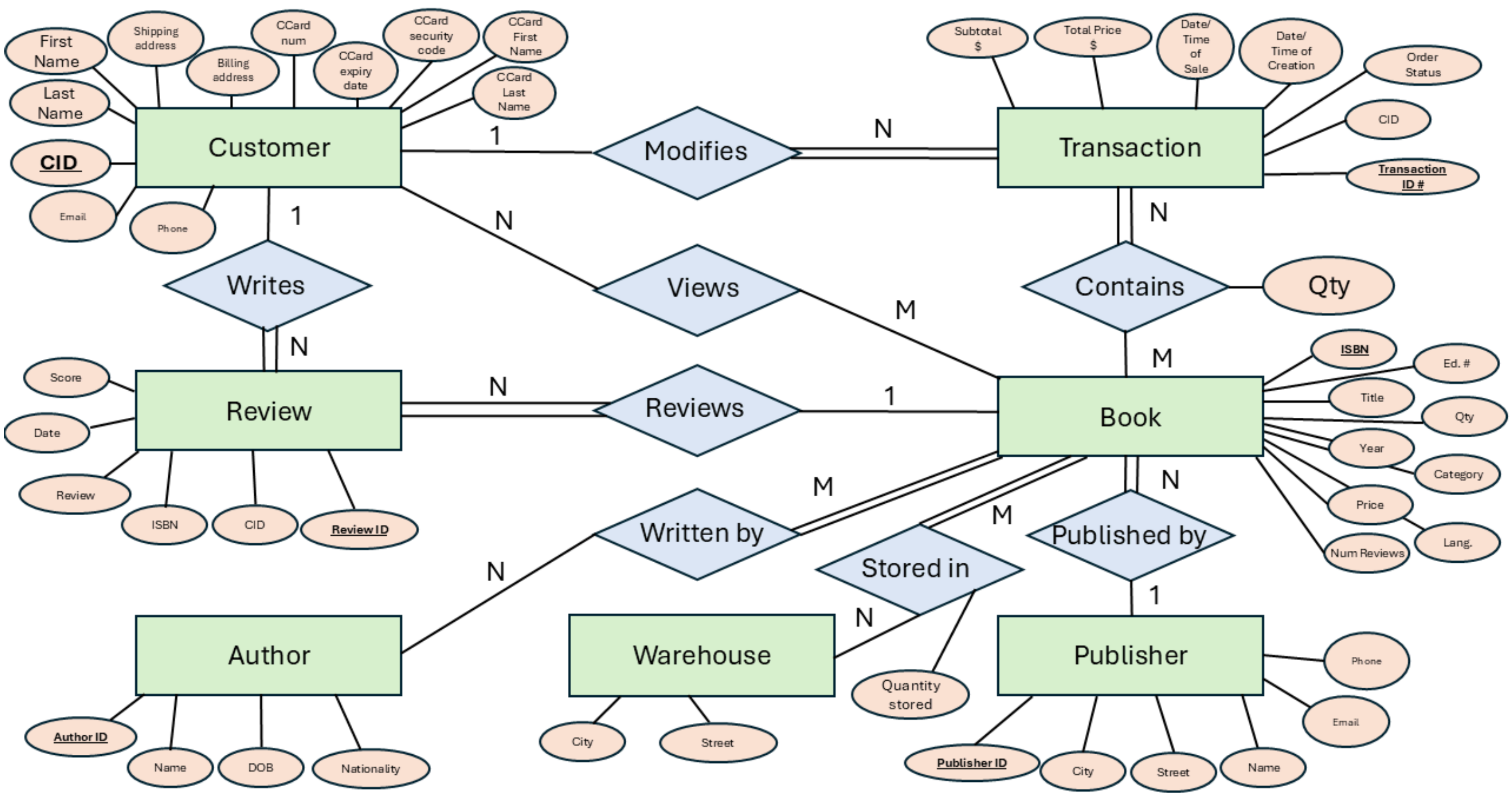
CSE 3241 Final Report

Connor Fricke, Sierra Reis, Ethan Conley

DATABASE DESCRIPTION:

ER Diagram



Normalizations

CUSTOMER - BCNF
REVIEW - BCNF
AUTHOR - BCNF
WAREHOUSE - BCNF
PUBLISHER - BCNF
BOOK - BCNF
TRANSACTION - BCNF
WRITTEN_BY - BCNF
VIEWS - BCNF
CONTAINS - BCNF
STORED_IN - BCNF

Schema

CUSTOMER: *(<u>CID</u>, FirstName, LastName, ShippingAddress, BillingAddress, CCardNum, CCardExpiryDate, CCardSecCode, CCardFirstName, CCardLastName, Email, Phone)*

{CID} → {FirstName, LastName, ShippingAddress, BillingAddress, CCardNum, CCardExpiryDate, CCardSecCode, CCardFirstName, CCardLastName, Email, Phone}

{CCardNum} → {CID, FirstName, LastName, ShippingAddress, BillingAddress, CCardExpiryDate, CCardSecCode, CCardFirstName, CCardLastName, Email, Phone}

{Email} → {<u>CID</u>, FirstName, LastName, ShippingAddress, BillingAddress, CCardNum, CCardExpiryDate, CCardSecCode, CCardFirstName, CCardLastName, Phone}

Assumptions:

- *Each customer account can only have one corresponding credit card attached to it*
- *A credit card can only be used for one customer account*
- *Each customer account can only have one email attached to it*
- *An email address can only be used for one customer account*
- *Customer accounts do not need a phone, it is optional (and thus not a key)*

These assumptions make it so that CCardNum and Email are both candidate keys

REVIEW: *(<u>ReviewID</u>, {CID}, {ISBN}, Review, ReviewDate, Score)*

{ReviewID} → {CID, ISBN, Review, ReviewDate, Score}

{CID, ISBN} → {ReviewID, Review, ReviewDate, Score} *(Each CID can only leave 1 review per book)*

AUTHOR: *(<u>AuthorID</u>, AuthorName, DOB, Nationality)*

{AuthorID} → {AuthorName, DOB, Nationality}

WAREHOUSE: *(<u>City</u>, <u>Street</u>)*

{City} → {Street} *(Limiting to one warehouse per city)*

PUBLISHER: *(<u>PublisherID</u>, City, Street, State, PublisherName, Email, PhoneNumber)*

{PublisherID} → {City, Street, PublisherName, Email, Phone}

{City, Street} → {PublisherID, PublisherName, Email, Phone} *(Assuming publisher only has 1 main HQ)*

{PublisherName} → {PublisherID, City, Street, Email, Phone} *(Assuming publisher's Name is unique)*

{Email} → {PublisherID, City, Street, PublisherName, Phone}

{Phone} → {PublisherID, City, Street, PublisherName, Email}

BOOK: *(<u>ISBN</u>, EditionNum, Title, Quantity, Publication_Year, Category, Price, Language, Num_Reviews, Publisher_ID)*

{ISBN} → {EditionNum, Title, Quantity, Publication_Year, Category, Price, Language, Num_Reviews, Publisher_ID}

In database:

Publication_Year is Year

Quantity is Qty

TRANSACTION: *(<u>TransactionID</u>, {CID}, OrderStatus, DateTimeCreation, DateTimeSale, Subtotal, TotalPrice)*

{TransactionID} → {CID, OrderStatus, DateTimeCreation, DateTimeSale, Subtotal, TotalPrice}

{CID, DateTimeCreation} → {TransactionID, OrderStatus, DateTimeSale, Subtotal, TotalPrice}

{CID, DateTimeSale} → {TransactionID, OrderStatus, DateTimeCreation, Subtotal, TotalPrice}

WRITTEN_BY: *({ISBN}, {AuthorID})*

{ISBN} → {AuthorID}

VIEWS: *({CID}, {ISBN})*

(No functional dependencies, this is a simple bridge table where neither FK is functionally dependent on the other and there are no non-key attributes.)

CONTAINS: *({TransactionID}, {ISBN}, Qty)*

{TransactionID, ISBN} → {Qty}

STORED_IN: *({City}, {Street}, {ISBN}, Quantity_Stored)*

{City, Street, ISBN} → {Quantity_Stored}

In database Quantity_Stored is just Qty

Indexes

Indexes are important tools for improving the performance of this database. Query operations will be happening constantly while users are browsing the site for books, thus maximizing the efficiency of the most common queries is essential for the performance of the website.

In the relevant queries, natural joins are a common tool. There are several queries which rely on a long chain of joins through as many as 5 to 7 entity relations and bridge tables. For example, the query which obtains the most profitable author on the site follows:

```
SELECT AuthorName
FROM Author * WrittenBy * Book * Contains * Transactions
GROUP BY AuthorName
HAVING TotalPrice = (
      SELECT MAX(TotalPrice)
      FROM Author * WrittenBy * Book * Contains * Transactions
      GROUP BY AuthorName
);
```

This query, of course, joins five separate relations, which may each have many hundreds of thousands or even millions of rows when *BitsNBooks* becomes world renowned. These joins benefit greatly from a hash-based index on the relevant primary and foreign keys of the relations.

Even though SQLite only implements a tree-based index, these joins still have a great performance boost to gain. By creating an index for the five tables above (on AuthorID, ISBN, and TransactionID, etc) we can greatly increase the performance of similar queries.

More generally, the tables which will have the most rows of data will be the Book, Review, Customer, Transaction, and corresponding bridge tables, so it will be helpful to create and keep updated indices as follows:

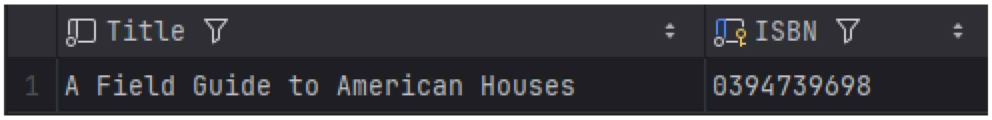
```
-- Indexes to help speed up JOIN queries
CREATE INDEX CIDSearch ON Customer (CID);
CREATE INDEX ISBNSearch ON Book (ISBN);
CREATE INDEX AuthorIDSearch ON Author (AuthorID);
CREATE INDEX ReviewSearch ON Review (CID);
CREATE INDEX BookReviewSearch ON Review (ISBN);
CREATE INDEX TransactionLookup ON Transactions (CID);
CREATE INDEX WrittenByIdx ON WrittenBy (AuthorID, ISBN);
CREATE INDEX ContainsIdx ON Contains (TransactionID, ISBN);
CREATE INDEX ViewsIdx ON Views (CID, ISBN);

-- Customers may often want to view large portions of the database,
-- such as using a search bar to search
-- for books of a certain title or from a certain author. We will
-- attempt to speed these queries up with indexes like the following:

-- so users can search for books by name
CREATE INDEX BookTitleSearch ON Book (Title);
-- so users can search for authors by name
CREATE INDEX AuthorNameSearch ON Author (AuthorName);
-- so users can quickly filter or sort books by price
CREATE INDEX PriceFilterSearch ON Book (Price);
```

Views

1) Find the titles and ISBNs for all books with less than 5 copies in stock.
   This view is useful because it allows workers to view which books are running low in order to order more of them.

   $\pi_{title,\,ISBN}(\sigma_{qty < 5}(Book))$

   ```
   SELECT Title, ISBN
   FROM Book
   WHERE Quantity < 5;
   ```



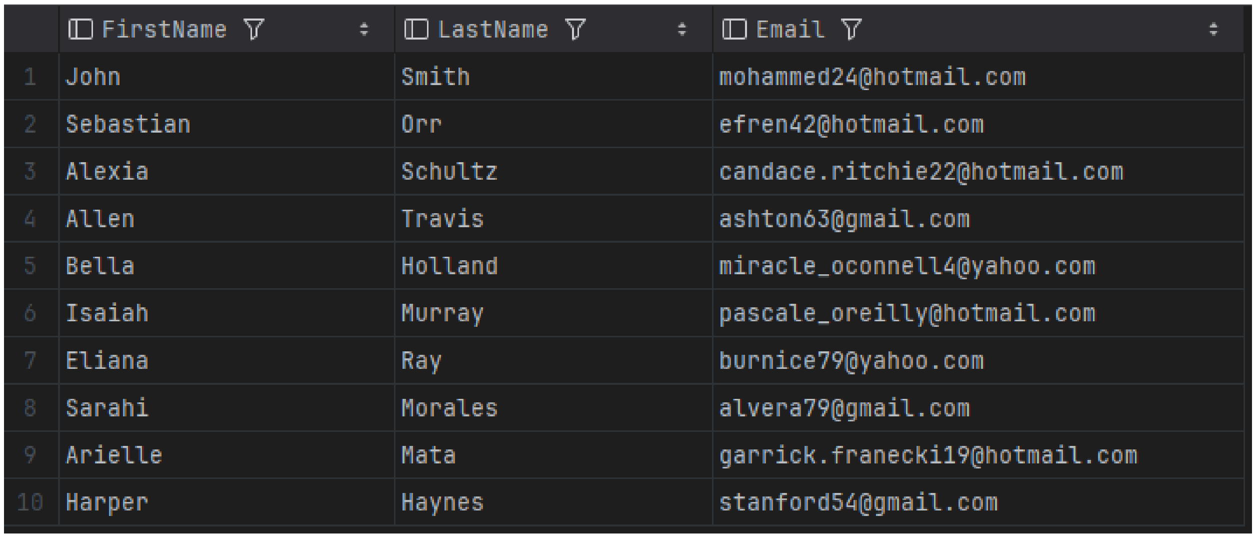
	Title	ISBN
1	A Field Guide to American Houses	0394739698

2) Provide a list of customer names and e-mail addresses for customers who have spent more than the average customer.
   This view is useful because it allows marketing teams to send out promotional material or deals to customers who have spent more than the average customer.

