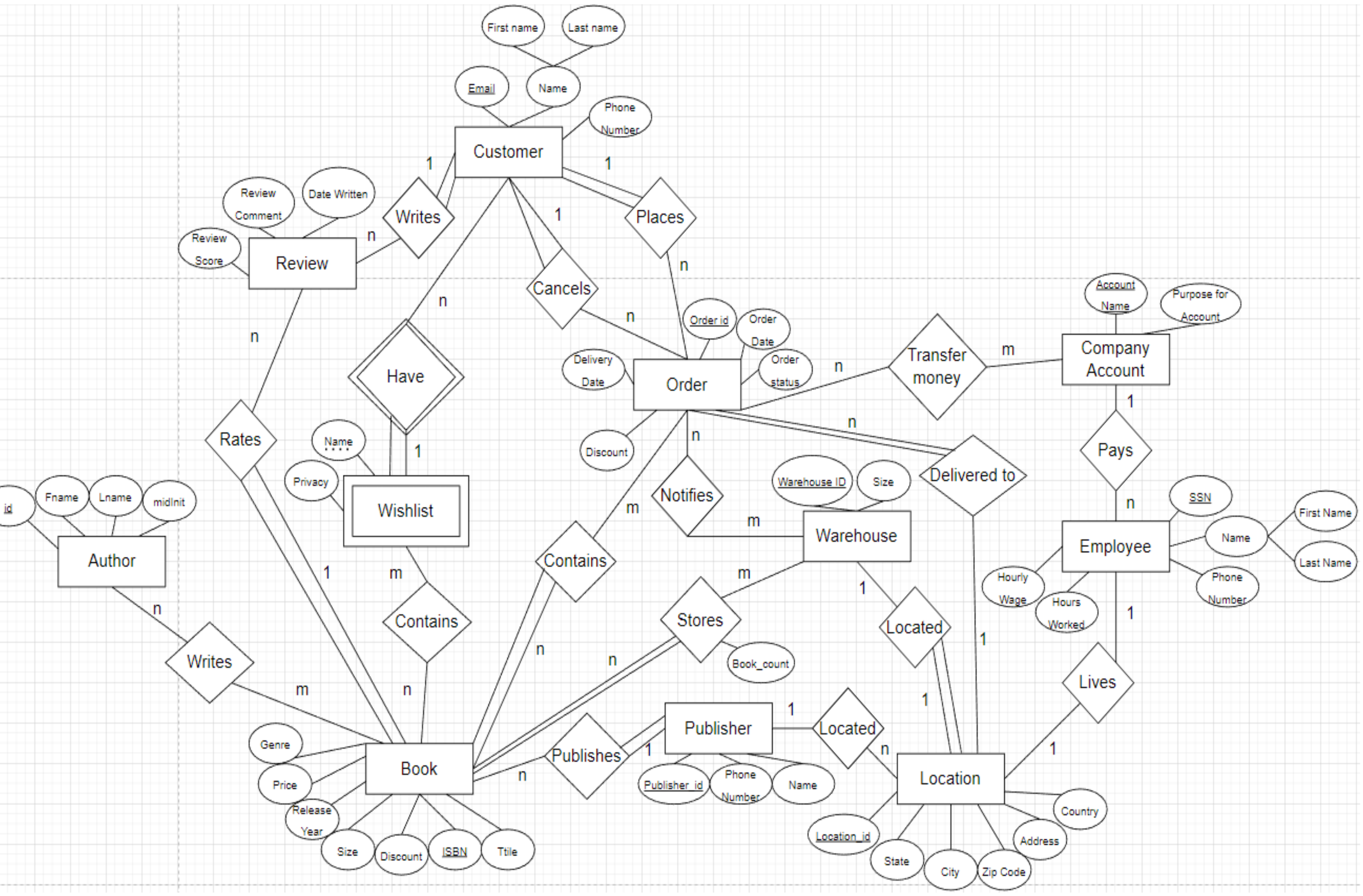
Names: Faisal Yaeish and Will Blanton
Date: April 20, 2023

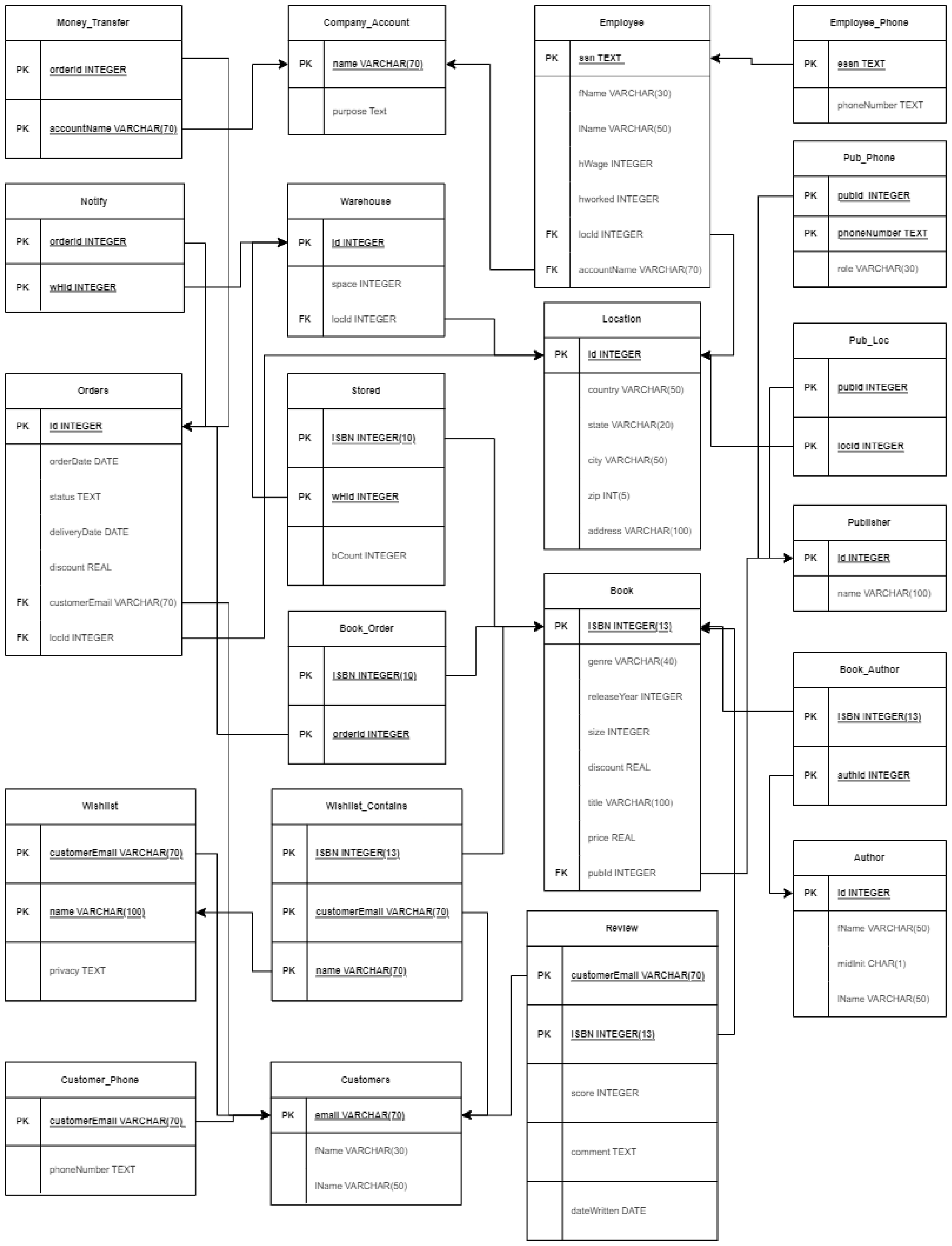
Part I – The Final Report

Section 1 - Database Description

ER-model



Relational Schema



Normalization

- **Author**
    - 3NF
        - Author is in 3NF since all domain values are atomic (1NF), every non-key attribute is fully dependent on the key (2NF), and every non-key attribute ($fname,\ midInit,\ lName$) is non-transitively dependent on the key (id).
    - Functional Dependencies:
        - $\{id\} \to \{fname,\ midInit,\ lName\}$
- **Book**
    - 3NF
        - Book is in 3NF since all domain values are atomic (1NF), every non-key attribute is fully dependent on the key (2NF), and every non-key attribute ($Title,\ Genre,\ Price,\ Release\ year,\ Size,\ Discount,\ PubId$) is non-transitively dependent on the key ($ISBN$).
    - Functional Dependencies:
        - $\{ISBN\} \to \{Title,\ Genre,\ Price,\ Release\ year,\ Size,\ Discount,\ PubId\}$
- **Book_Author**
    - 3NF
        - There are no non-key attributes in Book_Author, so no non-key attributes are functionally dependent on the primary key. Therefore, the condition for 3NF is already satisfied.
    - Functional Dependencies:
        - Trivial dependency (no non-key attributes)
- **Book_Order**
    - 3NF
        - There are no non-key attributes in Book_Order, so no non-key attributes are functionally dependent on the primary key. Therefore, the condition for 3NF is already satisfied.
    - Functional Dependencies:
        - Trivial dependency (no non-key attributes)
- **Company_Account**
    - 3NF
        - Company_Account is in 3NF since all domain values are atomic (1NF), every non-key attribute is fully dependent on the key (2NF), and every non-key attribute ($Purpose$) is non-transitively dependent on the key ($Name$).
    - Functional Dependencies:
        - $\{Name\} \to \{Purpose\}$
- **Customer**

- 3NF
  - Customer is in 3NF since all domain values are atomic (1NF), every non-key attribute is fully dependent on the key (2NF), and every non-key attribute ($Fname, Lname$) is non-transitively dependent on the key ($Email$).
- Functional Dependencies:
  - $\{Email\} \rightarrow \{Fname, Lname\}$

- **Customer_Phone**
  - 3NF
    - Customer_Phone is in 3NF since all domain values are atomic (1NF), every non-key attribute is fully dependent on the key (2NF), and every non-key attribute ($PhoneNumber$) is non-transitively dependent on the key ($CustomerEmail$).
  - Functional Dependencies:
    - $\{\text{Customer}Email\} \rightarrow \{PhoneNumber\}$
- **Employee**
  - 3NF
    - Employee is in 3NF since all domain values are atomic (1NF), every non-key attribute is fully dependent on the key (2NF), and every non-key attribute ($Fname, Lname, hWage, hWorked, LocId, aNum$) is non-transitively dependent on the key ($SSNl$).
  - Functional Dependencies:
    - $\{SSN\} \rightarrow \{Fname, Lname, hWage, hWorked, LocId, aNum\}$
- **Employee_Phone**
  - 3NF
    - Employee_Phone is in 3NF since all domain values are atomic (1NF), every non-key attribute is fully dependent on the key (2NF), and every non-key attribute ($PhoneNumber$) is non-transitively dependent on the key (SSN).
  - Functional Dependencies:
    - $\{SSN\} \rightarrow \{Phone\ Number\}$
- **Location**
  - 3NF
    - Location is in 3NF since all domain values are atomic (1NF), every non-key attribute is fully dependent on the key (2NF), and every non-key attribute ($Country, State, City, Zip, Address$) is non-transitively dependent on the key ($LocationId$).
  - Functional Dependencies:
    - $\{LocationId\} \rightarrow \{Country, State, City, Zip, Address\}$
- **Money_Transfer**

    - 3NF
        - There are no non-key attributes in Money_Transfer, so no non-key attributes are functionally dependent on the primary key. Therefore, the condition for 3NF is already satisfied.
    - Functional Dependencies:
        - Trivial dependency (no non-key attributes)
- **Notify**
    - 3NF
        - There are no non-key attributes in Notify, so no non-key attributes are functionally dependent on the primary key. Therefore, the condition for 3NF is already satisfied.
    - Functional Dependencies:
        - Trivial dependency (no non-key attributes)
- **Order**
    - 3NF
        - Order is in 3NF since all domain values are atomic (1NF), every non-key attribute is fully dependent on the key (2NF), and every non-key attribute ($OrderDate, Status, DeliveryDate, Discount, CustomerEmail, LocId$) is non-transitively dependent on the key ($OrderId$).
    - Functional Dependencies:
        - $\{OrderId\} \rightarrow \{OrderDate, Status, DeliveryDate, Discount, CustomerEmail, LocId\}$
- **Pub_Loc**
    - 3NF
        - There are no non-key attributes in Pub_loc, so no non-key attributes are functionally dependent on the primary key. Therefore, the condition for 3NF is already satisfied.
    - Functional Dependencies:
        - Trivial dependency (no non-key attributes)
- **Pub_Phone**
    - 3NF
        - Pub_Phone is in 3NF since all domain values are atomic (1NF), every non-key attribute is fully dependent on the key (2NF), and every non-key attribute ($Role$) is non-transitively dependent on the key ($PubId, PhoneNumber$).
    - Functional Dependencies:
        - $\{PubId, PhoneNumber\}\rightarrow\{Role\}$
- **Publisher**
    - 3NF

