475 Database Systems Project

Topic: An e-commerce sales database for the book retailer *Twice Sold Tales*

Presented by: Team iHOP

Members: Brendan Amort, Daisy Liu, Maxim Karamurzin

Table of Contents

Database Description

Online sales platforms are a very common application of databases. For example, one of the most prominent eCommerce platforms, Amazon, employs massive databases to store the records needed to function well. Amazon and the countless other internet sales sites practically rely on the use of databases. For this project, an eCommerce-centric database will be created to record and manage the online sales of Twice Sold Tales, a bookstore located in Seattle, Washington. The database will hold data related to product information, customer information, shopping cart information, payment information, shipping details, and order information. Using a database such as this one will support multiple user functions. As a customer, you could view the collection of thousands of books, add them to your shopping cart, and convert that shopping cart into an order complete with payment information and shipping information. Multiple payments will be supported. On the store side of things, the database also serves as an inventory management tool. It can keep track of titles, authors, ISBNs, prices, and other relevant details.

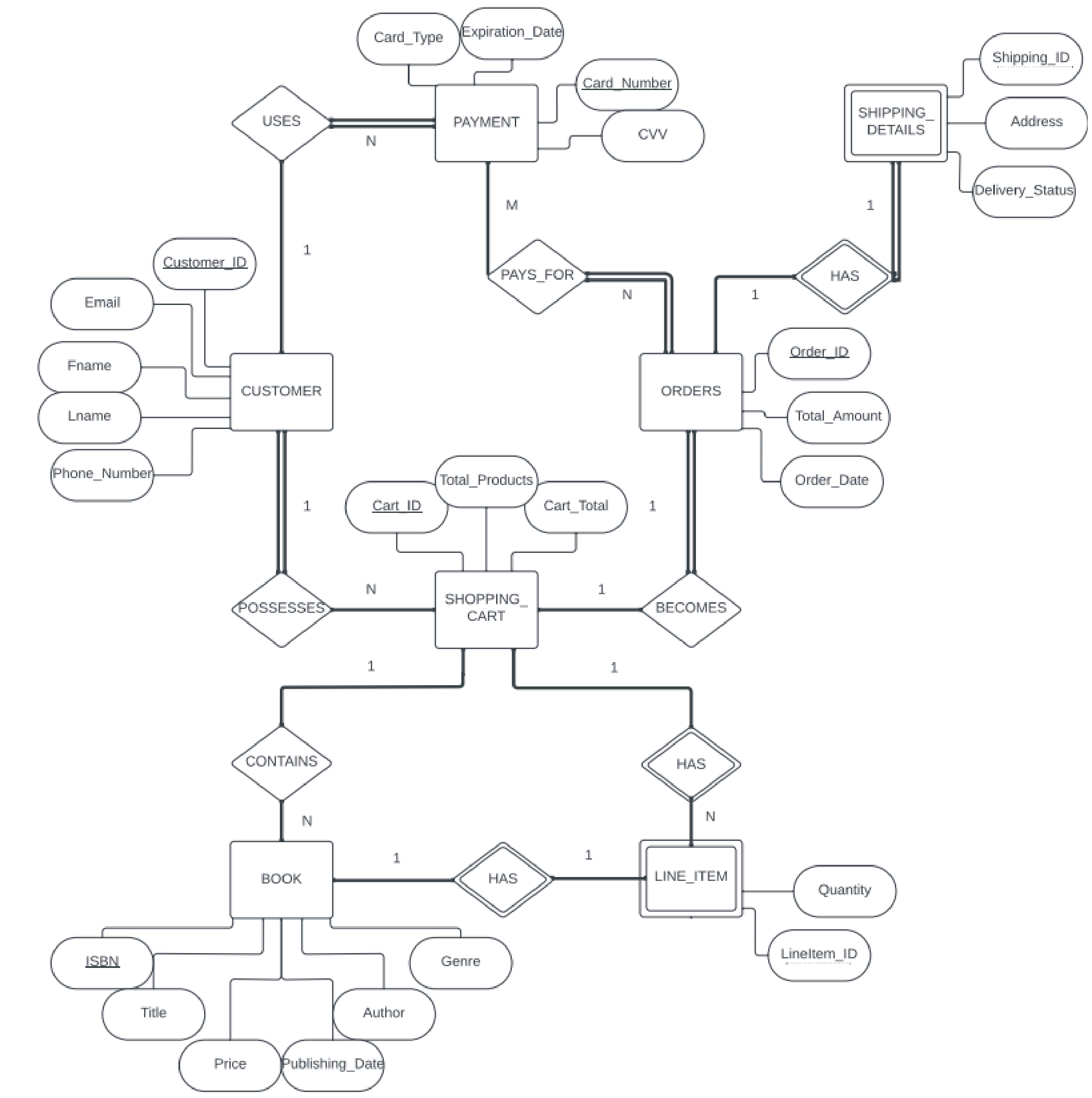
By creating this database, this project allows us to experiment with and utilize the core concepts taught in our CSS 475 class, such as ER diagrams, relational tables, SQL queries, and the various software tools required for the project.

ER Diagram and Relational Model

ER Diagram

Figure 1 is an Entity-Relationship diagram that represents a simple model of an e-commerce database for an online bookstore. It contains a total of 7 entities: BOOK, CUSTOMER, SHOPPING_CART, PAYMENT, ORDER, LINE_ITEM, and SHIPPING_DETAILS.

