Bit & Books

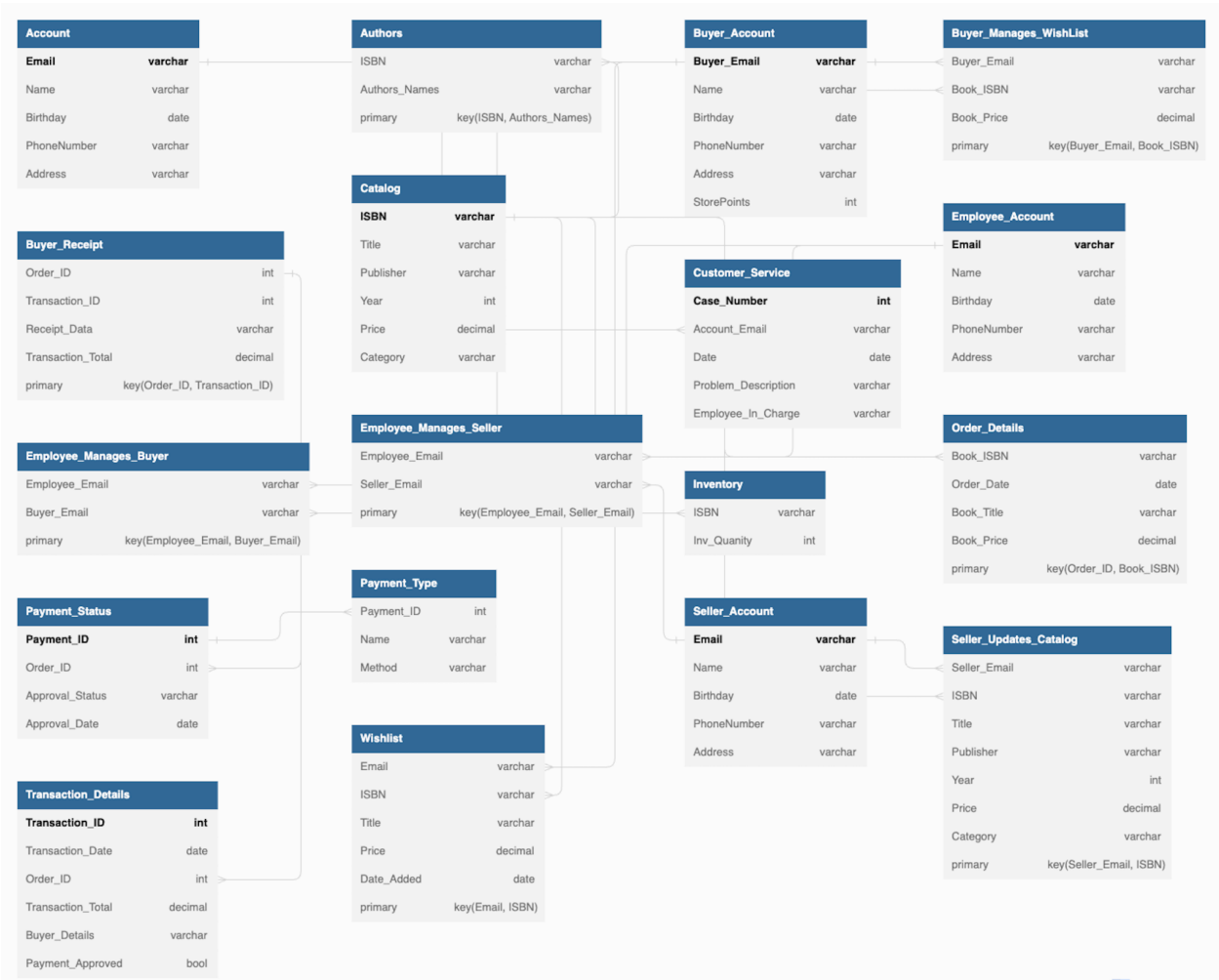
CSE 3241

Anmol Kumar & Jay Ramesh

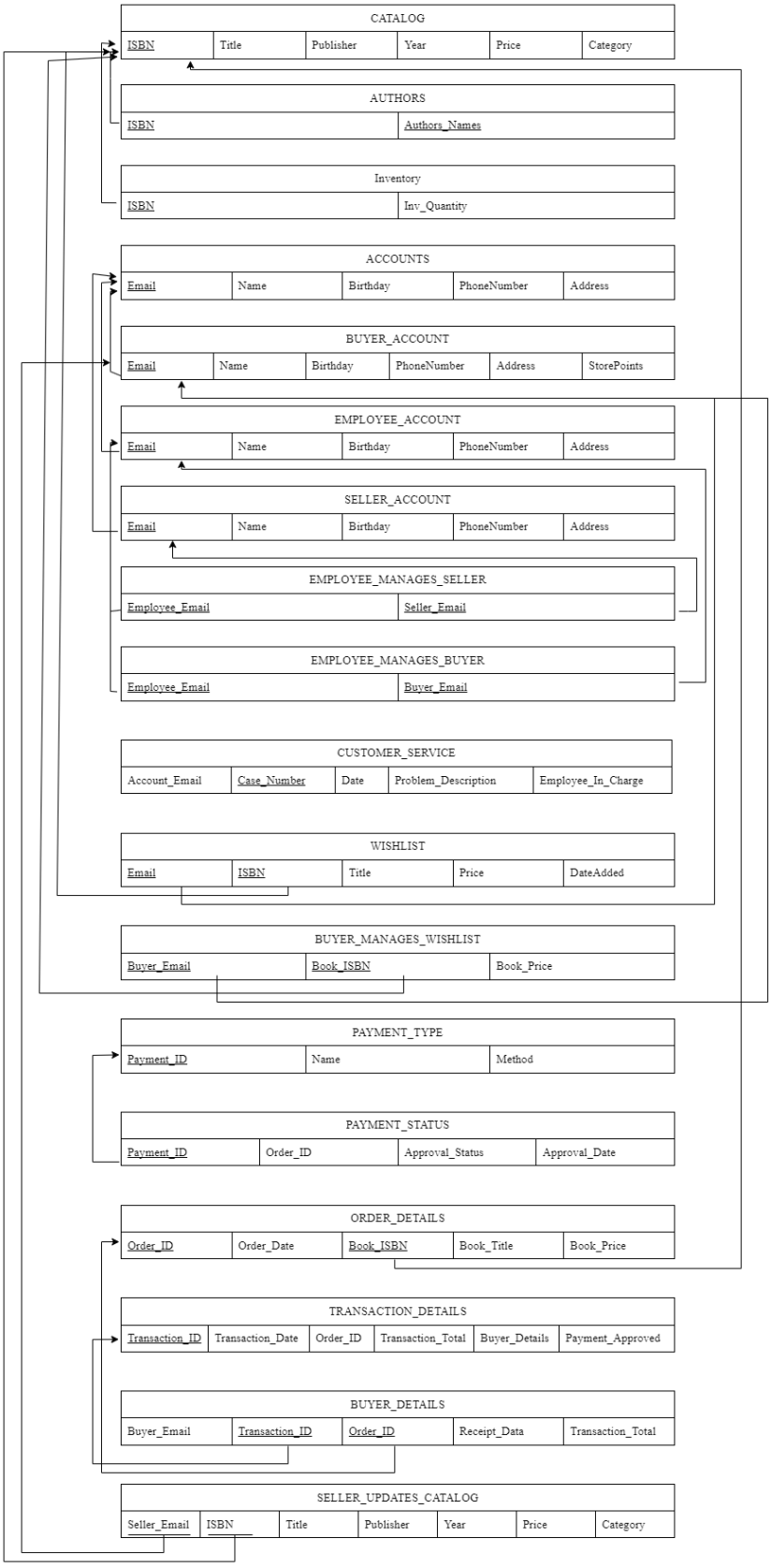
SP ‘23

Section 1 - Database Description

ER – Model



Relational Schema



Database fully normalized/justifications

ACCOUNT
(Account) = {Email, Name, Birthday, PhoneNumber, Address}
Primary Key = Email

3NF = {Email → Name, Birthday, PhoneNumber, Address}

This table is already in 1NF since all keys/attributes are atomic. Additionally, the table is already in 2NF since there is a composite primary/candidate key. This table is in 3NF because it has no transitive dependencies. Every non-key trait/attribute depends on a singular superkey and not another non-key. Under the assumption that the phone number and address can have multiple accounts.

AUTHORS
(Authors) = {ISBN, Authors_Names}
Primary Key = Combination of both ISBN and Authors_Names

3NF = {ISBN, Authors_Names}

This table is already in 1NF since all keys/attributes are atomic. Additionally, the table is already in 2NF since there is a composite primary/candidate key. This table is in 3NF because it has no transitive dependencies. Since there are no non-key attributes, there are no transitive dependencies.

BUYER_ACCOUNT
(Buyer_Account) = {Buyer_Email, Name, Birthday, PhoneNumber, Address, StorePoints}
Primary Key = Buyer_Email

3NF = {Buyer_Email→ Name, Birthday, PhoneNumber, Address, StorePoints}

This table is already in 1NF since all keys/attributes are atomic. Additionally, the table is already in 2NF since there is a composite primary/candidate key. This table is in 3NF because it has no transitive dependencies. Every non-key trait/attribute depends on a singular superkey and not another non-key. Under the assumption that the phone number and address can have multiple accounts.

BUYER_MANAGES_WISHLIST
(Buyer_Manages_WishList) = {Buyer_Email, Book_ISBN, Book_Price}
Primary Key = Buyer_Email, Book_ISBN

3NF = {Buyer_Email, Book_ISBN → Book_Price}

This table is already in 1NF since all keys/attributes are atomic. Additionally, the table is already in 2NF since there is a composite primary/candidate key. This table is in 3NF because it has no transitive dependencies. Every non-key trait/attribute depends on a singular superkey and not another non-key. Under the assumption that the combination of the Buyer's email and the book ISBN will provide a unique identifier key.

BUYER_RECEIPT
(Buyer_Receipt) = {Order ID, Transaction_ID, Receipt_Data, Transaction_Total}
Primary Key = Order_ID, Transaction_ID

3NF = {Order ID, Transaction_ID → Receipt_Data, Transaction_Total}

This table is already in 1NF since all keys/attributes are atomic. Additionally, the table is already in 2NF since there is a composite primary/candidate key. This table is in 3NF because it has no transitive dependencies. Every non-key trait/attribute depends on a singular superkey and not another non-key. Each Order_ID is unique to each order that occurs.

CATALOG
(Catalog) = {ISBN, Title, Publisher, Year, Price, Category}
Primary Key = ISBN

3NF = {ISBN → Title, Publisher, Year, Price, Category}

This table is already in 1NF since all keys/attributes are atomic. Additionally, the table is already in 2NF since there is a composite primary/candidate key. This table is in 3NF because it has no transitive dependencies. Every non-key trait/attribute depends on a singular superkey and not another non-key.