   $TotalSpent \rightarrow {}_{CID}\mathcal{F}_{SUM(TotalPrice)}(Customer)$

   $\pi_{FirstName,\,LastName,\,Email}(\sigma_{TotalSpent.TotalPrice > \mathcal{F}_{AVG(TotalSpent)}}(TotalSpent * (Customer * Transactions)))$

   ```
   SELECT FirstName, LastName, Email
   FROM Customer
   NATURAL JOIN Transactions
   GROUP BY CID
   HAVING SUM(Transactions.TotalPrice) > (
           SELECT AVG(TotalSpent)
           FROM (
                   SELECT SUM(Transactions.TotalPrice) as TotalSpent
                   FROM Transactions
                   GROUP BY CID
           )
   );
   ```

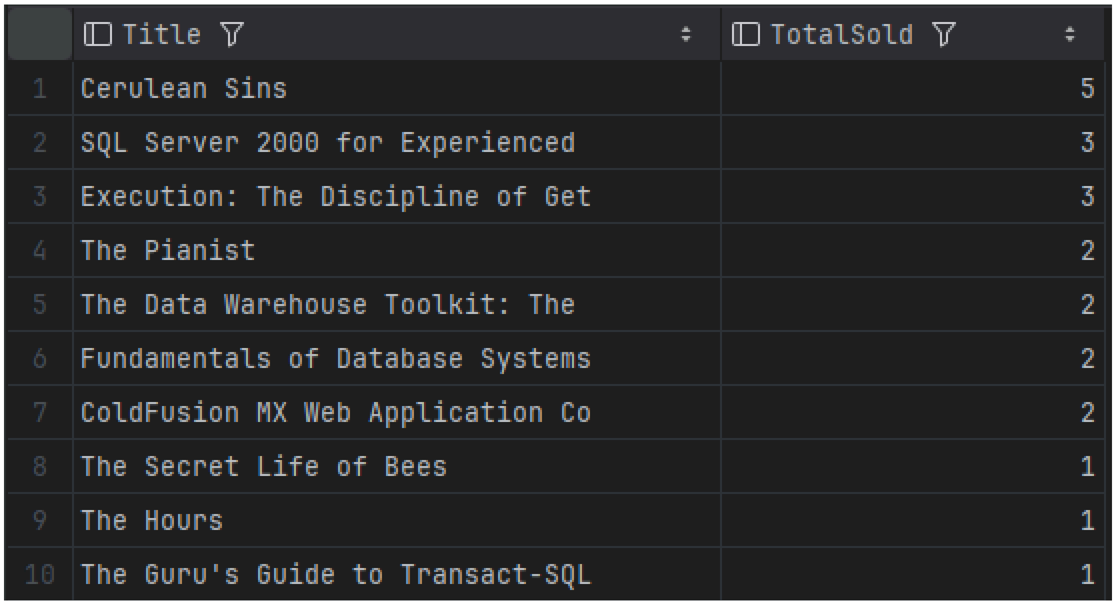


	FirstName	LastName	Email
1	John	Smith	mohammed24@hotmail.com
2	Sebastian	Orr	efren42@hotmail.com
3	Alexia	Schultz	candace.ritchie22@hotmail.com
4	Allen	Travis	ashton63@gmail.com
5	Bella	Holland	miracle_oconnell4@yahoo.com
6	Isaiah	Murray	pascale_oreilly@hotmail.com
7	Eliana	Ray	burnice79@yahoo.com
8	Sarahi	Morales	alvera79@gmail.com
9	Arielle	Mata	garrick.franecki19@hotmail.com
10	Harper	Haynes	stanford54@gmail.com

(Note: The emails here do not correspond to the person's actual name, they were randomly generated)

3) 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.
This is useful because it allows for employees and customers to see the most and least popular books. It is useful to workers because they can see what books are very popular and preemptively stock up on them or prioritize ordering similar books and to see which books are not popular and could be deprioritized. It is useful to customers because they can see which books are popular and that other people are ordering so they can join in on it.
$TotalSold \rightarrow {}_{ISBN}\mathcal{F}_{Sum(Qty)}(Contains)$
$\pi_{Title,\, TotalSold.Qty}(TotalSold\ *\ Book)$
*No sort in relational algebra.

```
SELECT Title, SUM(Contains.Qty) as TotalSold
FROM Book
NATURAL JOIN Contains
GROUP BY Title
ORDER BY TotalSold DESC;
```



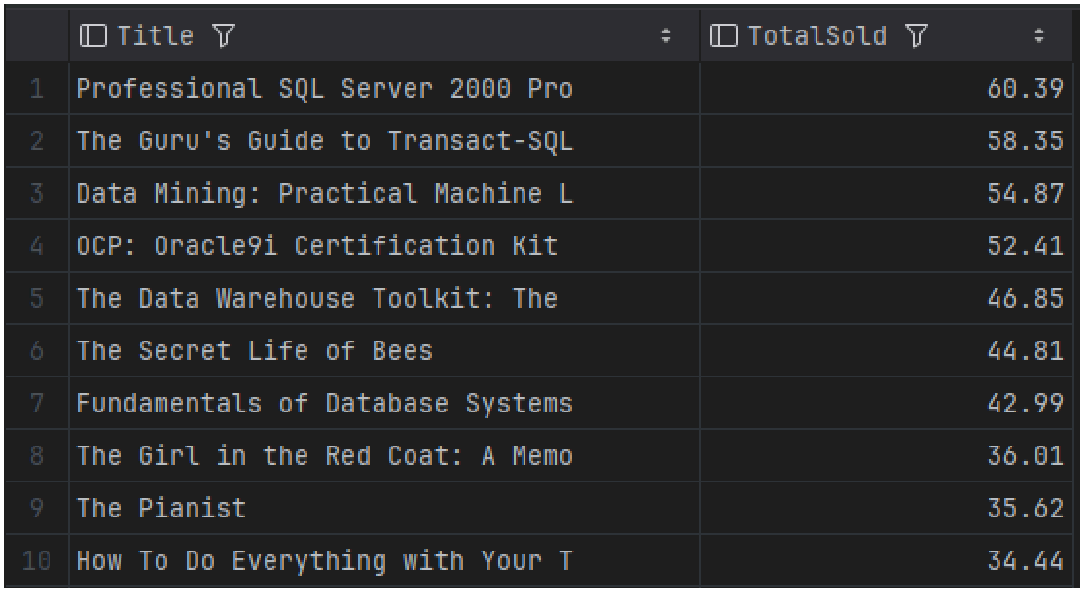
	Title	TotalSold
1	Cerulean Sins	5
2	SQL Server 2000 for Experienced	3
3	Execution: The Discipline of Get	3
4	The Pianist	2
5	The Data Warehouse Toolkit: The	2
6	Fundamentals of Database Systems	2
7	ColdFusion MX Web Application Co	2
8	The Secret Life of Bees	1
9	The Hours	1
10	The Guru's Guide to Transact-SQL	1

4) 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.
This view is useful because it allows workers to see which books have made the most and least money.

$_{Title}\mathcal{F}_{SUM(Transactions.TotalPrice)}(Transactions\ *\ (Contains\ *\ Books))$

*No sort in relational algebra.

```
SELECT Title, SUM(Transactions.TotalPrice) as TotalSold
FROM Book
NATURAL JOIN Contains
NATURAL JOIN Transactions
GROUP BY Title
ORDER BY TotalSold DESC;
```



	Title	TotalSold
1	Professional SQL Server 2000 Pro	60.39
2	The Guru's Guide to Transact-SQL	58.35
3	Data Mining: Practical Machine L	54.87
4	OCP: Oracle9i Certification Kit	52.41
5	The Data Warehouse Toolkit: The	46.85
6	The Secret Life of Bees	44.81
7	Fundamentals of Database Systems	42.99
8	The Girl in the Red Coat: A Memo	36.01
9	The Pianist	35.62
10	How To Do Everything with Your T	34.44

5) Find the most popular author in the database (i.e. the one who has sold the most books) This view is useful to customers because they can see which author other customers like the most.

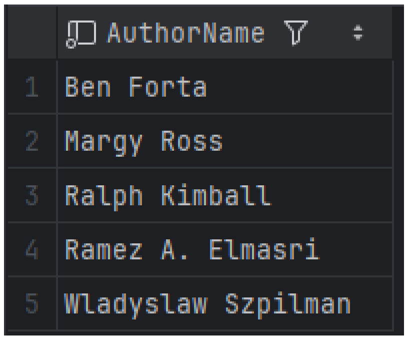
REL ALG

$TotalBooks \rightarrow {}_{AuthorName}\mathcal{F}_{SUM(Contains.Qty)}(Contains\ *\ Book\ *\ WrittenBy\ *\ Author)$

$MaxQty \rightarrow \mathcal{F}_{MAX(TotalBooks.Qty)}(TotalBooks)$

$\pi_{AuthorName}(\sigma_{TotalBooks.Qty\ =\ MaxQty}(TotalBooks))$

```
SELECT AuthorName
FROM Author
NATURAL JOIN WrittenBy
NATURAL JOIN Book
NATURAL JOIN Contains
GROUP BY AuthorName
HAVING Contains.Qty = (

    SELECT MAX(Contains.Qty)
    FROM Author
    NATURAL JOIN WrittenBy
    NATURAL JOIN Book
    NATURAL JOIN Contains
    GROUP BY AuthorName
);
```



	AuthorName
1	Ben Forta
2	Margy Ross
3	Ralph Kimball
4	Ramez A. Elmasri
5	Wladyslaw Szpilman

(5-way tie between these authors)

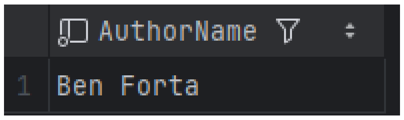
6) Find the most profitable author in the database for this store (i.e. the one who has brought in the most money)
   This view is more useful to workers compared to view 6. It allows workers to see which author is the most profitable for them.

   $TotalRevenue \rightarrow {}_{AuthorName}\mathcal{F}_{Sum(TotalPrice)}(Author\ *\ WrittenBy\ *\ Book\ *\ Contains\ *\ Transactions)$

   $MaxRevenue \rightarrow \mathcal{F}_{MAX(TotalPrice)}(TotalRevenue)$

   $\pi_{AuthorName}(\sigma_{TotalRevenue.TotalPrice\ =\ MaxRevenue}(TotalRevenue))$

```
SELECT AuthorName
FROM Author
NATURAL JOIN WrittenBy
NATURAL JOIN Book
NATURAL JOIN Contains
NATURAL JOIN Transactions
GROUP BY AuthorName
HAVING TotalPrice = (
        SELECT MAX(TotalPrice)
        FROM Author
        NATURAL JOIN WrittenBy
        NATURAL JOIN Book
        NATURAL JOIN Contains
        NATURAL JOIN Transactions
GROUP BY AuthorName
);
```



	AuthorName
1	Ben Forta

USER MANUAL:

Entities & Attributes

CUSTOMER - Represents the customers using the database to buy books.

(<u>CID</u>, FirstName, LastName, ShippingAddress, BillingAddress, CCardNum, CCardExpiryDate, CCardSecCode, CCardFirstName, CCardLastName, Email, Phone)

- CID - **Integer** - Customer ID. Number used to uniquely identify customers.
- FirstName - **String** - First name of the customer in their account.
- LastName - **String** - Last name of the customer.
- ShippingAddress - **String** - Address used to ship books to customers.
- BillingAddress - **String** - Address used to charge customers.
- CCardNum - **Integer** - Credit Card Number attached to a customer's account. 16 digits long. Only one allowed per customer account.
- CCardExpiryDate - **Date** - Expiration date of customer’s credit card.
- CCardSecCode - **Integer** - Credit Card security code. 3 digits long.
- CCardFirstName - **String** - First name of the owner of the credit card.
- CCardLastName - **String** - Last name of the owner of the credit card.
- Email - **String** - Email address used to contact customers. Unique per customer. Customer accounts cannot have more than 1 email and the same email cannot be used to create multiple accounts.
- Phone - **String** - Phone number used to contact customers. 10 character long string. Does not include hyphens, slashes, or other characters to break up the number.

REVIEW - Represents the reviews customers have posted on books in the database.

(<u>ReviewID</u>, {CID}, {ISBN}, Review, ReviewDate, Score)

- ReviewID - **Integer** - Number used to uniquely identify reviews.
- CID - **Integer** - Customer ID number of the account that wrote the review. Foreign Key to Customer entity.
- ISBN - **Integer** - 10-digit ISBN number of the book the review is posed on. Foreign Key to Book entity.
- Review - **String** - The string text review that the customer posted for the book.
- ReviewDate - **Date** - The posting date of the review on the book.
- Score - **Integer** - An integer review score from 1 to 5 of the rating a customer gave to the book.

AUTHOR - Represents the authors in the database. There does not need to be books in the database written by a particular author for the author to be in the database.

(<u>AuthorID</u>, Name, DOB, Nationality)

- AuthorID - **Integer** - Number used to uniquely identify authors in the database.
- AuthorName - **String** - Name of the author as it would appear on their books.
- DOB - **Date** - Date representing the date of birth of the author.
- Nationality - **String** - Nationality of the author.

WAREHOUSE - Represents warehouses used to store the books in the database.

(<u>City</u>, <u>Street</u>)

- City - **String** - City where the warehouse is located.
- Street - **String** - Address where the warehouse is specifically located. Includes the street number of the warehouse.

PUBLISHER - Represents the publishers in the database. There does not need to be books in the database published by a particular publisher for them to be in the database.

(<u>PublisherID</u>, City, Street, PublisherName, Email, Phone)

- PublisherID - **Integer** - Number used to uniquely identify publishers in the database.
- City - **String** - City where the publisher's main office is located.
- State - **2 characters** - The two-character abbreviation of the state where the publisher's main office is located.
- Street - **String** - Address where the publisher's main office is located. Includes the street number.
- PublisherName - **String** - Name of the publisher as it would appear on books.
- Email - **String** - Email of the publisher.
- PhoneNumber - **String** - Phone number of the publisher including extension code and hyphens used to break up the phone number.

BOOK - Represents the books stored in the database.

(ISBN, EditionNum, Title, Quantity, Publication_Year, Category, Price, Language, Num_Reviews, Publisher_ID)

- ISBN - **Integer** - 10 digit long number used to uniquely identify books in the database.
- EditionNum - **String** - The edition of the book. 3 character long string (i.e. 1st, 2nd).
- Title - **String** - Title of the book.
- Quantity - **Integer** - How many copies of the book are held across all warehouses.
- Publication_Year - **Integer** - Publication year of the book.
- Category - **String** - Category/Genre of the book.
- Price - **Float** - Price of the book before tax is applied.
- Language - **String** - Language the book is written in.
- Num_Reviews - **Integer** - Number of reviews that customers have given the book.
- Publisher_ID - **Integer** - ID of the publisher that published the book. Foreign key to Publisher entity.

TRANSACTION - Represents transactions made by customers buying books in the database.

(TransactionID, {CID}, OrderStatus, DateTimeCreation, DateTimeSale, Subtotal, TotalPrice)

- TransactionID - **Integer** - Number used to uniquely identify customer transactions.
- CID - **Integer** - Customer ID of customer who created the transaction either by putting books in a digital “cart” or by making a purchase. Foreign Key to Customer entity.
- OrderStatus - **String** - Single word description of the order status. (“Ordered”, “Processed”, “Shipped”, “Delivered”, “Canceled”)
- DateTimeCreation - **DateTime** - date and time when the transaction was “created” on the website.
- DateTimeSale - **DateTime** - date and time when the customer paid for the order.
- Subtotal - **Float** - Order cost prior to addition of tax and shipping.
- TotalPrice - **Float** - Order cost after addition of tax and shipping.