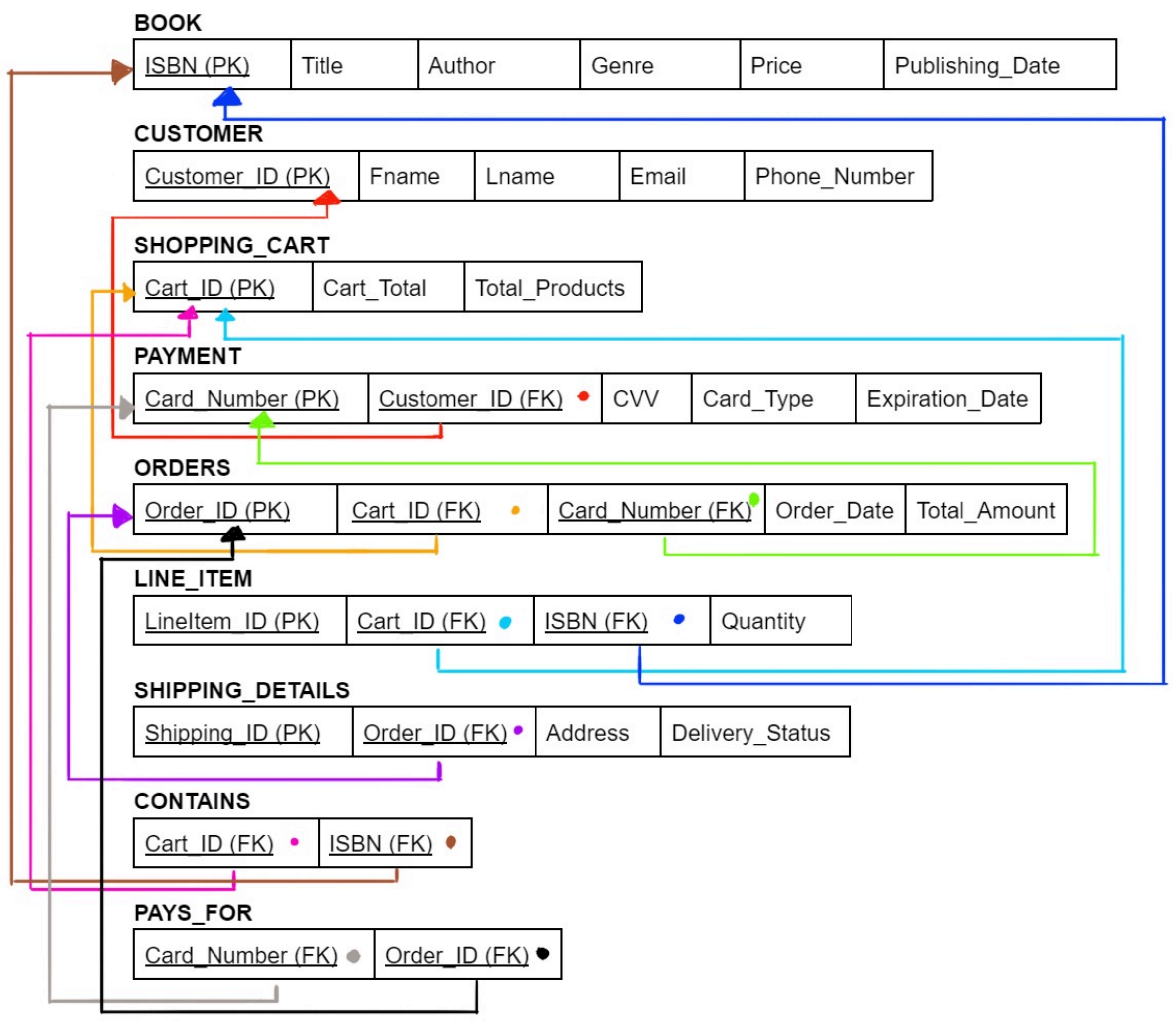
Figure 1. ER Diagram



Relational Schema

Figure 2 depicts the Relational Model for the online bookstore database, complete with color-coded arrows. For more information on design choices, see the 'assumptions' section at the end of this document.

Figure 2. Relational Model



Domain Constraints

BOOK:

- ISBN (PK): String of 13 characters using unique International Standard Book Number
- Title: String of up to 255 characters
- Author: String of up to 255 characters, and must contain a space for separation between first name and last name (i.e. xxxx xxxxx)
- Genre: String of up to 50 characters
- Price: Double that allows for exactly 2 decimal places (e.g. 17.99). Uses USD as its unit.
- Publishing_Date: String in the format of "mm/dd/yyyy." The first two digits must be between 1 and 12, the following two digits must be between 1-31 (depending on the month, and year when the month is February), and the last four digits must be a positive 4 digit integer up until 2023

CUSTOMER:

- Customer_ID (PK): String of 9 unique alphanumeric characters
- Fname: String of up to 50 characters for customer's first name
- Lname: String of up to 50 characters for customer's last name
- Email: String of up to 255 characters, using valid email format (e.g. uniquestring@organizationname.com)
- Phone_Number: String that is 12 in length and consists of numbers and dashes in the following format: xxx-xxx-xxxx

SHOPPING_CART:

- Cart_ID (PK): Unique string of 9 alphanumeric characters
- Cart_Total: The total price of all products in the shopping cart: Double that allows for exactly two decimal places. In USD. Cart total must be 0 if total products is 0. Price should be greater than or equal to 0.
- Total_Products: an int that must be greater than or equal to. Total products must be 0 if cart total is 0.

PAYMENT:

- Card_Number (PK): unique String of exactly 16 digits
- Customer_ID (FK): String of 9 unique alphanumeric characters
- CVV: an int of 3 or 4 digits
- Card_Type: String that is 4-15 in length and one of the following: {Visa, Mastercard, AmericanExpress, Discover}
- Expiration_Date: A String of 5 characters in the format mm/yy. The first two digits must be between 1 and 12, and the last two digits must be 23 or higher

ORDERS:

- Order_ID (PK): Unique string of 9 alphanumeric characters
- Cart_ID (FK): String of 9 unique alphanumeric characters
- Card_Number (FK): unique String of exactly 16 digits

- Order_Date: Date in the format of "YYYY-MM-DD." the first four digits must be a positive 4 digit integer up until 2023, the second two digits must be between 1 and 12, the last two digits must be between 1-31 (depending on the month, and year when the month is February), and Total_Amount: Double that allows for exactly two decimal places. In USD. Total cost of the entire order (items, shipping, tax, and other fees)

LINE_ITEM:
- LineItem_ID (PK(partial key)): Unique string of 9 alphanumeric characters
- ISBN (PK, FK): String of 13 characters using unique International Standard Book Number
- Cart_ID (PK, FK): Unique string of 9 alphanumeric characters
- Quantity: int from 1 - 100
- Price: Double that allows for exactly 2 decimal places (e.g. 17.99). Uses USD as its unit.

