BOOKSTORE MANAGEMENT SYSTEM

UCS310 Database Management Project File

END-Semester Evaluation

Submitted by:

NAME:	Roll Number:
Pridhi Singla	102117001
Pranjali Sharma	102117003
Jasmine Das	102117016
Nandini Naithani	102297003

BE Second Year, CSE

Group No: 2CS1

Submitted to: Dr. Payal Goyal

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

TABLE OF CONTENTS

1. REQUIREMENT ANALYSIS

The bookstore wants to develop a database management system to manage information about their books, customers, and orders. The system should allow the bookstore to add, update, and delete information about books, customers, and orders. It should also enable the bookstore to retrieve information about books, customers, orders, and generate reports based on the stored information.
To fulfill these requirements, we need to design a database schema that includes tables for books, customers, orders, and order items. The books table will store information about books sold by the bookstore, including the book ID, title, author, publisher, price, and stock. The customers table will store information about the bookstore's customers, including the customer ID, name, email address, and phone number. The orders table will store information about orders made by customers, including the order ID, the customer ID who made the order, and the order date. The order items table will store information about the books ordered by customers, including the order ID, book ID, and quantity.

The database management system should allow the bookstore to perform the following operations:

1. Add, update, and delete books from the books table.

2. Add, update, and delete customers from the customer table.

3. Add, update, and delete orders from the orders table.

4. Add, update, and delete order items from the order items table.

5. Retrieve information about books in stock, including the book ID, title, author, publisher, price, and stock.

6. Retrieve information about customers who have placed orders, including the customer ID, name, email address, and phone number.

7. Retrieve information about orders made by customers, including the order ID, the customer ID who made the order, and the order date.

8. Generate reports based on the stored information, such as the total revenue generated by a book, the top 5 best-selling books, and the number of orders made on a specific date.

We also need to write SQL queries to retrieve the required information from the database, such as retrieving all books in stock, retrieving all customers who have placed orders, retrieving the total revenue generated by a book, retrieving the top 5 best-selling books, and retrieving the number of orders made on a specific date. These queries should be optimized for performance and accuracy to ensure that the system can handle large amounts of data efficiently.

The database management system should also ensure the data integrity and consistency by enforcing constraints, such as primary keys, foreign keys, and unique keys. It should also provide efficient indexing and querying capabilities to handle large amounts of data efficiently. Finally, the system should be scalable, secure, and easy to use by the bookstore staff.

1. Security:
   - Implement user authentication and access control to ensure that only authorized users can access and modify the database.
   - Implement backup and recovery mechanisms to protect against data loss or corruption.