WRITTEN_BY - Bridge table between Book entities written by Author entities.

({ISBN}, {AuthorID})

- ISBN - **Integer** - 10 digit long number used to uniquely identify books in the database. Foreign Key to Book entity.
- AuthorID - **Integer** - Number used to uniquely identify authors in the database. Foreign Key to Author entity.

VIEWS - Bridge table between Customer entities viewing Book entities.

({CID}, {ISBN})

- CID - **Integer** - Customer ID. Number used to uniquely identify customers. Foreign Key to Customer entity.
- ISBN - **Integer** - 10 digit long number used to uniquely identify books in the database. Foreign Key to Book entity.
-

CONTAINS - Bridge table between Transaction entities containing Book entities.

({TransactionID}, {ISBN}, Qty)

- TransactionID - **Integer** - Number used to uniquely identify customer transactions. Foreign Key to Transaction entity.
- ISBN - **Integer** - 10 digit long number used to uniquely identify books in the database. Foreign Key to Book entity.
- Qty - **Integer** - Quantity of the individual books contained in the transaction. On a per book basis.

STORED_IN - Bridge table between Book entities being stored in Warehouse entities.

({City}, {Street}, {ISBN}, Quantity_Stored)

- City - **String** - City where the warehouse is located. Foreign Key to Warehouse entity.
- Street - **String** - Address where the warehouse is specifically located. Includes the street number of the warehouse. Foreign Key to Warehouse entity.
- ISBN - **Integer** - 10 digit long number used to uniquely identify books in the database. Foreign Key to Book entity.
- Quantity_Stored - **Integer** - Quantity of the individual books stored in the warehouse. On a per book basis.

Sample SQL Queries

1. Find the titles of all books by Pratchett that cost less than $10

   $\pi_{Title}(\sigma_{AuthorName = 'Pratchett'\ AND\ Price < 10}\ (Book\ *\ (Writtenby\ *\ Author))$

   ```
   SELECT Title FROM Book
   NATURAL JOIN WrittenBy
   NATURAL JOIN Author
   WHERE AuthorName LIKE ‘%Pratchett%’ AND Price < 10;
   ```

2. Give all the titles and their dates of purchase made by a single customer (you choose how to designate the customer)

   $\pi_{Customer.CID,\ Title,\ DateTime\ Sale}(\sigma_{Customer.CID = 1}\ (Transactions\ *_{TransactionID}(Contains\ *_{ISBN}\ Book))))$

   ```
   SELECT Title, DateTimeSale
   FROM Book NATURAL JOIN Contains
   NATURAL JOIN Transactions
   WHERE CID = 1;
   ```

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

   $\pi_{title,\ ISBN}(\sigma_{qty < 5}(Book))$

   ```
   SELECT Title, ISBN
   FROM Book
   WHERE Quantity < 5;
   ```

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

   $\pi_{customer.FirstnName,\ customer.LastName,Book.title}\ (Customer * (Transaction * (Contains * (Book * (Writtenby * (\sigma_{authorID='Pratchett'}(Author)))))))$

   ```
   SELECT FirstName, LastName, Title
   FROM Customer NATURAL JOIN Transactions
   NATURAL JOIN Contains
   NATURAL JOIN Book
   ```

```
NATURAL JOIN WrittenBy
NATURAL JOIN Author
WHERE AuthorName LIKE ‘%Pratchett%’;
```

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

$\mathcal{F}_{SUM\ (Contains.Qty)}\ (\sigma_{Customer.CID=1}(Transactions\ *_{TransactionID} Contains)$

```
SELECT Sum(Qty)
FROM Transactions NATURAL JOIN Contains
WHERE CID = 1;
```

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

$\mathcal{F}_{Customer.CID,MAX(SUM(Contains.Qty))}($
${}_{CID}\mathcal{F}_{SUM(Contains.qty)}(Customer\ *\ (Transaction\ *\ Contains))$
$)$

```
From Customer NATURAL JOIN (
        SELECT CID, Sum(Qty) as TotalBooks
        FROM Transactions NATURAL JOIN Contains
        GROUP BY CID
)
WHERE TotalBooks = (
        SELECT MAX (TotalBooks)
        FROM (
                SELECT CID, Sum(Qty) as TotalBooks
                FROM Transactions NATURAL JOIN Contains
                GROUP BY CID
        )
)
```

7. Find all Warehouses that store books written by Pratchett

$\pi_{address}(\sigma_{Author.AuthorName\ =\ 'Pratchett'}\ (StoredIn\ \ *\ (Books\ *\ (Author\ *\ WrittenBy)))$

```
SELECT City, Street
FROM Warehouse
NATURAL JOIN StoredIn
NATURAL JOIN WrittenBy
NATURAL JOIN Author
WHERE AuthorName LIKE ‘%Pratchett%’;
```

8. Find the average number of reviews written by customers

$\mathcal{F}_{AVG(COUNT(*))}({}_{CID}\mathcal{F}_{COUNT(*)}(Review))$

```
SELECT AVG(ReviewCount)
FROM (
        SELECT CID, COUNT(*) AS ReviewCount
        FROM Review
        GROUP BY CID
);
```

9. Find the average review score for books written by Pratchett

$\mathcal{F}_{AVG(Review.score)}(\sigma_{Author.Name='Pratchett'}(Review * (Book * (Writtenby * Author))))$

```
SELECT AVG(Score)
FROM Review
NATURAL JOIN Book
NATURAL JOIN WrittenBy
NATURAL JOIN Author
WHERE AuthorName LIKE ‘%Pratchett%’;
```

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

$\pi_{FirstName,\, LastName,\, SUM(TotalPrice)}(Customer * ({}_{CID}\mathcal{F}_{SUM(TotalPrice)}(Customer * Transactions)))$

```
SELECT FirstName, LastName, SUM(Transactions.TotalPrice) as TotalSpent
FROM Customer
NATURAL JOIN Transactions
GROUP BY CID;
```

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

$TotalSpent \rightarrow {}_{CID}\mathcal{F}_{SUM(TotalPrice)}(Transactions)$

$\pi_{FirstName,\, LastName,\, Email}(\sigma_{TotalSpent.TotalPrice > \mathcal{F}_{AVG(TotalSpent)}}(TotalSpent * (Customer * Transactions)))$

```
SELECT FirstName, LastName, Email
FROM Customer
NATURAL JOIN Transactions
GROUP BY CID
HAVING SUM(Transactions.TotalPrice) > (
        SELECT AVG(TotalSpent)
        FROM (
                SELECT SUM(Transactions.TotalPrice) as TotalSpent
                FROM Transactions
                GROUP BY CID
        )
);
```

12. 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.

    $TotalSold \rightarrow {}_{ISBN}\mathcal{F}_{Sum(Qty)}(Contains)$

    $\pi_{Title,\, TotalSold.Qty}(TotalSold\ *\ Book)$

    *No sort in relational algebra.

```
SELECT Title, SUM(Contains.Qty) as TotalSold
FROM Book
NATURAL JOIN Contains
GROUP BY Title
ORDER BY TotalSold DESC;
```

13. 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.

    ${}_{Title}\mathcal{F}_{SUM(Transactions.TotalPrice)}(Transactions\ *\ (\,Contains\ *\ Books))$

    *No sort in relational algebra.

```
SELECT Title, SUM(Transactions.TotalPrice) as TotalSold
FROM Book
NATURAL JOIN Contains
NATURAL JOIN Transactions
GROUP BY Title
ORDER BY TotalSold DESC;
```

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

$TotalBooks \rightarrow {}_{AuthorName}\mathcal{F}_{SUM(Contains.Qty)}(Contains\ *\ Book\ *\ WrittenBy\ *\ Author)$

$MaxQty \rightarrow \mathcal{F}_{MAX(TotalBooks.Qty)}(TotalBooks)$

$\pi_{AuthorName}(\sigma_{TotalBooks.Qty\ =\ MaxQty}(TotalBooks))$

```
SELECT AuthorName
FROM Author
NATURAL JOIN WrittenBy
NATURAL JOIN Book
NATURAL JOIN Contains
GROUP BY AuthorName
HAVING Contains.Qty = (
        SELECT MAX(Contains.Qty)
        FROM Author
        NATURAL JOIN WrittenBy
        NATURAL JOIN Book
        NATURAL JOIN Contains
        GROUP BY AuthorName

);
```

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

$TotalRevenue \rightarrow {}_{AuthorName}\mathcal{F}_{Sum(TotalPrice)}(Author\ *\ WrittenBy\ *\ Book\ *\ Contains\ *\ Transactions)$

$MaxRevenue \rightarrow \mathcal{F}_{MAX(TotalPrice)}(TotalRevenue)$

$\pi_{AuthorName}(\sigma_{TotalRevenue.TotalPrice\ =\ MaxRevenue}(TotalRevenue))$

```
SELECT AuthorName
FROM Author
NATURAL JOIN WrittenBy
NATURAL JOIN Book
NATURAL JOIN Contains
NATURAL JOIN Transactions
GROUP BY AuthorName
```

```
HAVING TotalPrice = (
        SELECT MAX(TotalPrice)
        FROM Author
        NATURAL JOIN WrittenBy
        NATURAL JOIN Book
        NATURAL JOIN Contains
        NATURAL JOIN Transactions
        GROUP BY AuthorName
);
```

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

$TotalRevenue \rightarrow {}_{AuthorName}\mathcal{F}_{Sum(TotalPrice)}(Author\ *\ WrittenBy\ *\ Book\ *\ Contains\ *\ Transactions)$

$MaxRevenue \rightarrow \mathcal{F}_{MAX(TotalPrice)}(TotalRevenue)$

$MostProfitableAuthor \rightarrow \pi_{AuthorName}(\sigma_{TotalRevenue.TotalPrice\ =\ MaxRevenue}(TotalRevenue))$

$\pi_{FirstName,\ LastName,\ Email,\ Phone,\ ShipppingBillingAddress}(\sigma_{AuthorName\ =\ MostProfitableAuthor}(Customer\ *\ Transactions\ *\ Contains\ *\ Book\ *\ WrittenBy\ *\ Author))$

```
SELECT FirstName, LastName, Email, Phone, ShippingBillingAddress
FROM Customer
NATURAL JOIN Transactions
NATURAL JOIN Contains
NATURAL JOIN Book
NATURAL JOIN WrittenBy
NATURAL JOIN Author
WHERE AuthorName IN (
                SELECT AuthorName
                FROM Author
                NATURAL JOIN WrittenBy
                NATURAL JOIN Book
                NATURAL JOIN Contains
                NATURAL JOIN Transactions
                GROUP BY AuthorName
                HAVING SUM(TotalPrice) = (
                        SELECT MAX(TotalRevenue)
                        FROM (
                                SELECT AuthorName, SUM(TotalPrice) AS TotalRevenue
                                FROM Author
                                NATURAL JOIN WrittenBy
                                NATURAL JOIN Book
```

```
                    NATURAL JOIN Contains
                    NATURAL JOIN Transactions
                    GROUP BY AuthorName
                ) AS AuthorTotalRevenue
            )
);
```

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

$TotalSpent \rightarrow {}_{CID}\mathcal{F}_{SUM(TotalPrice)}(Transactions)$

$AvgSpent \rightarrow \mathcal{F}_{SUM(TotalPrice)}(TotalSpent)$

$CustomersOverAvg \rightarrow \sigma_{TotalSpent.TotalPrice > AvgSpent}(TotalSpent)$

$\pi_{AuthorName}(CustomersOverAvg * Customer * Transactions * Contains * Book * WrittenBy * Author)$

```
SELECT AuthorName
FROM Author
NATURAL JOIN WrittenBy
NATURAL JOIN Book
NATURAL JOIN Contains
NATURAL JOIN Transactions
NATURAL JOIN Customer
WHERE CID IN (
        SELECT CID
        FROM Customer
        NATURAL JOIN Transactions
        GROUP BY CID
        HAVING SUM(TotalPrice) > (
                SELECT AVG(TotalSpent)
                FROM (
                        SELECT SUM(TotalPrice) as TotalSpent
                        FROM Transactions
                        GROUP BY CID
                )
        )
);
```

Inserting Data

BOOK - (<u>ISBN</u>, EditionNum, Title, Quantity, Publication_Year, Category, Price, Language, Num_Reviews, Publisher_ID)

INSERT INTO BOOK
VALUES (*ISBN, EditionNum, Title, Quantity, Publication_Year, Category, Price, Language, Num_Reviews, Publisher_ID*);

Example:
INSERT INTO BOOK
VALUES (1463027589, '2nd', 'Ranger Guide to Shoshone National Forest', 27, 1989, 'Outdoors Education', 13.99, 'English', 56, 'Wyoming Press');

PUBLISHER - *(<u>PublisherID</u>, City, Street, State, PublisherName, Email, PhoneNumber)*

INSERT INTO PUBLISHER
VALUES (*PublisherID, City, Street, State, PublisherName, Email, PhoneNumber*);

Example:
INSERT INTO PUBLISHER
VALUES (74, 'Cheyenne', '350 W 24th St', 'WY', 'Wyoming Press', 'contactwp@wyopress.com', '+1-307-233-1746');

AUTHOR - *(<u>AuthorID</u>, AuthorName, DOB, Nationality)*

INSERT INTO Author
VALUES (*AuthorID, AuthorName, DOB, Nationality*);

Example:
INSERT INTO Author
VALUES (163, 'Delilah A. June', 08-23-1946, 'American')

CUSTOMER - *(<u>CID</u>, FirstName, LastName, ShippingAddress, BillingAddress, CCardNum, CCardExpiryDate, CCardSecCode, CCardFirstName, CCardLastName, Email, Phone)*

INSERT INTO Customer
VALUES (*CID, FirstName, LastName, ShippingAddress, BillingAddress, CCardNum, CCardExpiryDate, CCardSecCode, CCardFirstName, CCardLastName, Email, Phone*)

Example:
INSERT INTO Customer
VALUES (1330, 'Henry', 'Jules', '1537 Pine St. Boulder, Colorado, 80302', '1537 Pine St. Boulder, Colorado, 80302', 7591357169425158, 02-31-1990, 757, 'Henry', 'Jules', 'henryj89tfl@gmail.com', 3033062414)

Deleting Data

BOOK - (ISBN, EditionNum, Title, Quantity, Publication_Year, Category, Price, Language, Num_Reviews, Publisher_ID)

```
DELETE FROM BOOK
WHERE ISBN = XXXXXXXXX;
```

Example:

DELETE FROM BOOK
WHERE ISBN = 1463027589;

Deleting a book from the database will have the highest impact compared to deleting any other entity. When deleting a book from the database, data will also need to be deleted from almost every other entity.

