INT generated always as identity,
CustomerID INT,
OrderDate DATE NOT NULL, TotalAmount DECIMAL(10, 2) NOT NULL,
OrderStatus varchar(40),
FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
primary key(orderid)
);

```
SQL> CREATE TABLE Order2 (OrderID INT generated always as identity,CustomerID INT, OrderDate DATE NOT NULL, TotalAmount DECIMAL(10, 2) NOT NULL,    O
rderStatus varchar(40),FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),primary key(orderid));

Table created.

SQL> desc order2;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ORDERID                                   NOT NULL NUMBER(38)
 CUSTOMERID                                         NUMBER(38)
 ORDERDATE                                 NOT NULL DATE
 TOTALAMOUNT                               NOT NULL NUMBER(10,2)
 ORDERSTATUS                                        VARCHAR2(40)
```

COMPLEX QUERIES:

1. Query for Calculate the total sales for each publisher

```
SELECT p.Name AS PublisherName, SUM(b.Price) AS TotalSales
FROM Publisher p
JOIN Book1 b ON p.PublisherID = b.PublisherID
GROUP BY p.PublisherID, p.Name;
```

Output:

```
SQL> SELECT p.Name AS PublisherName, SUM(b.Price) AS TotalSales FROM Publisher p JOIN Book1 b ON p.PublisherID = b.PublisherID GROUP BY p.PublisherID
, p.Name;

PUBLISHERNAME   TOTALSALES
--------------- ----------
Random House         59.97
manglo House         59.97
```

This SQL query is designed to calculate the total sales for each publisher by joining the Publisher and Book1 tables based on their PublisherID columns.

i. SELECT Clause:
- p.Name AS PublisherName: Selects the Name column from the Publisher table and aliases it as PublisherName.
- SUM(b.Price) AS TotalSales: Calculates the sum of the Price column from the Book1 table for each publisher and aliases it as TotalSales.

ii. FROM Clause:
- Publisher p: Specifies the Publisher table with an alias p.
- JOIN Book1 b ON p.PublisherID = b.PublisherID: Joins the Publisher table (p) with the Book1 table (b) using the PublisherID column, which exists in both tables.

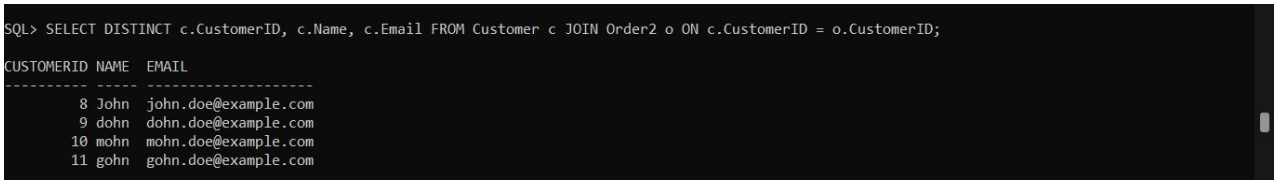
iii.GROUP BY Clause:

GROUP BY p.PublisherID, p.Name: Groups the result set by PublisherID and Name columns from the Publisher table (p). This means that the aggregation function (SUM in this case) will be applied for each unique combination of PublisherID and Name.

2.Query for Distinct Customer Information :

```
SELECT DISTINCT c.CustomerID, c.Name, c.Email
FROM Customer c
JOIN Order2 o ON c.CustomerID = o.CustomerID;
```

Output:



This SQL query retrieves distinct customer information who have placed orders by joining the Customer and Order2 tables based on their CustomerID column.

i. SELECT DISTINCT Clause:

-SELECT DISTINCT c.CustomerID, c.Name, c.Email: Selects distinct rows based on the combination of CustomerID, Name, and Email from the Customer table (c). This ensures that each combination of these columns appears only once in the result set.

ii. FROM Clause:

- FROM Customer c: Specifies the Customer table with an alias c.

iii. JOIN Clause:

- JOIN Order2 o ON c.CustomerID = o.CustomerID: Joins the Customer table (c) with the Order2 table (o) based on the CustomerID column. This establishes a relationship where each customer who has placed an order (CustomerID present in both tables) will be included in the result set.

3.Query for information about Book

SELECT BookID, Title, PublicationDate

FROM Book1WHERE PublicationDate > TO_DATE('2023-01-01',

'YYYY-MM-DD');

Output:

```
SQL> SELECT BookID, Title, PublicationDate FROM Book1 WHERE PublicationDate > TO_DATE('2023-01-01', 'YYYY-MM-DD');

    BOOKID TITLE                     PUBLICATI
---------- ------------------------- ---------
         2 The Great Adventure       15-JAN-23
         5 The Great Adventure       16-FEB-24
         6 The Great Adventure       17-FEB-24
         7 The Great Adventure       18-FEB-24
         8 The Great Adventure       14-FEB-24
         9 The Great Adventure       14-FEB-25

6 rows selected.
```

This SQL query retrieves information about books from the Book1 table where the PublicationDate is greater than January 1st, 2023.

i. SELECT Clause:

- SELECT BookID, Title, PublicationDate: Specifies the columns to be retrieved from the Book1 table. It selects BookID, Title, and PublicationDate columns.

ii.FROM Clause:

- FROM Book1: Specifies the Book1 table from which data is retrieved.

iii. WHERE Clause:

- WHERE PublicationDate > TO_DATE('2023-01-01', 'YYYY-MM-DD'): Filters the rows where the PublicationDate column is greater than January 1st, 2023. The TO_DATE function converts the string '2023-01-01' into a date format that Oracle can understand (YYYY-MM-DD).