        - Publisher is in 3NF since all domain values are atomic (1NF), every non-key attribute is fully dependent on the key (2NF), and every non-key attribute ($PhoneNumber,\ Name$) is non-transitively dependent on the key ($PublisherId$).
    - Functional Dependencies:
        - $\{PublisherId\} \rightarrow \{PhoneNumber,\ Name\}$
- **Review**
    - 3NF
        - Review is in 3NF since all domain values are atomic (1NF), every non-key attribute is fully dependent on the key (2NF), and every non-key attribute ($Score,\ Comment,\ Date$) is non-transitively dependent on the key ($CustomerEmail,\ ISBN$).
    - Functional Dependencies:
        - $\{CustomerEmail,\ ISBN\} \rightarrow \{Score,\ Comment,\ Date\}$
- **Stored**
    - 3NF
        - Stored is in 3NF since all domain values are atomic (1NF), every non-key attribute is fully dependent on the key (2NF), and every non-key attribute ($bCount$) is non-transitively dependent on the key ($ISBN,\ wHId$).
    - Functional Dependencies:
        - $\{ISBN,\ wHId\} \rightarrow \{bCount\}$
- **Warehouse**
    - 3NF
        - Warehouse is in 3NF since all domain values are atomic (1NF), every non-key attribute is fully dependent on the key (2NF), and every non-key attribute ($Space,\ LocId$) is non-transitively dependent on the key ($WarehouseId$).
    - Functional Dependencies:
        - $\{WarehouseId\} \rightarrow \{Space,\ LocId\}$
- **Wishlist**
    - 3NF
        - Wishlist is in 3NF since all domain values are atomic (1NF), every non-key attribute is fully dependent on the key (2NF), and every non-key attribute ($Privacy$) is non-transitively dependent on the key ($CustomerId,\ Name$).
    - Functional Dependencies:
        - $\{CustomerId,\ Name\} \rightarrow \{Privacy\}$
- **Wishlist_Contains**
    - 3NF

        - There are no non-key attributes in Wishlist_Contains, so no non-key attributes are functionally dependent on the primary key. Therefore, the condition for 3NF is already satisfied.
    - Functional Dependencies:
        - Trivial dependency (no non-key attributes)

Indexes

1. **ISBN Index -** The ISBN is a unique identifier for books. An index on the ISBN column can help to quickly find a book's details when the ISBN is known.

   ```
   CREATE INDEX idx_ISBN ON Book (ISBN)
   ```

   Rationale:
   The ISBN is a unique identifier for books. It is an essential piece of information required to identify a book uniquely. If you have a large database of books, finding a specific book's details based on its ISBN can be time-consuming, especially if the database is not indexed. Creating an index on the ISBN column will help to quickly locate a book's details when the ISBN is known.

2. Order Date Index: An index on the Order Date column can help to quickly locate orders placed on a specific date or within a specific date range.

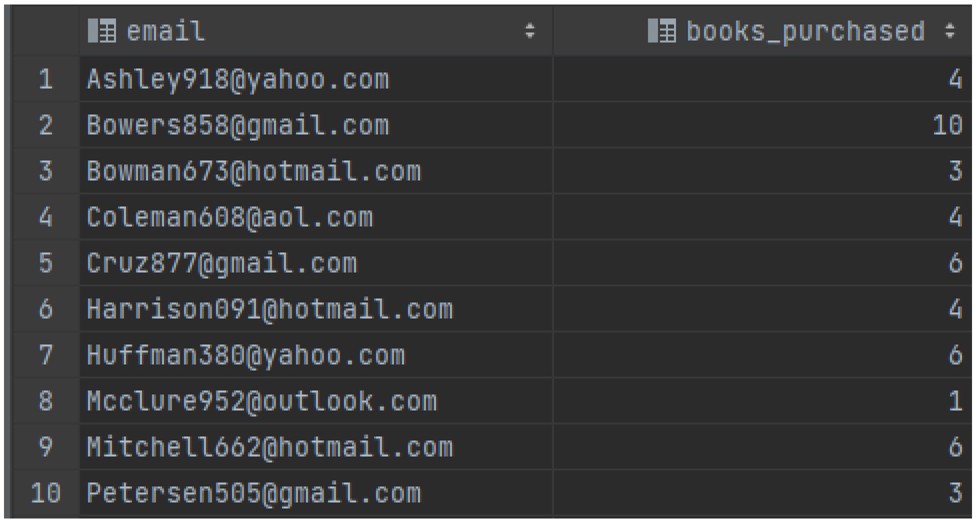
   ```
   CREATE INDEX idx_orderDate ON “Order” (orderDate);
   ```

   Rationale:
   If a query involves filtering or sorting by the Order Date column, an index can help to quickly locate the relevant rows, rather than scanning the entire table. This can result in faster query execution times and improved overall database performance.

Views

Finds the total number of books purchased by customers:

```
CREATE VIEW total_books_purchased_by_customer AS
SELECT email, COUNT(ISBN) AS books_purchased
FROM Customer, "Order", Book_order
WHERE Customer.email = "Order".customerEmail AND "Order".id = Book_order.orderId
GROUP BY Customer.email
```

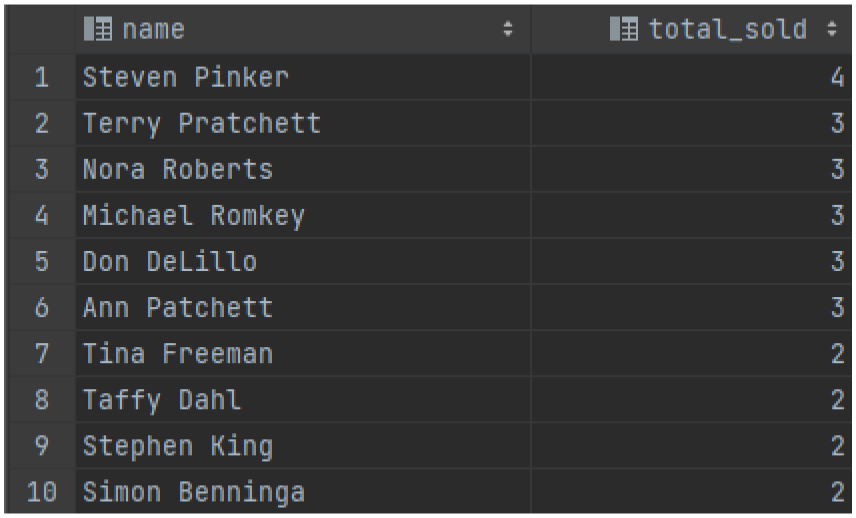


	email	books_purchased
1	Ashley918@yahoo.com	4
2	Bowers858@gmail.com	10
3	Bowman673@hotmail.com	3
4	Coleman608@aol.com	4
5	Cruz877@gmail.com	6
6	Harrison091@hotmail.com	4
7	Huffman380@yahoo.com	6
8	Mcclure952@outlook.com	1
9	Mitchell662@hotmail.com	6
10	Petersen505@gmail.com	3

This view can be useful because it allows us to see which customers buy a lot from our bookstore and possibly reward them for it.

Finds the total number of books sold by authors:

CREATE VIEW author_book_sales AS
SELECT fname, Count(Book.ISBN) AS total_sold
FROM Book, Author, Book_Author, Book_Order
WHERE Author.id = Book_Author.authId AND Book.ISBN = Book_Author.ISBN AND Book.ISBN = Book_Order.ISBN
GROUP BY fname
ORDER BY COUNT(Book.ISBN) DESC

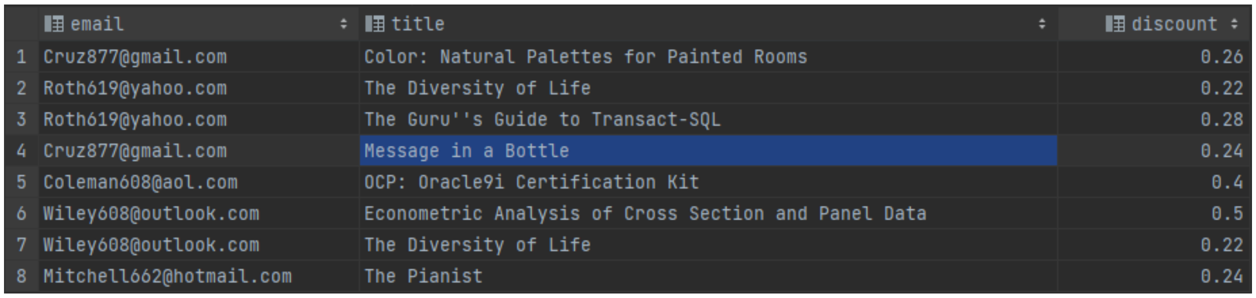


	name	total_sold
1	Steven Pinker	4
2	Terry Pratchett	3
3	Nora Roberts	3
4	Michael Romkey	3
5	Don DeLillo	3
6	Ann Patchett	3
7	Tina Freeman	2
8	Taffy Dahl	2
9	Stephen King	2
10	Simon Benninga	2

This view can be useful because it allows us to see which author is selling the most books.

Finds the books that are discounted on a customer's wishlist

CREATE VIEW discounted_books_on_wishlist AS
SELECT DISTINCT email, title, Book.discount
FROM Book, Wishlist_Contains, Book_Order, "Order", Customer
WHERE Book.discount > 0 AND Book.ISBN = Wishlist_Contains.ISBN AND Book.ISBN = Book_Order.ISBN AND Book_Order.orderId = "Order".id AND "Order".customerEmail = Customer.email



	email	title	discount
1	Cruz877@gmail.com	Color: Natural Palettes for Painted Rooms	0.26
2	Roth619@yahoo.com	The Diversity of Life	0.22
3	Roth619@yahoo.com	The Guru''s Guide to Transact-SQL	0.28
4	Cruz877@gmail.com	Message in a Bottle	0.24
5	Coleman608@aol.com	OCP: Oracle9i Certification Kit	0.4
6	Wiley608@outlook.com	Econometric Analysis of Cross Section and Panel Data	0.5
7	Wiley608@outlook.com	The Diversity of Life	0.22
8	Mitchell662@hotmail.com	The Pianist	0.24

This view produces a list of books on a customer's wishlist that are currently on sale or have a discount. It can be useful for notifying customers of sales and discounts on items they are interested in.

Transactions

a. Add a new book

INSERT INTO Book (ISBN, genre, releaseYear, size, discount, title price pubId)
VALUES (`0140447920, 'Classic Literature', 1869, 50, 0, 'The Idiot', 15.99, 100`)

b. Update order status

UPDATE Order
SET status = 'shipped'
WHERE id = 1234

c. Update Book

UPDATE Book
SET price = 24.99
WHERE ISBN = '1234567890'

Section 2 - User Manual

Table Descriptions

- **AUTHOR:**
  - Represents an author.
  - **Attributes:**
    - id: Unique number used to identify author.
      - INTEGER
    - fName: First name of the customer.
      - VARCHAR(50)
      - NOT NULL
    - midInit: Middle initial of the author.
      - CHAR(1)
    - lName: Last name of the author.
      - VARCHAR(50)
      - NOT NULL
  - PRIMARY KEY: (id)

- **BOOK**:
  - Represents a book.
  - **Attributes:**
    - ISBN: A unique number used to identify the book.
      - INTEGER(13)
    - genre: Genre of the book.
      - VARCHAR(40)
      - NOT NULL
    - releaseYear: Release year of the book.
      - INTEGER
      - NOT NULL
      - CHECK releaseYear <= 2023
    - size: Area of the book (cm^2).
      - INTEGER
      - NOT NULL
      - CHECK (size >= 0)
    - discount: Discount on book's price.
      - REAL
      - NOT NULL

            - DEFAULT 0
            - CHECK (0 <= discount < 1)
        - title: Title of the book.
            - VARCHAR(150)
            - NOT NULL
        - price: Price of the book (USD).
            - REAL
            - NOT NULL
            - CHECK (price > 0)
        - pubId: A unique number used to identify the book’s publisher.
            - INTEGER
            - NOT NULL
    - PRIMARY KEY: (ISBN)
    - FOREIGN KEYs:
        - pubId REFERENCES PUBLISHER (id)
            - ON UPDATE CASCADE

- **BOOK_AUTHOR**:
    - Represents a book's author.
    - One book can have multiple authors.
    - **Attributes:**
        - ISBN: ISBN of the book.
            - INTEGER(13)
        - authId: Book’s author.
            - INTEGER
    - PRIMARY KEY: (ISBN, authId)
    - FOREIGN KEYs:
        - ISBN references BOOK (ISBN)
            - ON DELETE CASCADE
            - ON UPDATE CASCADE
        - authID references AUTHOR(id)
            - ON DELETE CASCADE
            - ON UPDATE CASCADE

- **BOOK_ORDER**
    - Represents a book in an order.
    - **Attributes:**
        - ISBN: The ISBN of the book.
            - INTEGER(13)
            - NOT NULL

        - orderId: The ID of the order.
            - INTEGER
            - NOT NULL
    - PRIMARY KEY (ISBN, orderId)
    - FOREIGN KEYs:
        - ISBN REFERENCES BOOK (ISBN)
            - ON UPDATE CASCADE
            - ON DELETE CASCADE
        - orderId REFERENCES ORDER (id)
            - ON UPDATE CASCADE
            - ON DELETE CASCADE