CUSTOMER_SERVICE
(Customer_Service) = {Case_Number, Account_Email, Date, Problem_Description, Employee_In_Charge}
Primary Key = Case_Number

3NF = {Case_Number → Account_Email, Date, Problem_Description, Employee_In_Charge}

This table is already in 1NF since all keys/attributes are atomic. Additionally, the table is already in 2NF since there is a composite primary/candidate key. This table is in 3NF because it has no

transitive dependencies. Every non-key trait/attribute depends on a singular superkey and not another non-key. Under the assumption that an account can have multiple problems, and an employee can be in charge of multiple accounts

EMPLOYEE_ACCOUNT
(Employee_Account) = {Email, Name, Birthday, PhoneNumber, Address}
Primary Key = Email

3NF = {Emaill → Birthday, PhoneNumber, Address}

This table is already in 1NF since all keys/attributes are atomic. Additionally, the table is already in 2NF since there is a composite primary/candidate key. This table is in 3NF because it has no transitive dependencies. Every non-key trait/attribute depends on a singular superkey and not another non-key. Under the assumption that the phone number and address can have multiple accounts.

EMPLOYEE_MANAGES_BUYER
(Employee_Manages_Buyer) = {Employee_Email, Buyer_Email}
Primary Key = Combination of both Employee_Email and Buyer_Email

3NF = {Employee_Email, Buyer_Email}

This table is already in 1NF since all keys/attributes are atomic. Additionally, the table is already in 2NF since there is a composite primary/candidate key. This table is in 3NF because it has no transitive dependencies. Since there are no non-key attributes, there are no transitive dependencies.

EMPLOYEE_MANAGES_SELLER
(Employee_Manages_Seller) = {Employee_Email, Seller_Email}
Primary Key = Combination of both Employee_Email and Seller_Email

3NF = {Employee_Email,Seller_Email}

This table is already in 1NF since all keys/attributes are atomic. Additionally, the table is already in 2NF since there is a composite primary/candidate key. This table is in 3NF because it has no transitive dependencies. Since there are no non-key attributes, there are no transitive dependencies.

INVENTORY
(Inventory) = {ISBN, Inv_Quanity}
Primary Key = ISBN

3NF = {ISBN → Inv_Quanity}

This table is already in 1NF since all keys/attributes are atomic. Additionally, the table is already in 2NF since there is a composite primary/candidate key. This table is in 3NF because it has no transitive dependencies. Every non-key trait/attribute depends on a singular superkey and not another non-key.

ORDER_DETAILS
(Order_Details) = {Order_ID,, Order_Date, Book_Title, Book_Price}
Primary Key = Order_ID, Book_ISBN

3NF = {Order_ID, Book_ISBN→ Order_Date, Book_Title, Book_Price}

This table is already in 1NF since all keys/attributes are atomic. Additionally, the table is already in 2NF since there is a composite primary/candidate key. This table is in 3NF because it has no transitive dependencies. Every non-key trait/attribute depends on a singular combination superkey and not another non-key. Each order ID coupled with each individual book ISBN purchased creates a unique identifying primary key.

