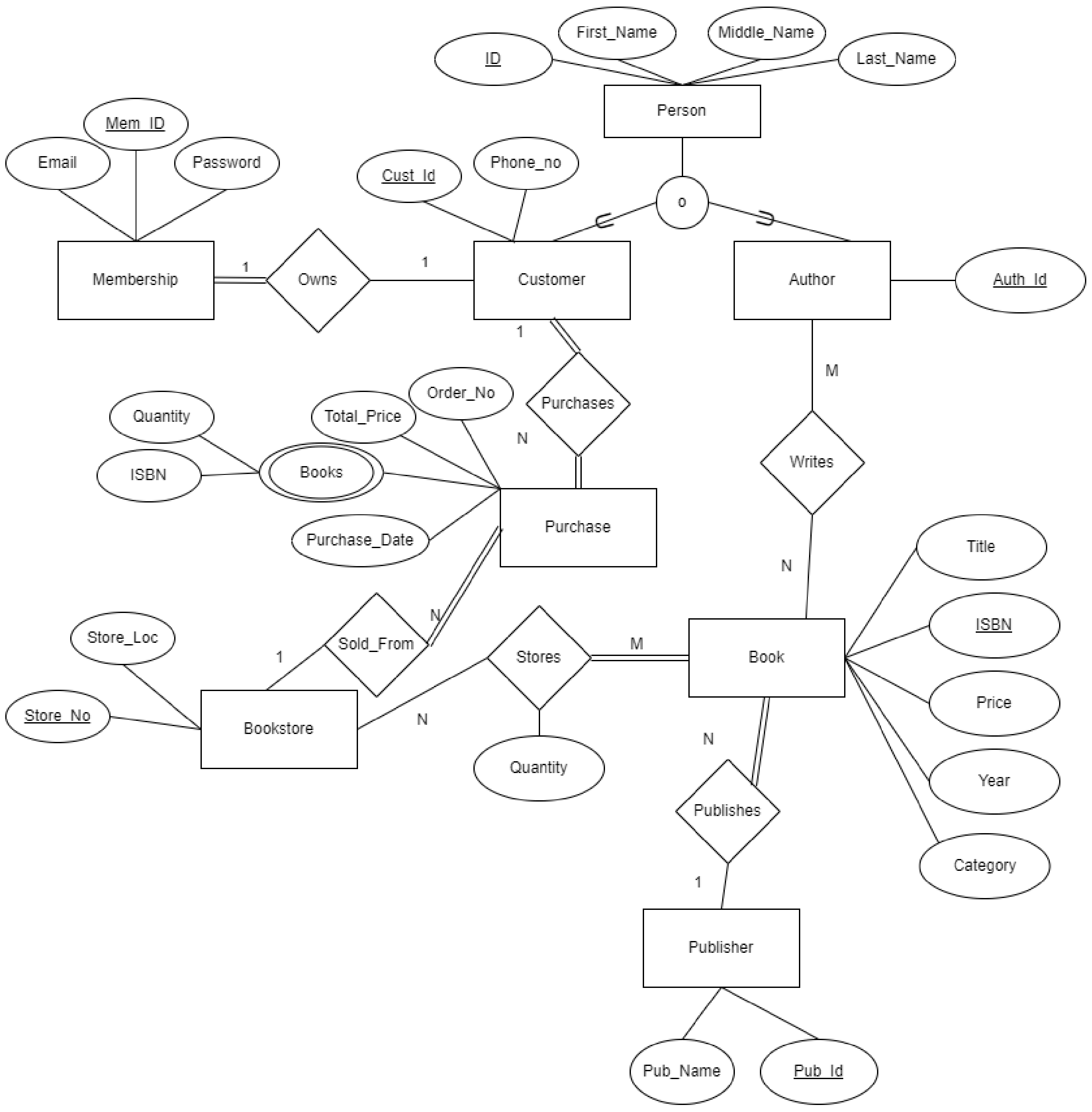
CSE 3241 Project Checkpoint 03 – SQL and More SQL

Names: Tyra Shin, Sean Kelley, Trey Thomas, Jack Kern, Justin Sparacino	Date: 3.16.24

As with all worksheets, these checkpoint parts must be submitted together on the Carmen Dropbox as a ZIP.