Customer: Tuples with the book's ISBN *must* be deleted in the Views bridge table.
Transaction: Tuples with the book's ISBN *must* be deleted in the Contains bridge table.
Review: Tuples with the book's ISBN *could* be deleted in the Review entity. However, it may be (very) useful to leave these in the database in the event the book is readded so that customers don't have their reviews deleted.
Author: Tuples with the book's ISBN *could* be deleted in the Written_By bridge table.
Warehouse: Tuples with the book's ISBN *could* be deleted in the Stored_In bridge table.
Publisher: Will not be affected by deleting a book since the publisher ID is stored in the book table and not in a bridge table or in the publisher entity.

PUBLISHER - *(PublisherID, City, Street, State, PublisherName, Email, PhoneNumber)*

```
DELETE FROM PUBLISHER
WHERE PublisherID = XXX;
```

Example:

DELETE FROM PUBLISHER
WHERE PublisherID = 74;

Deleting a publisher from the database will require updating the Book entity since the publisher ID is stored in the book entity. There are two ways to go about this. Deleted publisher IDs in the book table can be set to null, in which case the publisher for the book will be unknown but not much else is affected. Otherwise if a publisher is being deleted it usually means all of their published books are being removed from the database too, in which case the book's tuple with the publisher ID should be deleted which would follow the procedure for deleting a book entity.

AUTHOR - *(<u>AuthorID</u>, AuthorName, DOB, Nationality)*

```
DELETE FROM Author
WHERE AuthorID = XXX;
```

Example:

DELETE FROM Author
WHERE AuthorID = 163;

Deleting an author from the database works similarly in concept to deleting a publisher with the small change that there is a bridge table in between Author and Book instead of a direct connection. There are two ways to handle deleting an author. The author's tuple can be deleted and the tuples with the corresponding Author IDs in the Written_By bridge table can have their Author IDs set to null, in which case the authors for the book entities will have unknown authors. The other method follows how deleting a publisher is handled, as if an author is being deleted, it most likely means their books will also be removed from the database, in which case the procedure for deleting a book entity above should be used.

CUSTOMER - *(<u>CID</u>, FirstName, LastName, ShippingAddress, BillingAddress, CCardNum, CCardExpiryDate, CCardSecCode, CCardFirstName, CCardLastName, Email, Phone)*

```
DELETE FROM Customer
WHERE CID = XXXXX
```

Example:

DELETE FROM Customer
WHERE CID = 1330;

Deleting a customer entity will affect their corresponding Transaction and Review entities and the Views bridge table. The transactions with the corresponding CID can be deleted as they will have no corresponding customer, or they can more likely have the CID set to null without being deleted in order to maintain a history of what transactions have been made, potentially for financial/legal purposes. These transactions can then later be deleted once enough time has passed that a history of them is no longer useful based on their CIDs being null. The tuples in the Views bridge table with the corresp. CID should be deleted. The reviews that the customer has made can be deleted, but they should likely instead have the CID set to null so that other customers can still see their review and ratings of the books they gave reviews to.

Other entities/bridge table relationships:

TRANSACTION: *(TransactionID, {CID}, OrderStatus, DateTimeCreation, DateTimeSale, Subtotal, TotalPrice)*

```
DELETE FROM Transactions
WHERE TransactionID = 85;
```

REVIEW: *(ReviewID, {CID}, {ISBN}, Review, ReviewDate, Score)*

```
DELETE FROM Review
WHERE ReviewID = 13;
```

WAREHOUSE: *(City, Street)*

```
DELETE FROM Warehouse
WHERE City = 'Waterbury' AND Street = '94 Clover Lane';
```

WRITTEN_BY: *({ISBN}, {AuthorID})*

```
DELETE FROM WrittenBy
WHERE ISBN = 0324107502 AND AuthorID = 72;
```

VIEWS: *({CID}, {ISBN})*

```
DELETE FROM Views
WHERE CID = 7 AND ISBN = 0201615762;
```

CONTAINS: *({TransactionID}, {ISBN}, Qty)*

```
DELETE FROM Contains
WHERE TransactionID = 4 AND ISBN = 0425188361;
```

STORED_IN: *({City}, {Street}, {ISBN}, Quantity_Stored)*

```
DELETE FROM StoredIn
WHERE City = 'Manchester' AND Street = '87 Spruce Way' AND ISBN =
        0782140114;
```

PROJECT CHECKPOINTS:

Project Checkpoint 01

Names: Date: 01/23/2025

Ethan Conley, Connor Fricke, Sierra Reis

In a **NEATLY TYPED** document, provide the following:

1. Based on the requirements given in the project overview, list the entities to be modeled in this database. For each entity, provide a list of associated attributes.

   *[Entity: **Key Attribute**, Attribute]*

   Books: **ISBN**, Title, ~~Author(s), Publisher,~~ (promoted to full entities), Year, Price, Category, Qty, Edition, Language, Num Reviews

   Customer: **CID (customer id #)**, First name, Last name, Shipping/billing addresses, credit card information, contact info (email/phone)

   Transaction: **Transaction#**, CID, Subtotal $, Total $ (incl. tax, shipping, etc.), Date/Time of Sale, Date/Time of Creation, Order status

   Publisher: **Publisher ID**, Publisher Name, Location, Contact Info

2. Based on the requirements given in the project overview, what are the various relationships between entities? (For example, "CUSTOMER entities purchase BOOK entities").

   CUSTOMER entities can *view* BOOK entities (N <-> M)

   CUSTOMER entities can *create and modify* TRANSACTION entities (1 <-> N)

   TRANSACTION entities *contain* BOOK entities (N <-> M)

   - This relationship contains an attribute for Quantity

   PUBLISHER entities *publish* BOOK entities (N <-> M)

3. Propose at least two additional entities that it would be useful for this database to model beyond the scope of the project requirements. Provide a list of possible attributes for the additional entities and possible relationships they may have with each other and the rest of the entities in the database. Give a brief, one sentence rationale for why adding these entities would be interesting/useful to the stakeholders for this database project.

Review: **Review ID**, CID, ISBN, date of review, Score (i.e. stars out of 5), review text

Relationship: CUSTOMER *writes* REVIEW (1 <-> N)

Relationship: REVIEW(S) *is for* BOOK (N <-> 1)

- Useful entity to have so that customers can leave reviews and read reviews of others, and the admin can see book performances
- We will be assuming there are no anonymous reviews allowed

Author: **Author ID**, Name, DOB, Nationality

Relationship: AUTHOR(S) *writes* BOOK (N <-> M)

- Useful entity to store information about a specific author in case users want to know more about an author and what books they write, and to query about all the books by specific authors that are held in stock.

Warehouse: **Address (Compound - City, Street, etc.)**, Book ISBN, Quantity Stored

Relationship: WAREHOUSE(S) *contains* BOOKS (N <-> M)

- Useful entity to more specifically inform employees about book stock and where it is so that they may more accurately manage book storage.

4. Give at least four examples of some informal queries/reports that it might be useful for this database might be used to generate. Include one example for each of the additional entities you proposed in question 3 above.

1. Query all reviews a customer has written and their associated books.
2. Query all books that an author writes
3. Query the transaction history of a particular customer
4. Query transactions status so that a customer can check up on their order tracking information
5. Query all books below a threshold quantity in the database so that an admin knows to order more from the publisher to maintain stock.

5. Suppose we want to add a new publisher to the database. How would we do that given the entities and relationships you've outlined above? Given your above description, is it possible to add a new publisher to your database without knowing the title of any books they have published? If not, revise your model to allow for publishers to be added as separate entities.

You would add a publisher type entity to a publisher table, which does not contain an attribute for the books the publisher has published (for saving memory). To be able to add a publisher without knowing any of the books they have published, the relationship between Publisher and Books must not be a required relationship. This would be no problem to implement, because the N <-> M relationship means that a bridge table is required for the publisher <-> book relationship. There simply wouldn't be any rows that contain this publisher's ID.

6. Determine at least three other informal update operations and describe what entities would need to have attributes altered and how they would need to be changed given your above descriptions. Include one example for each of the additional entities you proposed in question 3 above.

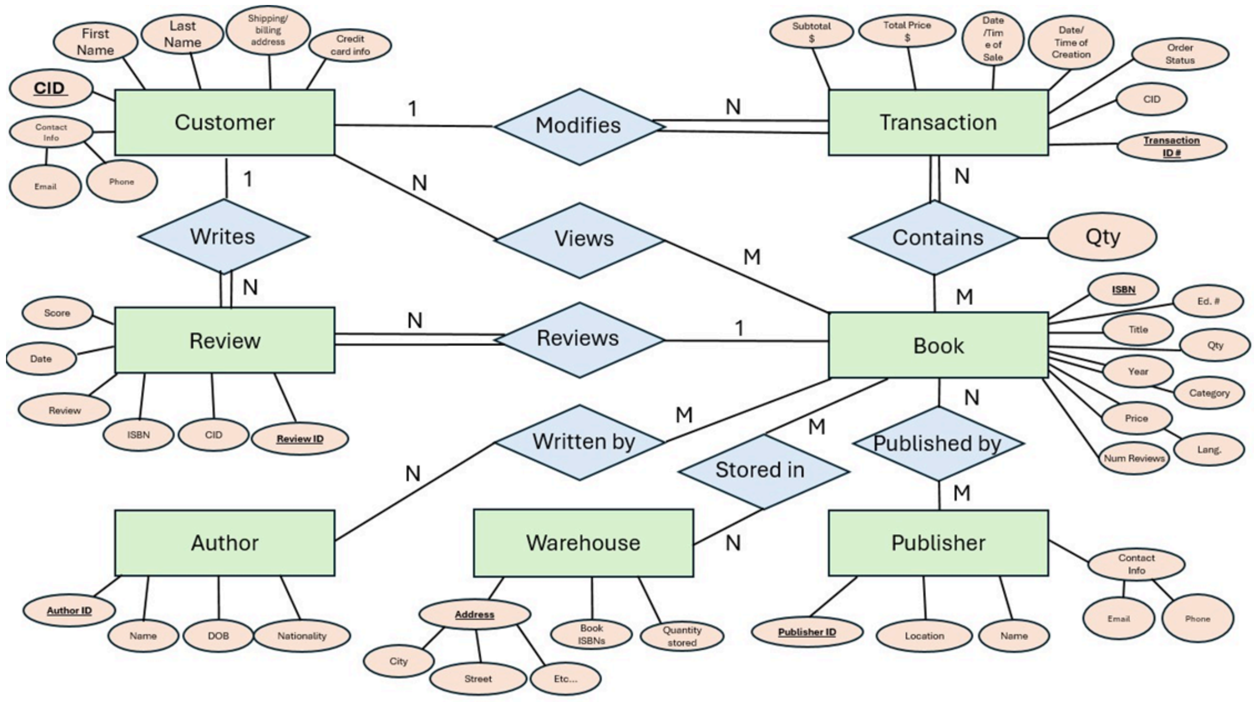
1. Create / modify transaction update requires updating attributes of the transaction but also requires updating the quantity available for sale (attribute of BOOK entity and potentially WAREHOUSE entity if included). *Example*: user buys 10 copies of a book. The transaction must be updated to reflect these 10 books, but also the book entity has a quantity that must be decreased by 10 when this transaction occurs.

2. Create / modify author information would only require updating the author entities because the information for an author is only stored in the Author entity. As long as the primary key associated with an Author does not change, then this update would automatically propagate through the database. *Example:* if an author has their DOB entered into the system, all book entities which reference that author with a foreign key would automatically see that change reflected via the relation.

3. Updating/Adding reviews – requires updating the review entity themselves, of course. Some sort of attribute on books like an average rating would require updating to reflect the new review accounted for in the average. Though the average rating attribute could be implemented as a *derived* attribute such that the update need not be made manually but is instead accounted for by calculation of the derived attribute. *Example*: One hundred new customers leave reviews on a book in a single day. Obviously each of these reviews needs their own review instances within the database. If the avg rating attribute is calculated automatically (say, by the website querying all reviews and averaging them before displaying the rating), then the book entity need not be accessed by the update operation.

7. Provide an ER diagram for your database. Make sure you include all of the entities and relationships you determined in the questions above ***INCLUDING the entities for question 3 above*** and remember that ***EVERY*** entity in your model needs to connect to another entity in the model via some kind of relationship.

(Project Checkpoint 1 ER Diagram)

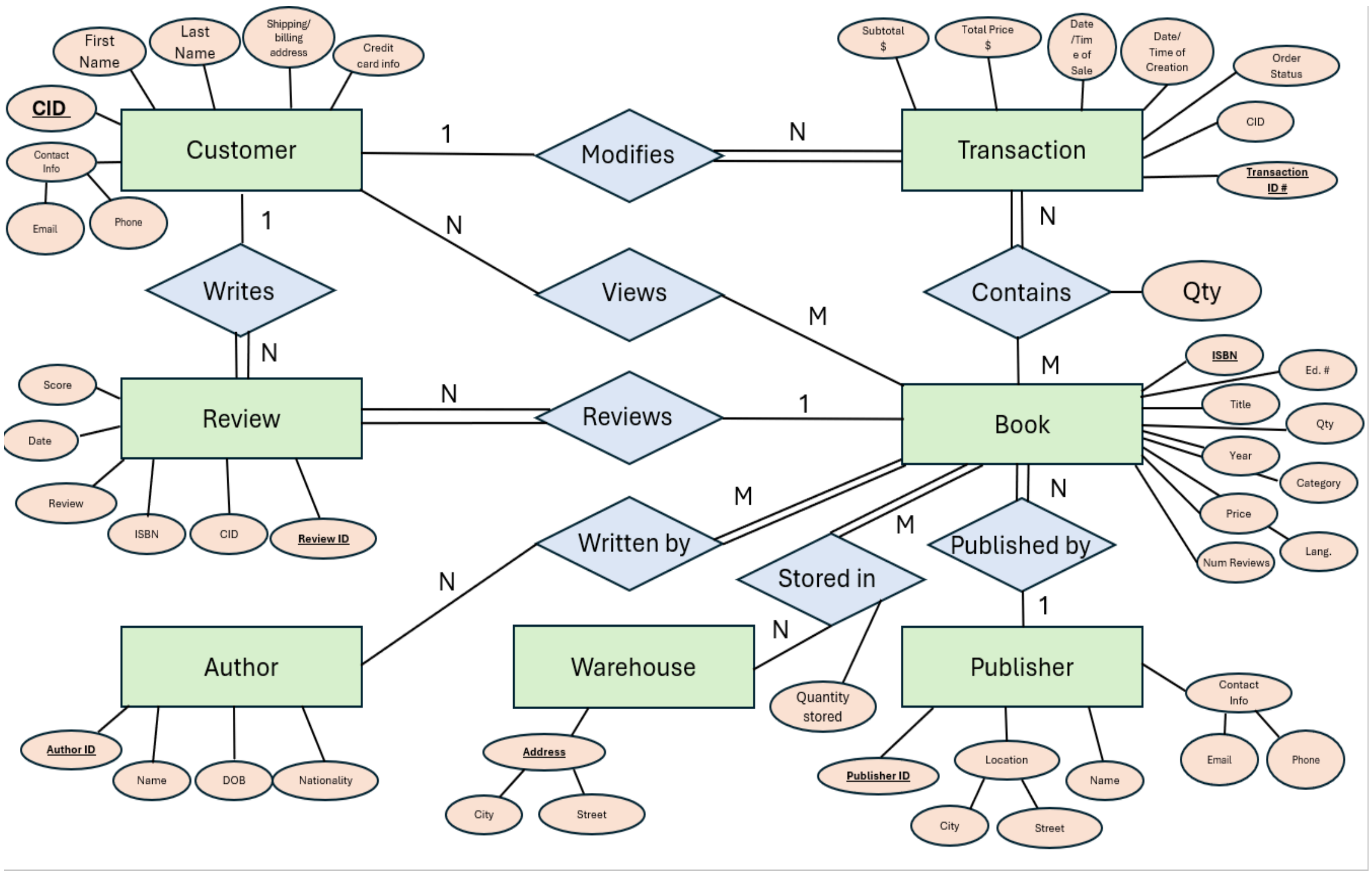


Project Checkpoint 02

Ethan Conley, Sierra Reis, Connor Fricke
14-Feb-2025

Symbols - ρ π σ ∩ ∪ ⋈ × ⟖

1) ER MODEL:



2) ER MODEL TO RELATIONAL SCHEMA

Primary keys are underlined, {foreign keys are curly braces}

CUSTOMER (CID, First name, Last name, shipping/billing address, credit card info, email, phone)

REVIEW (Review ID, {CID}, {ISBN}, Review, Date, Score)

AUTHOR (AuthorID, Name, DOB, Nationality)

WAREHOUSE (City, Street)

PUBLISHER (PublisherID, City, Street, Name, Email, Phone)

BOOK (ISBN, Edition_#, Title, Quantity, Publication_Year, Category, Price, Language, Num_Reviews)

TRANSACTION (Transaction_ID, {CID}, Order_Status, Date/Time_of_Creation, Date/Time_of_Sale, Subtotal, Total_Price)

WRITTEN_BY({ISBN},{authorID})

VIEWS({CID}, {ISBN})

CONTAINS({transactionID}, {ISBN}, qty)

STORED_IN({address}, {ISBN}, Quantity_stored)

3) QUERIES (RELATIONAL ALGEBRA FORM)

ρ π σ ∩ ∪ ⋈ × ⋊ ∗

a. Find the titles of all books by Pratchett that cost less than $10

$\pi_{Title}(\sigma_{Name = 'Pratchett'\ AND\ Price < 10}\ (Book\ *\ (Written\_By\ *\ Author))$

b. Give all the titles and their dates of purchase made by a single customer (you choose how to designate the customer)

$\pi_{Customer.CID,\ Title,\ Date/Time\ of\ Sale}(\sigma_{Customer.CID}\ (Customer\ *_{CID}\ (Transaction\ \ *_{TransactionID}(\ Contains\ *_{ISBN}\ Book))))$

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

$\pi_{title,\ ISBN}(\sigma_{qty < 5}(Book))$

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

$\pi_{customer.Fname,\ customer.Lname, Book.title}($
$Customer * (Transaction * (Contains * (Book * (Written\_By * (\sigma_{authorID='Pratchett'}(Author))))))$
$)$

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

$\mathcal{F}_{SUM\ Contains.Qty}\ (\sigma_{Customer.CID}(Customer *_{CID} (Transaction\ *_{TransactionID}(Contains *_{ISBN} Book))))$

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

$\mathcal{F}_{Customer.CID, MAX(SUM(Contains.Qty))}($
${}_{CID}\mathcal{F}_{SUM(Contains.qty)}(Customer\ *\ (Transaction\ *\ (Contains\ *\ (Book))))$
$)$

4) ADDITIONAL QUERIES
Come up with three additional interesting queries that your database can provide. Give what the queries are supposed to retrieve in plain English and then as relational algebra. Your queries should include joins and at least one should include an aggregate function. At least one of your queries should use “extra” entities you added to your model in Checkpoint 01.

a) Find all Warehouses that store books written by Pratchett
$\pi_{address}(\sigma_{Author.Name\,=\,'Pratchett'}\ (StoredIn\ \ *\ (Books\ *\ (Author\ *\ WrittenBy)))$

b) Find the average number of reviews written by customers
$\mathcal{F}_{AVG(COUNT(*))}({}_{CID}\mathcal{F}_{COUNT(*)}(Review))$

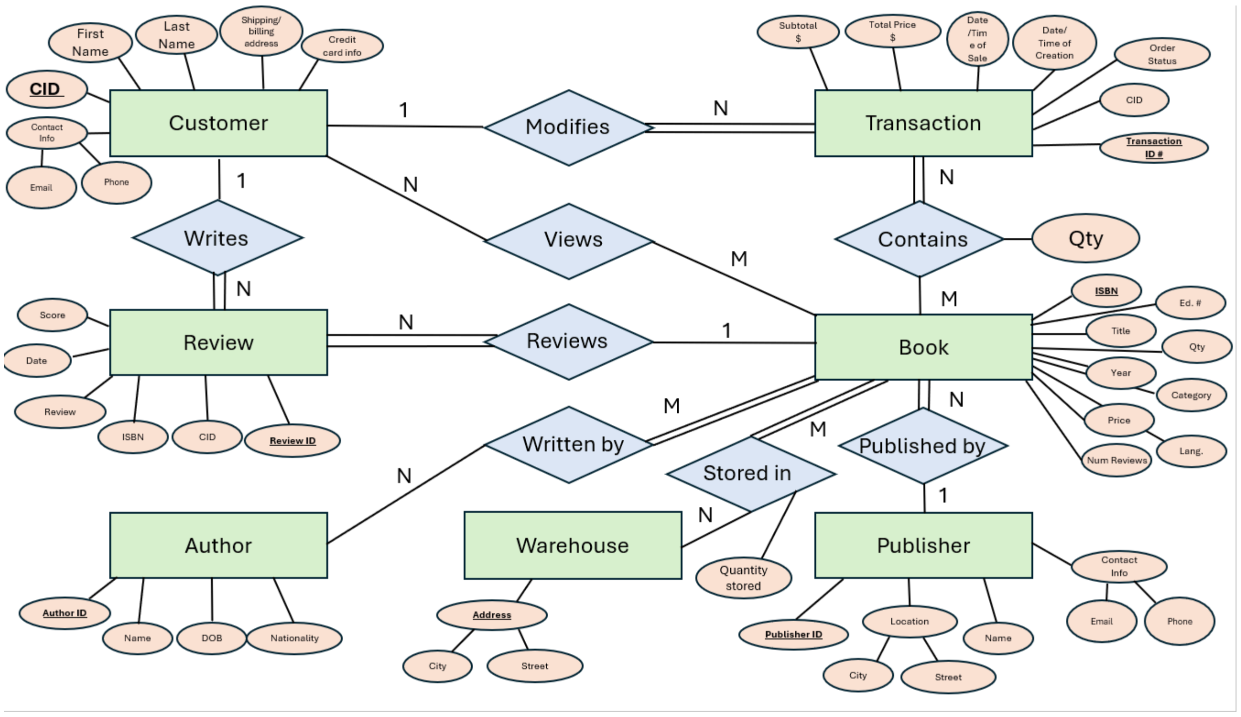
c) Find the average review score for books written by Pratchett

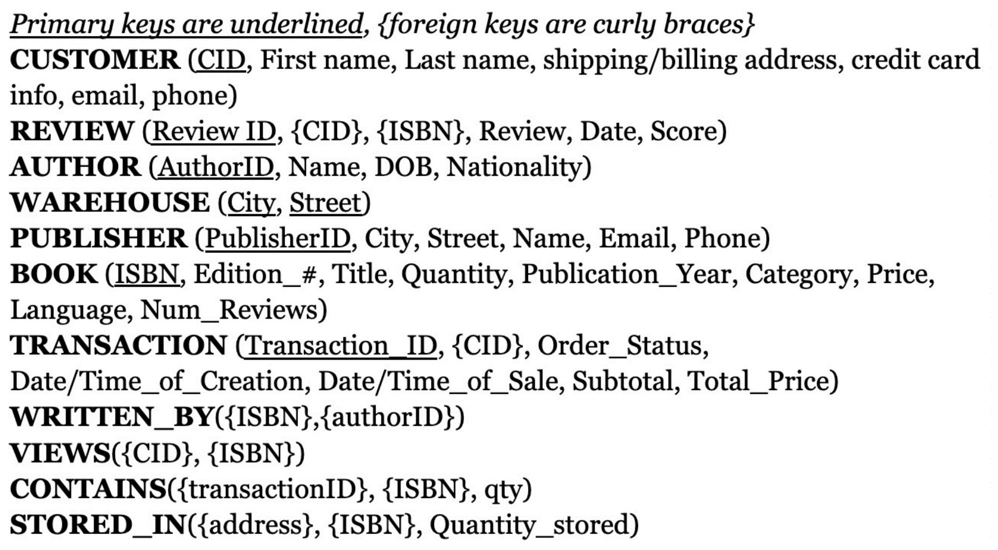
$\mathcal{F}_{AVG(Review.score)}(\sigma_{Author.Name='Pratchett'}(Review\ *\ (Book\ *\ (Writtenby\ *\ Author))))$

Project Checkpoint 03

Names: Ethan Conley, Connor Fricke, Sierra Reis
Date: 03/19/2025

1. 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.





Primary keys are underlined, {foreign keys are curly braces}
CUSTOMER (CID, First name, Last name, shipping/billing address, credit card info, email, phone)
REVIEW (Review ID, {CID}, {ISBN}, Review, Date, Score)
AUTHOR (AuthorID, Name, DOB, Nationality)
WAREHOUSE (City, Street)
PUBLISHER (PublisherID, City, Street, Name, Email, Phone)
BOOK (ISBN, Edition_#, Title, Quantity, Publication_Year, Category, Price, Language, Num_Reviews)
TRANSACTION (Transaction_ID, {CID}, Order_Status, Date/Time_of_Creation, Date/Time_of_Sale, Subtotal, Total_Price)
WRITTEN_BY({ISBN},{authorID})
VIEWS({CID}, {ISBN})
CONTAINS({transactionID}, {ISBN}, qty)
STORED_IN({address}, {ISBN}, Quantity_stored)

2. 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.

```
CREATE TABLE Customer(
        CID                         INT                 not null,
        FirstName                   VARCHAR(16)         not null,
        LastName                    VARCHAR(32)         not null,
        ShippingBillingAddress      VARCHAR(64)         not null,
        CreditCardInfo              VARCHAR(64)         not null,
        Email                       VARCHAR(32),
        Phone                       VARCHAR(16),
        Primary Key(CID)
);

CREATE TABLE Transactions(
        TransactionID               INT                 NOT NULL,
        CID                         INT                 NOT NULL,
        OrderStatus                 VARCHAR(16)         NOT NULL,
        DateTimeCreation            DATETIME,
        DateTimeSale                DATETIME,
        Subtotal                    DECIMAL(2, 2),
        TotalPrice                  DECIMAL(2, 2),
        PRIMARY KEY(Transaction_ID),
        FOREIGN KEY (CID) REFERENCES Customer(CID)
);
```

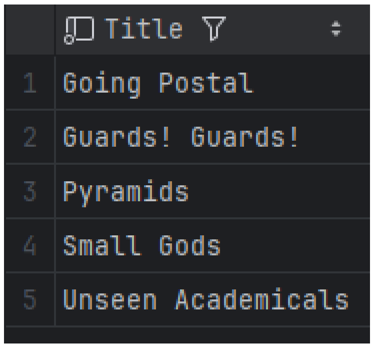
```
CREATE TABLE Review(
        ReviewID                int                     not null,
        CID                     int                     not null,
        ISBN                    char(10)                not null,
        Review                  text,
        ReviewDate              date,
        Score                   int,
        Primary Key(ReviewID),
        Foreign Key(CID) references Customer(CID),
        Foreign Key(ISBN) references Book(ISBN)
);

CREATE TABLE Book (
        ISBN                    char(10)                not null,
        EditionNumber           int,
        Title                   varchar(32)             not null,
        Quantity                int,
        PublicationYear         int,
        Category                varchar(32),
        Price                   DECIMAL(2,2)            not null,
        Language                varchar(16),
        NumReviews              int,
        PublisherID             int                     not null,
        Primary Key(ISBN),
        FOREIGN KEY (PublisherID) REFERENCES Publisher(PublisherID)
);

CREATE TABLE Author (
        AuthorID                int                     not null,
        Name                    varchar(64)             not null,
        DOB                     date,
        Nationality             varchar(64),
        Primary Key(AuthorID)
);

CREATE TABLE Warehouse (
        City                    varchar(32)             not null,
        Street                  varchar(32)             not null,
        Primary Key(City, Street)
);
```

```
CREATE TABLE Publisher(
        PublisherID             INT             NOT NULL,
        City                    VARCHAR(16),
        Street                  VARCHAR(64),
        Name                    VARCHAR(32)     NOT NULL,
        Email                   VARCHAR(32),
        Phone                   CHAR(10)        NOT NULL,
        PRIMARY KEY  (PublisherID)
);

CREATE TABLE WrittenBy (
        ISBN                    CHAR(10)        NOT NULL,
        AuthorID                INT             NOT NULL,
        FOREIGN KEY (ISBN) REFERENCES Book(ISBN),
        FOREIGN KEY (AuthorID) REFERENCES Author(AuthorID)
);

CREATE TABLE Views (
        CID                     INT             NOT NULL,
        ISBN                    CHAR(10)        NOT NULL,
        FOREIGN KEY(CID) REFERENCES Customer(CID),
        FOREIGN KEY(ISBN) REFERENCES Book(ISBN)
);

CREATE TABLE Contains (
        TransactionID           INT             NOT NULL,
        ISBN                    CHAR(10)        NOT NULL,
        Qty                     INT             NOT NULL,
        FOREIGN KEY (TransactionID) REFERENCES Transactions(TransactionID)
);

CREATE TABLE StoredIn (
        City                    VARCHAR(16)     NOT NULL,
        Street                  VARCHAR(64)     NOT NULL,
        ISBN                    CHAR(10)        NOT NULL,
        QuantityStored          INT             NOT NULL,
        FOREIGN KEY(City) REFERENCES Warehouse(City),
        FOREIGN KEY(Street) REFERENCES Warehouse(Street),
        FOREIGN KEY(ISBN) REFERENCES Book(ISBN)
);
```