PAYMENT_STATUS
(Payment_Status) = {Payment_ID, Order_ID, Approval_Status, Approval_Date}
Primary Key = Payment_ID

3NF = {Payment_ID → Order_ID, Approval_Status, Approval_Date}

This table is already in 1NF since all keys/attributes are atomic. Additionally, the table is already in 2NF since there is a composite primary/candidate key. This table is in 3NF because it has no transitive dependencies. Every non-key trait/attribute depends on a singular superkey and not another non-key

PAYMENT_TYPE
(Payment_Type) = {Payment_ID, Name, Method}
Primary Key = Payment_ID

3NF = {Payment_ID → Name, Method}

This table is already in 1NF since all keys/attributes are atomic. Additionally, the table is already in 2NF since there is a composite primary/candidate key. This table is in 3NF because it has no transitive dependencies. Every non-key trait/attribute depends on a singular superkey and not another non-key

SELLER_ACCOUNT
(Seller_Account) = {Email, Name, Birthday, PhoneNumber, Address}
Primary Key = Email

3NF = {Emaill → Birthday, PhoneNumber, Address}

This table is already in 1NF since all keys/attributes are atomic. Additionally, the table is already in 2NF since there is a composite primary/candidate key. This table is in 3NF because it has no transitive dependencies. Every non-key trait/attribute depends on a singular superkey and not another non-key. Under the assumption that the phone number and address can have multiple accounts.

SELLER_UPDATES_CATALOG
(Seller_Updates_Catalog) = {Seller_Email, ISBN, Title, Publisher, Year, Price, Category}
Primary Key = Combination of both Seller_Email and ISBN

3NF = {Seller_Email, ISBN, → Title, Publisher, Year, Price, Category}

This table is already in 1NF since all keys/attributes are atomic. Additionally, the table is already in 2NF since there is a composite primary/candidate key. This table is in 3NF because it has no transitive dependencies. Every non-key trait/attribute depends on a singular combination superkey and not another non-key. Seller_Email and ISBN, together, create the primary key that keeps this table in 3NF.

TRANSACTION_DETAILS
(Transaction_Details) = {Transaction_ID, Transaction_Date, Order_ID, Transaction_Total, Buyer_Details, Payment_Approved}

Primary Key = Tranaction_ID

3NF = {Transaction_ID → Transaction_Date, Order_ID, Transaction_Total, Buyer_Details, Payment_Approved}

This table is already in 1NF since all keys/attributes are atomic. Additionally, the table is already in 2NF since there is a composite primary/candidate key. This table is in 3NF because it has no transitive dependencies. Every non-key trait/attribute depends on a singular superkey and not another non-key.

WISHLIST
(Wishlist) = {Email, ISBN, Title, Price, Date_Added}
Primary Key = Combination of both Email and ISBN

3NF = {Email, ISBN → Title, Price, Date_Added}

This table is already in 1NF since all keys/attributes are atomic. Additionally, the table is already in 2NF since there is a composite primary/candidate key. This table is in 3NF because it has no transitive dependencies. Every non-key trait/attribute depends on a singular combination superkey and not another non-key. Email and ISBN, together, create the primary key that keeps this table in 3NF.

Relational Algebra and SQL for Each View

FIRST VIEW | FULL_CATALOG
The first view to create is a full catalog that contains the ISBN, Title, Author's Names, Publisher, and Price of all the books. This view would be useful for customers to have so that they have full access to what books Bits & Books provides. It would also be a useful guide to have for anyone wanting to just search for a book from either the ISBN, Author, or Publisher.

Relational Algebra View:
FULL_CATALOG ← π ISBN, Title, Authors_Names, Publisher, Price ((CATALOG ⋈ ISBN = ISBN AUTHORS))

SQL Statements:

```
CREATE VIEW FULL_CATALOG AS
        SELECT CATALOG.ISBN, CATALOG.Title, AUTHORS.Authors_Names,
        CATALOG.Publisher, CATALOG.Price FROM CATALOG
INNER JOIN AUTHORS ON CATALOG.ISBN = AUTHORS.ISBN;
```

FULL_CATALOG SAMPLE DATA

ISBN	Title	Authors_Names	Publisher	Price
1561586196	Building A Shed	Joseph Truini	Taunton Pr	$19.95
0061092177	Small Gods	Terry Pratchett	Harper	$7.99
00610502935	Going Postal	Terry Pratchett	Harper	$7.99
0061020656	Pyramids	Terry Pratchett	Harper	$7.99
0060855925	The Color of Magic	Terry Pratchett	Harper	$13.99
0061020648	Guards! Guards!	Terry Pratchett	Harper	$7.99

SECOND VIEW | ONE_WISH

The second view is to create ways to let customers know they have items in their wishlist and if those items are in stock. Bits & Books can use this to see which books from customers' wishlists they need to restock so that the customers can buy those books. Additionally, the store can use this view to remind customers they have items in their wishlist.

Relational Algebra:

ONE_WISH ← π Name, Email, Title, Inv_Quantity ((BUYER_ACCOUNT ⋈ Email = WISHLIST.Email) ⋈ WISHLIST.ISBN = INVENTORY.ISBN)

SQL Statements:

```
CREATE VIEW ONE_WISH
        AS SELECT BUYER_ACCOUNT.Name, BUYER_ACCOUNT.Email, WISHLIST.Title,
        INVENTORY.Inv_Quantity
FROM WISHLIST
INNER JOIN INVENTORY ON WISHLIST.ISBN = INVENTORY.ISBN
INNER JOIN BUYER_ACCOUNT ON WISHLIST.Email = BUYER_ACCOUNT.Email;
```

ONE_WISH SAMPLE DATA

Name	Email	Title	Inv_Quantity
John Smith	123test@test.net	Architecture: Form, Space, and Order	2
Bob Davis	456test@test.net	A Visual Dictionary of Architecture	2

Bob Davis	456test@test.net	The Magician's Assistant	5
Bob Davis	456test@test.net	Patron Saint of Liars	8
Bob Davis	456test@test.net	How the Mind Works	0
Petunia Brown	789test@test.net	The Language Instinct: How the Mind Creates Langua	4

Indexes

Indexes are data elements that essentially allow for queries to implement faster. Indexes provide a path for queries not search every row for specific data, and instead use Indexes in order to find the needed data faster. Indexes can form as hashtables or tree structures. Both of these data structures result in a faster time.

Two indexes we deemed important to create were:

1. Genre - this allows customers to just search for books in genres they are interested in

   SQL Query:

   CREATE INDEX Genre
   ON CATALOG(CATEGORY);
2. Price Range - this gives customers a filter to find books within price ranges

   SQL Query:

   CREATE INDEX Price_Range ON CATALOG(Price);

Section 2 - User Manual

A user manual describing the usage of your database, for use by developers who are going to be writing code to use your database.

ACCOUNT - This database is a superclass. It contains all the buyer, employee, and seller accounts.

