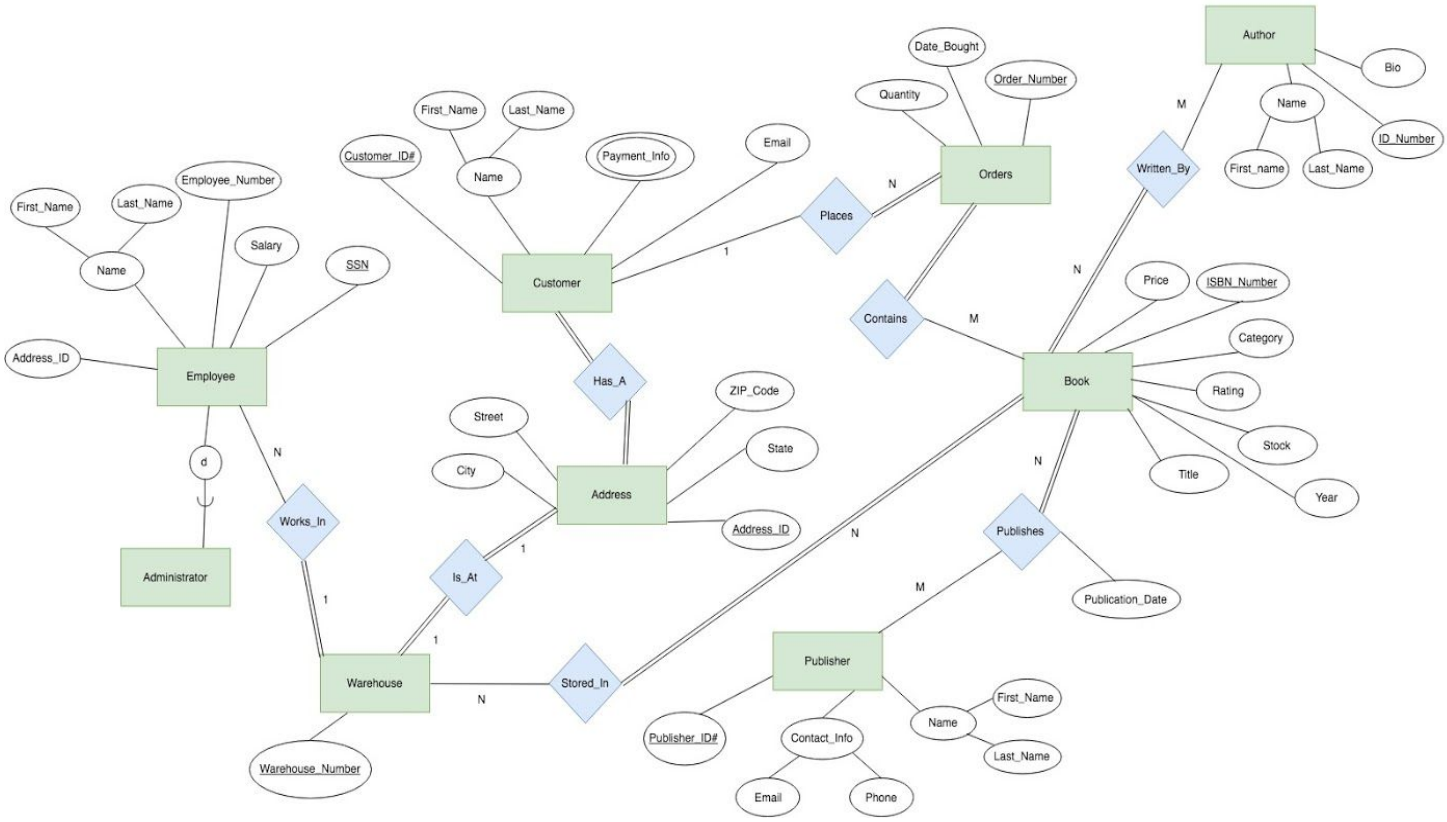
BITS & BOOKS DATABASE MANUAL
CSE 3241: Intro to Databases
Final Project

Emily Zeaman, Vicky Sandoval, Patrick Walter, Gilberto Molina Badillo, Nick Schumer