- **COMPANY_ACCOUNT**:
    - Represents one of the bookstore's bank accounts.
    - **Attributes:**
        - name: Name of the account.
            - VARCHAR(70)
        - purpose: Purpose of the account.
            - TEXT
            - NOT NULL
    - PRIMARY KEY(name)

- **CUSTOMER:**
    - Represents a customer's account information.
    - **Attributes:**
        - email: Email of the customer.
            - VARCHAR(70)
        - fName: First name of the customer.
            - VARCHAR(30)
            - NOT NULL
        - lName: Last name of the customer.
            - VARCHAR(50)
            - NOT NULL
    - PRIMARY KEY: (email)

- **CUSTOMER_PHONE:**
    - Represents a customer's phone number.
    - **Attributes:**
        - customerEmail: Email of the customer.
            - VARCHAR(70)

        - phoneNumber: Customer’s phone number.
            - TEXT
            - NOT NULL
            - CHECK LIKE '(___) ___-____') to ensure that it follows the format of a US phone number.
    - PRIMARY KEY: (customerEmail)
    - FOREIGN KEYs:
        - customerEmail REFERENCES CUSTOMER (email)
            - ON DELETE CASCADE
            - ON UPDATE CASCADE

- **EMPLOYEE**
    - Represents an employee of the company.
    - **Attributes:**
        - ssn: The social security number of the employee.
            - TEXT
            - CHECK (ssn Like '___-__-____') to ensure that it follows the format of a US SSN.
        - fName: The first name of the employee.
            - VARCHAR(30)
            - NOT NULL.
        - lName: The last name of the employee.
            - VARCHAR(50)
            - NOT NULL
        - hWage: The hourly wage of the employee.
            - INTEGER
            - NOT NULL
            - CHECK (hWage >= 0)
        - hWorked: The hours worked by the employee.
            - INTEGER
            - NOT NULL
            - CHECK (hWorked >= 0)
        - locId: The ID of the location where the employee lives.
            - INTEGER.
            - NOT NULL
        - accountName: The name of the company account that the employee is paid with.
            - VARCHAR(70)
            - NOT NULL
    - PRIMARY KEY: (ssn)

- FOREIGN KEYs:
  - locId REFERENCES Location(id)
    - ON UPDATE CASCADE
  - accountName REFERENCES COMPANY_ACCOUNT (name)
    - ON UPDATE CASCADE

- **EMPLOYEE_PHONE**
  - Represents the phone number of an employee.
  - **Attributes:**
    - essn: The social security number of the employee.
      - TEXT
      - NOT NULL
      - CHECK (essn Like '___-__-____') to ensure that it follows the format of a US SSN
    - phoneNumber: The phone number of the employee.
      - TEXT
      - NOT NULL
      - CHECK (phoneNumber LIKE '(___) ___-____') to ensure that it follows the format of a US phone number.
  - PRIMARY KEY: (essn)
  - FOREIGN KEYs:
    - essn REFERENCES EMPLOYEE (essn)
      - ON UPDATE CASCADE
      - ON DELETE CASCADE

- **LOCATION**
  - Represents a physical location.
  - **Attributes**
    - Id: A unique number used to identify location tuple.
      - INTEGER
    - country: The country that the location is in
      - VARCHAR(50)
      - NOT NULL
    - state: The state that the location is in
      - VARCHAR(20)
      - Only applicable in US
    - city: The city that the location is in
      - VARCHAR(50)
      - NOT NULL
    - zip: The zip code that the location is in.

        - INTEGER(5)
        - NOT NULL
      - address: The location’s address
        - VARCHAR(100)
        - NOT NULL
  - PRIMARY KEY: (id)

- **MONEY_TRANSFER**
  - Represents the transfer of money received from an order into a company account.
  - **Attributes:**
    - orderId:The ID of the order related to the money transfer.
      - INTEGER
      - NOT NULL
    - accountName: The name of the company account receiving the money transfer.
      - VARCHAR(70)
      - NOT NULL
  - PRIMARY KEY (orderId, accountName)
  - FOREIGN KEYs:
    - orderId REFERENCES Order (id)
      - ON UPDATE CASCADE
      - ON DELETE CASCADE
    - accountName REFERENCES COMPANY_ACCOUNT (name)
      - ON UPDATE CASCADE
      - ON DELETE CASCADE

- **NOTIFY**
  - Represents the notifications warehouses receive when an order requires their inventory.
  - **Attributes:**
    - orderId: An integer representing the ID of the order related to the notification.
      - INTEGER
      - NOT NULL
    - wHId: An integer representing the ID of the warehouse where that was notified.
      - INTEGER
      - NOT NULL
  - PRIMARY KEY (orderId, wHId)
  - FOREIGN KEYs:

        - orderId REFERENCES Order (id)
            - ON UPDATE CASCADE
            - ON DELETE CASCADE
        - wHId REFERENCES WAREHOUSE (id)
            - ON UPDATE CASCADE
            - ON DELETE CASCADE

- **"ORDER"**
    - Represents a customer’s order from the bookstore..
    - **Attributes:**
        - id: The unique ID of the order.
            - INTEGER
        - orderDate:Date when the order was placed.
            - DATE
            - NOT NULL
        - status: Status of the order.
            - TEXT
            - NOT NULL
            - CHECK (status IN ('Delivered', 'In Progress', 'Canceled'))
        - deliveryDate: Date when the order was delivered
            - DATE.
        - discount: Total discount applied to the order.
            - REAL
            - NOT NULL
            - DEFAULT 0
            - CHECK (discount >= 0 AND discount < 1)
        - customerEmail: Email of the customer who placed the order.
            - VARCHAR(70)
            - NOT NULL
        - locId: Location ID where the order was placed
            - INTEGER
            - NOT NULL
    - PRIMARY KEY (id)
    - FOREIGN KEYs:
        - customerEmail REFERENCES CUSTOMER (email)
            - ON DELETE SET NULL
            - ON UPDATE CASCADE
        - locId REFERENCES LOCATION (id)
            - ON DELETE SET NULL
            - ON UPDATE CASCADE

    - CONSTRAINTS
        - DATECHK CHECK (orderDate <= deliveryDate)
            - Ensure that the orderDate is before the deliveryDate.

- **PUB_LOC**
    - Represents the a publisher’s physical location.
    - **Attributes:**
        - pubId: The ID of the publisher.
            - INTEGER
            - NOT NULL
        - locId: The ID of the location.
            - INTEGER
            - NOT NULL
    - PRIMARY KEY (pubId, locId)
    - FOREIGN KEYs:
        - pubId REFERENCES PUBLISHER (id)
            - ON UPDATE CASCADE
            - ON DELETE CASCADE
        - locId REFERENCES LOCATION (id)
            - ON UPDATE CASCADE
            - ON DELETE CASCADE

- **PUB_PHONE:**
    - Represents a publisher's phone number.
    - **Attributes:**
        - pubId: A unique number used to identify a publisher.
            - INTEGER
        - phoneNumber: Publisher’s phone number.
            - TEXT
            - CHECK LIKE '(___) ___-____') to ensure that it follows the format of a US phone number.
        - role: Role of the phone number (e.g., sales, billing).
            - VARCHAR(30)
            - NOT NULL
    - PRIMARY KEY: (pubId, phoneNumber)
    - FOREIGN KEYs:
        - pubId REFERENCES PUBLISHER (id)
            - ON DELETE CASCADE
            - ON UPDATE CASCADE

- **PUBLISHER:**
    - Represents a book publisher.
    - **Attributes:**
        - id: A unique number used to identify the publisher.
            - INTEGER
        - fName: The first name of the publisher.
            - VARCHAR(30)
            - NOT NULL.
        - lName: The last name of the publisher.
            - VARCHAR(50)
            - NOT NULL
    - PRIMARY KEY: (id)

- **REVIEW**
    - Represents a review written by a customer for a book.
    - **Attributes:**
        - customerEmail: Email of the customer who wrote the review.
            - VARCHAR(70)
            - NOT NULL
        - ISBN:ISBN of the book for which the review is written.
            - INTEGER(13)
            - NOT NULL
        - score: Customer's rating of the book, which should be between 0 and 10.
            - INTEGER
            - NOT NULL
            - CHECK (score >= 0 AND score <= 10)
        - comment: Comment written by the customer for the book.
            - TEXT
        - dateWritten: Date when the review was written.
            - DATE
            - NOT NULL
    - PRIMARY KEY: (customerEmail, ISBN)
    - FOREIGN KEYs:
        - customerEmail REFERENCES CUSTOMER (email)
            - ON UPDATE CASCADE
        - ISBN REFERENCES BOOK (ISBN)
            - ON DELETE CASCADE
            - ON UPDATE CASCADE

- **STORED**

- Represents the inventory of a book in a warehouse.
- **Attributes:**
  - ISBN: The ISBN of the book being stored.
    - INTEGER(13)
    - NOT NULL
  - wHId: The ID of the warehouse where the book is stored.
    - INTEGER
    - NOT NULL
  - bCount: The number of books of the specified ISBN that are stored in the warehouse.
    - INTEGER
    - NOT NULL
    - DEFAULT 0
    - CHECK (bCount >= 0)
- PRIMARY KEY (ISBN, wHId)
- FOREIGN KEYs:
  - ISBN REFERENCES BOOK (ISBN)
    - ON UPDATE CASCADE
    - ON DELETE CASCADE
  - wHId REFERENCES WAREHOUSE (id)
    - ON UPDATE CASCADE
    - ON DELETE CASCADE

- **WAREHOUSE**
  - Represents a warehouse where the company stores its inventory.
  - **Attributes:**
    - id: A unique number used to identify a warehouse.
      - INTEGER
    - space: The available space for books in the warehouse (m^2).
      - INTEGER
      - NOT NULL
      - CHECK (space > 0)
    - locId: The ID of the location where the warehouse is situated.
      - INTEGER
      - NOT NULL
  - PRIMARY KEY: (id)
  - FOREIGN KEYs:
    - locId REFERENCES Location (id)

- **WISHLIST**

    - Represents a customer’s book wishlists.
    - A customer can have multiple wishlists with different names.
    - **Attributes**:
        - customerEmail: Email of the customer who created the wishlist
            - VARCHAR(70)
            - NOT NULL
        - name: Name of the wishlist
            - VARCHAR(70)
            - NOT NULL
        - privacy: Privacy setting of the wishlist, which can be 'Private' or 'Public'.
            - TEXT
            - NOT NULL
            - CHECK (privacy IN ('Private', 'Public'))
        - PRIMARY KEY: (customerEmail, name)
        - FOREIGN KEYs:
            - customerEmail REFERENCES CUSTOMER (email)
                - ON DELETE CASCADE
                - ON UPDATE CASCADE

- **WISHLIST_CONTAINS**
    - Represents a book that is in a customer's wishlist.
    - **Attributes:**
        - ISBN: The ISBN of the book in the wishlist.
            - INTEGER(13)
            - NOT NULL
        - customerEmail: The email of the customer who owns the wishlist.
            - VARCHAR(70)
            - NOT NULL
        - name: The name of the wishlist to which the book is on.
            - VARCHAR(70)
            - NOT NULL
    - PRIMARY KEY: (ISBN, customerEmail, name)
    - FOREIGN KEYs:
        - ISBN REFERENCES BOOK (ISBN)
            - ON UPDATE CASCADE
            - ON DELETE CASCADE
        - customerEmail REFERENCES CUSTOMER (email)
            - ON UPDATE CASCADE
            - ON DELETE CASCADE
        - name REFERENCES WISHLIST (name)

- ON UPDATE CASCADE
- ON DELETE CASCADE

SQL Queries

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

$Author \leftarrow \sigma_{fname='Pratchett'}(Author \bowtie_{id\,=\,authId} Book\_Author)$

$cheapBook \leftarrow \sigma_{\ Price\,<\,10}(Book)$

$result \leftarrow \pi_{title}(cheapBook \bowtie_{ISBN=ISBN} Author)$

SELECT Title
FROM Book, Author, Book_Author
WHERE fname = 'Pratchett' AND price < 10 AND Book.ISBN = Book_Author.ISBN AND Book_Author.authId= Author.id

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

$Single\_Customer \leftarrow \sigma_{\ email\,=\,'Bowers858@gmail.com'}(Customer)$

$Custmor\_Orders \leftarrow Single\_Customer \bowtie_{email=customerEmail} Order$

$Books\_in\_Orders \leftarrow Customer\_Orders \bowtie_{id\,=\,orderId} Book\_Order$

$result \leftarrow \pi_{Title,\,orderDate}(Books\_in\_Orders \bowtie_{ISBN\,=\,ISBN} Book)$

SELECT Title, orderDate
FROM Customer, Book, Book_Order, "Order"
WHERE email = 'Bowers858@gmail.com' AND Customer.email = "Order".customerEmail AND "Order".id = Book_Order.orderId AND Book_Order.ISBN = Book.ISBN

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

$\pi_{\ Title,\,ISBN}(\sigma_{\ BCount\,<\,5}(Stored \bowtie_{ISBN\,=\,ISBN} Book))$

SELECT Title, Book.ISBN
FROM Book, Stored
WHERE bCount < 5 AND Book.ISBN = Stored.ISBN