- Email - Primary key that holds the email address of the account holder (VARCHAR())
- Name - Name of the account holder (CHAR())
- Birthday - Birthday of the account holder (DATE())
- PhoneNumber - Phone Number of the account holder (INT)
- Address - Home address of the account holder (VARCHAR())
-

BUYER_ACCOUNT - This database is a specialization class of ACCOUNT. It contains all the buyer accounts.

- Email - Primary key that holds the email address of the customer account (VARCHAR())
- Name - Name of the account holder (CHAR())
- Birthday - Birthday of the account holder (DATE())
- PhoneNumber - Phone Number of the account holder (INT)
- Address - Home address of the account holder (VARCHAR())
- StorePoints - Buyer gets store points for purchasing books. Points can be used to get further discounts and exclusive access (INT)

EMPLOYEE_ACCOUNT - This database is a specialization class of ACCOUNT. It contains all the employee accounts.

- Email - Primary key that holds the email address of the employee account (VARCHAR())
- Name - Name of the employee (CHAR())
- Birthday - Birthday of the employee (DATE())
- PhoneNumber - Phone Number of the employee (INT)
- Address - Home address of the employee (VARCHAR())

SELLER_ACCOUNT - This database is a specialization class of ACCOUNT. It contains all the seller accounts.

- Email - Primary key that holds the email address of the seller (VARCHAR())
- Name - Name of the seller (CHAR())
- Birthday - Birthday of the seller (DATE())
- PhoneNumber - Phone Number of the seller (INT)
- Address - Home address of the seller (VARCHAR())

EMPLOYEE_MANAGES_BUYER - This relationship focuses on Employee Accounts overlooking Buyer Accounts in cases that any trouble occurs.

- Employee_Email - Employee Email that is in charge of the relation (VARCHAR())
- Buyer_Email - Buyer Email that may have a problem (VARCHAR())
- Both work together to create the primary keys

EMPLOYEE_MANAGES_SELLER - This relationship focuses on Employee Accounts overlooking Seller Accounts in cases that any trouble occurs.

- Employee_Email - Employee Email that is in charge of the relation (VARCHAR())
- Seller_Email - Seller Email that may have a problem (VARCHAR())
- Both work together to create the primary keys
-

CUSTOMER_SERVICE - This database represents and holds the problems that account have and need help from customer service (employees)

- Case_Number - Primary Key containing certain codes that identify the problem
- Account_Email - Email of the account that is having a problem (VARCHAR())
- Date - Date that the problem occurred (Date)
- Problem_Description - A brief description of the problem with the account (VARCHAR())
- Employee_InCharge - Email of the Employee who received the problem and is in charge of fixing the problem (VARCHAR())

CATALOG - This database holds all the information, apart from the Authors, of each book that is sold at Bits&Books

- ISBN - Primary key, series of digits that is unique to each book (VARCHAR())
- Title - Title of each book (VARCHAR())
- Publisher - Publisher of each book (VARCHAR())
- Year - Year the book was published (INT)
- Price - Price of each book (VARCHAR())
- Category - Genre of each book (VARCHAR())

AUTHORS - This database holds the authors of each book, tagged to each ISBN

- ISBN - Series of digits that is unique to each book (VARCHAR())
- Authors_Names - Author of each book. A book can have multiple authors, hence why the ISBN of a book may occur more than once (CHARS())
- Both ISBN and Authors_Names work together to create the primary keys

INVENTORY - This database holds the quantity of each book

- ISBN - Series of digits that is unique to each book (VARCHAR())

- Inv_Quantity - The quantity of each book (INT)

WISHLIST - This database represents all the book customers want to buy
- Email - Email address of the buyer account (VARCHAR())
- ISBN - Series of digits that is unique to each book (VARCHAR())
- Title - Title of each book (VARCHAR())
- Price - Price of each book (VARCHAR())
- DateAdded - Date that the book was added to the wishlist (DATE)
- Both Email and ISBN work together to create the primary keys

BUYER_MANAGES_WISHLIST - This database allows a customer to add books and manage which books they want to add or remove
- Buyer_Email - Email address of the buyer account (VARCHAR())
- Book_ISBN - Series of digits that is unique to each book (VARCHAR())
- Book_Price - Price of each book (VARCHAR())
- Both Buyer_Email and Book_ISBN work together to create the primary keys

PAYMENT_TYPE - This database represents the type of payment method that occurred when a customer purchases a book(s)
- Payment_ID - The primary key which consists of random numbers in order to create a unique code for each payment (INT)
- Name - Name of the individual purchasing the book (VARCHAR())
- Method - Method of payment, card or cash (VAARCHAR())

PAYMENT_STATUS - This database contains whether or not the purchase was approved or denied
- Payment_ID - The primary key which consists of random numbers in order to create a unique code for each payment (INT)
- Order_ID - Another unique key that represents each individual order (INT)
- Approval_Status - Whether or not the payment was approved or denied (BOOLEAN)
- Approval_Date - The date the payment was approved

ORDER_DETAILS - This database contains details of each order (i.e. the title, ISBN, and price, of each book). This order is what carries from the merchant to the buyer
- Order_ID - Unique key that represents each individual order (INT)
- Order_Date - The date the order was created (DATE)
- Book_ISBN - Series of digits that is unique to each book (VARCHAR())
- Book_Title - Title of each book (VARCHAR())
- Book_Price - Price of each book (VARCHAR())
- Both Order_ID and Book_ISBN work together to create the primary keys

TRANSACTION_DETAILS - This database contains of the transaction of each order. This transaction is what carries from the payment to the merchant

- Transaction_ID - The primary key used to identify each individual transaction (INT)
- Transaction_Date - Date that the transaction took place (DATE)
- Order_ID - Unique key that represents each individual order (INT)
- Transaction_Total - Total price of the transaction (DECIMAL())
- Buyer_Details - The buyer's email so the purchase can be linked to the consumer (VARCHAR())
- Payment_Approved - Whether or not the payment was approved (BOOLEAN)

BUYER_RECEIPT - This database contains all the transactions that have occurred and which individuals the transactions belong to

- Buyer_Email - The email of the buyer, unique to each customer (INT)
- Transaction_ID - Used to identify each individual transaction (INT)
- Order_ID - The Primary key that represents each individual order (INT)
- Receipt_Date - The date the receipt was created (DATE)
- Transaction_Total - Total price of the transaction (DECIMAL())

SELLER_UPDATES_CATALOG - This database holds all the books that a seller would like to update in the catalog in order to create a more accurate library for customers to look at

- Seller_Email - Email address of the seller (VARCHAR())
- ISBN - Series of digits that is unique to each book (VARCHAR())
- Title - Title of book (VARCHAR())
- Publisher - Publisher of the book (VARCHAR())
- Year - Year the book was published (VARCHAR())
- Price - Price of the book (DECIMAL())
- Category - Genre of the book (VARCHAR())

Sample Queries

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

Relational Algebra:
π Title ((σ Authors_Names LIKE '%Pratchett%' AND CAST(Price AS DECIMAL(10,2)) < 10.00) (CATALOG ⋈ ISBN=ISBN (AUTHORS)))