Part One:

Provide a current version of your ER Diagram and Relational Model as per Project Checkpoint 02. **If you were instructed to change the model for Project Checkpoint 02, make sure you use the revised versions of your models.**



Membership(Password, Mem_ID(FK))

Person(ID, First_Name, Middle_Name, Last_Name, Email)

Author(Auth_ID)

Writes(Author_ID, ISBN)

Purchase(Order_Num, Cust_ID(FK), Book(FK), Total_Price, Purchase_Date, Store_no(FK))

Publisher(Pub_ID, Pub_Name)

Customer(Cust_id(FK), Phone_no)

Book(ISBN, Pub_ID(FK), Year, Price, Title, Category)

Books(ISBN**,** Order_num, quantity)

Bookstore(Store_no, store_loc)

Stored(ISBN, Store_no, Quantity)

Part Two:

1. Given your relational schema, create a text file containing the SQL code to create your database schema. Use this SQL to create a database in SQLite. Populate this database with the data provided for the project as well as 20 sample records for each table that does not contain data provided in the original project documents.

2. Given your relational schema, provide the SQL to perform the following queries. If your schema cannot provide answers to these queries, revise your ER Model and your relational schema to contain the appropriate information for these queries. These queries should be provided in a plain text file named "WorksheetTwoSimpleQueries.txt":
   a. Find the titles of all books by Pratchett that cost less than $10
   ```
   SELECT Title
   FROM BOOK
   JOIN AUTHOR ON BOOK.Auth_Id = AUTHOR.Auth_Id
   JOIN PERSON ON AUTHOR.Auth_Id = PERSON.ID
   WHERE Pub_name = 'Pratchett' AND Price < 10;
   ```

   b. Give all the titles and their dates of purchase made by a single customer (you choose how to designate the customer)
   ```
   SELECT Title, Order_num
   FROM BOOK
   JOIN PURCHASE ON BOOK.ISBN = PURCHASE.Book
   WHERE Cust_Id = <customer_id>;
   ```

c. Find the titles and ISBNs for all books with less than 5 copies in stock

```
SELECT Title, ISBN
FROM BOOK
JOIN STORED ON BOOK.ISBN = STORED.ISBN
HAVING SUM(STORED.Quantity) < 5;
```

d. Give all the customers who purchased a book by Pratchett and the titles of Pratchett books they purchased

```
SELECT First_Name, Last_Name, Title
FROM PERSON
JOIN CUSTOMER ON PERSON.ID = CUSTOMER.Cust_Id
JOIN PURCHASE ON CUSTOMER.Cust_Id = PURCHASE.Cust_Id
JOIN BOOK ON PURCHASE.Book = BOOK.ISBN
JOIN PUBLISHER ON BOOK.Pub_Id = PUBLISHER.Pub_Id
WHERE Pub_Name = 'Pratchett';
```

e. Find the total number of books purchased by a single customer (you choose how to designate the customer)

```
SELECT SUM(Quantity)
FROM PURCHASE
JOIN BOOKS ON PURCHASE.Order_No = BOOKS.Order_no
JOIN CUSTOMER ON CUSTOMER.Cust_Id = PURCHASE.Cust_Id
JOIN PERSON ON PERSON.Id = CUSTOMER.Cust_Id
WHERE Last_Name = ‘smith’;
```

f. Find the customer who has purchased the most books and the total number of books they have purchased

```
SELECT Cust_Id, SUM(Quantity)  as Total_Books_Purchased
FROM PURCHASE
JOIN BOOKS ON PURCHASE.Order_No = BOOKS.Order_no
GROUP BY Cust_Id
ORDER BY Total_Books_Purchased DESC
LIMIT 1;
```

3. For Project Checkpoint 02, you were asked to come up with three additional interesting queries that your database can provide. Give what those queries are supposed to retrieve in plain English, as relational algebra and then as SQL. Your queries should include joins and at least one should include an aggregate function, and they should be the same as the queries you outlined for Worksheet 02. If you were instructed to fix the queries in Checkpoint 02, make sure you use the fixed queries here. These queries should be provided in a plain text file named “WorksheetTwoExtraQueries.txt”.