3. 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

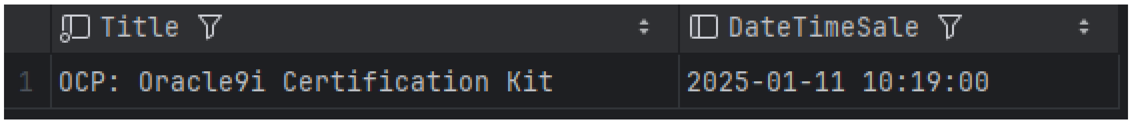
```
SELECT Title FROM Book
NATURAL JOIN WrittenBy
NATURAL JOIN Author
WHERE AuthorName LIKE ‘%Pratchett%’ AND Price < 10;
```



	Title
1	Going Postal
2	Guards! Guards!
3	Pyramids
4	Small Gods
5	Unseen Academicals

b. Give all the titles and their dates of purchase made by a single customer (you choose how to designate the customer)

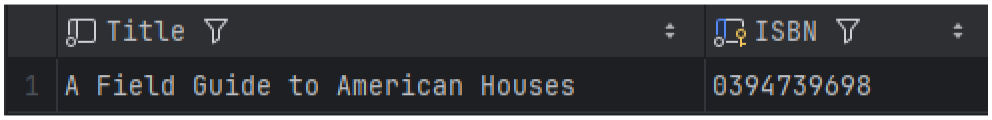
```
SELECT Title, DateTimeSale
FROM Book NATURAL JOIN Contains
NATURAL JOIN Transactions
WHERE CID = 1;
```



	Title	DateTimeSale
1	OCP: Oracle9i Certification Kit	2025-01-11 10:19:00

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

```
SELECT Title, ISBN
FROM Book
WHERE Quantity < 5;
```



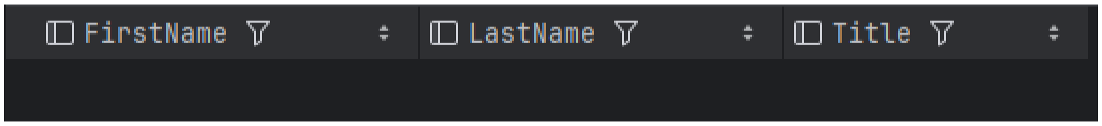
	Title	ISBN
1	A Field Guide to American Houses	0394739698

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

```
SELECT FirstName, LastName, Title
```

FROM Customer NATURAL JOIN Transactions
NATURAL JOIN Contains
NATURAL JOIN Book
NATURAL JOIN WrittenBy
NATURAL JOIN Author
WHERE AuthorName LIKE '%Pratchett%';

No output for our sample data,



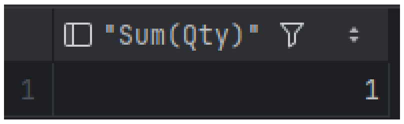
```
FirstName    LastName    Title
```

but would expect something like:

FirstName	**LastName**	**Title**
Jane	**Smith**	**Going Postal**

etc…

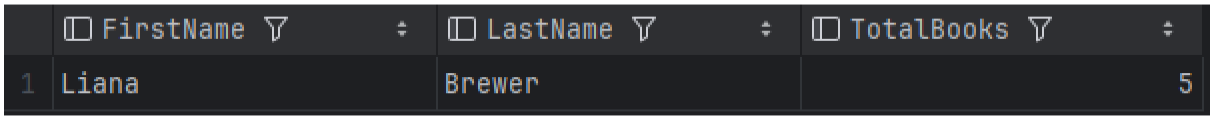
e. Find the total number of books purchased by a single customer (you choose how to designate the customer)
SELECT Sum(Qty)
FROM Transactions NATURAL JOIN Contains
WHERE CID = 1;



```
  "Sum(Qty)"
1          1
```

f. Find the customer who has purchased the most books and the total number of books they have purchased
SELECT FirstName, LastName, TotalBooks
From Customer NATURAL JOIN (
        SELECT CID, Sum(Qty) as TotalBooks
        FROM Transactions NATURAL JOIN Contains
        GROUP BY CID
)
WHERE TotalBooks = (
        SELECT MAX (TotalBooks)
        FROM (
                SELECT CID, Sum(Qty) as TotalBooks
                FROM Transactions NATURAL JOIN Contains

```
                    GROUP BY CID
            )
    )
```



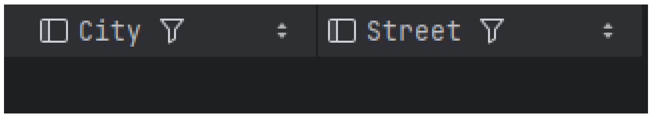
	FirstName	LastName	TotalBooks
1	Liana	Brewer	5

4. 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".

a) Find all Warehouses that store books written by Pratchett

```
SELECT City, Street
FROM Warehouse
NATURAL JOIN StoredIn
NATURAL JOIN WrittenBy
NATURAL JOIN Author
WHERE AuthorName LIKE '%Pratchett%';
```

No output for our sample data,



City	Street

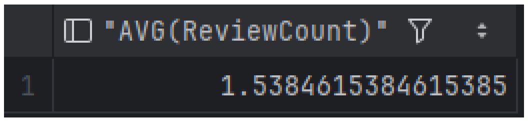
But would expect something like:

City	Street
Savannah	**123 Oak Avenue**

etc…

b) Find the average number of reviews written by customers

```
SELECT AVG(ReviewCount)
FROM (
        SELECT CID, COUNT(*) AS ReviewCount
        FROM Review
        GROUP BY CID
);
```

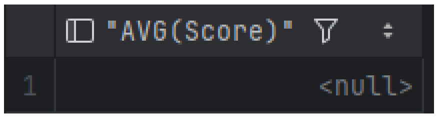


```
"AVG(ReviewCount)"
1    1.5384615384615385
```

c) Find the average review score for books written by Pratchett

```
SELECT AVG(Score)
FROM Review
NATURAL JOIN Book
NATURAL JOIN WrittenBy
NATURAL JOIN Author
WHERE AuthorName LIKE '%Pratchett%';
```

No output for our sample data,



```
"AVG(Score)"
1    <null>
```

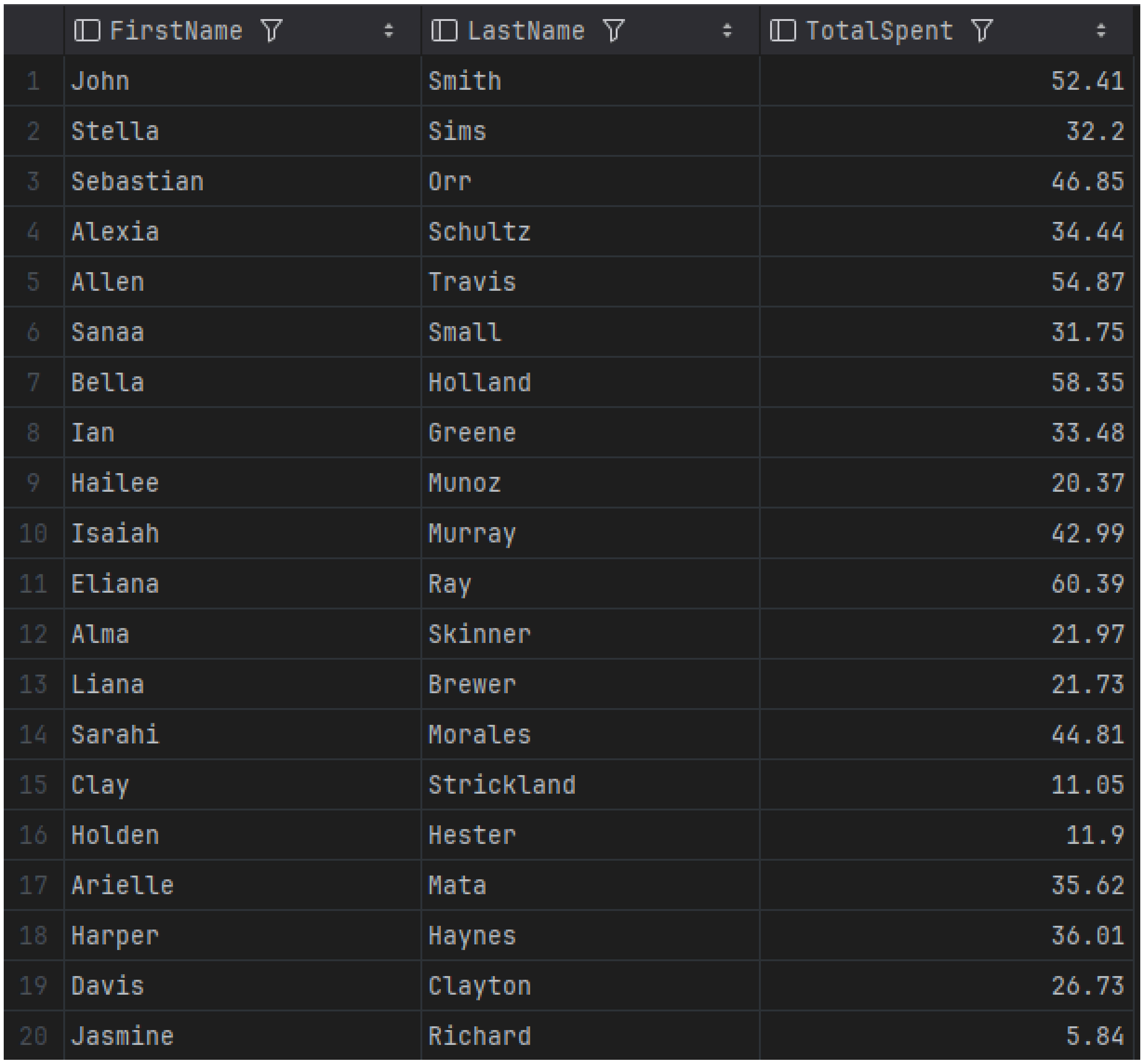
But would expect something like:

AVG(Score)
4.7

5. 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.

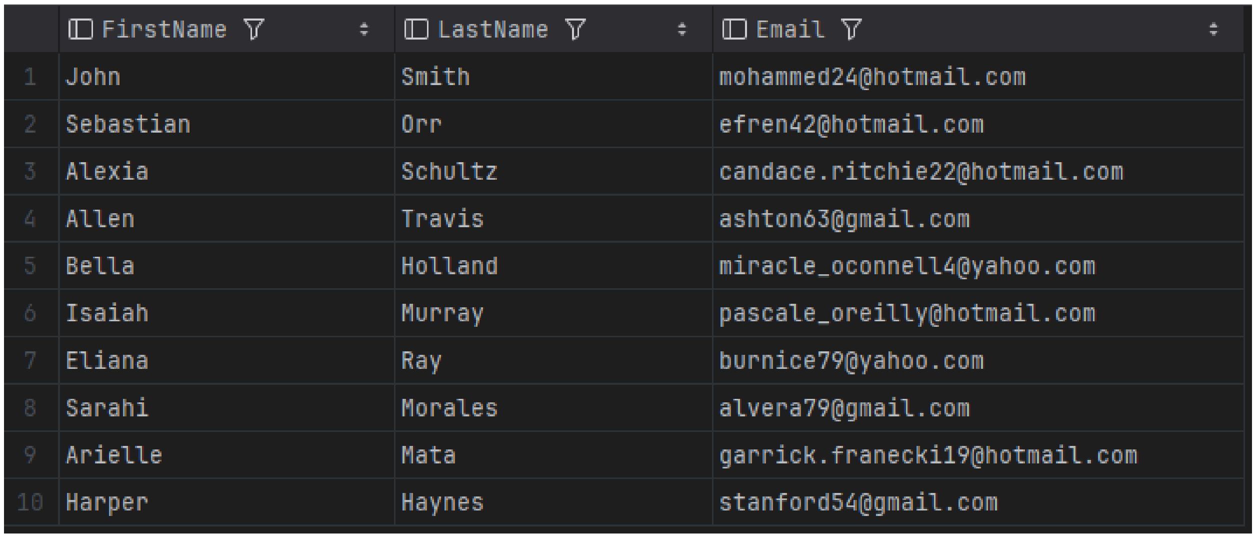
```
SELECT FirstName, LastName, SUM(Transactions.TotalPrice) as TotalSpent
FROM Customer
NATURAL JOIN Transactions
GROUP BY CID;
```



	FirstName	LastName	TotalSpent
1	John	Smith	52.41
2	Stella	Sims	32.2
3	Sebastian	Orr	46.85
4	Alexia	Schultz	34.44
5	Allen	Travis	54.87
6	Sanaa	Small	31.75
7	Bella	Holland	58.35
8	Ian	Greene	33.48
9	Hailee	Munoz	20.37
10	Isaiah	Murray	42.99
11	Eliana	Ray	60.39
12	Alma	Skinner	21.97
13	Liana	Brewer	21.73
14	Sarahi	Morales	44.81
15	Clay	Strickland	11.05
16	Holden	Hester	11.9
17	Arielle	Mata	35.62
18	Harper	Haynes	36.01
19	Davis	Clayton	26.73
20	Jasmine	Richard	5.84

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

```
SELECT FirstName, LastName, Email
FROM Customer
NATURAL JOIN Transactions
GROUP BY CID
HAVING SUM(Transactions.TotalPrice) > (
        SELECT AVG(TotalSpent)
        FROM (
                SELECT SUM(Transactions.TotalPrice) as TotalSpent
                FROM Transactions
                GROUP BY CID
        )
);
```