SQL Query:
```
SELECT Title FROM CATALOG
INNER JOIN AUTHORS on CATALOG.ISBN = AUTHORS.ISBN
WHERE AUTHORS.Authors_Names LIKE '%Pratchett%'
AND CAST(Price AS DECIMAL(10,2)) < 10.00;
```

Give all of the titles and dates for purchases made by a particular customer.
/ Let's find all the purchases made by John Smith */*

Relational Algebra:
π Book_Title, Receipt_Date ((σ Name='John Smith' ((ORDER_DETAILS ⋈ Order_ID=Order_ID (BUYER_RECEIPT)) ⋈ Buyer_Email=Email (BUYER_ACCOUNT)))))

SQL Query:
```
SELECT ORDER_DETAILS.Book_Title, BUYER_RECEIPT.Receipt_Date
FROM ORDER_DETAILS
INNER JOIN BUYER_RECEIPT ON ORDER_DETAILS.Order_ID =
BUYER_RECEIPT.Order_ID
INNER JOIN BUYER_ACCOUNT ON BUYER_RECEIPT.Buyer_Email =
BUYER_ACCOUNT.Email
WHERE BUYER_ACCOUNT.Name = 'John Smith';
```

List all of the books with less than 5 quantities in stock

Relational Algebra:
π Title (σ Inv_Quantity < 5 ((CATALOG ⋈ ISBN=ISBN (INVENTORY))))

SQL Query:
```
SELECT Title FROM CATALOG
INNER JOIN INVENTORY ON CATALOG.ISBN = INVENTORY.ISBN
WHERE INVENTORY.Inv_Quantity < 5;
```

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

Relational Algebra:

π BUYER_ACCOUNT.Name as Customer_Name, ORDER_DETAILS.Book_Title as Title, AUTHORS.Authors_Names as Authors
(σ AUTHORS.Authors_Names LIKE '%Pratchett%'
(((BUYER_ACCOUNT ⋈ᵢ Buyer_Receipt) ON BUYER_ACCOUNT.Email = BUYER_RECEIPT.Buyer_Email)
⋈ᵢ ((ORDER_DETAILS ON BUYER_RECEIPT.Order_ID = ORDER_DETAILS.Order_ID)
⋈ᵢ (CATALOG ON ORDER_DETAILS.Book_Title = CATALOG.Title))
⋈ᵢ (AUTHORS ON CATALOG.ISBN = AUTHORS.ISBN))

SQL Query:

```
SELECT BUYER_ACCOUNT.Name AS Customer_Name, ORDER_DETAILS.Book_Title AS Title, AUTHORS.Authors_Names AS Authors
FROM BUYER_ACCOUNT
INNER JOIN BUYER_RECEIPT ON BUYER_ACCOUNT.Email = BUYER_RECEIPT.Buyer_Email
INNER JOIN ORDER_DETAILS ON BUYER_RECEIPT.Order_ID = ORDER_DETAILS.Order_ID
INNER JOIN CATALOG ON ORDER_DETAILS.Book_Title = CATALOG.Title
INNER JOIN AUTHORS ON CATALOG.ISBN = AUTHORS.ISBN
WHERE AUTHORS.Authors_Names LIKE '%Pratchett%';
```

Find the total number of books purchased by a single customer
/ Let's assume the customer's name is 'Jaash Atluri' */*

Relational Algebra:

π BUYER_ACCOUNT.Name, count(ORDER_DETAILS.Book_ISBN) as Total_Books_Purchased (σ BUYER_ACCOUNT.Name = 'Jaash Atluri' ((BUYER_ACCOUNT ⋈ Buyer_Receipt) ⋈ ORDER_DETAILS))

SQL Query:

```
SELECT BUYER_ACCOUNT.Name, COUNT(ORDER_DETAILS.Book_ISBN) AS Total_Books_Purchased
FROM ORDER_DETAILS
INNER JOIN BUYER_RECEIPT ON ORDER_DETAILS.Order_ID = BUYER_RECEIPT.Order_ID
```

```
INNER JOIN BUYER_ACCOUNT ON BUYER_RECEIPT.Buyer_Email =
BUYER_ACCOUNT.Email
WHERE BUYER_ACCOUNT.Name = 'Jaash Atluri';
```

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

Relational Algebra:

σ rank=1 (γ Name, COUNT(Book_ISBN) AS Total_Books_Purchased ((ORDER_DETAILS ⋈ Order_ID=Order_ID (BUYER_RECEIPT) ⋈ Buyer_Email=Email (BUYER_ACCOUNT))) ÷ GROUP BY Name ÷ ORDER BY Total_Books_Purchased DESC)

SQL Query:

```
SELECT BUYER_ACCOUNT.Name, COUNT(ORDER_DETAILS.Book_ISBN) AS
Total_Books_Purchased
FROM ORDER_DETAILS
INNER JOIN BUYER_RECEIPT ON ORDER_DETAILS.Order_ID =
BUYER_RECEIPT.Order_ID
INNER JOIN BUYER_ACCOUNT ON BUYER_RECEIPT.Buyer_Email =
BUYER_ACCOUNT.Email
GROUP BY BUYER_ACCOUNT.Name
ORDER BY Total_Books_Purchased DESC
LIMIT 1;
```

Browse Catalog for a certain book

/ Customer is trying to find the book 'Small Gods' by Pratchett */*

Relational Algebra:

π CATALOG.Title, AUTHORS.Authors_Names, INVENTORY.Inv_Quantity AS Inventory y ((σ Title='Small Gods' ∧ Authors_Names LIKE '%Pratchett%' ((CATALOG ⋈ ISBN=ISBN (INVENTORY)) ⋈ ISBN=ISBN (AUTHORS))))

SQL Query:

```
SELECT CATALOG.Title, AUTHORS.Authors_Names, INVENTORY.Inv_Quantity AS
Inventory
FROM CATALOG
INNER JOIN INVENTORY ON CATALOG.ISBN = INVENTORY.ISBN
INNER JOIN AUTHORS ON CATALOG.ISBN = AUTHORS.ISBN
```

WHERE CATALOG.Title = 'Small Gods' AND AUTHORS.Authors_Names LIKE '%Pratchett%';

Customer can review their accounts
/ Requires Customer to input their email in order to review their own account */*
/ Assume the customer inputs '123test@test.net' */*

Relational Algebra:
σ Email='123test@test.net' (BUYER_ACCOUNT)

SQL Query:
SELECT * FROM BUYER_ACCOUNT
WHERE BUYER_ACCOUNT.Email = '123test@test.net';

Employee Reviews Sales

Relational Algebra:
π Book_ISBN, Book_Title, Inv_Quantity ((ORDER_DETAILS ⋈ Book_ISBN=ISBN INVENTORY))

SQL Query:
SELECT ORDER_DETAILS.Book_ISBN, ORDER_DETAILS.Book_Title, INVENTORY.Inv_Quantity
FROM ORDER_DETAILS
INNER JOIN INVENTORY ON ORDER_DETAILS.Book_ISBN = INVENTORY.ISBN;