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

$PRATCHETT \leftarrow \sigma_{Author.fname='Pratchett'}(Book\_Author \bowtie_{authId=id} Author)$
$PRATCHETT\_Book \leftarrow PRATCHETT \bowtie_{ISBN=ISBN} Book$
$PRATCHETT\_Order \leftarrow Order \bowtie_{id=orderId} (Book\_Order \bowtie_{ISBN=ISBN} PRATCHETT\_Book)$
$Result \leftarrow \pi_{email,\, title} (Customer \bowtie_{email=customerEmail} PRATCHETT\_Order)$

SELECT email, Title
FROM "Order", Customer, Book, Book_Author, Book_Order, Author
WHERE Author.fname = 'Pratchett' AND Customer.email = "Order".customerEmail AND "Order".id = Book_Order.orderId AND Book_Order.ISBN = Book.ISBN AND Book.ISBN = Book_Author.ISBN AND Book_Author.authId = Author.id

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

$Single\_Customer \leftarrow \sigma_{email = 'Bowers858@gmail.com'} (Customer)$
$Customer\_Orders \leftarrow Single\_Customer \bowtie_{email = customerEmail} Order$
$Books\_in\_Orders \leftarrow Customer\_Orders \bowtie_{id = orderId} Book\_Order$
$result \leftarrow A_{COUNT\ ISBN} (Books\_in\_Orders)$

SELECT COUNT(ISBN) AS total_books_purchased
FROM Customer, "Order", Book_Order
WHERE email = 'Bowers858@gmail.com' AND Customer.email = "Order".customerEmail AND "Order".id = Book_order.orderId

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

$Customer\_Orders \leftarrow Customer \bowtie_{email = customerEmail} Order$
$Books\_in\_Orders \leftarrow Customer\_Orders \bowtie_{id = orderId} Book\_Order$
$Number\_of\_Purchases \leftarrow {}_{email} A_{COUNT\ ISBN}(Book\_in\_Orders)$

```
SELECT Customer.email, COUNT(Book_order.ISBN) AS total_books_purchased
FROM Customer, "Order", Book_order
WHERE Customer.email = "Order".customerEmail AND "Order".id = Book_order.orderId
GROUP BY Customer.email
ORDER BY total_books_purchased DESC
LIMIT 1
```

g. Give the average hours worked of all the employees and their location id

$Employee\_Location \leftarrow Employee \bowtie_{Location\ id\ =\ Location\ id} Location$

$result \leftarrow \ _{Location\ id}A_{AVERAGE\ Hworked}(Employee\_Location)$

```
SELECT AVG(Hworked) AS avg_hours_worked
FROM Employee, Location
WHERE Employee.locId = Location.id
GROUP BY Location.id
```

h. Give all the customer_ids and names of customers who wrote a review with a score that is >= 4

$High\_rating \leftarrow \sigma_{Review\ Score\ >=\ 4}(Review)$

$Customer\_Good\_Review \leftarrow Customer \bowtie_{email\ =\ email} High\_rating$

$result \leftarrow \pi_{email,\ Fname,\ Lname} Customer\_Good\_Review$

```
SELECT email, Fname, Lname
FROM Review, Customer
WHERE Review.score >= 4 AND Review.customerEmail = Customer.email
```

i. Give all the emails and the name of their wishlist who have a certain book in their wishlist

$Specific\_Book \leftarrow \sigma_{Title\ =\ "Google\ Hacks"}(Book)$

$Customer \leftarrow Wishlist\_Contains \bowtie_{customerEmail\ =\ email} Customer$

$Wishlist\_Has\_Book \leftarrow Wishlist\_Contains \bowtie_{ISBN\ =\ ISBN} Specific\_Book$

$result \leftarrow \pi_{email,\ name} Wishlist\_Has\_Book$

```
SELECT email, name
FROM Customer, Book, Wishlist_Contains
```

```
WHERE Title = 'Google Hacks' AND Customer.email = Wishlist_Contains.customerEmail AND
Wishlist_Contains.ISBN = Book.ISBN
```

More SQL

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

$\pi_{fname,\ lname,\ SUM\ (\text{price})\ \rightarrow\ total\_\text{spent}}$
$\gamma_{email,\ SUM\ (\text{price})}$
$\sigma_{customer\ .\ email\ =\ order\ .\ customeremail\ AND\ order\ .\ id\ =\ book\_\text{order}\ .\ orderid\ AND\ book\_\text{order}\ .\ isbn\ =\ book\ .\ isbn}$ (customer × *book* × *order* × *book*_order)

```
SELECT Fname, Lname, SUM(Price) AS total_spent
FROM Customer, Book, "Order", Book_Order
WHERE Customer.email = "Order".customerEmail AND "Order".id = Book_Order.orderId AND
Book_Order.ISBN = Book.ISBN
GROUP BY email
```

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

π Fname, Lname, email (σ email ∈ (π email (ρ t1 (σ SUM(Price) > (π AVG(total_spent) (ρ t2 (γ email; AVG(total_spent); ρ t3 (σ email = customerEmail (Customer) ⋈ "Order".id = orderId (Book_Order ⋈ Book.ISBN = Book_Order.ISBN (Book)))))))) ⋈ email = customerEmail (Customer))

```
SELECT Fname, Lname, email
FROM Customer, Book, "Order", Book_Order
WHERE Customer.email = "Order".customerEmail AND "Order".id = Book_Order.orderId AND
Book_Order.ISBN = Book.ISBN
GROUP BY email
HAVING SUM(Price) > (
  SELECT AVG(total_spent)
  FROM (
     SELECT SUM(Price) AS total_spent
     FROM Customer, Book, "Order", Book_Order
     WHERE Customer.email = "Order".customerEmail AND "Order".id = Book_Order.orderId
AND Book_Order.ISBN = Book.ISBN
     GROUP BY email
  )
)
```

l. 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.

$\tau_{COUNT\ (*)\ \downarrow}$
$\gamma_{title,\ COUNT\ (*)}$
$\sigma_{book\ .\ isbn\ =\ book\_order\ .\ isbn}\ (\text{book} \times \textit{book}\_\text{order})$

```
SELECT Title, COUNT(*)
FROM Book, Book_Order
WHERE Book.ISBN = Book_Order.ISBN
GROUP BY Title
ORDER BY COUNT(*) DESC
```

m. 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.

$\tau_{COUNT\ (*)\ \downarrow}$
$\pi_{title,\ COUNT\ (*)\ *\ price}$
$\gamma_{title,}$
$\sigma_{book\ .\ isbn\ =\ book\_order\ .\ isbn}\ (\text{book} \times \textit{book}\_\text{order})$

```
SELECT Title, COUNT(*) * Price
FROM Book, Book_Order
WHERE Book.ISBN = Book_Order.ISBN
GROUP BY Title
ORDER BY COUNT(*) DESC
```

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

$\tau_{COUNT\ (*)\ \downarrow}$
$\pi_{name,\ COUNT\ (*)\ \rightarrow\ number\_of\_books\_sold}$
$\gamma_{name,\ COUNT\ (*)}$
$\sigma_{author\ .\ id\ =\ book\_author\ .\ authorid\ AND\ book\ .\ isbn\ =\ book\_author\ .\ isbn\ AND\ book\ .\ isbn\ =\ book\_order\ .\ isbn}\ (\text{book} \times \textit{author} \times \textit{book}\_\text{author} \times \textit{book}\_\text{order})$

```
SELECT name, COUNT(*) AS number_of_books_sold
FROM Book, Author, Book_Author, Book_Order
WHERE Author.id = Book_Author.authorId AND Book.ISBN = Book_Author.ISBN AND
Book.ISBN = Book_Order.ISBN
```

```
GROUP BY Name
ORDER BY COUNT(*) DESC
LIMIT 1
```

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

$\tau_{revenue \downarrow}$

$\pi_{name,\ SUM\ (\text{price}) \rightarrow revenue}$

$\gamma_{name,\ SUM\ (\text{price})}$

$\sigma_{author\,.\,id\,=\,book\_\text{author}\,.\,authorid\ AND\ book\,.\,isbn\,=\,book\_\text{author}\,.\,isbn\ AND\ book\,.\,isbn\,=\,book\_\text{order}\,.\,isbn}$ (book × *author* × *book*_author × *book*_order)

```
SELECT name, SUM(Price) AS revenue
FROM Book, Author, Book_Author, Book_Order
WHERE Author.id = Book_Author.authorId AND Book.ISBN = Book_Author.ISBN AND
Book.ISBN = Book_Order.ISBN
GROUP BY name
ORDER BY revenue DESC
LIMIT 1
```

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

π Fname, Lname, email (σ name ∈ (π name (γ name; SUM(Price) DESC; Book_Order ⋈ Book.ISBN = Book_Order.ISBN (Book_Author ⋈ Author.id = Book_Author.authorId (Author ⋈ Book.ISBN = Book_Author.ISBN (Book))))) ⋈ "Order".customerEmail = Customer.email ("Order" ⋈ Customer))

```
SELECT Fname, Lname, email
FROM Customer, Author, Book_Author, Book_Order, Book, "Order"
WHERE Author.id = Book_Author.authorId AND Book.ISBN = Book_Author.ISBN AND
Book.ISBN = Book_Order.ISBN AND Book_Order.orderId = "Order".id AND Customer.email =
"Order".customerEmail AND name IN (
  SELECT name
  FROM Book, Author, Book_Author, Book_Order
  WHERE Author.id = Book_Author.authorId AND Book.ISBN = Book_Author.ISBN AND
Book.ISBN = Book_Order.ISBN
  GROUP BY name
  ORDER BY SUM(Price) DESC
  LIMIT 1
)
```

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

π name (σ email ∈ (π email (ρ t1 (σ SUM(Price) > (π AVG(total_spent) (ρ t2 (γ email; AVG(total_spent); ρ t3 (σ email = customerEmail (Customer) ⋈ "Order".id = orderId (γ customerEmail, SUM(Price); ρ t4 (Book_Order ⋈ Book.ISBN = Book_Order.ISBN (Book ⋈ Book_Author.ISBN = Book.ISBN (Book_Author ⋈ Author.id = Book_Author.authorId (Author)))))))))))

```
SELECT DISTINCT name
FROM Book, Author, Book_Author, Book_Order, "Order", Customer
WHERE Author.id = Book_Author.authorId AND Book.ISBN = Book_Author.ISBN AND
Book.ISBN = Book_Order.ISBN AND Book_Order.OrderId = "Order".id AND email IN (
  SELECT email
   FROM Customer, Book, "Order", Book_Order
   WHERE Customer.email = "Order".customerEmail AND "Order".id = Book_Order.orderId AND
Book_Order.ISBN = Book.ISBN
   GROUP BY email
   HAVING SUM(Price) > (
      SELECT AVG(total_spent)
      FROM (
          SELECT SUM(Price) AS total_spent
          FROM Customer, Book, "Order", Book_Order
          WHERE Customer.email = "Order".customerEmail AND "Order".id = Book_Order.orderId
AND Book_Order.ISBN = Book.ISBN
          GROUP BY email
      )
   )
)
```

INSERT Syntax

- **Insert Author**
  - Dependencies
    - None
  - Syntax
    ```
    INSERT INTO     AUTHOR (id, fName, midInit, lName)
    VALUES          (100, 'Fyodor', NULL, 'Dostoevsky');
    ```

- **Insert Publisher**
  - Dependencies
    - None

- Syntax
  ```
  INSERT INTO     PUBLISHER (id, name)
  VALUES          (100, 'Penguin');
  ```

- **Insert Customer**
  - Dependencies
    - None
  - Syntax
  ```
  INSERT INTO     CUSTOMER (email, fName, lName)
  VALUES          ('email@hotmail.com', 'Fred', 'Jones');
  ```

- **Insert Book**
  - Dependencies
    - Need to insert publisher and assign it a unique Id if the book's publisher is not already in the database.
  - Defaults
    - Discount = 0
  - Syntax
  ```
  INSERT INTO     BOOK (ISBN, genre, releaseYear, size,
  discount, title, price, pubId)
  VALUES          (0140447920, 'Classic Literature', 1869,
  50, 0, 'The Idiot', 15.99, 100);
  ```

DELETE Syntax

- Delete Author
  - Dependencies
    - Delete cascades to BOOK_AUTHOR WHERE authId = 100;
  - Syntax
    DELETE FROM AUTHOR WHERE id = 100;

- Delete Publisher
  - Dependencies
    - Delete cascades to PUB_LOC WHERE pubId = 100
    - Delete cascades to PUB_PHONE WHERE pubId = 100
  - Syntax
    DELETE FROM BOOK WHERE pubId = 100;
    DELETE FROM PUBLISHER WHERE id = 100;

- Delete Customer
  - Dependencies

            - Delete cascades to CUSTOMER_PHONE WHERE customerEmail = 'email@hotmail.com';
            - Delete cascades to WISHLIST WHERE customerEmail = 'email@hotmail.com'
            - Delete cascades to WISHLIST_CONTAINS WHERE customerEmail = 'email@hotmail.com'
    - Syntax
      DELETE FROM "ORDER" WHERE customerEmail = 'email@hotmail.com';
      DELETE FROM CUSTOMER WHERE email = 'email@hotmail.com';

- Delete Book
    - Dependencies
        - Delete cascades to BOOK_AUTHOR WHERE ISBN = 0140447920
        - Delete cascades to BOOK_ORDER WHERE ISBN = 0140447920
        - Delete cascades to REVIEW WHERE ISBN = 0140447920
        - Delete cascades to STORED WHERE ISBN = 0140447920
        - Delete cascades to WISHLIST_CONTAINS WHERE ISBN = 0140447920
    - Syntax
      DELETE FROM BOOK WHERE ISBN = 0140447920;

1. Order:
   DELETE FROM “Order” WHERE id = 1234;

2. Customer
   DELETE FROM Customer WHERE Email = 'jane.doe@example.com';

3. Review
   DELETE FROM Review WHERE CustomerEmail = 'jane.doe@example.com';

4. Customer_Phone
   DELETE FROM Customer_Phone WHERE CustomerEmail = 'jane.doe@example.com';

5. Employee_Phone
   DELETE FROM Employee_Phone WHERE essn = '123-45-6789';

6. Employee
   DELETE FROM Employee WHERE SSN = '123-45-6789';

7. Location
   DELETE FROM Location WHERE id = 456;

8. Pub_Phone

```
DELETE FROM Pub_Phone WHERE PubId = 123 AND PhoneNumber =
'555-555-1212';
DELETE FROM Publisher WHERE id = 123;
DELETE FROM Location WHERE id = 789;
```

9. Warehouse
```
DELETE FROM Warehouse WHERE id = 456;
```

11. Wishlist
```
DELETE FROM Wishlist WHERE CustomerEmail = 'jane.doe@example.com' AND
Name = 'My Wishlist';
```

12. Book
```
DELETE FROM Book WHERE ISBN = '1234567890';
```

13. Author
```
DELETE FROM Author WHERE id = 123;
```

14. Company Account
```
DELETE FROM Company_Account WHERE Name = 'Marketing';
```

Section 3 - Graded Checkpoint Documents

Project Checkpoint 1: Original

CSE 3241 Project Checkpoint 01 – Entities and Relationships

Names: Faisal and Will
Date: February 13, 2023

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.