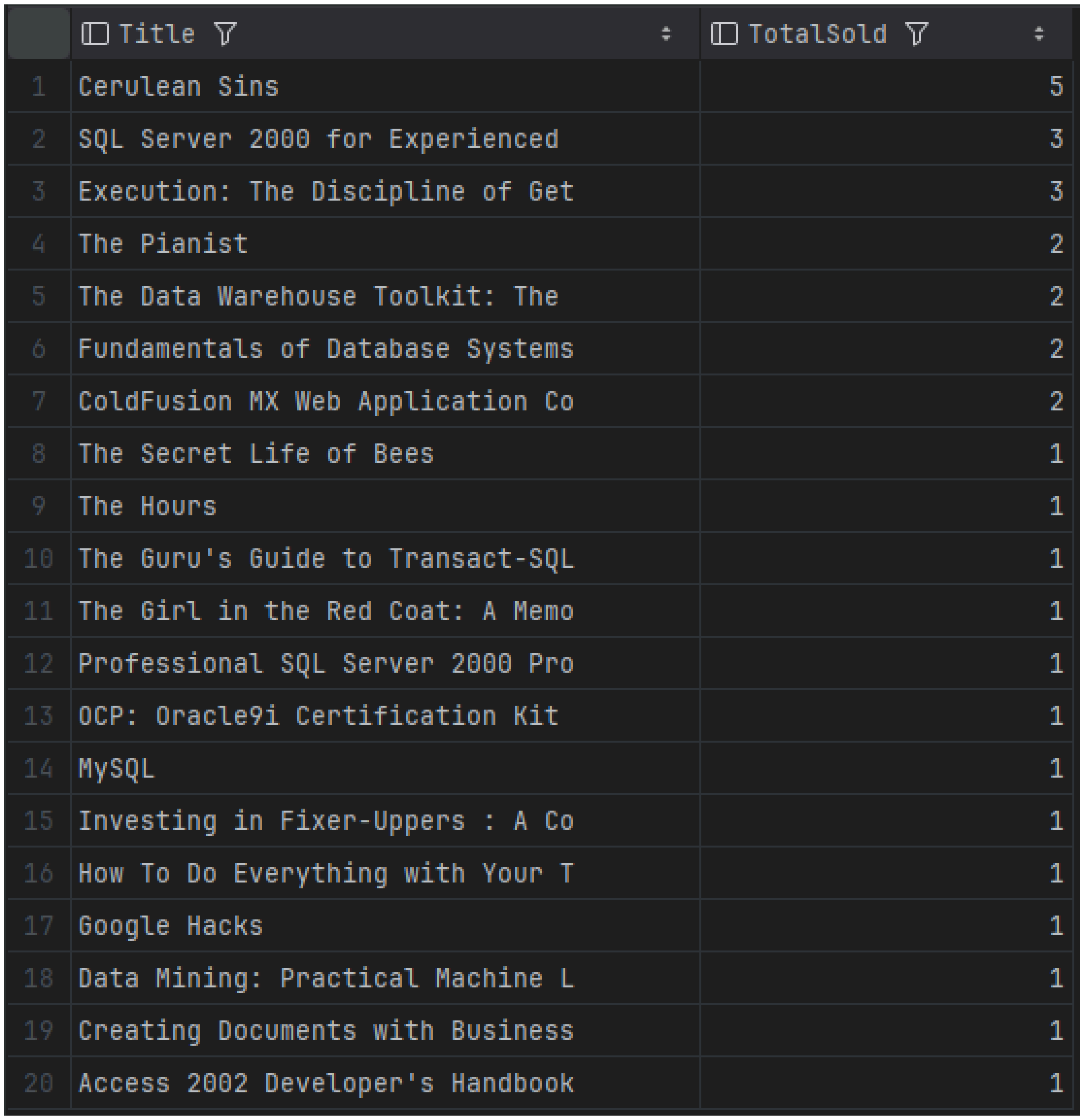


	FirstName	LastName	Email
1	John	Smith	mohammed24@hotmail.com
2	Sebastian	Orr	efren42@hotmail.com
3	Alexia	Schultz	candace.ritchie22@hotmail.com
4	Allen	Travis	ashton63@gmail.com
5	Bella	Holland	miracle_oconnell4@yahoo.com
6	Isaiah	Murray	pascale_oreilly@hotmail.com
7	Eliana	Ray	burnice79@yahoo.com
8	Sarahi	Morales	alvera79@gmail.com
9	Arielle	Mata	garrick.franecki19@hotmail.com
10	Harper	Haynes	stanford54@gmail.com

(Note: The emails do not correspond to the persons actual name, they were randomly generated)

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.

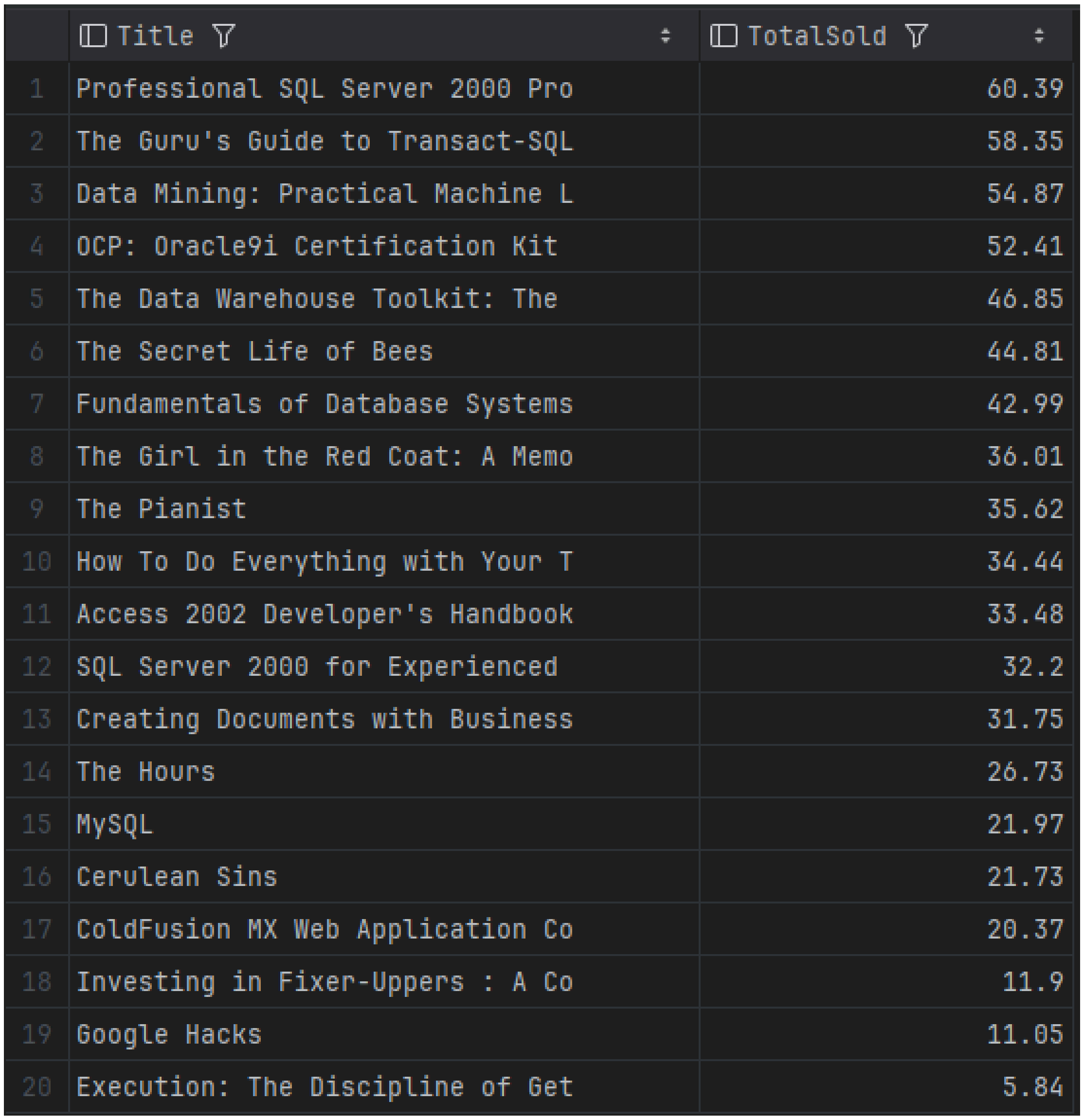
```
SELECT Title, SUM(Contains.Qty) as TotalSold
FROM Book
NATURAL JOIN Contains
GROUP BY Title
ORDER BY TotalSold DESC;
```



	Title	TotalSold
1	Cerulean Sins	5
2	SQL Server 2000 for Experienced	3
3	Execution: The Discipline of Get	3
4	The Pianist	2
5	The Data Warehouse Toolkit: The	2
6	Fundamentals of Database Systems	2
7	ColdFusion MX Web Application Co	2
8	The Secret Life of Bees	1
9	The Hours	1
10	The Guru's Guide to Transact-SQL	1
11	The Girl in the Red Coat: A Memo	1
12	Professional SQL Server 2000 Pro	1
13	OCP: Oracle9i Certification Kit	1
14	MySQL	1
15	Investing in Fixer-Uppers : A Co	1
16	How To Do Everything with Your T	1
17	Google Hacks	1
18	Data Mining: Practical Machine L	1
19	Creating Documents with Business	1
20	Access 2002 Developer's Handbook	1

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.

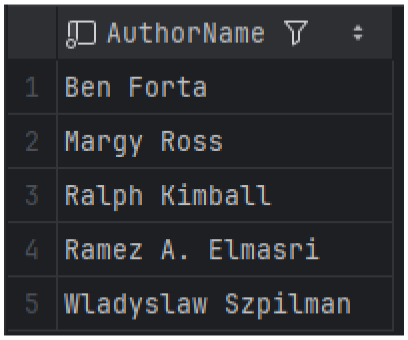
```
SELECT Title, SUM(Transactions.TotalPrice) as TotalSold
FROM Book
NATURAL JOIN Contains
NATURAL JOIN Transactions
GROUP BY Title
ORDER BY TotalSold DESC;
```



	Title	TotalSold
1	Professional SQL Server 2000 Pro	60.39
2	The Guru's Guide to Transact-SQL	58.35
3	Data Mining: Practical Machine L	54.87
4	OCP: Oracle9i Certification Kit	52.41
5	The Data Warehouse Toolkit: The	46.85
6	The Secret Life of Bees	44.81
7	Fundamentals of Database Systems	42.99
8	The Girl in the Red Coat: A Memo	36.01
9	The Pianist	35.62
10	How To Do Everything with Your T	34.44
11	Access 2002 Developer's Handbook	33.48
12	SQL Server 2000 for Experienced	32.2
13	Creating Documents with Business	31.75
14	The Hours	26.73
15	MySQL	21.97
16	Cerulean Sins	21.73
17	ColdFusion MX Web Application Co	20.37
18	Investing in Fixer-Uppers : A Co	11.9
19	Google Hacks	11.05
20	Execution: The Discipline of Get	5.84

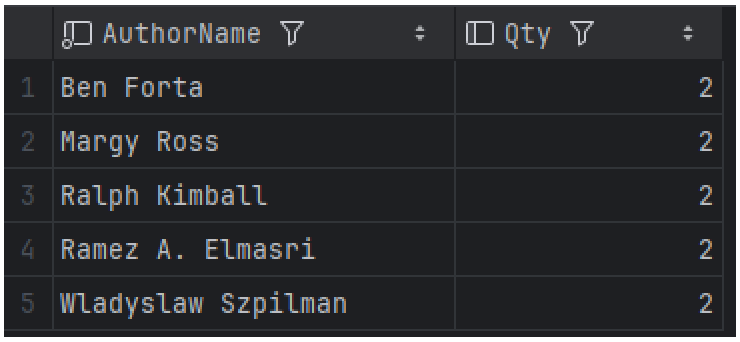
e. Find the most popular author in the database (i.e. the one who has sold the most books)

```
SELECT AuthorName
FROM Author
NATURAL JOIN WrittenBy
NATURAL JOIN Book
NATURAL JOIN Contains
GROUP BY AuthorName
HAVING Contains.Qty = (
        SELECT MAX(Contains.Qty)
        FROM Author
        NATURAL JOIN WrittenBy
        NATURAL JOIN Book
        NATURAL JOIN Contains
        GROUP BY AuthorName
);
```



	AuthorName
1	Ben Forta
2	Margy Ross
3	Ralph Kimball
4	Ramez A. Elmasri
5	Wladyslaw Szpilman

(Note: In our data they have each sold two books:

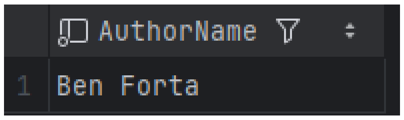


	AuthorName	Qty
1	Ben Forta	2
2	Margy Ross	2
3	Ralph Kimball	2
4	Ramez A. Elmasri	2
5	Wladyslaw Szpilman	2

)

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

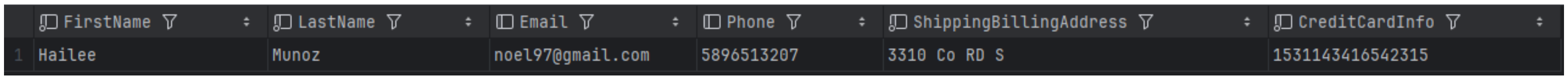
```
SELECT AuthorName
FROM Author
NATURAL JOIN WrittenBy
NATURAL JOIN Book
NATURAL JOIN Contains
NATURAL JOIN Transactions
GROUP BY AuthorName
HAVING TotalPrice = (
        SELECT MAX(TotalPrice)
        FROM Author
        NATURAL JOIN WrittenBy
        NATURAL JOIN Book
        NATURAL JOIN Contains
        NATURAL JOIN Transactions
        GROUP BY AuthorName
);
```



	AuthorName
1	Ben Forta

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

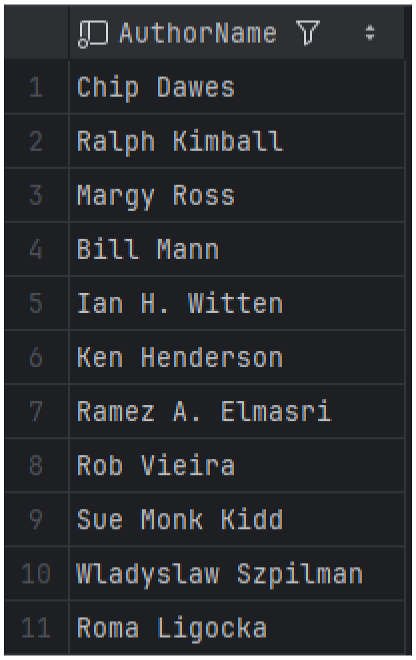
```
SELECT FirstName, LastName, Email, Phone, ShippingBillingAddress, CreditCardInfo
FROM Customer
NATURAL JOIN Transactions
NATURAL JOIN Contains
NATURAL JOIN Book
NATURAL JOIN WrittenBy
NATURAL JOIN Author
WHERE AuthorName = (
        SELECT AuthorName
        FROM Author
        NATURAL JOIN WrittenBy
        NATURAL JOIN Book
        NATURAL JOIN Contains
        NATURAL JOIN Transactions
        GROUP BY AuthorName
        HAVING TotalPrice =  (
                SELECT MAX(TotalPrice)
                FROM Author
                NATURAL JOIN WrittenBy
                NATURAL JOIN Book
                NATURAL JOIN Contains
                NATURAL JOIN Transactions
                GROUP BY AuthorName
        )
);
```