Find the total number of books the publisher Pratchett has published.

```
SELECT COUNT(Pub_Id)
FROM PUBLISHER
JOIN BOOK ON PUBLISHER.Pub_Id = BOOK.Pub_Id
WHERE Name = 'Pratchett';
```

Find the number of purchases made by customers with memberships.

```
SELECT COUNT(Member_num)
FROM MEMBERSHIP
JOIN PURCHASE ON MEMBERSHIP.Cust_Id = PURCHASE.Cust_Id;
```

Find and list the titles and ISBNs of books that <a certain author> has written that are published by Pratchett.

```
SELECT title, ISBN
FROM (
   SELECT *
   FROM AUTHOR
   JOIN WRITES ON AUTHOR.Auth_Id = WRITES.Author_ID
   JOIN BOOK ON WRITES.Book_ID = BOOK.ISBN
) AS AUTH_BOOK
JOIN PUBLISHER ON AUTH_BOOK.Publisher_ID = PUBLISHER.Pub_ID
WHERE Pub_Name = 'Pratchett', Auth_Id = <author_id>;
```

4. Given your relational schema, provide the SQL for the following more advanced queries. These queries may require you to use techniques such as nesting, aggregation using having clauses, and other techniques . If your database schema does not contain the information to answer to these queries, revise your ER Model and your relational schema to contain the appropriate information for these queries. **Note that if your database does contain the information but in non-aggregated form, you should NOT revise your model but instead figure out how to aggregate it for the query!** These queries should be provided in a plain text file named "WorksheetTwoAdvancedQueries.txt".

    a. Provide a list of customer names, along with the total dollar amount each customer has spent.

```
SELECT  First_Name, Last_Name, SUM(Price)
FROM CUSTOMER
JOIN PURCHASE ON CUSTOMER.Cust_Id = PURCHASE.Cust_Id
GROUP BY Cust_Id;
```

    b. Provide a list of customer names and e-mail addresses for customers who have spent more than the average customer.

```
SELECT First_Name, Last_Name, Email
FROM CUSTOMER
JOIN PURCHASE ON CUSTOMER.Cust_Id = PURCHASE.Cust_Id
WHERE Cust_id IN
        (SELECT Cust_ID
        FROM Purchase
        GROUP BY Cust_ID
        HAVING SUM(Total_Price) > (
                SELECT AVG(total_spent)
                FROM
                        (SELECT Cust_ID, SUM(Total_Price)
                        FROM Purchase
                        GROUP BY Cust_ID
                        )
        )
```

c. Provide a list of the titles in the database and associated total copies sold to customers, sorted from the title that has sold the most individual copies to the title that has sold the least.

```
SELECT Title, SUM(Quantity) as Total_Sold
FROM PURCHASE
JOIN BOOKS ON PURCHASE.Order_no = BOOKS.Order_no
JOIN BOOK ON BOOKS.ISBN = BOOK.ISBN
ORDER BY Total_Sold DESC
GROUP BY BOOK.ISBN;
```

d. Provide a list of the titles in the database and associated dollar totals for copies sold to customers, sorted from the title that has sold the highest dollar amount to the title that has sold the smallest.

```
SELECT Title, SUM(Quantity) * BOOK.Price as Total_Profit
FROM BOOK
JOIN BOOKS ON BOOK.ISBN = BOOKS.ISBN
ORDER BY Total_Profit DESC
GROUP BY BOOK.ISBN;
```

e. Find the most popular author in the database (i.e. the one who has sold the most books)

```
SELECT First_Name, Middle_Name, Last_Name
FROM PERSON
JOIN AUTHOR ON PERSON.Id = AUTHOR.Auth_Id
JOIN CUSTOMER ON PERSON.Id = CUSTOMER.Cust_Id
JOIN PURCHASE ON CUSTOMER.Cust_Id = PURCHASE.Cust_Id
JOIN AUTHOR ON WRITES.Auth_Id = AUTHOR.Auth_Id
ORDER BY COUNT(WRITES.ISBN) DESC
```

f. Find the most profitable author in the database for this store (i.e. the one who has brought in the most money)

```
SELECT First_Name, Last_Name SUM(BOOK_SUM.Total_Profit) as Author_Profit
FROM (
        SELECT ISBN, SUM(Quantity) * BOOKS.PRICE as Total_Profit
        FROM BOOKS
        GROUP BY ISBN
        ) as BOOK_SUM
```

```
JOIN BOOK ON BOOK_SUM.ISBN = BOOK.ISBN
JOIN WRITES ON BOOK.ISBN = WRITES.ISBN
JOIN AUTHOR ON WRITES.Auth_Id = AUTHOR.Auth_Id
JOIN PERSON ON AUTHOR.Auth_Id = PERSON.ID
ORDER BY Author_Profit DESC
GROUP BY AUTHOR.Auth_Id
LIMIT 1;
```