Employee reviews sales and needs to see what books need to be re-ordered
/* If the quantity is 0 and the book has been ordered before, needs to restock */

Relational Algebra:
π Book_ISBN, Book_Title, Inv_Quantity ((σ Inv_Quantity=0 ((ORDER_DETAILS ⋈ Book_ISBN=ISBN INVENTORY))))

SQL Query:
SELECT ORDER_DETAILS.Book_ISBN, ORDER_DETAILS.Book_Title, INVENTORY.Inv_Quantity
FROM ORDER_DETAILS
INNER JOIN INVENTORY ON ORDER_DETAILS.Book_ISBN = INVENTORY.ISBN
WHERE INVENTORY.Inv_Quantity = 0;

Extra Queries

Find the person with the most amount of books in their wishlist

Relational Algebra:
π BUYER_ACCOUNT.name, COUNT(Book_ISBN) AS Total_Books
(σ BUYER_ACCOUNT.Email = BUYER_MANAGES_WISHLIST.Buyer_Email
(BUYER_ACCOUNT ⋈ᵢ BUYER_MANAGES_WISHLIST))
γ BUYER_ACCOUNT.name; Total_Books β max(Total_Books)

SQL Query:
```
SELECT BUYER_ACCOUNT.name, COUNT(Book_ISBN) AS Total_Books
FROM BUYER_ACCOUNT
INNER JOIN BUYER_MANAGES_WISHLIST ON BUYER_ACCOUNT.Email =
BUYER_MANAGES_WISHLIST.Buyer_Email
GROUP BY Name
ORDER BY Total_Books DESC
LIMIT 1;
```

Find the reason for a specific customer service problem.
/ Customers name is Maggie Brown */*

Relational Algebra:
π ACCOUNTS.Name, CUSTOMER_SERVICE.Problem_Description
(σ ACCOUNTS.Name = 'Maggie Brown' (CUSTOMER_SERVICE ⋈ᵢ ACCOUNTS))

SQL Query:
```
SELECT ACCOUNTS.Name, CUSTOMER_SERVICE.Problem_Description
FROM CUSTOMER_SERVICE
INNER JOIN ACCOUNTS ON CUSTOMER_SERVICE.Account_Email = ACCOUNTS.Email
WHERE ACCOUNTS.Name = 'Maggie Brown';
```

In ascending order, get the price of each customer's wishlist

Relational Algebra:
WISHLIST ← σ WISHLIST.Price LIKE '%$%' (WISHLIST)
WISHLIST ← WISHLIST {Price ← REPLACE(WISHLIST.Price, '$', '')}

π BUYER_ACCOUNT.name, CAST(WISHLIST.Price AS DECIMAL(10,2)) AS Total_Price
((BUYER_ACCOUNT ⋈$_i$ WISHLIST)
γ BUYER_ACCOUNT.name; sum(Price) AS Total_Price) β Total_Price

SQL Query:

```
UPDATE WISHLIST
SET Price = REPLACE(Price, '$', '')
WHERE WISHLIST.Price LIKE '%$%';
SELECT BUYER_ACCOUNT.name, CAST(WISHLIST.Price AS DECIMAL(10,2)) AS
Total_Price
FROM BUYER_ACCOUNT
INNER JOIN WISHLIST ON BUYER_ACCOUNT.Email = WISHLIST.Email
GROUP BY Name
ORDER BY Total_Price ASC;
```

Find the top 5 customers who have accumulated the most store points and their current store point balances.

Relational Algebra

π(Name, StorePoints) τ(5)(σ(Buyer_Account) ⋈ (StorePoints DESC))

SQL Query:

```
SELECT Name, StorePoints FROM Buyer_Account
ORDER BY StorePoints DESC
LIMIT 5;
```

Find the average price of a book in each category - keep the price at two decimals
/ Update CATALOG TO GET RID OF '$', then cast AVG command since all the values are number */*
Relational Algebra:

π(CATEGORY, avg_price) (γ(CATEGORY, AVG(Price) as avg_price) (CATALOG))

SQL Query:

```
UPDATE CATALOG
SET Price = REPLACE(Price, '$', '')
WHERE CATALOG.Price LIKE '%$%';
SELECT CATEGORY, ROUND(AVG(Price), 2) AS avg_price FROM CATALOG
GROUP BY category;
```

Find the most used method payment type, and display the counts of each payment method in desc order

Relational Algebra:

π(Method, Num_Transactions) (
  γ(Method, COUNT(*) as Num_Transactions) (
    σ(Method = 'Card' ∨ Method = 'Cash')(PAYMENT_TYPE))

SQL Query:

```
SELECT Method, COUNT(*) as Num_Transactions FROM PAYMENT_TYPE
WHERE Method = 'Card' OR Method = 'Cash'
GROUP BY Method
ORDER BY Num_Transactions DESC;
```

INSERT and DELETE Syntaxes

When it comes to inserting data, all dependencies must be taken in account. For example, when adding a book to AUTHORS and INVENTORY, since the they are dependent on the ISBN, the values of the book (especially the ISBN) must be added to CATALOG first. This is because the ISBN is the primary key, or part of the primary key, for both of the tables. Below is the following code when insert the book ‘The Catcher in the Rye’).

```
INSERT INTO CATALOG (ISBN, Title, Publisher, Year, Price, CATEGORY) VALUES
('9783161484100', 'The Catcher in the Rye', 'Little, Brown and Company', 1951, '10.99',
'Fiction');
INSERT INTO AUTHORS (ISBN, Authors_Names) VALUES
('9783161484100', 'J. D. Salinger');
INSERT INTO INVENTORY (ISBN, Inv_Quantity) VALUES
('9783161484100', 50);
```

The same application logic applies when deleting data. Since AUTHORS and INVENTORY have a dependence of ISBN from CATALOG, when deleting book via ISBN in CATALOG, the same ISBN must be deleted from AUTHORS and INVENTORY. Below is an example using the ISBN of the book ‘The Catcher in the Rye’.

```
DELETE FROM CATALOG WHERE ISBN = '9783161484100';
DELETE FROM AUTHORS WHERE ISBN = '9783161484100';
DELETE FROM INVENTORY WHERE ISBN = '9783161484100';
```

Section 3 - Graded Checkpoint Documents

Below are the graded checkpoint documents.

Project Checkpoint 3 was a binary file, and therefore are unable to show it here. However, all the checkpoints and their pdf are in the zipped file.

Additionally, in the zipped file is the SQL file, the binary file, all the data CSV and XLSX files, adn this final project PDF.

CSE 3241 Project Checkpoint 01 – Entities and Relationships

Names: Jared Malto, Anmol Kumar Date: 1/27/23

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.