	FirstName	LastName	Email	Phone	ShippingBillingAddress	CreditCardInfo
1	Hailee	Munoz	noel97@gmail.com	5896513207	3310 Co RD S	1531143416542315

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

```
SELECT AuthorName
FROM Author
NATURAL JOIN WrittenBy
NATURAL JOIN Book
NATURAL JOIN Contains
NATURAL JOIN Transactions
NATURAL JOIN Customer
WHERE CID IN (
        SELECT CID
        FROM Customer
        NATURAL JOIN Transactions
        GROUP BY CID
        HAVING SUM(TotalPrice) > (
                SELECT AVG(TotalSpent)
                FROM (
                        SELECT SUM(TotalPrice) as TotalSpent
                        FROM Transactions
                        GROUP BY CID
                )
        )
);
```

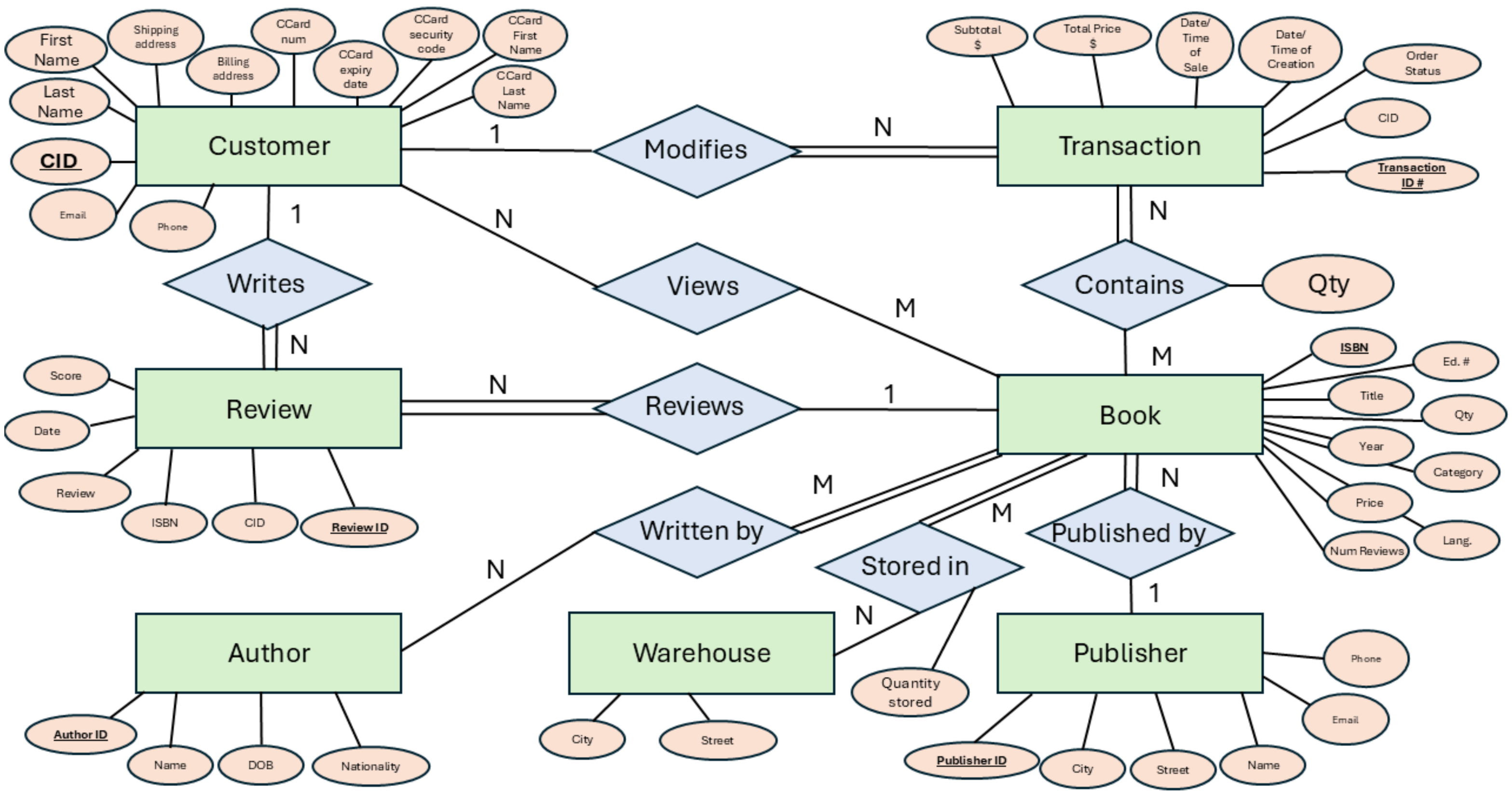


	AuthorName
1	Chip Dawes
2	Ralph Kimball
3	Margy Ross
4	Bill Mann
5	Ian H. Witten
6	Ken Henderson
7	Ramez A. Elmasri
8	Rob Vieira
9	Sue Monk Kidd
10	Wladyslaw Szpilman
11	Roma Ligocka

Project Checkpoint 04

Names: Connor Fricke, Sierra Reis, Ethan Conley
Date: 03/30/2025

1. Provide a current version of your ER Diagram and Relational Model as per Project Checkpoint 03. **If you were instructed to change the model for Project Checkpoint 03, make sure you use the revised versions of your models.**



Primary keys are underlined, *{foreign keys are curly braces}*

CUSTOMER (CID, First_Name, Last_Name, Shipping_Address, Billing_Address, CCard_Num, CCard_Expiry_Date, CCard_Sec_Code, CCard_First_Name, CCard_Last_Name, Email, Phone)
REVIEW (ReviewID, {CID}, {ISBN}, Review, Date, Score)
AUTHOR (AuthorID, Name, DOB, Nationality)
WAREHOUSE (City, Street)
PUBLISHER (PublisherID, City, Street, Name, Email, Phone)

BOOK (ISBN, Edition#, Title, Quantity, Publication_Year, Category, Price, Language, Num_Reviews)
TRANSACTION (TransactionID, {CID}, Order_Status, Date/Time_of_Creation, Date/Time_of_Sale, Subtotal, Total_Price)
WRITTEN_BY ({ISBN}, {AuthorID})
VIEWS ({CID}, {ISBN})
CONTAINS ({TransactionID}, {ISBN}, Qty)
STORED_IN ({Address}, {ISBN}, Quantity_Stored)

2. For each relation schema in your model, indicate the functional dependencies. Think carefully about what you are modeling here - make sure you consider all the possible dependencies in each relation and not just the ones from your primary keys. For example, a customer's credit card number is unique, and so will uniquely identify a customer even if you have another key in the same table (in fact, if the customer can have multiple credit card numbers, the dependencies can get even more involved).

Functional Dependencies:

CUSTOMER: *(CID, First_Name, Last_Name, Shipping/Billing_Address, Credit_Card_Info, Email, Phone)*

{CID} → {First_Name, Last_Name, Shipping/Billing_Address, Credit_Card_Info, Email, Phone}

{Credit_Card_Info} → {CID, First_Name, Last_Name, Shipping/Billing_Address, Email, Phone}

{Email} → {CID, First_Name, Last_Name, Shipping/Billing_Address, Credit_Card_Info, Phone}

(Assumption - Each CID can only have one corresponding credit card attached to it)

(Assumption - Customer accounts do not need a phone, it is optional (and thus not a key))

(This was updated in part three, so this info is for the previous relation diagram)

REVIEW: *(ReviewID, {CID}, {ISBN}, Review, Date, Score)*

{ReviewID} → {CID, ISBN, Review, Date, Score}

{CID, ISBN} → {ReviewID, Review, Date, Score} *(Each CID can only leave 1 review per book)*

AUTHOR: *(AuthorID, Name, DOB, Nationality)*

{AuthorID} → {Name, DOB, Nationality}

WAREHOUSE: *(City, Street)*

{City} → {Street} *(Limiting to one warehouse per city)*

PUBLISHER: *(PublisherID, City, Street, Name, Email, Phone)*

{PublisherID} → {City, Street, Name, Email, Phone}

{City, Street} → {PublisherID, Name, Email, Phone} *(Assuming publisher only has 1 main HQ)*

{Name} → {PublisherID, City, Street, Email, Phone} *(Assuming publisher's Name is unique)*

{Email} → {PublisherID, City, Street, Name, Phone}

{Phone} → {PublisherID, City, Street, Name, Email}

BOOK: *(ISBN, Edition#, Title, Quantity, Publication_Year, Category, Price, Language, Num_Reviews)*

{ISBN} → {Edition#, Title, Quantity, Publication_Year, Category, Price, Language, Num_Reviews}

TRANSACTION: *(TransactionID, {CID}, Order_Status, Date/Time_of_Creation, Date/Time_of_Sale, Subtotal, Total_Price)*

{TransactionID} → {CID, Order_Status, Date/Time of Creation, Date/Time of Sale, Subtotal, Total_Price}

{CID, Date/Time_of_Creation} → {TransactionID, Order_Status, Date/Time_of_Sale, Subtotal, Total_Price}

{CID, Date/Time_of_Sale} → {TransactionID, Order_Status, Date/Time_of_Creation, Subtotal, Total_Price}

WRITTEN_BY: *({ISBN}, {AuthorID})*

{ISBN} → {AuthorID}

VIEWS: *({CID}, {ISBN})*

(No functional dependencies, this is a simple bridge table where neither FK is functionally dependent on the other and there are no non-key attributes.)

CONTAINS: *({TransactionID}, {ISBN}, Qty)*

{TransactionID, ISBN} → {Qty}

STORED_IN: *({Address}, {ISBN}, Quantity_Stored)*

{Address, ISBN} → {Quantity_Stored}

3. For each relation schema in your model, determine the highest normal form of the relation. If the relation is not in 3NF, rewrite your relation schema so that it is in at least 3NF.

CUSTOMER - 0NF *(Credit Card Info and Billing/Shipping Address are multivalued)*
REVIEW - BCNF
AUTHOR - BCNF
WAREHOUSE - BCNF
PUBLISHER - BCNF
BOOK - BCNF
TRANSACTION - BCNF
WRITTEN_BY - BCNF
VIEWS - BCNF
CONTAINS - BCNF
STORED_IN - BCNF

Rewrites:

- Customer Credit_Card_Info non-atomic, violating 1NF. Rewrite to contain CCard_Num, CCard_Sec_Code, CCard_Expiry_Date, CCard_First_Name, and CCard_Last_Name.

- Split Shipping/Billing_Address into Shipping_Address and Billing_Address attributes

UPDATED BCNF CUSTOMER: *(<u>CID</u>, First_Name, Last_Name, Shipping_Address, Billing_Address, CCard_Num, CCard_Expiry_Date, CCard_Sec_Code, CCard_First_Name, CCard_Last_Name, Email, Phone)*

{CID} → {First_Name, Last_Name, Shipping_Address, Billing_Address, CCard_Num, CCard_Expiry_Date, CCard_Sec_Code, CCard_First_Name, CCard_Last_Name, Email, Phone}

{CCard_Num} → {CID, First_Name, Last_Name, Shipping_Address, Billing_Address, CCard_Expiry_Date, CCard_Sec_Code, CCard_First_Name, CCard_Last_Name, Email, Phone}

{Email} → {<u>CID</u>, First_Name, Last_Name, Shipping_Address, Billing_Address, CCard_Num, CCard_Expiry_Date, CCard_Sec_Code, CCard_First_Name, CCard_Last_Name, Phone}

(Assumption - Each CID can only have one corresponding credit card attached to it)

(Assumption - Customer accounts do not need a phone, it is optional (and thus not a key))

4. For each relation schema in your model that is in 3NF but not in BCNF, either rewrite the relation schema to BCNF or provide a short justification for why this relation should be an exception to the rule of putting relations into BCNF.

(BCNF - left side functional dependency must be a superkey)

VIEWS - cannot be in BCNF or is automatically BCNF because there are no functional dependencies

5. For your database, propose at least two interesting views that can be built from your relations. These views must involve joining at least two tables together each and must include some kind of aggregation in the view. Each view must also be able to be described by a one or two sentence description in plain English. Provide the code for constructing your views along with the English language description of what the view is supposed to be providing.

1) **Get a list of Customers with book categories they have viewed frequently, in order to provide recommendations of other books in the same or similar categories.**

CREATE TEMP VIEW Recommendations AS

> **SELECT CID, Customer.First_Name, Customer.Last_Name, SUM(Book.Category) AS num_views**
>
> **FROM Customer NATURAL JOIN Views NATURAL JOIN Book**

```
        GROUP BY Book.Category HAVING num_views > 3;
```

2) Get a list of Books with an average Review Score less than 2.5 and more than 20 reviews to know which books to buy less of/pull from shelves

```
CREATE TEMP VIEW Low_Score_Books AS

        SELECT Book.ISBN, Num_Reviews, AVG(Review.Score) as Avg_Score

        FROM Book NATURAL JOIN Review

        WHERE Num_Reviews > 20

        GROUP BY Book.ISBN

        HAVING Avg_Score < 2.5
```

Project Checkpoint 04 Revisions

It was not clearly enough explained why the Customer entity was in BCNF, so the following assumptions were added (in red) for the Customer table that had not been included in the original submission.

UPDATED BCNF CUSTOMER: *(<u>CID</u>, First_Name, Last_Name, Shipping_Address, Billing_Address, CCard_Num, CCard_Expiry_Date, CCard_Sec_Code, CCard_First_Name, CCard_Last_Name, Email, Phone)*

{CID} → {First_Name, Last_Name, Shipping_Address, Billing_Address, CCard_Num, CCard_Expiry_Date, CCard_Sec_Code, CCard_First_Name, CCard_Last_Name, Email, Phone}

{CCard_Num} → {CID, First_Name, Last_Name, Shipping_Address, Billing_Address, CCard_Expiry_Date, CCard_Sec_Code, CCard_First_Name, CCard_Last_Name, Email, Phone}

{Email} → {<u>CID</u>, First_Name, Last_Name, Shipping_Address, Billing_Address, CCard_Num, CCard_Expiry_Date, CCard_Sec_Code, CCard_First_Name, CCard_Last_Name, Phone}

Assumptions:

- *Each customer account can only have one corresponding credit card attached to it*
- *A credit card can only be used for one customer account*
- *Each customer account can only have one email attached to it*
- *An email address can only be used for one customer account*
- *Customer accounts do not need a phone, it is optional (and thus not a key)*

These assumptions make it so that CCard_Num and Email are both candidate keys and thus all functional dependencies in the Customer table originate from candidate keys.