SHIPPING_DETAILS:
- Shipping_ID (PK): Unique string of 9 alphanumeric characters
- Order_ID (FK): Unique string of 9 alphanumeric characters
- Address: String of up to 255 characters
- Delivery_Status: Boolean that is true if delivered, and false if not

CONTAINS:
- Cart_ID (FK): Unique string of 9 alphanumeric characters
- ISBN (FK): String of 13 characters using unique International Standard Book Number

PAYS_FOR:
- Card_Number (FK): unique String of exactly 16 digits
- Order_ID (FK): Unique string of 9 alphanumeric characters

ASSUMPTIONS:
- Not every book has a shopping cart (books don't need to be added to a shopping cart to exist)
- Not every shopping cart in the database must have a book (empty cart)
- Shopping carts can exist without a customer; Customers must have a shopping cart (which is automatically created empty by default)
- A customer can have multiple shopping carts (when a transaction is completed, a new, empty shopping cart is generated)
- A book can be added to many shopping carts, and a shopping cart can contain many books
- A customer can have many different entity instances of payment information (credit card, debit card, etc), but a single instance of payment information can only have one customer
- A customer is not directly related to a book. It is only related through a shopping cart (purchases can only be made through a shopping cart)

- Books are the only type of product available
- A shopping cart that is generated without customer information (if they are browsing as a guest) will set the related information to a default. For example, email would be defaultemail@gmail.com, and first name would be default.
- All primary keys imply the constraint of not having a null value
- Orders must have a cart and payment information
- Payment information can exist without an order (when payment details are saved to an account)
- Shipping details can exist without an order (when shipping details are saved to an account)
- Orders must have shipping details (if picking up in store, shipping details will be store location)
- Since payment information is linked to customer information, orders don't need customer id. Customers are related to orders through shopping cart and payment information only
- Cart total does not include shipping fee, but total amount under orders does

---

Sample Queries

Display all books in the price range of $15-$20.
Show the inventory stock information for a specific book.
List customer details for order number #iHOP
List product information for an item or all items in a customer’s shopping cart
Update a customer’s payment information
Delete product from a customer’s shopping cart
Add payment information for a customer

Discussion

When designing an ERD (Entity-Relationship Diagram) and relational schema for an eCommerce-centric database focusing on book sales, like for Twice Sold Tales, several design decisions, concerns, assumptions, and limitations should be documented. This documentation is crucial for future development, maintenance, and potential scalability of the database. Here’s a comprehensive overview:

Design Decisions:

1. We had the idea of separating shopping carts and orders, due to the need to separate a customer that would add books to the cart, but not place an order, and customers that would both add books to the cart and make a purchase.

2. Because the book database focuses on eCommerce, we kept the user interface in mind and chose to use the "LINE_ITEM" entity purely because this is a common strategy in actual eCommerce platforms.

Concerns and Considerations:

1. Security and Privacy: Customer data protection is important since this database deals with payment information. The database must be designed with security measures to protect sensitive information.
2. Data Integrity: Ensuring accuracy and consistency of data through constraints, foreign keys, and transaction controls is a major concern.

Limitations:

1. This design focuses on the online sales aspect of Twice Sold Tales. Modifications would be needed for it to be used for a local physical store system.
2. This database doesn't contain information about employees or the physical stores. However, this is appropriate considering that the focus is on the eCommerce aspect.

Tooling Assessment

Three primary factors were used to evaluate the potential tools used for the database: team familiarity/prior experience, cost effectiveness, and ability to integrate with other tools.

	TOOL	**RATIONALE**
IDE	Visual Studio	● Comprehensive extension support ● Team familiarity ● Easy integration with other tools
DBMS	MySQL	● Easy integration with Visual Studio ● Prominent in the industry ● Cost effective - free
HOSTING	Microsoft Azure	● Cost effective - free option, and students get $100 credit ● Compatible with MySQL ● Team familiarity
UI	HTML, CSS, Javascript **	● Traditional, fundamental tools for web applications such as eCommerce ● Team familiarity

**Update 4.3: UI was created using Power Apps due to easy integration with Microsoft Azure

Other Tools Used For Project

- LucidChart: Used for creating the ER Diagram

    - Supports multiple editors simultaneously
- Procreate: Used to create relational tables
- Data tooling is listed below.

Database Access:

MySQL Workbench:

1. Download MySQL and MySQL Workbench
    a. https://dev.mysql.com/downloads/mysql/
    b. https://dev.mysql.com/downloads/workbench/
2. Database
3. Connect to Database
4. Hostname: tstserverfinal.mysql.database.azure.com
5. Username: ihop
6. Password: 9rB!2pQ#4wXz

Data Generation:

A variety of sources were used to generate the data used to populate the database.

- BOOKS
    - Data for books was obtained using ChatGPT.
- Mockaroo was used to generate mock data for:
    - CUSTOMER
    - SHOPPING_CART
    - PAYMENT
    - ORDERS
    - LINE_ITEM
    - SHIPPING_DETAILS
- There were some issues in generating a valid date for expiration date in payment, date in orders, and addresses in shipping details. For now, those are set to the same valid dates and addresses (order date 2023-11-27, expiration date 11/24, address school address 18115 Campus Way NE, Bothell, WA 98011) to focus on testing other inserts. However, we have tested the domain constraints necessary. Proper inserts will be completed in the next iteration.
- Contains and Pays_for are only made of foreign keys and were considered to be of lower priority. They will be added by the next iteration to complete the database.
- Updated in Version 4.3: All data in the database was obtained using ChatGPT

Tables Used:

BOOKS
CUSTOMER
SHOPPING_CART

PAYMENT
ORDERS
LINE_ITEM
SHIPPING_DETAILS
CONTAINS
PAYS_FOR

Inserts:

The following sections list out a few of the inserts we used to populate our database, as well as inserts we made to test certain constraints.

Example Inserts:

- **BOOK**:
  - insert into CUSTOMER (Customer_ID, Fname, Lname, Email, Phone_Number) values ('4T27q9vWQ', 'Dominik', 'Ragate', 'dragate0@prweb.com', '343-260-4655');
- **SHOPPING_CART**:
  - insert into SHOPPING_CART (Cart_ID, Cart_Total, Total_Products) values ('o7mLyC0qC', 4845.15, 194);
- **PAYMENT**:
  - insert into PAYMENT (Card_Number, CVV, Card_Type, Expiration_Date, Customer_ID) values ('0534452817427615', 9121, 'Mastercard', '05/28', '8CHkYcs67');
- **ORDERS**:
  - insert into ORDERS (Order_ID, Order_Date, Total_Amount, Cart_ID, Card_Number) values ('oFKP68521', '2023-11-26', 4954.79, 'o7mLyC0qC', '7984415712508945');
- **LINE_ITEM**:
  - INSERT INTO LINE_ITEM (LineItem_ID, ISBN, Cart_ID, Quantity, Unit_Price) VALUES ('L1A2B3C4D', '9783161484100', 'o7mLyC0qC', 2, 19.99);
- **SHIPPING_DETAILS**:
  - insert into SHIPPING_DETAILS (Shipping_ID, Order_ID, Address, Delivery_Status) values ('8291hgcUs', 'xhYr14337', '18115 Campus Way NE, Bothell, WA 98011', true);

Bad Inserts:

CUSTOMER CONSTRAINT TESTING

1. Testing the Phone_Number CHECK Constraint

INSERT INTO CUSTOMER (Customer_ID, Fname, Lname, Email, Phone_Number)
VALUES ('X1Y2Z3A4B', 'Alice', 'Smith', 'alice.smith@example.com', '123-4567');

- Constraint Tested: This insert statement is testing the Phone_Number CHECK constraint which expects the phone number to follow the format '--____' (3 digits, a hyphen, 3 digits, another hyphen, and 4 digits).
- Expected Outcome: The query should fail because the provided phone number '123-4567' does not match the required format.
- Outcome: Successfully failed to insert.