   CUSTOMER- customer id, username, name, address, phone number
   BOOK- ISBN, Name, author first name, author last name, price, genre, publisher
   EMPLOYEE- employee id, username, name, address, work phone number, position
   SELLER- seller id, mailing address, publishers worked with, contact within the company
   TECHNICAL STAFF- employee id, username, name, address, work phone number, last time and date logged into company database
   PUBLISHERS- name, employee contact, account number, [list of titles]

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

   EMPLOYEE oversees SELLER
   TECHNICAL STAFF is a division of EMPLOYEES
   SELLER owns the right to given BOOK

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.

   A wish list entity is always useful for customers to keep tract of items they want to purchase but cannot at the present. Keeping track of this could help customers remember what they want, and they can even share it with friends.

   A sale entity is also a good implementation. If the company needed to clear inventory fast it can go under this entity and hopefully it can sell.

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.
   a. Find a book on sale that is less than $3

b. Find a book on my friend's wish list in the adventure genre
c. Find the main employee contact for a given seller company
d. Find who made the last update to the database

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.

   How we built the database above would not allow for a publisher to be added without knowing any book titles because I have it as an attribute under the book entity. After Revision, I have a PUBLISHER entity with various information about them as attributes. For example, I added an attribute of a list of titles so we can easily look up who published what.

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.
   a. Update the list of titles the store carries from the publisher entity
   b. Update a person's wish list (add to wish list)
   c. Update a price on a sale item (mark down the item even more)

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.

CSE 3241 Project Checkpoint 02 – Relational Model and Relational Algebra

Names	Date
Anmol Kumar	02-21-23

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

1. Provide a current version of your ER Model as per Project Checkpoint 01. If you were instructed to change the model for Project Checkpoint 01, make sure you use the revised version of your ER Model.

   CUSTOMER- customer id, username, name, address, phone number, transaction

   BOOK- ISBN, Name, author first name, author last name, price, genre, publisher

   EMPLOYEE- employee id, username, name, address, work phone number, position

   SELLER- seller id, mailing address, publishers worked with, contact within the company

   TECHNICAL STAFF- employee id, username, name, address, work phone number, last time and date logged into company database

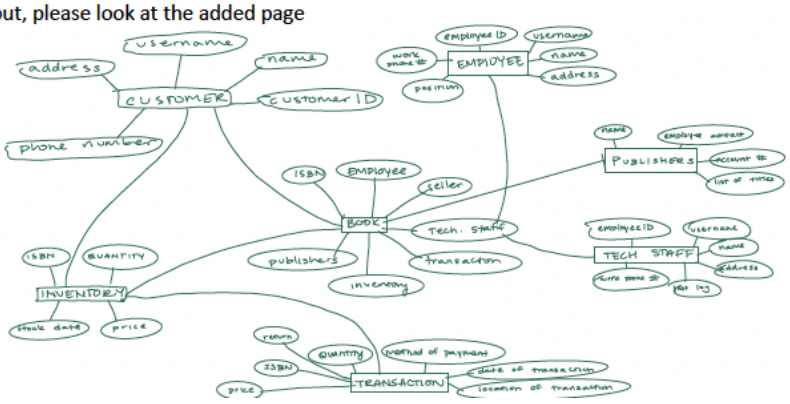
   PUBLISHERS- name, employee contact, account number, [list of titles]

   INVENTORY – ISBN, quantity, price, stock date

   TRANSACTION – price, ISBN, quantity, method of payment, date of transaction, location of transaction, return, max purchase, customer ID

2. Map your ER model to a relational schema. Indicate all primary and foreign keys.

   I drew this out, please look at the added page



3. Given your relational schema, provide the relational algebra 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:
   a. Find the titles of all books by Pratchett that cost less than $10

      $\sigma_{price < \$10}\ (\pi_{authorLastName = 'Pratchett'}\ Book)$

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

      $\pi_{(BookTitle,\ dateOfPurchase)}\ (\sigma_{CUSTOMER}\ (Book * Transaction))$

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

      $\pi\ (BOOK\ TITLE,\ ISBN)\ (\sigma_{INVENTORY.STOCK < 5})$

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

      $\pi_{(CUSTOMER.name,\ Book.authorLastName = 'Pratchett', book.title)}\ (CUSTOMER * BOOK)$

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

      $\pi\ SUM\ (CUSTOMER.transactions)$

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

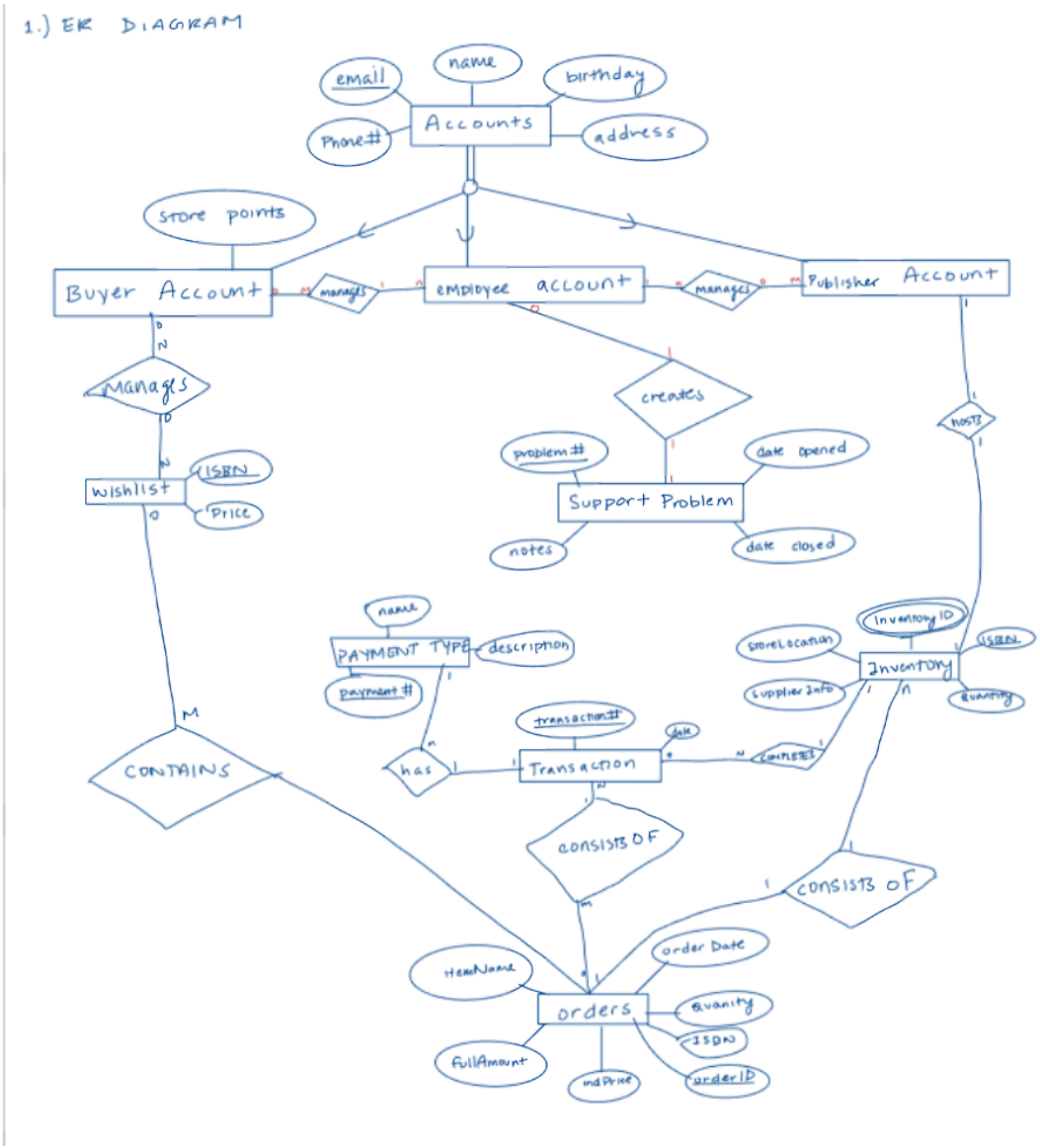
      $\pi\ transactions.maxpurchase\ (CUSTOMER \bowtie_{customer.customerID = transaction.customerID} TRANSACTION)$