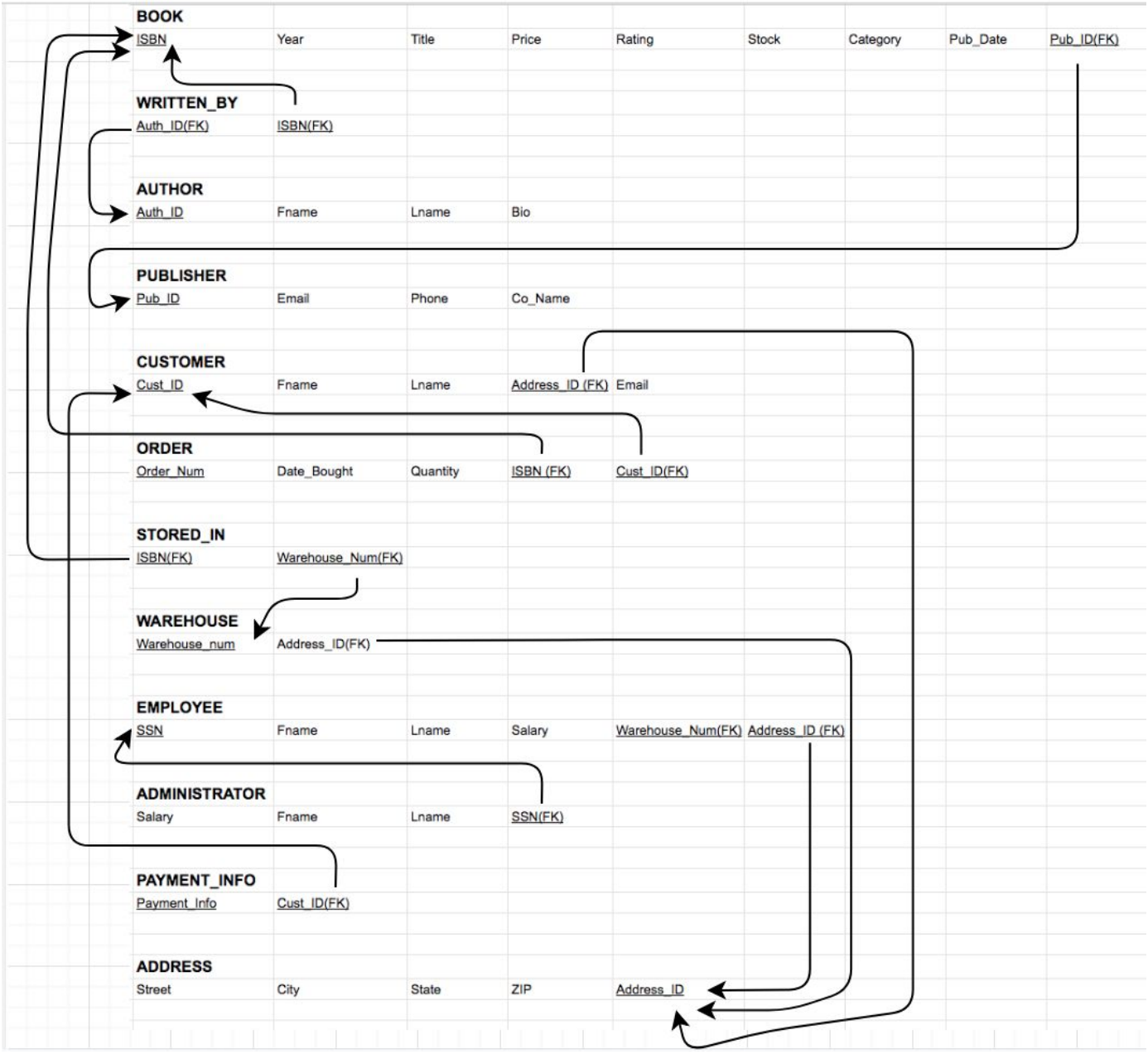
Part I: The Final Report

Section I: Database Description

ER Model:



Relational Schema



Database Normalization and Justification

BOOK

- Functional Dependencies
  - {ISBN} → {Year, Title, Price, Rating, Stock, Category, Pub_Date, Pub_ID}

- BCNF - ISBN leads to all of the other attributes as the superkey and with no other functional dependencies so this relation is in BCNF.

WRITTEN_BY

- Functional Dependencies
  - {Auth_ID, ISBN} → {Year, Title, Price, Rating, Stock, Category, Pub_Date, Pub_ID, Fname, Lname, Bio}
- BCNF - The pair Auth_ID and ISBN leads to all of the other attributes as the superkey and with no other functional dependencies so this relation is in BCNF.

AUTHOR

- Functional Dependencies
  - {Auth_ID} → {Fname, Lname, Bio}
- BCNF - An assumption made for this relation is that no Fname and Lname pair is unique. Author_ID leads to all of the other attributes as the superkey and with no other functional dependencies so this relation is in BCNF.

PUBLISHER

- Functional Dependencies
  - {Pub_ID} → {Email, Phone, Co_Name}
- BCNF - Pub_ID leads to all of the other attributes as the superkey and with no other functional dependencies so this relation is in BCNF.