2. Testing the Email CHECK Constraint

   INSERT INTO CUSTOMER (Customer_ID, Fname, Lname, Email, Phone_Number)
   VALUES ('B4C5D6E7F', 'Bob', 'Jones', 'bob.jonesatexample.com', '234-567-8901');

- Constraint Tested: This insert statement is testing the Email CHECK constraint which expects the email to contain an '@' symbol.
- Expected Outcome: The query should fail because the provided email 'bob.jonesatexample.com' does not include an '@' symbol.
- Outcome: Successfully failed to insert.

3. Testing a NOT NULL Constraint (e.g., Fname)

   INSERT INTO CUSTOMER (Customer_ID, Lname, Email, Phone_Number)
   VALUES ('C7D8E9F0G', 'Johnson', 'johnson@example.com', '345-678-9012');

- Constraint Tested: This insert statement is testing the NOT NULL constraint for the Fname field, which expects a first name to be provided.
- Expected Outcome: The query should fail because the Fname (first name) field is left out, violating the NOT NULL constraint.
- Outcome: Successfully failed to insert.

SHOPPING_CART CONSTRAINT TESTING

1. Testing the Cart_Total CHECK Constraint

   INSERT INTO SHOPPING_CART (Cart_ID, Cart_Total, Total_Products)
   VALUES ('A1B2C3D4E', -10000.00, 50);

   - Constraint Tested: This is testing the Cart_Total CHECK constraint which requires the total to be greater than 0

- Expected Outcome: The query should fail because the cart total of -1000.00 is less than 0
- Outcome: Successfully failed to insert.

2. Testing the Total_Products CHECK Constraint

```
INSERT INTO SHOPPING_CART (Cart_ID, Cart_Total, Total_Products)
VALUES ('F5G6H7I8J', 500.00, -250);
```

- Constraint Tested: This query tests the Total_Products CHECK constraint that requires the number of products to be greater than 0.
- Expected Outcome: The query should fail because the total number of products (-250) is less than 0.
- Outcome: Successfully failed to insert.

3. Testing the Combined Total_Products and Cart_Total CHECK Constraint

```
INSERT INTO SHOPPING_CART (Cart_ID, Cart_Total, Total_Products)
VALUES ('K9L0M1N2O', 0.00, 10);
```

- Constraint Tested: This tests the combined CHECK constraint which requires that if Total_Products is greater than 0, then Cart_Total should also be greater than 0 (and vice versa).
- Expected Outcome: The query should fail because it violates the condition that if there are products in the cart (Total_Products > 0), the cart total (Cart_Total) cannot be 0.00.
- Outcome: Successfully failed to insert.

PAYMENT CONSTRAINT TESTING

1. Testing the CVV CHECK Constraint

```
INSERT INTO PAYMENT (Card_Number, CVV, Card_Type, Expiration_Date, Customer_ID)
VALUES ('1234567890123456', 99, 'Visa', '07/25', '8CHkYcs67');
```

- Constraint Tested: This is testing the CVV CHECK constraint which requires the CVV to be between 100 and 9999.
- Expected Outcome: The query should fail because the CVV value (99) is outside the specified range.
- Outcome: Successfully failed to insert.

2. Testing the Card_Type CHECK Constraint

INSERT INTO PAYMENT (Card_Number, CVV, Card_Type, Expiration_Date, Customer_ID)
VALUES ('6543210987654321', 123, 'JCB', '10/27', '8CHkYcs67');

- Constraint Tested: This tests the Card_Type CHECK constraint which requires the card type to be either 'Visa', 'Mastercard', 'AmericanExpress', or 'Discover'.
- Expected Outcome: The query should fail because 'JCB' is not one of the allowed card types.
- Outcome: Successfully failed to insert.

3. Testing the Expiration_Date CHECK Constraint

INSERT INTO PAYMENT (Card_Number, CVV, Card_Type, Expiration_Date, Customer_ID)
VALUES ('9876543210123456', 456, 'Visa', '13/26', '8CHkYcs67');

- Constraint Tested: This tests the Expiration_Date CHECK constraint, particularly the month part, which requires the first two characters (month) to be between '01' and '12'.
- Expected Outcome: The query should fail because '13' is not a valid month.
- Outcome: Successfully failed to insert.

ORDERS CONSTRAINT TESTING

1. Testing the Total_Amount Domain Constraint (Negative Value)

INSERT INTO ORDERS (Order_ID, Order_Date, Total_Amount, Cart_ID, Card_Number)
VALUES ('A1B2C3D4E', '2023-11-28', -50.00, 'o7mLyC0qC', '7984415712508945');

- Constraint Tested: This is testing the domain constraint of Total_Amount which should not allow negative values.
- Expected Outcome: The query should fail because the total amount (-50.00) is a negative value, which is not valid.
- Outcome: Successfully failed to insert.

2. Testing Foreign Key Constraint for Cart_ID

INSERT INTO ORDERS (Order_ID, Order_Date, Total_Amount, Cart_ID, Card_Number)
VALUES ('K9L01N2O', '2023-11-28', 200.00, 'fdjalskht', '7984415712508945');

- Constraint Tested: This tests the foreign key constraint for Cart_ID, which should reference a valid Cart_ID in the SHOPPING_CART table.

- Expected Outcome: The query should fail because 'NonExistentCart' does not exist in the SHOPPING_CART table.
- Outcome: Successfully failed to insert.

SHIPPING_DETAILS CONSTRAINT TESTING

1. Testing the Address NOT NULL Constraint

INSERT INTO SHIPPING_DETAILS (Shipping_ID, Order_ID, Delivery_Status)
VALUES ('A1B2C3D4E', 'xhYr14337', false);

- Constraint Tested: This tests the NOT NULL constraint for the Address field.
- Expected Outcome: The query should fail because the Address field is mandatory and has been omitted.
- Outcome: Successfully failed to insert.