4. 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.

   BEST SELLERS - Book name, ISBN, Publisher
   CLEARENCE – discounted price, book name, ISBN, time on shelf
   DISPLAY BOOK – book on display, book name, inventory stock, ISBN

CSE 3241 Project Checkpoint 04 – Functional Dependencies and Normal Forms

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.**



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).

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.

ACCOUNT
(Account) = {Email, Name, Birthday, PhoneNumber, Address}
Primary Key = Email
3NF = {Email → Name, Birthday, PhoneNumber, Address}

AUTHORS
(Authors) = {ISBN, Authors_Names}
Primary Key = Combination of both ISBN and Authors_Names
3NF = {ISBN, Authors_Names}

BUYER_ACCOUNT
(Buyer_Account) = {Buyer_Email, Name, Birthday, PhoneNumber, Address, StorePoints}
Primary Key = Buyer_Email
3NF = {Buyer_Email→ Name, Birthday, PhoneNumber, Address, StorePoints}

BUYER_MANAGES_WISHLIST
(Buyer_Manages_WishList) = {Buyer_Email, Book_ISBN, Book_Price}
Primary Key = Buyer_Email, Book_ISBN
3NF = {Buyer_Email, Book_ISBN → Book_Price}

BUYER_RECEIPT
(Buyer_Receipt) = {Order ID, Transaction_ID, Receipt_Data, Transaction_Total}
Primary Key = Order_ID, Transaction_ID
3NF = {Order ID, Transaction_ID → Receipt_Data, Transaction_Total}

CATALOG
(Catalog) = {ISBN, Title, Publisher, Year, Price, Category}
Primary Key = ISBN
3NF = {ISBN → Title, Publisher, Year, Price, Category}

CUSTOMER_SERVICE
(Customer_Service) = {Case_Number, Account_Email, Date, Problem_Description, Employee_In_Charge}
Primary Key = Case_Number
3NF = {Case_Number → Account_Email, Date, Problem_Description, Employee_In_Charge}

EMPLOYEE_ACCOUNT
(Employee_Account) = {Email, Name, Birthday, PhoneNumber, Address}
Primary Key = Email
3NF = {Emaill → Birthday, PhoneNumber, Address}

EMPLOYEE_MANAGES_BUYER
(Employee_Manages_Buyer) = {Employee_Email, Buyer_Email}
Primary Key = Combination of both Employee_Email and Buyer_Email
3NF = {Employee_Email, Buyer_Email}

EMPLOYEE_MANAGES_SELLER
(Employee_Manages_Seller) = {Employee_Email, Seller_Email}
Primary Key = Combination of both Employee_Email and Seller_Email
3NF = {Employee_Email,Seller_Email}

INVENTORY
(Inventory) = {ISBN, Inv_Quanity}
Primary Key = ISBN
3NF = {ISBN → Inv_Quanity}

ORDER_DETAILS
(Order_Details) = {Order_ID,, Order_Date, Book_Title, Book_Price}
Primary Key = Order_ID, Book_ISBN

3NF = {Order_ID, Book_ISBN→ Order_Date, Book_Title, Book_Price}
PAYMENT_STATUS
(Payment_Status) = {Payment_ID, Order_ID, Approval_Status, Approval_Date}
Primary Key = Payment_ID
3NF = {Payment_ID → Order_ID, Approval_Status, Approval_Date}
PAYMENT_TYPE
(Payment_Type) = {Payment_ID, Name, Method}
Primary Key = Payment_ID
3NF = {Payment_ID → Name, Method}
SELLER_ACCOUNT
(Seller_Account) = {Email, Name, Birthday, PhoneNumber, Address}
Primary Key = Email
3NF = {Emaill → Birthday, PhoneNumber, Address}
SELLER_UPDATES_CATALOG
(Seller_Updates_Catalog) = {Seller_Email, ISBN, Title, Publisher, Year, Price, Category}
Primary Key = Combination of both Seller_Email and ISBN
3NF = {Seller_Email, ISBN, → Title, Publisher, Year, Price, Category}
TRANSACTION_DETAILS
(Transaction_Details) = {Transaction_ID, Transaction_Date, Order_ID, Transaction_Total, Buyer_Details, Payment_Approved}
Primary Key = Tranaction_ID
3NF = {Transaction_ID → Transaction_Date, Order_ID, Transaction_Total, Buyer_Details, Payment_Approved}
WISHLIST
(Wishlist) = {Email, ISBN, Title, Price, Date_Added}
Primary Key = Combination of both Email and ISBN
3NF = {Email, ISBN → Title, Price, Date_Added}

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.

a. Don't need to provide BCNF

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.

a. FULL CATALOG – Displays all the details of each b
   i. ISBN
   ii. Title
   iii. Author
   iv. Publisher
   v. Year
   vi. Price
   vii. Category/Genre
b. Wishlist Cart – Displays if books in Wishlist are in stock or not

On the next page is the code for each of the views

```
6. /* Display a Full Catalog
   the ISBN, Title, Authors, Publisher, and Price of all books */
CREATE VIEW FULL_CATALOG AS
SELECT CATALOG.ISBN, CATALOG.Title, AUTHORS.Authors_Names,
CATALOG.Publisher, CATALOG.Price
FROM CATALOG
INNER JOIN AUTHORS ON CATALOG.ISBN = AUTHORS.ISBN;

/* Display ONE_WISH
   Customer Email, Name, Items in their Wishlist
 */
CREATE VIEW ONE_WISH AS
SELECT BUYER_ACCOUNT.Name, BUYER_ACCOUNT.Email, WISHLIST.Title,
INVENTORY.Inv_Quantity
FROM WISHLIST
INNER JOIN INVENTORY ON WISHLIST.ISBN = INVENTORY.ISBN
INNER JOIN BUYER_ACCOUNT ON WISHLIST.Email = BUYER_ACCOUNT.Email;
```