CUSTOMER

- Functional Dependencies
  - {Cust_ID} → {Fname, Lname, Address_ID, Email}
- BCNF - It was assumed that no Fname and Lname pairs were unique. Cust_ID leads to all of the other attributes as the superkey and with no other functional dependencies so this relation is in BCNF.

ORDER

- Functional Dependencies
  - {Order_Num, ISBN} → {Date_Bought, Quantity, Cust_ID}
- BCNF - The pair Order_Num and ISBN leads to all of the other attributes as the superkey and with no other functional dependencies so this relation is in BCNF.

STORED_IN

- Functional Dependencies
  - {Warehouse_num, ISBN} → {Address_ID, Year, Title, Price, Rating, Stock, Category, Pub_Date, Pub_ID}
- BCNF - The pair Warehouse_num and ISBN leads to all of the other attributes as the superkey and with no other functional dependencies so this relation is in BCNF.

WAREHOUSE

- Functional Dependencies
  - {Warehouse_num} → {Address_ID}
- BCNF - It was assumed that a Address_ID could correspond to multiple Warehouse_num's. Warehouse_num leads to all of the other attributes as the superkey and with no other functional dependencies so this relation is in BCNF

EMPLOYEE

- Functional Dependencies
  - {SSN} → {Fname, Lname, Salary, Warehouse_Num, Address_ID}
- BCNF - It was assumed that no Fname and Lname pairs were unique. SSN leads to all of the other attributes as the superkey and with no other functional dependencies so this relation is in BCNF.

ADMINISTRATOR

- Functional Dependencies
  - {SSN} → {Salary, Fname, Lname}
- BCNF - SSN leads to all of the other attributes as the superkey and with no other functional dependencies so this relation is in BCNF.

PAYMENT_INFO

- Functional Dependencies
  - {Payment_Info} → {Cust_ID}
- BCNF - It was assumed that a Cust_ID could correspond to multiple Payment_Info's. Payment_Info leads to all of the other attributes as the superkey and with no other functional dependencies so this relation is in BCNF.

ADDRESS

- Functional Dependencies
  - {Address_ID} → {Street, City, State, ZIP}
  - {ZIP} → {City, State}
- BCNF - Address_ID leads to all of the other attributes as the superkey and with no other functional dependencies so this relation is in BCNF.

Database Indexes

Book Indices

title_index

CREATE INDEX title_index ON BOOK(Title) USING HASH;

We want to use a hash index on the titles of the books because hash indices are best used for equality testing, and people will most likely be searching for a particular title of a book rather than a range of book titles.

price_index
CREATE INDEX price_index ON BOOK(Price) USING BTREE;
We want to use a tree index on the price of books because it is likely that users will be searching for books within a certain price range, and trees are best suited for searching a range of values.

rating_index
CREATE INDEX rating_index ON BOOK(Rating) USING BTREE;
We want to use a tree index on the rating of books because it is likely that users will be searching for books that are better than three stars, for example, and trees are best suited for searching a range of values.

category_index
CREATE INDEX category_index ON BOOK(Category) USING HASH;
We want to use a hash index on the category of the books because hash indices are best used for equality testing, and people will most likely be searching for a books in a certain category.

authName_index
CREATE INDEX authName_index ON AUTHOR(Fname, Lname) USING HASH;
We want to use a hash index on the name of the author because hash indices are best used for equality testing, and people will most likely be searching for a particular author.

pubName_index
CREATE INDEX pubName_index ON PUBLISHER(Name) USING HASH;
We want to use a hash index on the name of the publisher because hash indices are best used for equality testing, and people will most likely be searching for a particular publisher.

custName_index
CREATE INDEX custName_index ON CUSTOMER(Fname, Lname) USING HASH;
We want to use a hash index on the name of the customer because hash indices are best used for equality testing, and people will most likely be searching for a particular person.

dateOrdered_index
CREATE INDEX dateOrdered_index ON ORDERS(Date_Bought) USING BTREE;

We want to use a tree index on the dates of orders because tree indices are best used for searching ranges, and we will most likely be searching for orders that occurred before or after certain dates.

empSal_index
CREATE INDEX empSal_index ON EMPLOYEE(Salary) USING BTREE;
We want to use a tree index on the salary of employees because tree indices are best used for searching ranges, and we will most likely be searching for salaries that are greater or less than a certain criteria.