2. Testing the Foreign Key Constraint for Order_ID

INSERT INTO SHIPPING_DETAILS (Shipping_ID, Order_ID, Address, Delivery_Status)
VALUES ('F5G6H7I8J', 'qwertyuio', '123 Main St, Anytown, USA', true);

- Constraint Tested: This tests the foreign key constraint for Order_ID, which should reference a valid Order_ID in the ORDERS table.
- Expected Outcome: The query should fail because 'qwertyuio' does not exist in the ORDERS table.
- Outcome: Successfully failed to insert.

3. Testing the Delivery_Status Domain Constraint (Invalid Data Type)

INSERT INTO SHIPPING_DETAILS (Shipping_ID, Order_ID, Address, Delivery_Status)
VALUES ('K9L0M1N2O', 'xhYr14337', '456 Another St, Anytown, USA', 'NotDelivered');

- Constraint Tested: This tests the data type constraint for Delivery_Status, which is expected to be a BOOLEAN (true or false).
- Expected Outcome: The query should fail because 'NotDelivered' is not a valid BOOLEAN value.
- Outcome: Successfully failed to insert.

Normalization:

Normal Form Assessment

BOOK Table

- 1NF: Achieved. Each attribute has atomic values, and there are no multivariate attributes.
- 2NF: Achieved. It's in 1NF, and all non-key attributes (Title, Author, Genre, Price, Publishing_Date) are fully functionally dependent on the primary key (ISBN).
- 3NF: Achieved. It's in 2NF, and there are no transitive dependencies.
- BCNF: Achieved. It's in 3NF, and every determinant is a candidate key.

CUSTOMER Table

- 1NF: Achieved. Each attribute has atomic values, and there are no multivariate attributes.
- 2NF: Achieved. All attributes are fully dependent on Customer_ID.
- 3NF: Achieved. There are no transitive dependencies.
- BCNF: Achieved. Every determinant is a candidate key.

SHOPPING_CART Table

- 1NF: Achieved. Each attribute has atomic values, and there are no multivariate attributes.
- 2NF: Achieved. All attributes are fully dependent on Cart_ID.
- 3NF: Achieved. There are no transitive dependencies.
- BCNF: Achieved. Every determinant is a candidate key.

PAYMENT Table

- 1NF: Achieved. Each attribute has atomic values, and there are no multivariate attributes.
- 2NF: Achieved. All attributes are fully dependent on Card_Number.
- 3NF: Achieved. There are no transitive dependencies.
- BCNF: Achieved. Every determinant is a candidate key.

ORDERS Table

- 1NF: Achieved. Each attribute has atomic values, and there are no multivariate attributes.
- 2NF: Achieved. All attributes are fully dependent on Order_ID.
- 3NF: Achieved. There are no transitive dependencies.
- BCNF: Achieved. Every determinant is a candidate key.

LINE_ITEM Table

- 1NF: Achieved. Each attribute has atomic values, and there are no multivariate attributes.
- 2NF: Achieved. Attributes depend on the composite primary key.
- 3NF: Achieved. There are no transitive dependencies.
- BCNF: Achieved. Every determinant is a candidate key.

SHIPPING_DETAILS Table

- 1NF: Achieved. Each attribute has atomic values, and there are no multivariate attributes.

- 2NF: Achieved. All attributes are fully dependent on Shipping_ID.
- 3NF: Achieved. There are no transitive dependencies.
- BCNF: Achieved. Every determinant is a candidate key.

CONTAINS Table (Junction Table)
1NF, 2NF, 3NF, BCNF: Achieved. This is a junction table to handle many-to-many relationships with only foreign keys as its attributes.

PAYS_FOR Table (Junction Table)
1NF, 2NF, 3NF, BCNF: Achieved. Similar to CONTAINS, this is a junction table for many-to-many relationships.

Normalization Improvements

1. Decompose AUTHOR from BOOK Table: If multiple books can have the same author, creating a separate AUTHOR table can reduce redundancy. This table would have Author_ID as a primary key and Author Name as another attribute. The BOOK table would then reference Author_ID.

2. Separate Address Details in CUSTOMER Table: For more complex applications, address-related attributes (street, city, postal code, etc.) could be moved to a separate ADDRESS table to handle multiple addresses for a single customer or shared addresses

---

SQL Query Statements:

Note: Complex queries are noted in the purpose section.

SQL Query	Purpose

SELECT Genre, COUNT(*) AS NumberOfBooks
FROM BOOK GROUP BY Genre
ORDER BY NumberOfBooks DESC LIMIT 1;
	Find the Most Popular Genre Based on Number of Titles

– Disable safe update mode
UPDATE BOOK SET Price = Price * 1.1
WHERE Genre = 'Fantasy';
	Update Prices of a Specific Genre

SELECT Author
FROM BOOK GROUP BY Author
HAVING COUNT(DISTINCT Genre) > 1;
	Find Authors with Multiple Books in Different Genres

SELECT Customer_ID, Fname, Lname
FROM CUSTOMER
ORDER BY Customer_ID DESC LIMIT 5;
	Customers with Most Recent Registrations

SELECT Customer_ID, Fname, Lname
FROM CUSTOMER
WHERE Phone_Number IS NULL;
	Find Customers with No Phone Number Provided

SELECT *
FROM CUSTOMER
ORDER BY Lname, Fname;
	List Customers Alphabetically by Last Name

SELECT *
FROM CUSTOMER
WHERE Fname LIKE '%Dan%' OR Lname LIKE '%Doe%';
	Search for Customer by Partial Name

SELECT *
FROM SHOPPING_CART
ORDER BY Cart_Total DESC LIMIT 5;
	Carts with Highest Total Value

SELECT * FROM SHOPPING_CART WHERE Total_Products > 10;	Carts with More than 10 Products

SQL Query	Purpose

– Disable safe update mode
UPDATE SHOPPING_CART SC
SET Cart_Total = (
   SELECT SUM(LI.Unit_Price * LI.Quantity)
   FROM LINE_ITEM LI
   WHERE LI.Cart_ID = SC.Cart_ID
);
	Update Cart Total After Price Change in Books

SELECT *
FROM PAYMENT
WHERE Expiration_Date LIKE '%23';
	Payments Expiring This Year