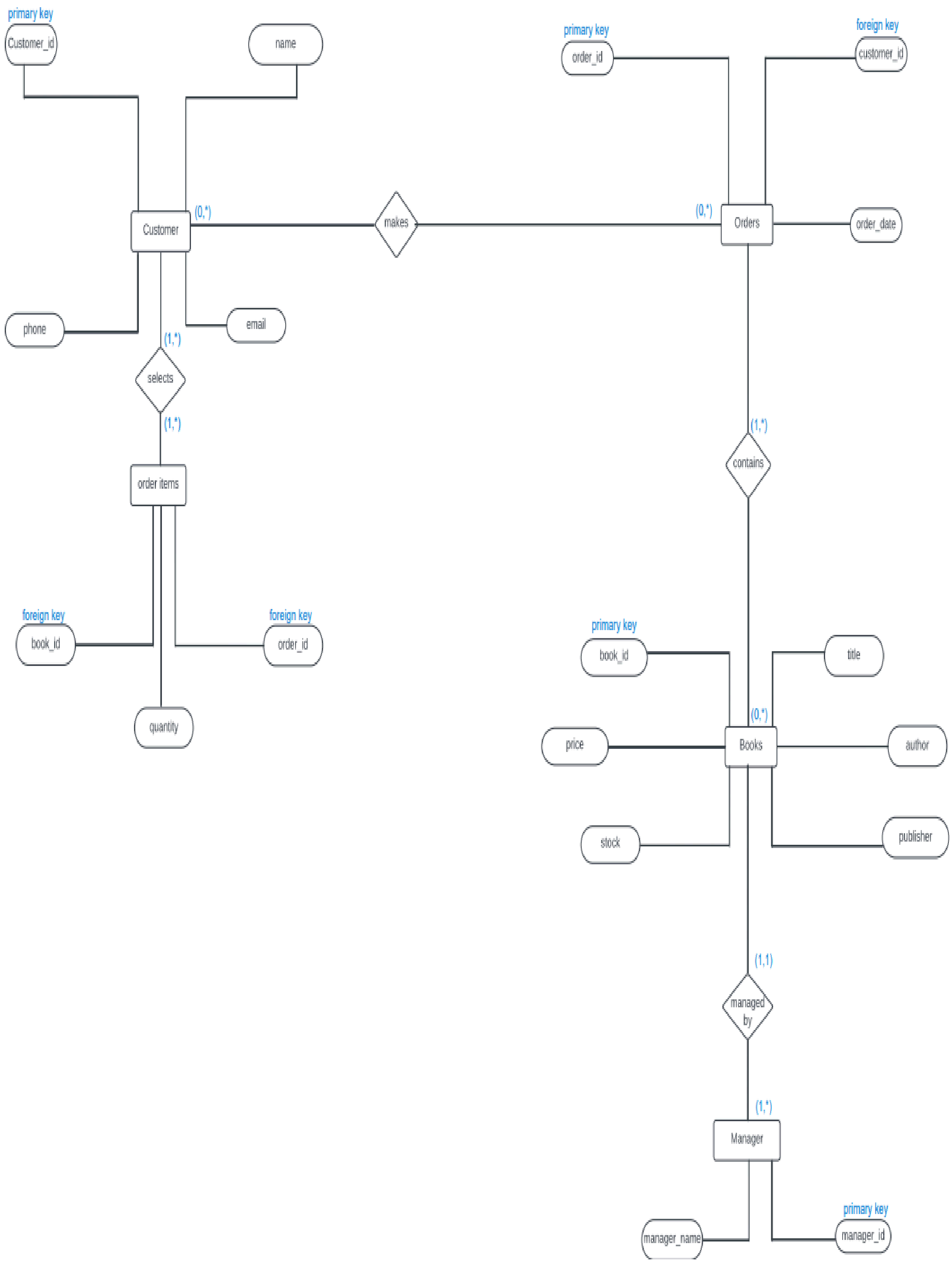
2.Scalability:

- Design the database schema and queries to handle a large volume of data efficiently.
- Implement a distributed database architecture to support multiple locations or online sales channels.

3.Usability:

- Design a user-friendly interface that allows bookstore staff to perform operations and generate reports easily.
- Provide user documentation and training to ensure that bookstore staff can use the system effectively

2. ER DIAGRAM



3. ER TO TABLES

1. **The "Customers" table has a many-to-many relationship with the "Orders" table. Here, 3 tables will be required**

- **Customer (<u>Customer_id</u> , name ,email ,phone)**

- Makes (<u>order_id</u>, <u>Customer_id</u>)
- Order (<u>order_id</u>, Customer_id, order_date)

Customer

<u>Customer_id</u>	name	email	phone

Orders

<u>order_id</u>	Customer_id	order_date

Makes

<u>Customer_id</u>	<u>Order_id</u>

2.

The Customers" table has a many-to-many relationship with the "Order items" table. Here, 3 tables will be required.

- Customer (<u>Customer_id</u> ,Customer_name ,customer_email ,phone_numb)
- Selects (<u>Customer_id</u> <u>,Order_id)</u>
- Order items(order_id, book_id,quantity)

Customer

<u>Customer_id</u>	name	email	phone

Order items

order_id	quantity	book_id

Selects

<u>Customer_id</u>	<u>order_id</u>

The "Book" table has a many-to-many relationship with the "Orders" table.
Here, 3 tables will be required.

- **Book (book_id, title, author, price, publisher, stock)**
- **Contains (book_id, order_id)**
- **Order (order_id, Customer_id, order_date)**

Book

book_id	title	author	publisher	stock	price

Orders

order_id	Customer_id	order_date

Contains

book_id	order_id

4.
The "Manager" table has a many-to-one relationship with the "Book" table.
Here, 2 tables will be required.

- **Manager (manager_id, book_id, manager_name)**
- **Book (book_id,book_title, author_name, publisher, stock, price)**

Manager

manager_id	Manager_name

Book

book_id	manager_id	title	author	publisher	stock	price

Therefore the final tables formed are:

Customer:

Customer_id	name	email	phone

Orders

order_id	Customer_id	order_date

5

Order items

order_id	quantity	book_id

Book

book_id	manager_ id	title	author	publisher	stock	price

Manager

manager_id	Manager_name

Contains

book_id	order_id

Makes

Customer_id	Order_id

4. NORMALISATION

First we find all the Functional Dependencies:

For the books table:
• book_id → {title, author_name, publisher, price, stock, manager_id}
• book_title, author_name → {publisher, price, stock}
For the customer table:
• customer_id → {name, customer_email, phone_number}
For the orders table:
• order_id → {customer_id, order_date}
For the order_items table:
• {order_id, book_id} → quantity
For the manager table:
• manager_id → manager_name

1NF
All the tables are already in 1NF because there are no multivalued / composite attributes in the tables, and each cell contains only atomic values.

2NF
All the tables are in 1NF and there is no partial dependency of any non-primary key attribute on the primary key. So it is already in 2NF.

3NF
In the books table, we have
• book_id → {book_title, author_name, publisher, price, stock, manager,_id}
• book_title, author_name → {publisher, price, stock}
Where book_id is the primary key. Hence there is transitive dependency.
To remove this we remove transitive dependent attributes from the relation that violets 3NF and place them in a new relation along with the non-prime attributes due to which transitive dependency occurred.

So new tables formed are:

Book

book_id	manager_id	book_title	author_name

7

Book_Details

book_title		author_name	book_id	Publisher	Stock	Price

Where book_id is the foreign key of book_details referencing book table.

In rest of the tables, there is no transitive dependency and are already in 2NF, hence they are in 3NF.

BCNF

All tables are in 3NF and for every functional dependency X ⟶ Y, X is the primary key of the table. Hence it is in BCNF.

4NF

All the tables are in BCNF and there are no multivalued dependencies. So all tables are in 4NF.

5NF

All the tables are in 4NF and they cannot have a lossless decomposition in to any number of smaller tables. So it is in 5NF.

5. SQL AND PL SQL

```
CREATE TABLE manager (
   manager_id INTEGER PRIMARY KEY,
   manager_name VARCHAR(100) NOT NULL
);
CREATE TABLE books (
 book_id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
manager_id INTEGER NOT NULL,
 title VARCHAR(100) NOT NULL,
 author VARCHAR(100) NOT NULL,
 publisher VARCHAR(100) NOT NULL,
 price NUMBER(10,2) NOT NULL,
 stock INTEGER NOT NULL,
FOREIGN KEY (manager_id) REFERENCES manager (manager_id)
);
CREATE TABLE customers (
 customer_id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
 name VARCHAR(100) NOT NULL,
 email VARCHAR(100) NOT NULL,
 phone VARCHAR(20) NOT NULL
);

CREATE TABLE orders (
 order_id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
 customer_id INTEGER NOT NULL,
 order_date DATE DEFAULT SYSDATE,
 FOREIGN KEY (customer_id) REFERENCES customers (customer_id)
);

CREATE TABLE order_items (
 order_id INTEGER NOT NULL,
 book_id INTEGER NOT NULL,
 quantity INTEGER NOT NULL,
 PRIMARY KEY (order_id, book_id),
 FOREIGN KEY (order_id) REFERENCES orders (order_id),
 FOREIGN KEY (book_id) REFERENCES books (book_id)
);
```

```
INSERT INTO manager VALUES('24','Ram');
INSERT INTO manager VALUES('27','Sham');
INSERT INTO manager VALUES('10','Balram');

INSERT INTO books VALUES('237','24','Magic Tree','Abhinav','Sushant','1500','20');
INSERT INTO books VALUES('452','27','Winter Fairy','Shubham','Manoj','2000','10');
INSERT INTO books VALUES('480','10','Enchanted','Aryan','Vikas','2500','8');

INSERT INTO customers VALUES('1','Pranjali','pranjali15@gmail.com','9834652376');
INSERT INTO customers VALUES('2','Jasmine','jasmine17@gmail.com','7640271548');
INSERT INTO customers VALUES('3','Nandini','nandini5@gmail.com','9843725476');

INSERT INTO orders VALUES('39','1','15-JAN-2022');
INSERT INTO orders VALUES('72','2','27-APR-2022');
INSERT INTO orders VALUES('58','3','23-MAY-2022');

INSERT INTO order_items VALUES('39','237','7');
INSERT INTO order_items VALUES('72','452','5');
INSERT INTO order_items VALUES('58','480','3');

select * from manager;
select * from books;
select * from customers;
select * from orders;
select *from order_items;
```

```
Table created.

Table created.

Table created.

Table created.

Table created.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.
```

```
1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.
```