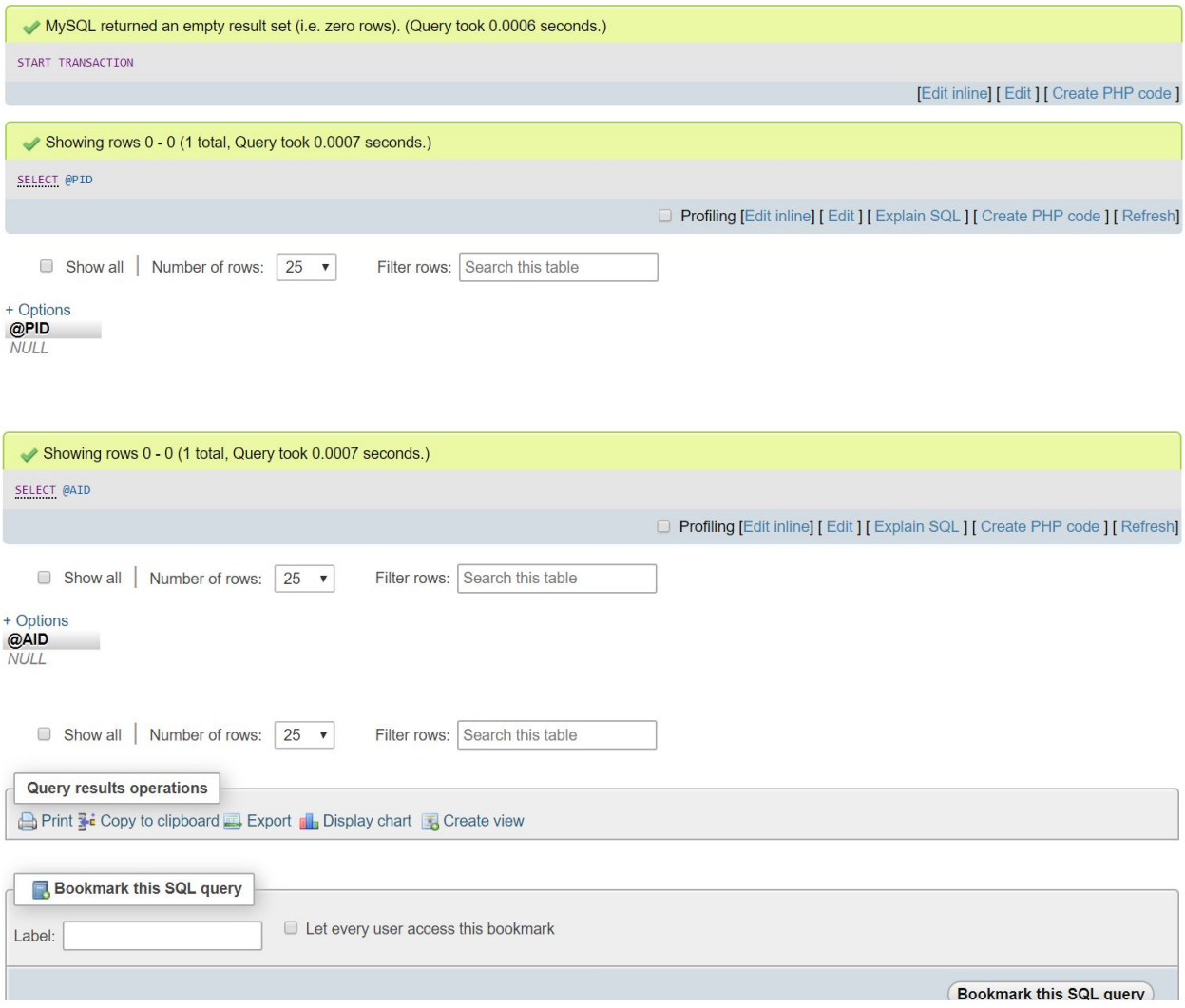
empName_index
CREATE INDEX empName_index ON EMPLOYEE(Fname, Lname) USING HASH;
We want to use a hash index on the name of the employee because hash indices are best used for equality testing, and we will likely be searching for a particular employee.

city_index
CREATE INDEX city_index ON ADDRESS(City, State) USING HASH;
We want to use a hash index on the city and state in the address table because hash indices are best used for equality testing, and we will likely be searching for a particular city.

Sample Transactions

```
START TRANSACTION;
SELECT @PID;
SELECT @AID;
SELECT @PID:= COUNT(*) FROM PUBLISHER;
SET @PID := @PID + 1;
INSERT INTO PUBLISHER VALUES(@PID, 'consumerservices@penguinrandomhouse.com', 2124143581, 'G.P. Putnam\'s Sons');
INSERT INTO BOOK VALUES (0735219117, 2018, 'Where the Crawdads Sing', 15.40, 5, 10, ‘08/14/2018’, @PID, 0, 'Young Adult', 0);
SELECT @AID:= COUNT(*) FROM AUTHOR;
SET @AID= @AID + 1;
INSERT INTO AUTHOR VALUES(@AID, 'Delia', 'Owens', 'A great book writer!');
INSERT INTO WRITTEN_BY VALUES(@AID, 0735219117);
INSERT INTO STORED_IN VALUES(0735219117, 2);
COMMIT;
END TRANSACTION
```

This transaction adds a new book, *Where the Crawdads Sing*, to the database. The first thing it does is check to see how many publishers there are and whether or not the publisher for the book is already in the PUBLISHER table. If the publisher isn't already in the table, then it is added. It then adds in the book to the BOOK table. Next, it checks to see whether or not Delia Owens is already in the AUTHOR table, and if she isn't, she is added. Finally, the WRITTEN_BY table is updated so that there is a relationship between *Where the Crawdads Sing* and Delia Owens, and also the STORED_IN table is updated so that the books are stored in warehouse 2.