Book:

- ISBN
- Title
- Genre
- Price
- Author(s)
- Release year
- Size
- Discount

Customer:

- Customer_id
- Email address
- Name
    - First name
    - Last name
- Phone number
- Address

Order:

- Order_id
- Order date
- Order status
- Delivery Date
- Order Cost
- Total Discount (coupons and such included)

Warehouse:
- Warehouse id
- Size

Location:

- Location_id
- State
- Address
- City
- Zip code

Publisher:

- Publisher_id
- Phone number
- Name

Reviews:

- <u>Review_id</u>
- Review score
- Review comment
- Date written

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 place ORDER entities
- CUSTOMER entities cancel ORDER entities
- CUSTOMER entities buy BOOK entities
- ORDER entities contain BOOK entities
- ORDER entities notify WAREHOUSE entities
- ORDER entities delivered to LOCATION entities
- REVIEW entities rate BOOK entities
- PUBLISHER entities publish BOOK entities
- WAREHOUSE entities store BOOK entities
- CUSTOMER entities write REVIEW entities
- PUBLISHER entities are located at LOCATION entities
- WAREHOUSE entities are located at LOCATION entities

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.

Employee:

- <u>SSN</u>
- Name
    - First name
    - Last name
- Phone number
- Address
- Hours Worked
    - Normal Hours
    - Overtime Hours
- Hourly wage

Company Account:

- Account number
- Account name
- Purpose for account

Wishlist (weak entity that is dependent on the customer entity):

- Name
- Privacy

Relationships:

- EMPLOYEE entities live at LOCATION entities
- CUSTOMER entities have WISHLIST entities
- WISHLIST entities contain BOOK entities
- ORDER entities transfer money to COMPANY ACCOUNT entities
- COMPANY ACCOUNT entities pay EMPLOYEE entities

The wishlist entity would be useful because the company could track which books the customers want, which would allow them to send the customers emails when there are discounts on books in their wishlists.
The employee entity and company account entity are useful because they provide an organized way of taking in revenue and using that revenue to pay employees.

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. List the titles of books written by a certain author
2. List all orders made by a certain customer
3. List all warehouses that have a certain book
4. Get the book that has sold the most copies
5. Get the SSN of a certain employee
6. Get the hours worked for a certain employee
7. Get books on a customers wishlist that have a discount > 0

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

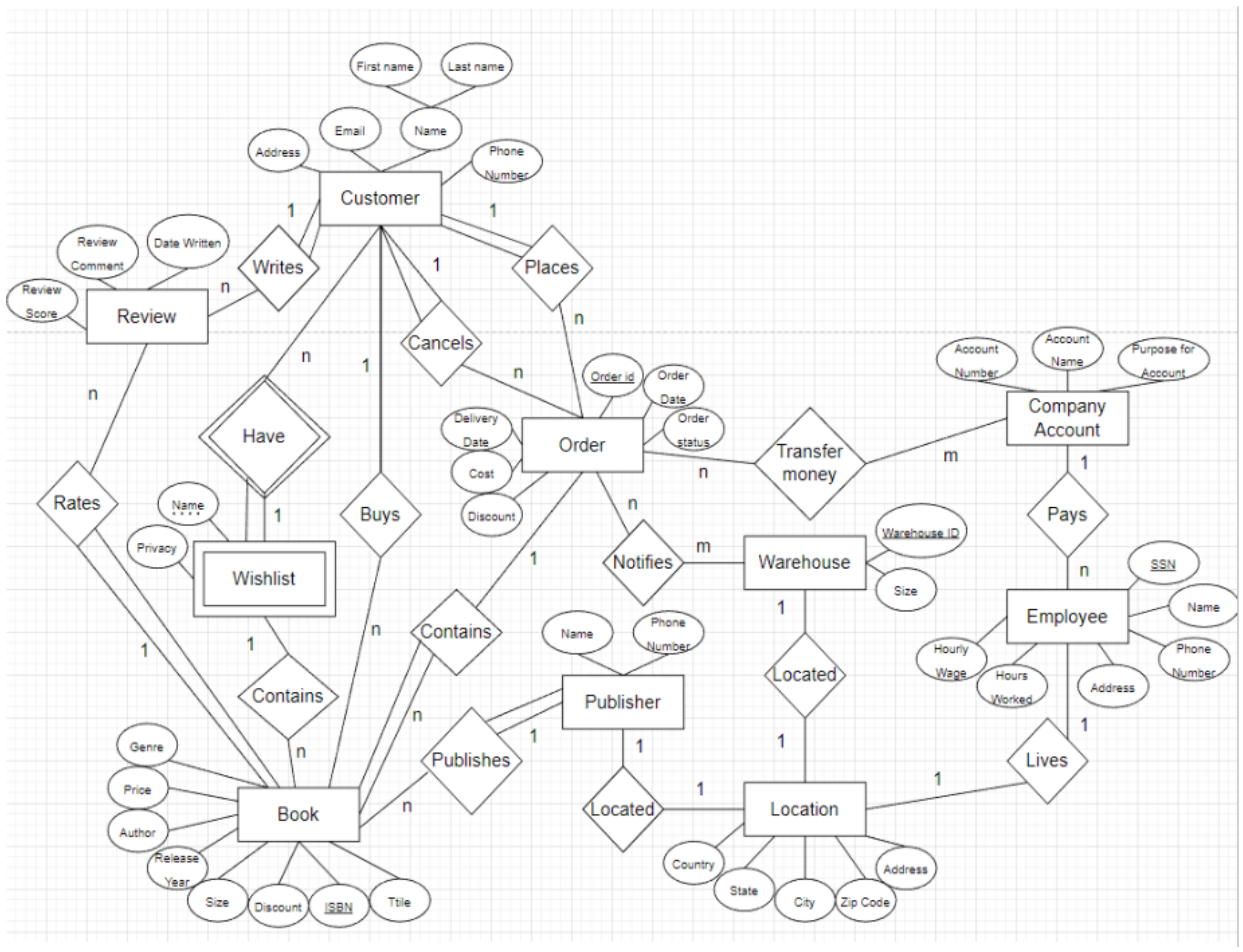
above description, is it possible to add a new publisher to your database without knowing the title of any books they have published? If not, revise your model to allow for publishers to be added as separate entities.

To add a new publisher to the database, create a new publisher entity with the desired attributes, create a located relation to the location(s) that the publisher is based in, and create a publish relation to any books in the database that are published by them.

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. Add a new customer to the database
   ○ A customer entity would need to be created with the desired attributes ○ A located relation would need to be made with the location that the customer lives in
   i. A new city entity would need to be added for the customer's home city if it is not already in the database. This city entity would have the corresponding city name and state
2. Add a new employee
   ○ A new employee entity would need to be created with its associated attributes
   ○ A location relation would need to be made with the location where the employee lives in
   ○ A company account relation would need to be made to pay the employee
3. Adding new books
   ○ A new book entity would need to be created with its associated attributes ○ A publishes relation would need to be made with the book's publisher 4. Adding a new wishlist
   ○ A wishlist entity would need to be created with the desired name and privacy setting (public or private)
   ○ A has relation with the customer that created the wishlist

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: Revised

CSE 3241 Project Checkpoint 01 – Entities and Relationships

Names: Faisal and Will
Date: February 13, 2023

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.

Book:

- ISBN
- Title
- Genre
- Price
- Author(s)
- Release year
- Size
- Discount

Customer:

- Customer_id
- Email address
- Name
  - First name
  - Last name
- Phone number
- Address

Order:

- Order_id
- Order date
- Order status
- Delivery Date
- Order Cost
- Total Discount (coupons and such included)

Warehouse:

- Warehouse id
- Size

Location:

- Location_id
- State
- Address
- City
- Zip code

Publisher:

- Publisher_id
- Phone number
- Name

Reviews:

- Review_id
- Review score
- Review comment
- Date written

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 place ORDER entities
- CUSTOMER entities cancel ORDER entities
- CUSTOMER entities buy BOOK entities
- ORDER entities contain BOOK entities
- ORDER entities notify WAREHOUSE entities
- ORDER entities delivered to LOCATION entities
- REVIEW entities rate BOOK entities
- PUBLISHER entities publish BOOK entities
- WAREHOUSE entities store BOOK entities
- CUSTOMER entities write REVIEW entities
- PUBLISHER entities are located at LOCATION entities
- WAREHOUSE entities are located at LOCATION entities

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.

Employee:

- SSN
- Name
  - First name
  - Last name
- Phone number
- Address
- Hours Worked
  - Normal Hours
  - Overtime Hours

- Hourly wage

Company Account:

- <u>Account number</u>
- Account name
- Purpose for account

Wishlist (weak entity that is dependent on the customer entity):

- <u>Name</u>
- Privacy

Relationships:

- EMPLOYEE entities live at LOCATION entities
- CUSTOMER entities have WISHLIST entities
- WISHLIST entities contain BOOK entities
- ORDER entities transfer money to COMPANY ACCOUNT entities
- COMPANY ACCOUNT entities pay EMPLOYEE entities

The wishlist entity would be useful because the company could track which books the customers want, which would allow them to send the customers emails when there are discounts on books in their wishlists.

The employee entity and company account entity are useful because they provide an organized way of taking in revenue and using that revenue to pay employees.

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. List the titles of books written by a certain author
2. List all orders made by a certain customer
3. List all warehouses that have a certain book
4. Get the book that has sold the most copies
5. Get the SSN of a certain employee
6. Get the hours worked for a certain employee
7. Get books on a customers wishlist that have a discount > 0