– Disable safe update mode
DELETE FROM PAYMENT
WHERE Expiration_Date < CURRENT_DATE;
	Delete Expired Payment Methods

SELECT * FROM ORDERS
WHERE Total_Amount > 100;
	Orders with Total Amount Above $100

SELECT *
FROM ORDERS
ORDER BY Order_Date DESC LIMIT 5;
	Most Recent 5 Orders

SELECT *
FROM ORDERS
WHERE Order_Date > CURRENT_DATE - INTERVAL 1 MONTH;
	Orders Placed in Last Month

SELECT MONTH(Order_Date) AS Month, COUNT(*)
FROM ORDERS
GROUP BY Month
ORDER BY Month;
	Complex query
	
	Count Orders by Month

SELECT B.ISBN, B.Title, COUNT(*) AS OrderCount
FROM LINE_ITEM L
JOIN BOOK B ON L.ISBN = B.ISBN
GROUP BY B.ISBN, B.Title
ORDER BY OrderCount DESC
LIMIT 5;
	Complex query
	
	List Most Frequently Ordered Books

SQL Query	Purpose

SELECT ISBN, SUM(Unit_Price * Quantity) AS TotalSales
FROM LINE_ITEM
GROUP BY ISBN;
	Complex query
	
	Find Total Sales of Each Book

SELECT L1.ISBN AS Book1, L2.ISBN AS Book2, COUNT(*) AS TimesBoughtTogether
FROM LINE_ITEM L1
JOIN LINE_ITEM L2 ON L1.Cart_ID = L2.Cart_ID AND L1.ISBN < L2.ISBN
GROUP BY L1.ISBN, L2.ISBN
ORDER BY TimesBoughtTogether
DESC LIMIT 5;
	Complex query
	
	Books Often Bought Together

SELECT * FROM LINE_ITEM
WHERE Cart_ID = 'CART12345';
	List Line Items for a Specific Cart

SELECT * FROM SHIPPING_DETAILS	Track Orders Currently Out for Delivery

WHERE Delivery_Status = false;	

– Disable safe update mode
UPDATE SHIPPING_DETAILS
SET Delivery_Status = true
WHERE Shipping_ID = 'SHIP12345';
	Update Delivery Status for an Order

SELECT c.Customer_ID, c.Fname, c.Lname, SUM(o.Total_Amount) AS Total_Spent, (SELECT SUM(li.Quantity) FROM LINE_ITEM li JOIN ORDERS o2 ON li.Cart_ID = o2.Cart_ID WHERE o2.Customer_ID = c.Customer_ID) AS Total_Books_Bought FROM CUSTOMER c JOIN ORDERS o ON c.Customer_ID = o.Customer_ID JOIN SHOPPING_CART sc ON o.Cart_ID = sc.Cart_ID JOIN LINE_ITEM li ON sc.Cart_ID = li.Cart_ID WHERE o.Order_Date > ‘2023-12-01’ AND o.Order_Date <= ‘2023-12-31’ GROUP BY c.Customer_ID, c.Fname, c.Lname;	Complex query Show customer information along with their total spent amount and total books for December 2023

SELECT Customer_ID, AVG(Total_Amount) as Avg_Amount_Spent
FROM ORDERS
WHERE Customer_ID IN (
SELECT Customer_ID
FROM ORDERS
GROUP BY Customer_ID
	Complex query
	Find average total amount spent in orders for each customer who has spent above the overall average amount in the orders table

```
    HAVING AVG(Total_Amount) > (SELECT
AVG(Total_Amount) FROM ORDERS)
)
GROUP BY Customer_ID;
```

```
SELECT
    C.Fname,
    C.Lname,
    COUNT(LI.ISBN) as Total_Books_Ordered
FROM
    CUSTOMER C
JOIN
    ORDERS O ON C.Customer_ID =
O.Customer_ID
JOIN
    LINE_ITEM LI ON O.Order_ID =
LI.Order_ID
WHERE
    C.Customer_ID IN (
        SELECT
            Customer_ID
        FROM
            ORDERS
        GROUP BY
            Customer_ID
        HAVING
            COUNT(Order_ID) > 1
    )
GROUP BY
    C.Customer_ID, C.Fname, C.Lname;
```
	Complex query
	
	Retrieves the first and last names of customers along with the total count of books they've ordered, limited to those who have placed more than one order

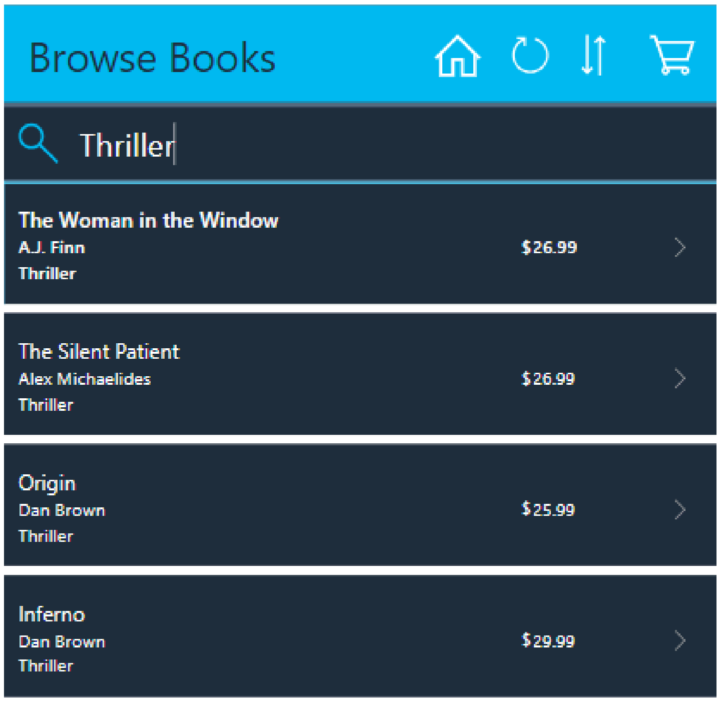
UI:

Link to the UI:
https://make.powerapps.com/e/Default-f6b6dd5b-f02f-441a-99a0-162ac5060bd2/canvas/?action=edit&app-id=%2Fproviders%2FMicrosoft.PowerApps%2Fapps%2Fb660784e-b115-4ca9-982f-033742b9404f

Note: Access has already been shared with the UW emails of the graders and the professor.

The following section is dedicated to screenshots of our UI and the corresponding query.

CRUD (**read**): List the title, author, genre, and price of the books that have the genre "Thriller."