Your SQL query has been executed successfully.

```
SELECT @PID:= COUNT(*) FROM PUBLISHER
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

+ Options

@PID:= COUNT(*)
10

Query results operations

Print Copy to clipboard Export Display chart Create view

Bookmark this SQL query

Label: Let every user access this bookmark

Bookmark this SQL query

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0006 seconds.)

```
SET @PID := @PID + 1
```

[Edit inline] [Edit] [Create PHP code]

1 row inserted. (Query took 0.0078 seconds.)

```
INSERT INTO PUBLISHER VALUES(@PID, 'consumerservices@penguinrandomhouse.com', 2124143581, 'G.P. Putnam\'s Sons')
```

[Edit inline] [Edit] [Create PHP code]

1 row inserted. (Query took 0.0045 seconds.)

```
INSERT INTO BOOK VALUES (0735219117, 2018, 'Where the Crawdads Sing', 15.40, 5, 10, '08/14/2018', @PID, 0, 'Young Adult', 0, 0000-00-00)
```

[Edit inline] [Edit] [Create PHP code]

Warning: #1265 Data truncated for column 'PUB_DATE' at row 1

Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available.

Your SQL query has been executed successfully.

```
SELECT @AID:= COUNT(*) FROM AUTHOR
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

+ Options

@AID:= COUNT(*)
20


```
START TRANSACTION;
SET @CID := 0;
SET @AID := 0;
SET @ONum := 0;
SELECT @CID := COUNT(*) FROM CUSTOMER;
SET @CID := @CID + 1;
SELECT @AID := COUNT(*) FROM ADDRESS;
SET @AID := @AID + 1;
INSERT INTO ADDRESS VALUES('188 E Oakland Ave', 'Columbus', 'Ohio', 43201,@AID);
INSERT INTO CUSTOMER VALUES(@CID, 'Pat', 'Walter', @AID, 'walter.373@osu.edu');
INSERT INTO PAYMENT_INFO VALUES(123412341234, @CID);
SELECT @ONum := COUNT(*) FROM ORDERS;
SET @ONum := @ONum + 1;
INSERT INTO ORDERS VALUES(@ONum, '04/21/2019', 2, 0735219117, @CID);
UPDATE BOOK SET Stock = Stock - 2 WHERE ISBN = 0735219117;
COMMIT;
```

This transaction adds a new customer with its payment information and address to the database. The new customer then buys two copies of *Where the Crawdads Sing*. This order is then added to the ORDER table, and finally the stock of *Where the Crawdads Sing* is updated to reflect that two books have just been bought.

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0004 seconds.)

```
START TRANSACTION
```

[Edit inline] [Edit] [Create PHP code]

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0004 seconds.)

```
SET @CID := 0
```

[Edit inline] [Edit] [Create PHP code]

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0005 seconds.)

```
SET @AID := 0
```

[Edit inline] [Edit] [Create PHP code]

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0007 seconds.)

```
SET @ONum := 0
```

[Edit inline] [Edit] [Create PHP code]

Your SQL query has been executed successfully.

```
SELECT @CID := COUNT(*) FROM CUSTOMER
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

+ Options

@CID := COUNT(*)
6

Query results operations

Print Copy to clipboard Export Display chart Create view

Bookmark this SQL query

Label:

Let every user access this bookmark

Bookmark this SQL query

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0005 seconds.)

```
SET @CID := @CID + 1
```

[Edit inline] [Edit] [Create PHP code]

Your SQL query has been executed successfully.