MANAGER_ID	MANAGER_NAME
24	Ram
27	Sham
10	Balram

ORDER_ID	BOOK_ID	QUANTITY
39	237	7
72	452	5
58	480	3

				PUBLISHER	PRICE	STOCK
				Sushant	1500	20
				Manoj	2000	10
480	10	Enchanted	Aryan	Vikas	2500	8

CUSTOMER_ID	NAME	EMAIL	PHONE
1	Pranjali	pranjali15@gmail.com	9834652376
2	Jasmine	jasmine17@gmail.com	7640271548
3	Nandini	nandini5@gmail.com	9843725476

Download CSV

3 rows selected.

ORDER_ID	CUSTOMER_ID	ORDER_DATE
39	1	15-JAN-22
72	2	27-APR-22
58	3	23-MAY-22

```
CREATE OR REPLACE FUNCTION get_order_total (
   p_order_id IN INTEGER
) RETURN NUMBER IS
   v_total NUMBER := 0;
BEGIN
   SELECT SUM(quantity * price)
   INTO v_total
   FROM order_items oi
   JOIN books b ON oi.book_id = b.book_id
   WHERE oi.order_id = p_order_id;
   RETURN v_total;
END;
/

CREATE OR REPLACE TRIGGER update_stock
AFTER INSERT ON order_items
FOR EACH ROW
DECLARE
  v_stock books.stock%TYPE;
BEGIN

  SELECT stock INTO v_stock FROM books WHERE book_id = :NEW.book_id;


  UPDATE books SET stock = v_stock - :NEW.quantity WHERE book_id = :NEW.book_id;
END;
/


CREATE OR REPLACE TRIGGER prevent_negative_price
BEFORE INSERT,UPDATE ON books
FOR EACH ROW
BEGIN
  IF :NEW.price < 0 THEN
    RAISE_APPLICATION_ERROR(-20007, 'Price cannot be negative.');
  END IF;
END;
/


CREATE OR REPLACE PROCEDURE update_book_stock (
   p_book_id IN INTEGER,
   p_stock IN INTEGER
) IS
BEGIN
   UPDATE books
   SET stock = p_stock
   WHERE book_id = p_book_id;
   COMMIT;
END;
/
```

```
INSERT INTO orders (customer_id)
VALUES (1);
INSERT INTO order_items (order_id, book_id, quantity)
VALUES (1, 1, 2);


SELECT get_order_total(39) FROM dual;



BEGIN
  update_book_stock(237, 150);
END;
/

select * from books;
DECLARE
  CURSOR c1 IS
    SELECT title, author
    FROM books;

  v_title books.title%TYPE;
  v_author books.author%TYPE;
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO v_title, v_author;
    EXIT WHEN c1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Title: ' || v_title || ', Author: ' || v_author);
  END LOOP;
  CLOSE c1;
END;
/

*/
```

```
Function created.

Trigger created.

Trigger created.

Procedure created.

1 row(s) inserted.
```

GET_ORDER_TOTAL(39)
10500

```
INSERT INTO books VALUES('127','24','Harry Potter','Nandini','Pridhi',-150,20);
Trigger created.

Procedure created.

ORA-20007: Price cannot be negative. ORA-06512: at "SQL_OMEJBOGADKFRSNRGXIVJGTWCW.PREVENT_NEGATIVE_PRICE", line 3
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

6. CONCLUSION

In conclusion, the development of a database management system for a bookstore is an essential step to effectively manage the store's books, customers, and orders. The system should enable the bookstore to add, update, and delete information about books, customers, and orders, retrieve information, and generate reports based on the stored information.

To achieve this, we designed a normalized database schema that includes tables for books, customers, orders, and order items. The schema ensures data integrity and consistency by enforcing constraints such as primary keys, foreign keys, and unique keys. We also wrote SQL queries to retrieve the required information and optimized them for performance and accuracy to handle large amounts of data efficiently.

To ensure the security of the system, we implemented user authentication and access control, as well as backup and recovery mechanisms to protect against data loss or corruption. We also designed the system to be scalable and user-friendly by implementing a distributed database architecture and providing user documentation and training.

Overall, the developed database management system will help the bookstore to efficiently manage its books, customers, and orders, leading to improved customer satisfaction, increased sales, and streamlined operations