Browse Books

Thriller

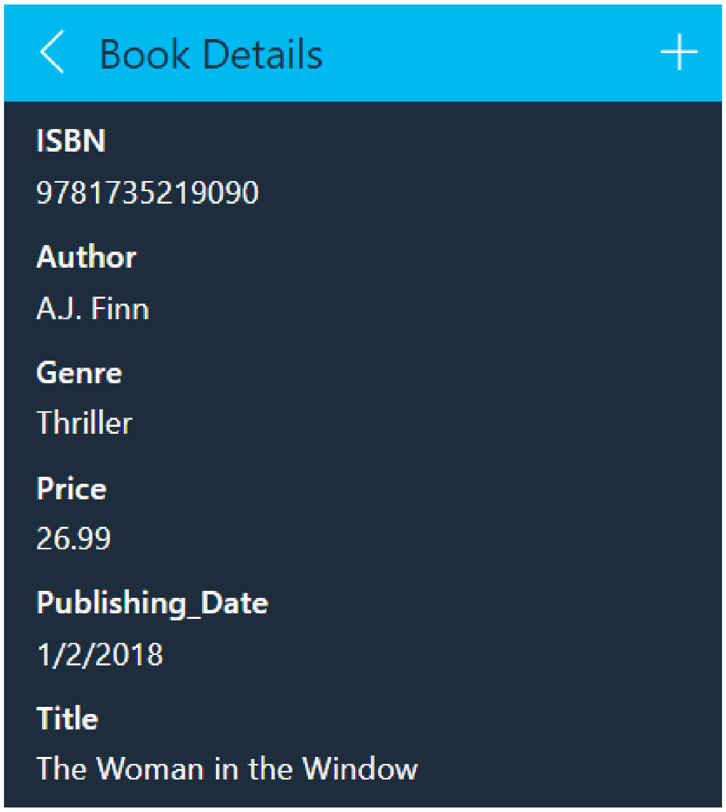
The Woman in the Window
A.J. Finn
Thriller
$26.99

The Silent Patient
Alex Michaelides
Thriller
$26.99

Origin
Dan Brown
Thriller
$25.99

Inferno
Dan Brown
Thriller
$29.99

CRUD (**read**): List the ISBN, author, genre, price, publishing date, and title of the book where the title is The Woman in the Window


Book Details

ISBN
9781735219090

Author
A.J. Finn

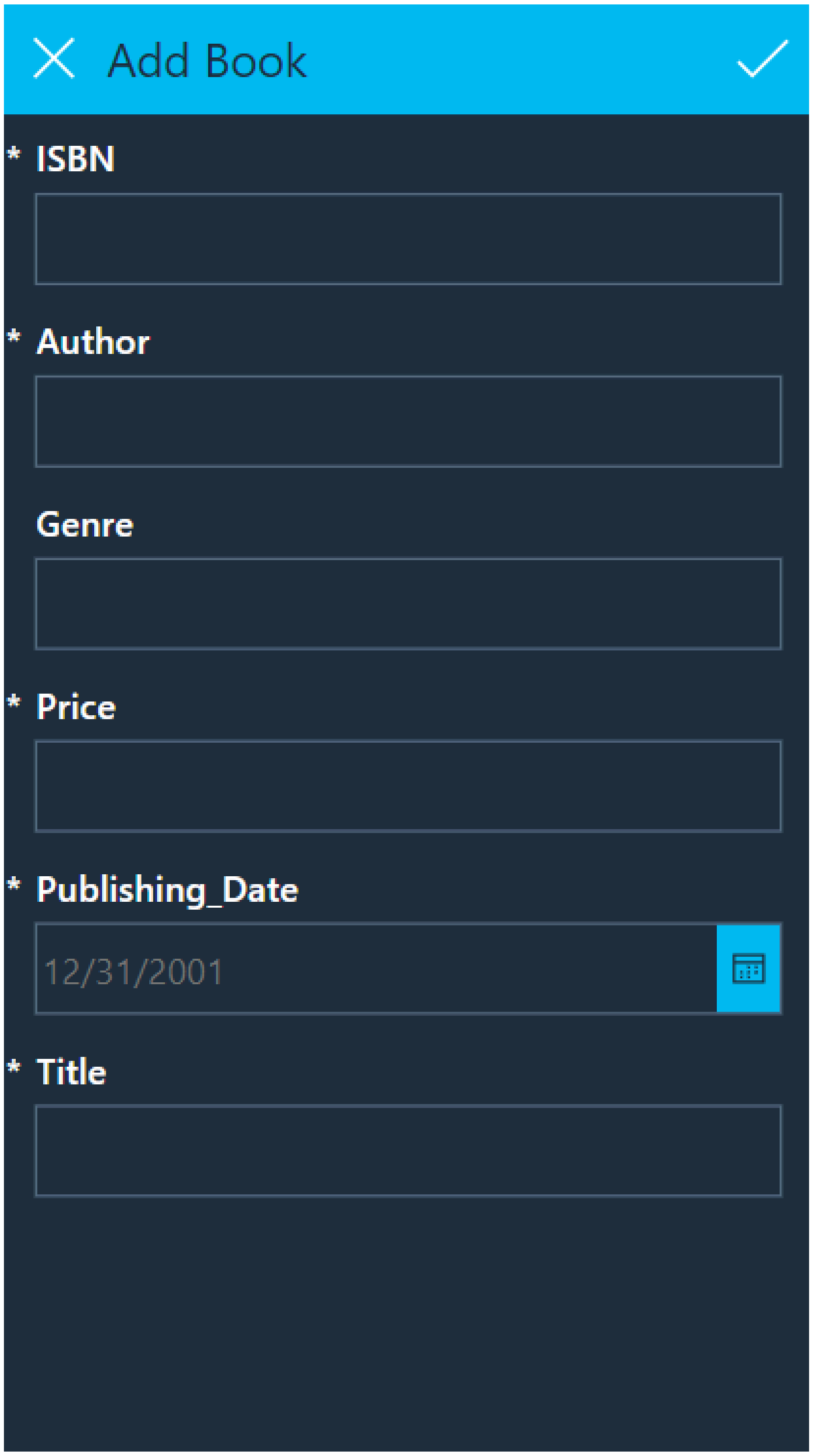
Genre
Thriller

Price
26.99

Publishing_Date
1/2/2018

Title
The Woman in the Window

CRUD: This interface allows the user to **create** a tuple for the BOOK entity and adds it to the table