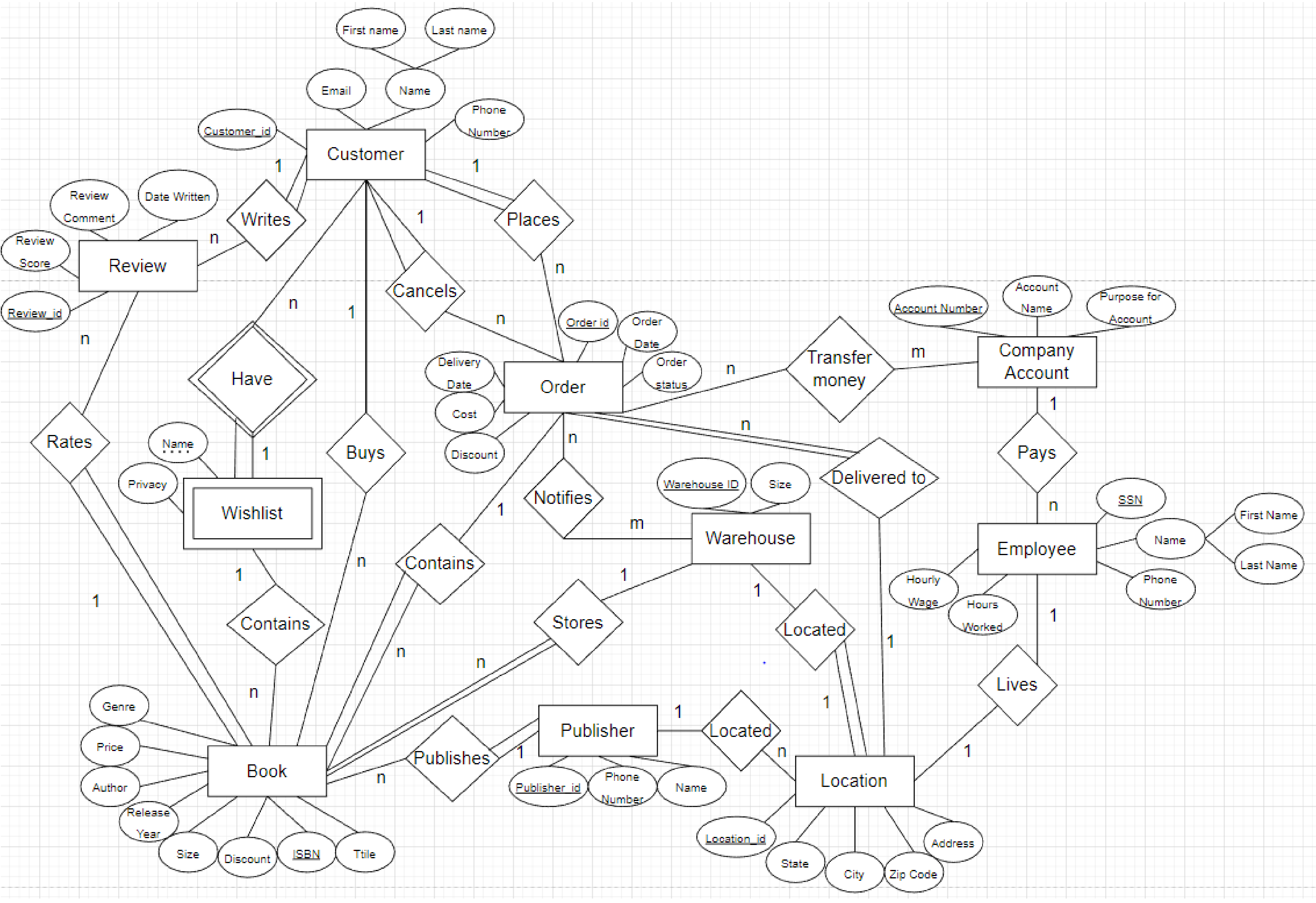
5. Suppose we want to add a new publisher to the database. How would we do that given the entities and relationships you've outlined above? Given your above description, is it possible to add a new publisher to your database without knowing the title of any books they have published? If not, revise your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

To add a new publisher to the database, create a new publisher entity with the desired attributes, create a located relation to the location(s) that the publisher is based in, and create a publish relation to any books in the database that are published by them.

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. Add a new customer to the database
    - A customer entity would need to be created with the desired attributes
    - A located relation would need to be made with the location that the customer lives in
        i. A new city entity would need to be added for the customer's home city if it is not already in the database. This city entity would have the corresponding city name and state
2. Add a new employee
    - A new employee entity would need to be created with its associated attributes
    - A location relation would need to be made with the location where the employee lives in
    - A company account relation would need to be made to pay the employee
3. Adding new books
    - A new book entity would need to be created with its associated attributes
    - A publishes relation would need to be made with the book's publisher
4. Adding a new wishlist
    - A wishlist entity would need to be created with the desired name and privacy setting (public or private)
    - A have relation with the customer that created the wishlist
    - A contains relation with book needs to be created

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.



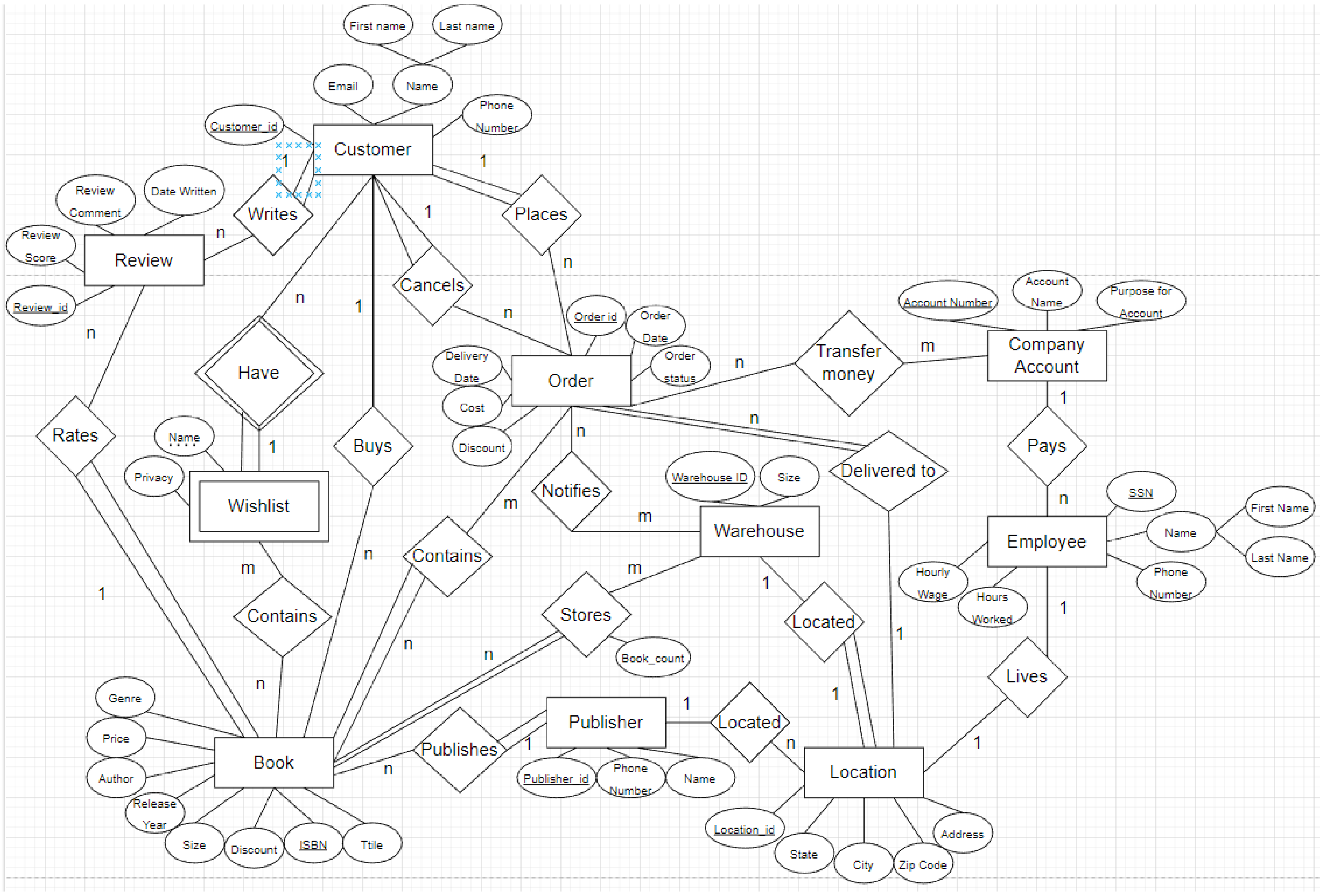
Project Checkpoint 2: Original

CSE 3241 Project Checkpoint 02 – Relational Model and Relational Algebra

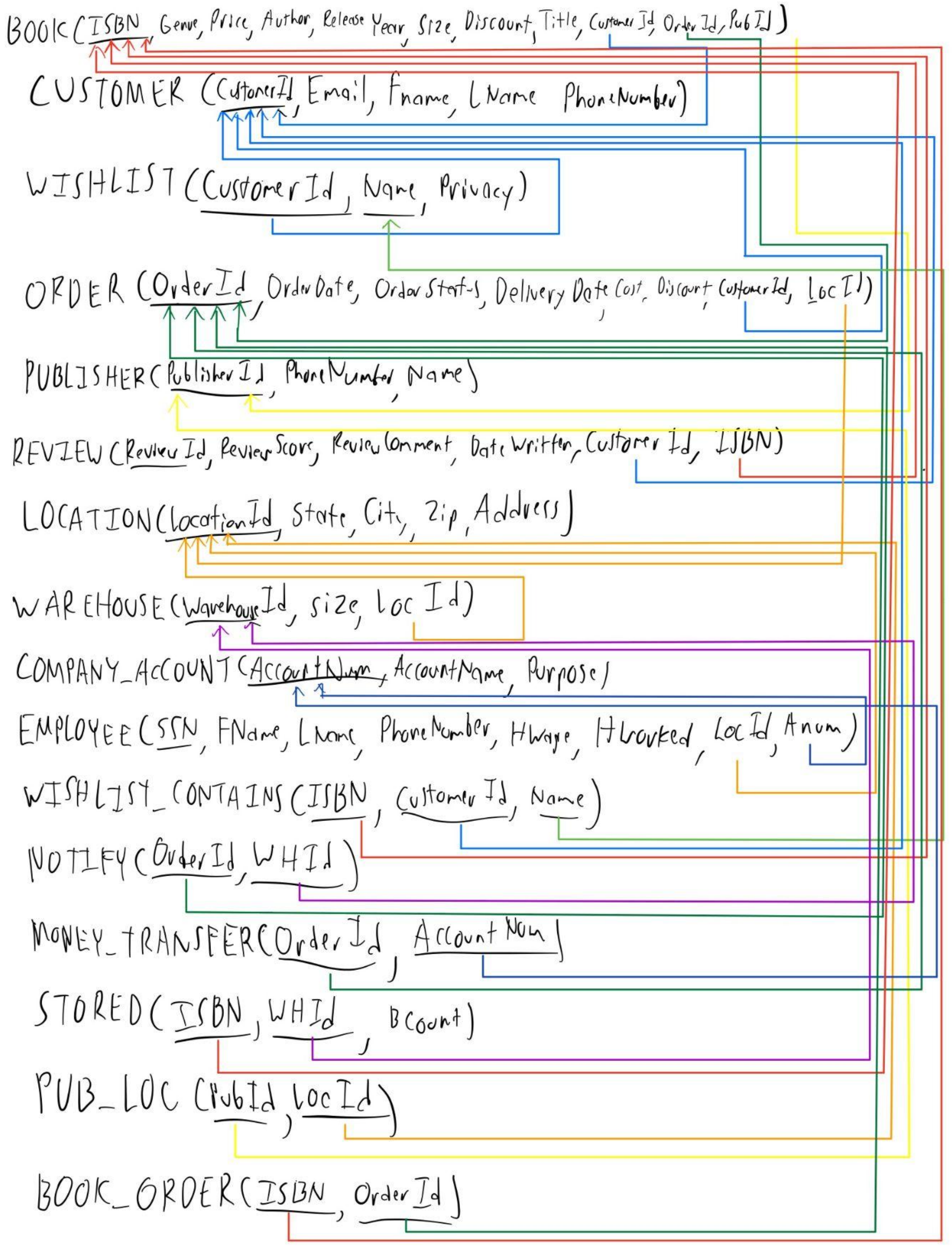
Names: Faisal and Will
Date: 2/21/2023

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.



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


BOOK(ISBN, Genre, Price, Author, Release Year, Size, Discount, Title, Customer Id, Order Id, Pub Id)
CUSTOMER (CustomerId, Email, Fname, LName PhoneNumber)
WISHLIST(CustomerId, Name, Privacy)
ORDER (OrderId, OrderDate, OrderStatus, Delivery Date, Cost, Discount, CustomerId, Loc Id)
PUBLISHER(PublisherId, PhoneNumber, Name)
REVIEW(ReviewId, ReviewScore, ReviewComment, DateWritten, Customer Id, ISBN)
LOCATION(LocationId, State, City, Zip, Address)
WAREHOUSE(WarehouseId, size, Loc Id)
COMPANY_ACCOUNT(AccountNum, AccountName, Purpose)
EMPLOYEE(SSN, FName, LName, PhoneNumber, HWage, HWorked, LocId, Anum)
WISHLIST_CONTAINS(ISBN, CustomerId, Name)
NOTIFY(OrderId, WHId)
MONEY_TRANSFER(OrderId, AccountNum)
STORED(ISBN, WHId, BCount)
PUB_LOC(PubId, LocId)
BOOK_ORDER(ISBN, OrderId)

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

$$\sigma_{Author = 'Pratchett'\ AND\ Price < 10}(Book)$$

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

$Single\_Customer \leftarrow \sigma_{Customer\ id = 32167}(Customer)$

$Custmor\_Orders \leftarrow Single\_Customer \bowtie_{Customer\ id = Customer\ id} Order$

$Books\_in\_Orders \leftarrow Customer\_Orders \bowtie_{Order\ id = Order\ id} Book$

$result \leftarrow \pi_{Title,\ Order\ date}(Books\_in\_Orders)$

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

$$\pi_{Title,\ ISBN}(\sigma_{BCount < 5}(Stored))$$

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

$PRATCHETT \leftarrow \sigma_{Name = "Pratchett"}(BOOK \bowtie_{PubId = PublisherId} PUBLISHER)$

$\pi_{CustomerId,\ Title}(CUSTOMER \bowtie_{CUSTOMER.CustomerID = PRATCHETT.CustomerId} PRATCHETT)$

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

$Single\_Customer \leftarrow \sigma_{Customer\ id = 32167}(Customer)$

$Customer\_Orders \leftarrow Single\_Customer \bowtie_{Customer\ id = Customer\ id} Order$

$Books\_in\_Orders \leftarrow Customer\_Orders \bowtie_{Order\ id = Order\ id} Book\_Order$

$result \leftarrow A_{COUNT\ ISBN}(Books\_in\_Orders)$

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

$Customer\_Orders \leftarrow Customer \bowtie_{Customer\ id\ =\ Customer\ id} Order$

$Books\_in\_Orders \leftarrow Customer\_Orders \bowtie_{Order\ id\ =\ Order\ id} Book\_Order$

$Number\_of\_Purchases \leftarrow_{CustomerId} A_{COUNT\ ISBN}(Book\_in\_Orders)$

$result \leftarrow_{CustomerId} A_{MAX\ count\ \_ISBN}(Number\_of\_Purchases)$

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.