```
SELECT @AID := COUNT(*) FROM ADDRESS
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

+ Options

@AID := COUNT(*)
18

Query results operations

Print Copy to clipboard Export Display chart Create view

Bookmark this SQL query

Label: Let every user access this bookmark

Bookmark this SQL query

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0005 seconds.)

```
SET @AID := @AID + 1
```

[Edit inline] [Edit] [Create PHP code]

1 row inserted. (Query took 0.0041 seconds.)

```
INSERT INTO ADDRESS VALUES('188 E Oakland Ave', 'Columbus', 'Ohio', 43201,@AID)
```

[Edit inline] [Edit] [Create PHP code]

1 row inserted. (Query took 0.0024 seconds.)

```
INSERT INTO CUSTOMER VALUES(@CID, 'Pat', 'Walter', @AID, 'walter.373@osu.edu')
```

[Edit inline] [Edit] [Create PHP code]

1 row inserted. (Query took 0.0031 seconds.)

```
INSERT INTO PAYMENT_INFO VALUES(123412341234, @CID)
```

[Edit inline] [Edit] [Create PHP code]

Your SQL query has been executed successfully.

```
SELECT @ONum := COUNT(*) FROM ORDERS
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

+ Options

@ONum := COUNT(*)
3

Query results operations

Print Copy to clipboard Export Display chart Create view

Bookmark this SQL query

Label: Let every user access this bookmark

Bookmark this SQL query

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0005 seconds.)

```
SET @ONum := @ONum + 1
```

[Edit inline] [Edit] [Create PHP code]

1 row inserted. (Query took 0.0029 seconds.)

```
INSERT INTO ORDERS VALUES(@ONum, '04/21/2019', 2, 0735219117, @CID)
```

[Edit inline] [Edit] [Create PHP code]

Warning: #1265 Data truncated for column 'DATE_BOUGHT' at row 1

1 row affected. (Query took 0.0026 seconds.)

```
UPDATE BOOK SET Stock = Stock - 2 WHERE ISBN = 0735219117
```

[Edit inline] [Edit] [Create PHP code]

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0026 seconds.)

```
COMMIT
```

[Edit inline] [Edit] [Create PHP code]

START TRANSACTION;
INSERT INTO EMPLOYEE VALUES(123121234, ‘Patt’, ‘Walterr’, 0, 64005, 2, 4);
UPDATE EMPLOYEE SET Salary = Salary + 5000 WHERE Salary < 70000;
UPDATE EMPLOYEE SET Salary = Salary - 2500 WHERE SALARY >= 75000;
COMMIT;
END TRANSACTION

First, this transaction inserts a new employee into the table. This transaction then updates the salaries of employees. Everyone making less than 70 grand gets a 5 grand bonus, while those making over 75 grand get a 2.5 grand deduction from their salary.

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0006 seconds.)

```
START TRANSACTION
```

[Edit inline] [Edit] [Create PHP code]

1 row inserted. (Query took 0.0083 seconds.)

```
INSERT INTO EMPLOYEE VALUES(123121234, 'Patt', 'Walterr', 0, 64005, 2, 4)
```

[Edit inline] [Edit] [Create PHP code]

4 rows affected. (Query took 0.0022 seconds.)

```
UPDATE EMPLOYEE SET Salary = Salary + 5000 WHERE Salary < 70000
```

[Edit inline] [Edit] [Create PHP code]

2 rows affected. (Query took 0.0015 seconds.)

```
UPDATE EMPLOYEE SET Salary = Salary - 2500 WHERE SALARY >= 75000
```

[Edit inline] [Edit] [Create PHP code]

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0036 seconds.)