Add Book

* ISBN

* Author

Genre

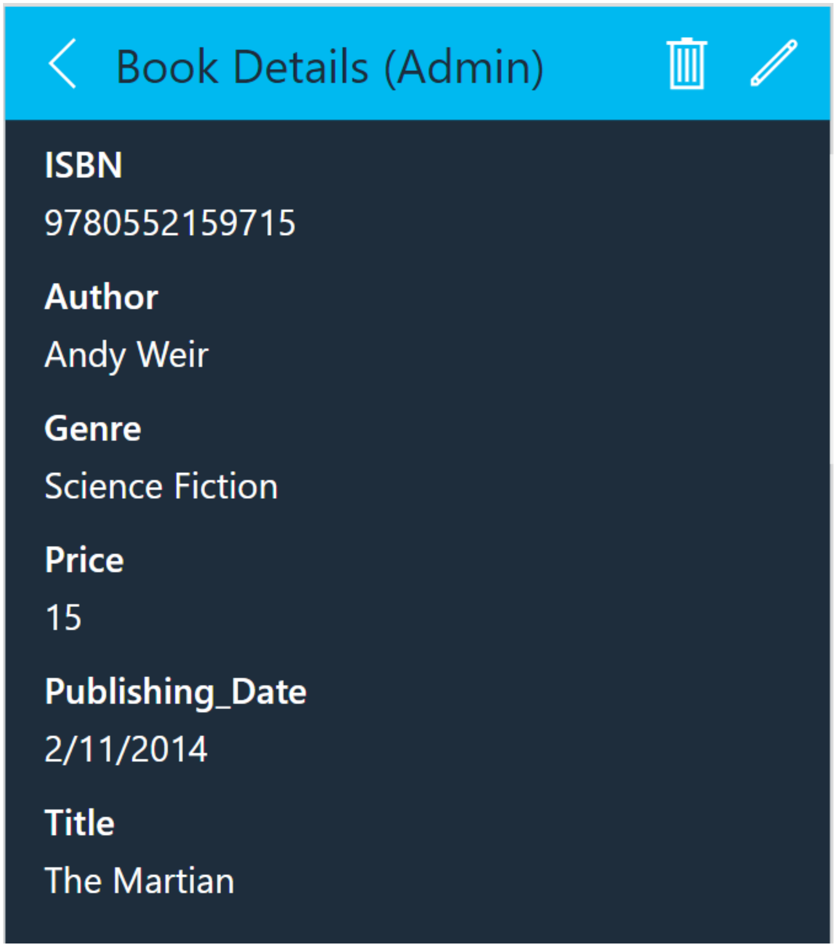
* Price

* Publishing_Date

12/31/2001

* Title

CRUD: This interface has selected a specific tuple from BOOK. The garbage can icon **deletes** the currently selected record. The pencil icon allows the user to **update** any of the fields shown in the selected tuple.


Book Details (Admin)

ISBN
9780552159715

Author
Andy Weir

Genre
Science Fiction

Price
15

Publishing_Date
2/11/2014

Title
The Martian

Evaluation:

A lot of the effort put into this project was focused on making sure that our tables were accurate and efficient. However, when creating data to insert into our database, we ran into a few issues, such as pricing not making any sense (for example, 2 books totaled up to $4000), attributes that should not have existed since they were redundant, and attributes that should've been foreign keys were not declared as foreign keys. Therefore, we ended up needing to recreate our tables multiple times. In the future, for similar projects, more care would need to be put into planning so that this situation does not repeat itself.

There was a lot of trouble with creating a user interface; no one really understood how to do that, and learning in the little amount of time we had among other obligations due to outside circumstances was difficult. With more time, we would have spent it on increasing the accuracy of how our database would be reflected in the real world. While we were filling out sample inserts, it wasn't necessary for us to consider automatically calculating the cart total, so we overlooked the idea of having the database automatically calculate the cart total based on the items in the cart. Furthermore, as we are making a database for a second-hand bookstore, we should have indicated another attribute for the book quality.

Additionally, during our project, the host of the Azure database's subscription expired. Therefore, we had to remake our entire database using another member's free trial. However, this was not too difficult, as we had all the files we needed to recreate the database in a Google Drive folder.

Project schedule:

Deliverables	**Description**	**Date**
1. Proposal	Define the project purpose and goals. Discuss initial ideas.	10/19/2023
2. Tooling	Choose software tools and methodologies for project management.	10/22
3. Design Documentation	Detailed planning and documentation of the system's design.	10/22
a. ERD	Design Entity-Relationship Diagrams for the database.	
b. Assumptions and specs	Define project specifications and note any assumptions.	
4. Create Database	Initial setup and creation of the database system.	11/18
5. Populate Database	Use sample data to test basic functionalities.	11/20
a. Input Sample Data	Use sample data to test basic functionalities.	
b. Testing	Run tests to ensure data integrity and correct functionality.	
6. SQL Query Statements	Design and create necessary SQL statements for operations.	11/30
a. Write Queries	List and develop SQL commands required for the system.	
b. Testing Phase	Verify SQL statements operate	

	correctly.	
7. Normalization	Review and optimize the database structure.	12/6
8. User Interface Strategy	Plan and design the user interface for the system.	12/10

Contribution:

Everyone will contribute to each portion of the project. The deliverables will be reviewed by the group when each deliverable is completed, and changes and corrections will be made as needed.

Version Control:

Version 2.1: Modified the ER diagram to look more professional. Added more tables to the database, and included those tables in the ER diagram and the relational model. Updated domain constraints and assumptions to include new entities and attributes.

Version 2.2: Organized sections to be in a logical order. Added details to the database description, including purpose, use, and entities. Schedule was broken down to include more details on when each deliverable should be completed by. Tooling assessment section was added.

Version 2.3: Discussion section was added. A table of contents was generated.

Version 3.0: Added missing foreign keys to the relational model. Updated relational model diagram and domain constraints.

Version 3.1: Included data generation information, and listed the names of the tables used. Added database access information

Version 3.2: Changed the type of order date to date, instead of string in domain restrictions. ORDER has been changed to ORDERS.

Version 3.3: Clarified constraint on total items and total price to be greater than 0. Added bad inserts, as well as example inserts.

Version 4.0: Normalization section was added.

Version 4.1: SQL query statements were added.

Version 4.2: Evaluation section was added. Line_Item was modified to remove redundancy regarding price.

Version 4.3: Data generation was modified to indicate the change in tooling. UI screenshots and relevant queries were added. Changed the UI section of data tooling to match what we ended up using.