1. Give the average hours worked of all the employees and their location id

   $Employee\_Location \leftarrow Employee \bowtie_{Location\ id\ =\ Location\ id} Location$
   $result \leftarrow\ _{Location\ id} A_{AVERAGE\ Hworked}(Employee\_Location)$

2. Give all the customer_ids and names of customers who wrote a review with a score that is >= 4

   $High\_rating \leftarrow \sigma_{Review\ Score\ >=\ 4}(Review)$
   $Customer\_Good\_Review \leftarrow Customer \bowtie_{Customer\ id\ =\ Customer\ id} High\_rating$
   $result \leftarrow \pi_{Customer\ id,\ Fname,\ Lname} Customer\_Good\_Review$

3. Give all the customer_ids and the name of their wishlist who have a certain book in their wishlist

   $Specific\_Book \leftarrow \sigma_{Title\ =\ "Intro\ to\ Linear\ Algebra"}(Book)$
   $Wishlist\_Has\_Book \leftarrow Wishlist \bowtie_{ISBN\ =\ ISBN} Specific\_Book$
   $result \leftarrow \pi_{Customer\ id,\ Name} Wishlist\_Has\_Book$

Project Checkpoint 3: Original

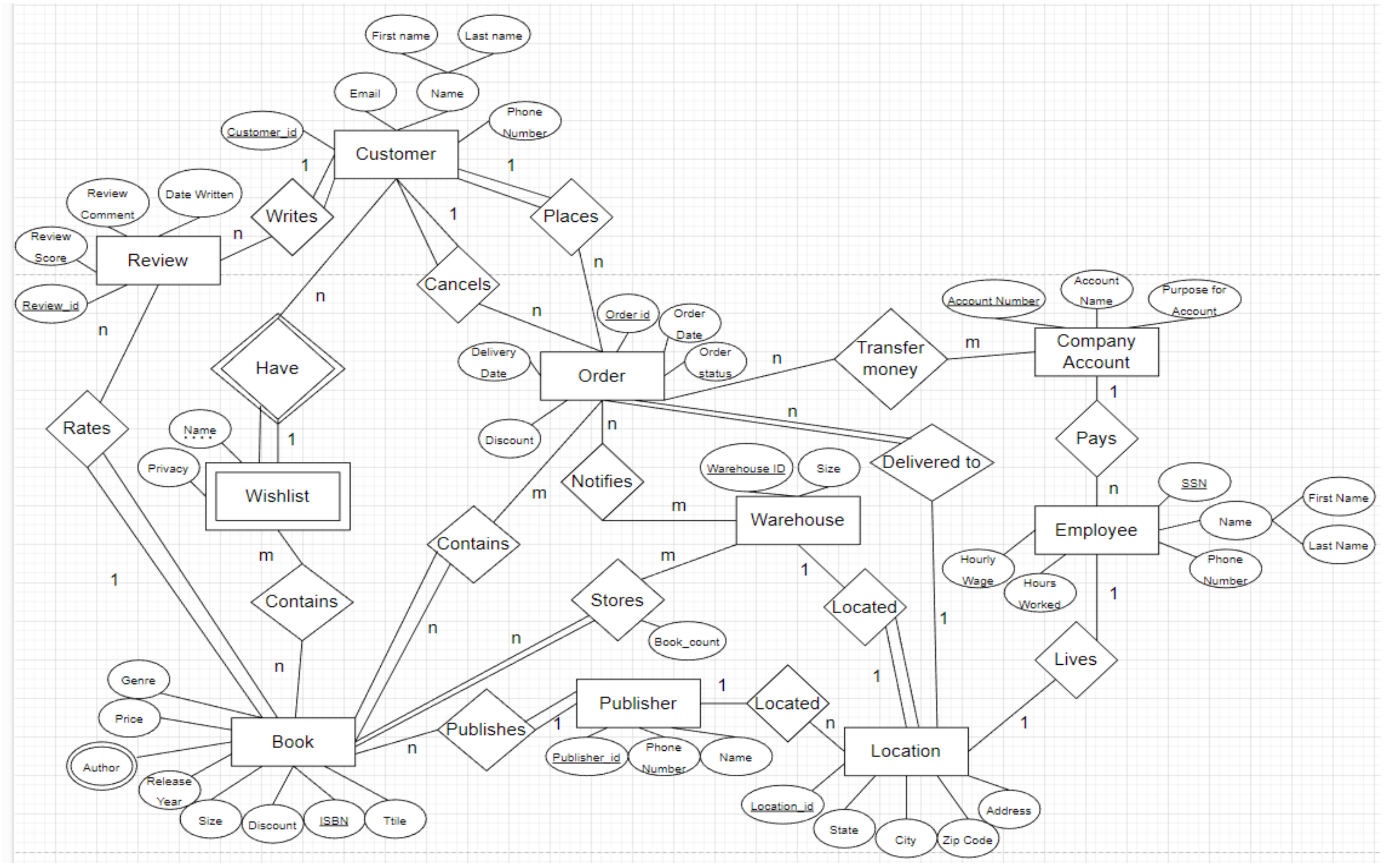
CSE 3241 Project Checkpoint 03 – SQL and More SQL

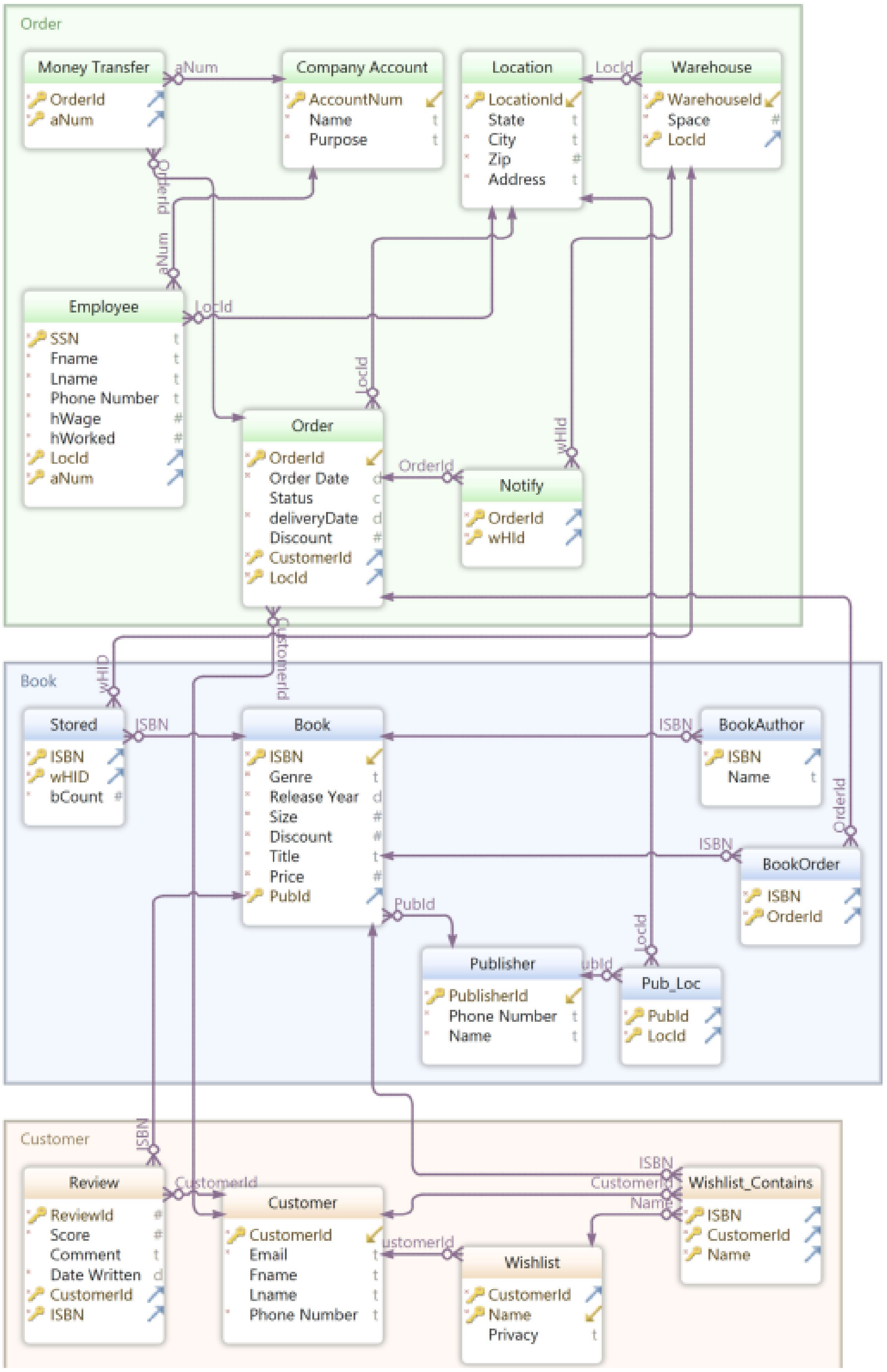
Names: Faisal and Will

Date: April 3, 2023

Submitted to the Carmen Dropbox

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




Order
Money Transfer
OrderId
aNum
Company Account
AccountNum
Name
Purpose
Location
LocationId
State
City
Zip
Address
Warehouse
WarehouseId
Space
LocId
Employee
SSN
Fname
Lname
Phone Number
hWage
hWorked
LocId
aNum
Order
OrderId
Order Date
Status
deliveryDate
Discount
CustomerId
LocId
Notify
OrderId
wHId
Book
Stored
ISBN
wHID
bCount
Book
ISBN
Genre
Release Year
Size
Discount
Title
Price
PubId
BookAuthor
ISBN
Name
BookOrder
ISBN
OrderId
Publisher
PublisherId
Phone Number
Name
Pub_Loc
PubId
LocId
Customer
Review
ReviewId
Score
Comment
Date Written
CustomerId
ISBN
Customer
CustomerId
Email
Fname
Lname
Phone Number
Wishlist
CustomerId
Name
Privacy
Wishlist_Contains
ISBN
CustomerId
Name

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.

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, Author
WHERE name = "Pratchett" AND price < 10 AND Book.ISBN = Author.ISBN
```

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

```
SELECT Title, Order Date
FROM Customer, Order, Book
WHERE CustomerId = 32167 AND Customer.CustomerId = Order.CustomerId AND
Order.OrderId = Book.OrderId
```

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

```
SELECT Title, ISBN
FROM Book, Stored
WHERE Bcount < 5 AND Book.ISBN = Stored.ISBN
```

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

```
SELECT CustomerId, Title
FROM Customer, Book
WHERE author = "Pratchett" AND Customer.CustomerId = Book.CustomerId
```

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

```
SELECT COUNT(ISBN)
FROM Customer, Order, Book_order
WHERE CustomerId = 32167 AND Customer.CustomerId = Order.CustomerId AND
```

Order.OrderId = Book_order.OrderId

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

```
SELECT MAX(COUNT(ISBN))
FROM Customer, Order, Book_order
WHERE Customer.CustomerId = Order.CustomerId AND
Order.OrderId = Book_order.OrderId
```

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. Give the average hours worked of all the employees and their location id

$Employee\_Location \leftarrow Employee \bowtie_{Location\ id\ =\ Location\ id} Location$

$result \leftarrow {}_{Location\ id}\, A_{AVERAGE\ Hworked}(Employee\_Location)$

```
SELECT AVERAGE(Hworked)
FROM Employee, Location
WHERE Employee.LocationId = Location.LocationId
GROUP BY LocationId
```

b. Give all the customer_ids and names of customers who wrote a review with a score that is >= 4

$High\_rating \leftarrow \sigma_{Review\ Score\ > = 4}(Review)$

$Customer\_Good\_Review \leftarrow Customer \bowtie_{Customer\ id\ =\ Customer\ id} High\_rating$

$result \leftarrow \pi_{Customer\ id,\ Fname,\ Lname} Customer\_Good\_Review$

```
SELECT CustomerId, Fname, Lname
FROM Review, Customer
WHERE Review Score >= 4 AND Review.CustomerId = Customer.CustomerId
```

c. Give all the customer_ids and the name of their wishlist who have a certain book in their wishlist

$Specific\_Book \leftarrow \sigma_{Title = "Intro\ to\ Linear\ Algebra"}(Book)$
$Wishlist\_Has\_Book \leftarrow Wishlist \bowtie_{ISBN = ISBN} Specific\_Book$
$result \leftarrow \pi_{Customer\ id,\ Name} Wishlist\_Has\_Book$

```
SELECT CustomerId, Name
FROM Book, Wishlist
WHERE Title = “Intro to Linear Algebra” AND Wishlist.ISBN = Book.ISBN
```

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 Fname, Lname, SUM(Price)
FROM Customer, Book
WHERE Customer.CustomerId = Book.CustomerId
GROUP BY CustomerId
```