```
COMMIT
```

[Edit inline] [Edit] [Create PHP code]

Section II: User Manual

Entity Representations

BOOK

Contains the information about each individual book that the Bits & Books have ever had or currently have in stock as well as its identifying information.

- ISBN - (INT) this is the primary key of this relation and is a unique commercial book identifier
- Year - (INT) this is the year that the book was first written
- Title - (VARCHAR(50)) the title of the book
- Price - (FLOAT) the price assigned to the book by Bits & Books
- Rating - (INT) the one to five star rating the book has been assigned by customers
- Stock - (INT) the number of books across all warehouses
- Category - (VARCHAR(30)) this is the genre the book belongs in
- Pub_Date - (DATE) the date that current book edition was published
- Pub_ID - (INT) this is a foreign key to the PUBLISHER relation and represents the publishing company that published this book

WRITTEN_BY

Contains a given author as well as the corresponding ISBN number of what book they have written.

- Auth_ID - (INT) this is a foreign key to the AUTHOR relation and is a unique identification number for an author
- ISBN - (INT) this is a foreign key to the BOOK relation and is a commercial book identifier

AUTHOR

Contains all of the authors that have written a book that has been or currently is in stock at Bits & Books as well as some of their identifying information.

- Auth_ID - (INT) this is the primary key of this relation and is a unique identification number for an author
- Fname - (VARCHAR(30)) this is the first name of the author
- Lname - (VARCHAR(30)) this is the last name of the author
- Bio - (VARCHAR(50)) this a short description about the author

PUBLISHER

Contains all of the publishers that have provide a book as well as some identifying information.

- Pub_ID - (INT) this is the primary key of this relation and represents the publishing company that published this book
- Email - (VARCHAR(30)) the email of the publishing company's point of contact
- Phone - (CHAR(11)) the phone number of the publishing company's point of contact
- Co_Name - (VARCHAR(30)) this is the company name of the publishing company

CUSTOMER

Contains all of the customers that have purchased a book as well as some of their identifying information.

- Cust_ID - (INT) this is the primary key of this relation and is a unique identifier for a given customer
- Fname - (VARCHAR(30)) this is the first name of the customer
- Lname - (VARCHAR(30)) this is the last name of the author
- Address_ID - (INT) this is a foreign key to the ADDRESS relation and represents the address of the customer
- Email - (VARCHAR(30)) the email of the customer

ORDER

Contains all of the orders that have everything ver been placed as well as the corresponding book and the customer who made the purchase.

- Order_Num - (INT) this is the primary key of this relation and a unique identifier for every order made
- Date_Bought - (DATE) the date that the order was completed
- Quantity - (INT) the number of books purchased in the order
- ISBN - (INT) this is a foreign key to the BOOK relation and is a unique commercial book identifier
- Cust_ID - (INT) this is is a unique identifier for a given customer

STORED_IN

Contains a given warehouse where books are stored as well as the corresponding ISBN number of the book they have stored there.

- ISBN - (INT) this is a foreign key to the BOOK relation and is a unique commercial book identifier
- Warehouse_num - (INT) this is a foreign key to the WAREHOUSE relation and is a unique identifier of the warehouse the book is stored in

WAREHOUSE

Contains all of the warehouses where the books are stored as well as some of its identifying information.

- Warehouse_num - (INT) this is the primary key of this relation and is a unique identifier of the warehouse the book is stored in
- Address_ID - (INT) this is a foreign key to the ADDRESS relation and represents the address of the warehouse

EMPLOYEE

Contains all of the employees that work at a warehouse as well as some of their identifying information and work information.

- SSN - (INT) this is the primary key of this relation and is the employee's social security number
- Fname - (VARCHAR(30)) this is the first name of the employee
- Lname - (VARCHAR(30)) this is the last name of the author

- Salary - (INT) the amount of money the employee makes in a year
- Warehouse_Num - (INT) a unique identifier of the warehouse the book is stored in
- Address_ID - (INT) this is a foreign key to the ADDRESS relation and represents the address of the employee

ADMINISTRATOR

Contains all of the employees that are administrators at a warehouse as well as some of their identifying information and work information.

- Salary - (INT) the amount of money the administrator makes in a year
- Fname - (VARCHAR(30)) this is the first name of the administrator
- Lname - (VARCHAR(30)) this is the last name of the author
- SSN - (INT) this is the primary key of this relation and is the employee's social security number

PAYMENT_INFO

Contains a given customer as well as the corresponding payment info that they have used to place an order.

- Payment_Info - (VARCHAR(19)) this is the primary key of this relation and is the card number a customer uses
- Cust_ID - (INT) this is a unique identifier for a given customer

ADDRESS

Contains all of the address information for the warehouses, employees, administrators, customers, and orders.

- Street - (VARCHAR(30)) this is the name of the house number and street name the address is in
- City - (VARCHAR(15)) this is the name of the city the address is in
- State - (VARCHAR(12)) this is the name of the state the address is in
- ZIP - (INT) this is the zip code of the address
- Address_ID - (INT) this is the primary key of the relation and represents the address of the customer

Sample SQL Queries:

- Find the titles of all books by Pratchett that cost less than $10
  - TABLE1 ← ($\sigma_{\text{Lname = 'Pratchett'}}$(BOOK * (AUTHOR* WRITTEN BY)))
  - TABLE2 ← ($\sigma_{\text{Price < 10}}$(TABLE1)
  - RESULT ← π_{Title}(TABLE2)

  - ```
    SELECT B.Title
    FROM BOOK AS B, AUTHOR AS A, WRITTEN_BY AS W
    WHERE B.Price <10 AND B.ISBN = W.ISBN
    AND W.Auth_ID = A.Auth_ID AND A.FLNAME = "PRATCHETT";
    ```
    ```

- Give all the titles and their dates of purchase made by a single customer (you choose how to designate the customer)
  - $\pi_{\text{Title}}(\pi_{\text{Date\_Bought}}(\sigma_{\text{CustID} = \text{`00001'}}(\text{ORDER}))$

  - ```
    SELECT B.Title, O.DATE_BOUGHT
    FROM BOOK AS B, CUSTOMER AS C, ORDERS AS O
    WHERE B.ISBN = O.ISBN
    AND O.Cust_ID = C.Cust_ID
    AND C.Cust_ID = 1;
    ```
- Find the titles and ISBNs for all books with less than 5 copies in stock
  - $\pi_{\text{Title, ISBN}}(\sigma_{\text{Stock} < 5}(\text{BOOK}))$