g. Provide a list of customer information for customers who purchased anything written by the most profitable author in the database.

```
SELECT c.*
FROM Customer c
JOIN Purchase p ON c.Cust_ID = p.Cust_ID
JOIN Book b ON p.ISBN = b.ISBN
JOIN Writes w ON b.ISBN = w.ISBN
WHERE w.Auth_ID = (
   SELECT w.Auth_ID
   FROM Writes w
   JOIN Book b ON w.ISBN = b.ISBN
   GROUP BY w.Auth_ID
   ORDER BY SUM(b.Price) DESC
   LIMIT 1
);
```

h. Provide the list of authors who wrote the books purchased by the customers who have spent more than the average customer.

```
SELECT a.*
FROM Author a
JOIN Writes w ON a.Auth_ID = w.Auth_ID
JOIN Book b ON w.ISBN = b.ISBN
JOIN Purchase p ON b.ISBN = p.ISBN
WHERE p.Total_Price > (
   SELECT AVG(Total_Price)
   FROM Purchase
);
```

Once you have completed all of the questions for Part Two, create a ZIP archive containing the binary SQLite file and the three text files and submit this to the Carmen Dropbox. **Make sure your queries work against your database and provide your expected output before you submit them!**

```
CREATE TABLE PERSON (
  ID INT NOT NULL,
  First_Name TEXT NOT NULL,
  Middle_Name TEXT,
  Last_Name TEXT NOT NULL,
    PRIMARY KEY(ID)
);

CREATE TABLE CUSTOMER (
    Cust_Id INT NOT NULL,
    Phone_Oo INT,
    FOREIGN KEY (Cust_Id) REFERENCES PERSON(ID),
    PRIMARY KEY(Cust_Id);
);

CREATE TABLE AUTHOR (
    Auth_Id INT NOT NULL,
    FOREIGN KEY (Auth_Id) REFERENCES PERSON(ID),
    PRIMARY KEY(Auth_Id)
);

CREATE TABLE MEMBERSHIP (
    Mem_Id INT NOT NULL,
    Email TEXT NOT NULL,
    Password TEXT NOT NULL,
    FOREIGN KEY (Mem_Id) REFERENCES CUSTOMER(Cust_Id),
    PRIMARY KEY(Mem_Id);
);

CREATE TABLE PURCHASE (
    Price REAL NOT NULL,
    Books TEXT NOT NULL,
    Date DATE NOT NULL,
    Order_No INT NOT NULL,
    Cust_Id INT NOT NULL,
    FOREIGN KEY (Cust_Id) REFERENCES CUSTOMER(Cust_Id),
    FOREIGN KEY (Order_No) REFERENCES BOOKS(Order_No),
    PRIMARY KEY(Order_num);
);

CREATE TABLE BOOKSTORE (
    Store_Id INT NOT NULL,
    Name TEXT NOT NULL,
    PRIMARY KEY(Store_Id);
);

CREATE TABLE BOOK (
    Title TEXT NOT NULL,
    ISBN INT NOT NULL,
    Price REAL NOT NULL,
    Year INT,
    Category TEXT,
    Auth_Id INT NOT NULL,
    Publ_Id INT NOT NULL,
    FOREIGN KEY (Auth_Id) REFERENCES AUTHOR(Auth_Id),
    FOREIGN KEY (Publ_Id) REFERENCES AUTHOR(Publ_Id),
    PRIMARY KEY(ISBN)
);

CREATE TABLE PUBLISHER (
    Publ_Id INT NOT NULL,
    PRIMARY KEY(Pub_Id);
);

CREATE TABLE STORES (
    QUANTITY INT NOT NULL,
    Store_Id INT NOT NULL,
    ISBN INT NOT NULL,
    FOREIGN KEY (ISBN) REFERENCES BOOK(ISBN),
    FOREIGN KEY (Store_Id) REFERENCES BOOKSTORE(Store_Id),
    PRIMARY KEY(ISBN, Store_Id);
);

CREATE TABLE BOOKS (
    QUANTITY INT NOT NULL,
    Order_num INT NOT NULL,
    ISBN INT NOT NULL,
    FOREIGN KEY (ISBN) REFERENCES BOOK(ISBN),
    FOREIGN KEY (Order_num) REFERENCES BOOKSTORE(Order_num)
    PRIMARY KEY(ISBN, Order_num)
);
```

Feedback: All the required data is missing, and you didn't supply an ERD and relational schema. I can't really give any meaningful feed back to this project. You need to populate all book data, and make sure every query works properly, and comes back with proper data.

How we addressed it:

- We created a c# program to insert the data. We also ensured the ERD and relational schema is up to date.