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

```
SELECT Fname, Lname, Email
FROM Customer, Book
WHERE Customer.CustomerId = Book.CustomerId
HAVING SUM(Price) > (
        SELECT AVG(Price)
        FROM (
                SELECT SUM(Price)
                FROM Customer, Book
                WHERE Customer.CustomerId = Book.CustomerId
                GROUP BY CustomerId
        )
)
```

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, COUNT(*)
FROM Book, BookOrder
JOIN Book.ISBN = BookOrder.ISBN
GROUP BY Title
ORDER BY COUNT(*) DESC
```

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, COUNT(*) * Price
FROM Book, BookOrder
JOIN Book.ISBN = BookOrder.ISBN
GROUP BY Title
ORDER BY (COUNT(*) * Price) DESC
```

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

```
SELECT Name, COUNT(*)
FROM Book, BookAuthor, BookOrder
WHERE Book.ISBN = BookAuthor.ISBN AND Book.ISBN = BookOrder.ISBN
GROUP BY Name
ORDER BY COUNT(*)
```

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

```
SELECT Name, SUM(Price)
FROM Book, BookAuthor, BookOrder
WHERE Book.ISBN = BookAuthor.ISBN AND Book.ISBN = BookOrder.ISBN
GROUP BY Name
ORDER BY SUM(Price)
```

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

```
SELECT Fname, Lname, email
```

```
    FROM Customer, BookAuthor, BookOrder, Book, Order
    WHERE Book.ISBN = BookAuthor.ISBN AND Book.ISBN = BookOrder.ISBN AND Book
    Order.OrderId = Order.OrderId AND Customer.CustomerId = Order.CustomerId AND
    Name IN (
        SELECT Name, SUM(Price)
        FROM Book, BookAuthor, BookOrder
        WHERE Book.ISBN = BookAuthor.ISBN AND Book.ISBN = BookOrder.ISBN
        GROUP BY Name
        ORDER BY SUM(Price)
    )
```

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

```
    SELECT Name
    FROM Book, BookAuthor
    WHERE Book.ISBN = BookAuthor.ISBN AND Book.ISBN = BookOrder.ISBN AND Book
    Order.OrderId = Order.OrderId AND CustomerId IN (
        SELECT CustomerId
        FROM Customer, Book
        WHERE Customer.CustomerId = Book.CustomerId
        HAVING SUM(Price) > (
            SELECT AVG(Price)
            FROM (
                SELECT SUM(Price)
                FROM Customer, Book
                WHERE Customer.CustomerId = Book.CustomerId
                GROUP BY CustomerId
            )
        )
    )
```

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

Project Checkpoint 3: Revisions

- “See Section 1: Relational Schema Page 2 for the revised relational schema)
- “See Section 2: SQL Queries Pages 24-24 for the revised SQL queries”

Project Checkpoint 4: Original

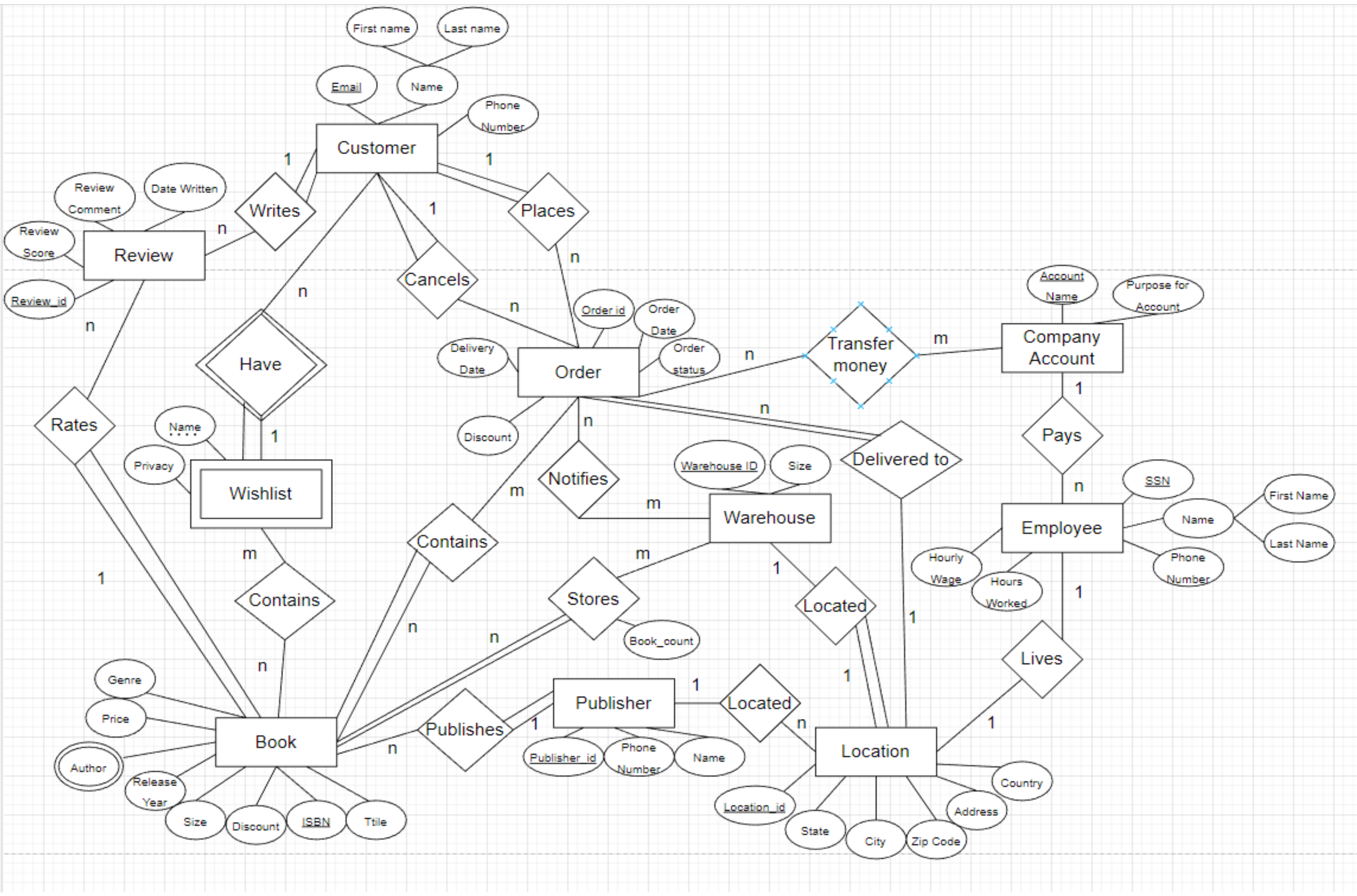
CSE 3241 Project Checkpoint 04 – Functional Dependencies and Normal Forms

Names: Will, Faisal

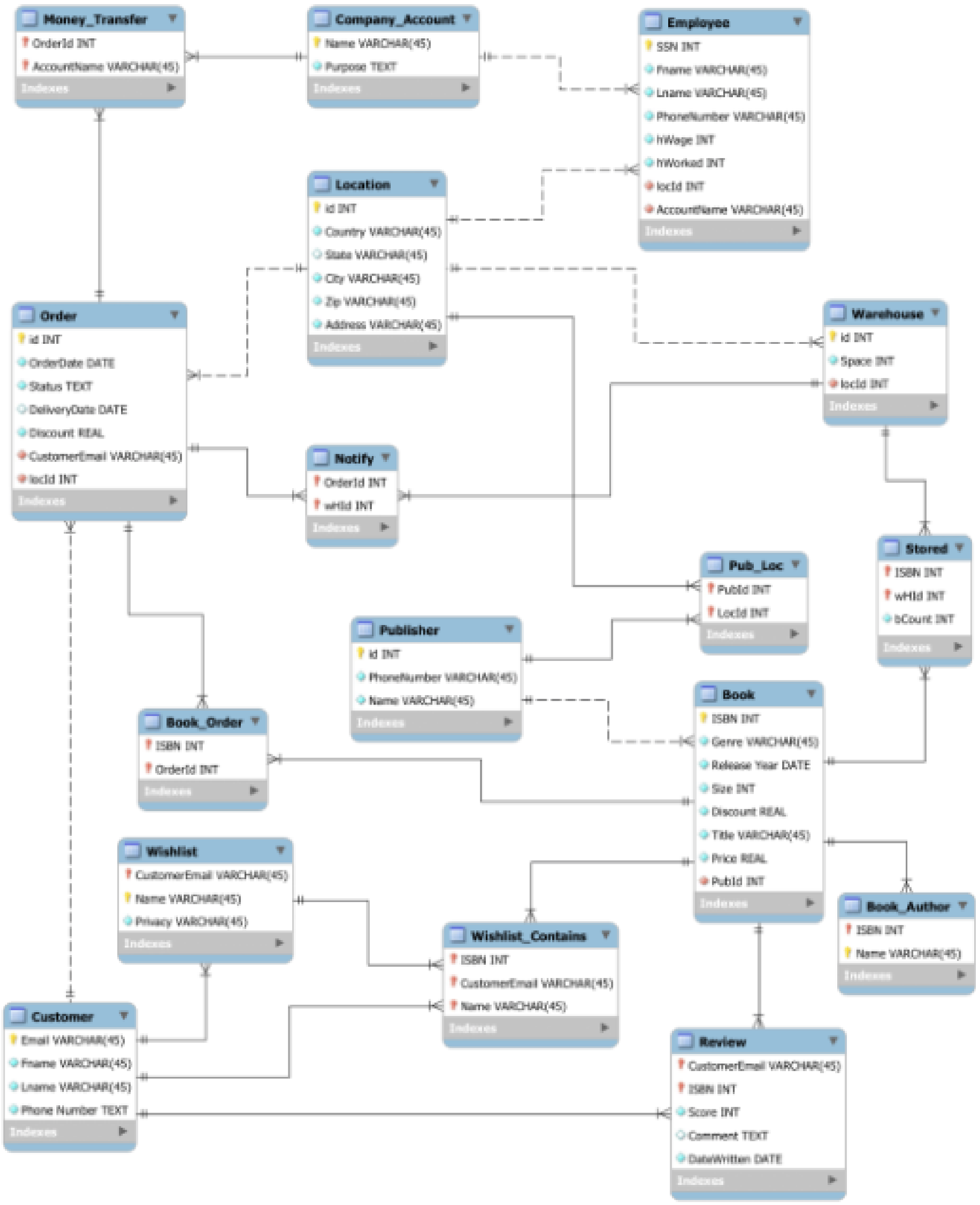
Date 4/18/2023

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

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).

Book:
- $\{ISBN\} \rightarrow \{Title, Genre, Price, Release\ year, Size, Discount, PubId\}$

Publisher:
- $\{PublisherId\} \rightarrow \{PhoneNumber, Name\}$

Customer:
- $\{Email\} \rightarrow \{Fname, Lname, PhoneNumber\}$

Order:
- $\{OrderId\} \rightarrow \{Order\ Date, Status, Delivery\ Date, Discount, CustomerEmail, LocId\}$

Warehouse:
- $\{WarehouseId\} \rightarrow \{Space, LocId\}$

Location:
- $\{LocationId\} \rightarrow \{Country, State, City, Zip, Address\}$

Review:
- $\{CustomerEmail, ISBN\} \rightarrow \{Score, Comment, Date\}$

Employee:
- $\{SSN\} \rightarrow \{Fname, Lname, hWage, hWorked, LocId, aNum\}$

Employee_Phone:
- $\{SSN\} \rightarrow \{Phone\ Number\}$

Company Account:
- $\{Name\} \rightarrow \{Purpose\}$

Stored:
- $\{ISBN, wHId\} \rightarrow \{bCount\}$

Wishlist:
- $\{CustomerId, Name\} \rightarrow \{Privacy\}$

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.
    - Book: 3NF
    - Book_Author: 3NF
    - Customer: 3NF
    - Wishlist: 3NF
    - Order: 3NF
    - Publisher: 3NF

- Review: 3NF
- Location: 3NF
- Warehouse: 3NF
- Company_Account: 3NF
- Employee: 3NF
- Employee_Phone: 3NF
- Wishlist_Contains: 3NF
- Notify: 3NF
- Money_Transfer: 3NF
- Stored: 3NF
- Pub_Loc: 3NF
- Book_Order: 3Nf

4. 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.

   Finds the total number of books purchased by customers:

```
CREATE VIEW total_books_purchased_by_customer AS
SELECT COUNT(ISBN)
FROM Customer, Order, Book_order
WHERE Customer.Email = Order.Email AND Order.OrderId = Book_order.OrderId
GROUP BY Customer.CustomerId
```

   Finds the total number of books sold by authors:

```
CREATE VIEW author_book_sales AS
SELECT Name,Count
FROM Book, BookAuthor, BookOrder
WHERE Book.ISBN = BookAuthor.ISBN AND Book.ISBN = BookOrder.ISBN
GROUP BY Name
ORDER BY COUNT(*)
```

Project Checkpoint 4: Revised

- “See Section 1: Relational Schema Page 2 for the revised relational schema”

- “See Section 1: Normalization Pages 3-7 for the revised functional dependencies and normalized forms”