4.Query For top 5 customers based on their total spending from the Customer and Order2 tables:

SELECT c.CustomerID, c.Name, c.Email, SUM(o.TotalAmount) AS TotalSpent

FROM Customer c

JOIN Order2 o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID, c.Name, c.Email

ORDER BY TotalSpent DESC

FETCH FIRST 5 ROWS ONLY;

Outputs:

```
SQL> SELECT c.CustomerID, c.Name, c.Email, SUM(o.TotalAmount) AS TotalSpent FROM Customer c JOIN Order2 o ON c.CustomerID = o.CustomerID GROUP BY c.C
ustomerID, c.Name, c.Email ORDER BY TotalSpent DESC FETCH FIRST 5 ROWS ONLY;

CUSTOMERID NAME  EMAIL               TOTALSPENT
---------- ----- ------------------- ----------
         8 John  john.doe@example.com     471.25
         9 dohn  dohn.doe@example.com        200
        11 gohn  gohn.doe@example.com        150
        10 mohn  mohn.doe@example.com         85
```

This SQL query retrieves the top 5 customers based on their total spending from the Customer and Order2 tables.

i. SELECT Clause:

- SELECT c.CustomerID, c.Name, c.Email, SUM(o.TotalAmount) AS TotalSpent: Specifies the columns to be retrieved. It selects CustomerID, Name, and Email from the Customer table (c). Additionally, it calculates the total spending (SUM(o.TotalAmount)) for each customer and aliases it as TotalSpent.

ii. FROM Clause:

- FROM Customer c: Specifies the Customer table with an alias c.

iii. JOIN Clause:

- JOIN Order2 o ON c.CustomerID = o.CustomerID: Joins the Customer table (c) with the Order2 table (o) based on the CustomerID column. This establishes a relationship where each order (o) is associated with its respective customer (c).

iv. GROUP BY Clause:

- GROUP BY c.CustomerID, c.Name, c.Email: Groups the result set by CustomerID, Name, and Email columns from the Customer table (c). This ensures that the SUM function calculates the total amount spent for each unique customer.

v. ORDER BY Clause:

- ORDER BY TotalSpent DESC: Sorts the result set by the TotalSpent column in descending order. This means the customers with the highest total spending will appear first.

vi. FETCH FIRST 5 ROWS ONLY:

- FETCH FIRST 5 ROWS ONLY: Limits the result set to the first 5 rows, ensuring that only the top 5 customers based on total spending are returned.

5.Query for calculates the average price of books published by each publisher

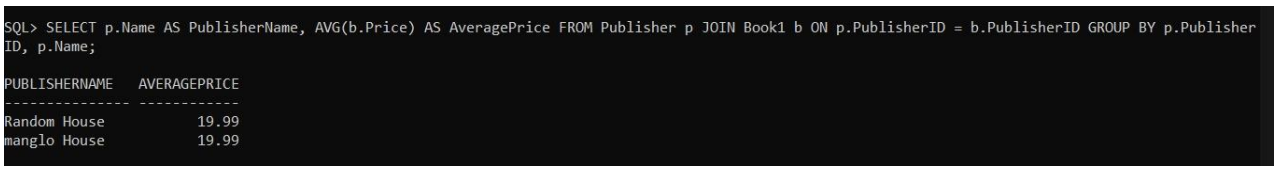
```
SELECT p.Name AS PublisherName, AVG(b.Price) AS AveragePrice

FROM Publisher p

JOIN Book1 b ON p.PublisherID = b.PublisherID

GROUP BY p.PublisherID, p.Name;
```

Output:



This SQL query calculates the average price of books published by each publisher. :

i. SELECT Clause:

- SELECT p.Name AS PublisherName, AVG(b.Price) AS AveragePrice: Specifies the columns to be retrieved in the result set. It selects the Name column from the Publisher table aliased as PublisherName, and calculates the average (AVG) of the Price column from the Book1 table aliased as AveragePrice.

ii. FROM Clause:

- FROM Publisher p: Specifies the Publisher table with an alias p.

iii. JOIN Clause:

- JOIN Book1 b ON p.PublisherID = b.PublisherID: Joins the Publisher table (p) with the Book1 table (b) based on the PublisherID column. This establishes a relationship where each book (b) is associated with its respective publisher (p).

iv. GROUP BY Clause:

- GROUP BY p.PublisherID, p.Name: Groups the result set by PublisherID and Name columns from the Publisher table (p). This means that the AVG function will be applied for each unique combination of PublisherID and Name.

CONCLUSION

In summary, the Entity-Relationship (ER) diagram is an essential tool for designing and understanding the structure of a database system.

This ER diagram provides a clear and comprehensive blueprint for developing a Book Management System, ensuring that the database is logically structured and capable of handling various operations related to books, customers, and transactions. Understanding and implementing such a diagram is fundamental to building robust, scalable, and efficient database systems.

Our Bookstore Inventory System aims to provide a robust and scalable solution for managing the dynamic requirements of a modern bookstore. By implementing this system, bookstores can achieve better inventory control, improved customer service, and streamlined operations, ultimately contributing to their growth and success.