  - ```
    SELECT  B.Title, B.ISBN
    FROM BOOK AS B
    WHERE B.stock < 5;
    ```
- Give all the customers who purchased a book by Pratchett and the titles of Pratchett books they purchased
  - Book_AuthorID $\leftarrow \pi_{\text{Cust\_ID, Title, Auth\_ID}}(\text{ORDER*WRITTEN BY})$
    Author_Name $\leftarrow \pi_{\text{Cust\_ID, Title, ALname}}(\text{BOOK\_AUTHORID}*\rho(\text{Lname, ALName})\text{AUTHOR})$
    RESULT $\leftarrow \pi_{\text{Fname, Lname, Title}}(\text{CUSTOMER} *(\sigma_{\text{ALname} = \text{“Pratchett”}}(\text{AUTHOR\_NAME}))$

  - ```
    SELECT C.Fname, C.Lname, B.Title
    FROM CUSTOMER AS C, BOOK AS B, WRITTEN_BY AS W, ORDERS AS O,
    AUTHOR AS A
    WHERE B.ISBN = O.ISBN
    AND C.Cust_ID = O.Cust_ID
    AND B.ISBN= W.ISBN
    AND W.Auth_ID = A.AUTH_ID
    AND A.LNAME = "Pratchett";
    ```

- Find the total number of books purchased by a single customer (you choose how to designate the customer)
  - TABLE1 $\leftarrow \sigma_{\text{cust\_id} = \text{“123456”}}(\text{CUSTOMER})$
    RESULT $\leftarrow \mathbf{F}_{\text{COUNT ISBN}}(\text{TABLE1} \bowtie_{\text{CUSTOMER.cust\_id} = \text{ORDER.cust.id}} \text{ORDER}))$

  - ```
    SELECT SUM(O.quantity)
    FROM ORDERS AS O, CUSTOMER AS C, BOOK AS B
    WHERE B.ISBN = O.ISBN
    AND C.Cust_ID = O.Cust_ID
    AND C.Cust_ID = 1;
    ```

- Find the customer who has purchased the most books and the total number of books they have purchased
  - PURCHASED ← CUSTOMER $\bowtie_{CUSTOMER.cust\_id = ORDER.cust.id}$ ORDER
    MAX_TABLE ← $_{cust\_ID}\mathbf{F}_{COUNT\ ISBN}$(PURCHASED)
    RESULT ← $_{cust\_ID}\mathbf{F}_{Max\ count\_isbn}$(MAX_TABLE)

  - ```
    SELECT TT.Lname, MAX(TT.total)
    FROM (SELECT C.LName AS LName, SUM(O.Quantity) AS total
    FROM CUSTOMER AS C, BOOK AS B, ORDERS AS O
    WHERE C.Cust_ID = O.Cust_ID
    AND B.ISBN = O.ISBN
    GROUP BY C.LName) AS TT;
    ```

- Select all books written by authors with the last name "Smith"
  - TABLE1 ← $\boldsymbol{\sigma}_{last\_name = \text{"Smith"}}$(AUTHOR)
    RESULT ← TABLE1 $\bowtie_{TABLE1..auth\_ID = WRITTEN\_BY.auth\_ID}$ WRITTEN_BY

  - ```
    SELECT B.ISBN
    FROM BOOK AS B, WRITTEN_BY AS W, AUTHOR AS A
    WHERE B.ISBN = W.ISBN AND A.AUTH_ID = W.AUTH_ID
    AND A.Lname = 'Smith';
    Find the average price of all books
    SELECT AVG(B.Price)
    FROM BOOK AS B;
    ```

- Find the average price of all books
  - $\mathbf{F}_{AVERAGE\ price}$(BOOK)

  - ```
    SELECT AVG(B.Price)
    FROM BOOK AS B;
    ```

- Customers who bought books that were stored in Warehouse "2"
  - TABLE1 ← BOOK $\bowtie_{ISBN = ISBN\_num}$ STORED_IN
    TABLE2 ← TABLE1$\bowtie_{Table1.Warehouse\_num = Warehouse.warehouse\_num}$ WAREHOUSE
    TABLE3 ← TABLE2$\bowtie_{Table1.Cust\_ID = Customer.Cust\_ID}$ CUSTOMER
    RESULT ← $\boldsymbol{\pi}_{Fname,\ Lname}(\boldsymbol{\sigma}_{Warehouse\_num = \text{"2"}}$(TABLE3))

  - ```
    SELECT C.Fname, C.Lname
    FROM CUSTOMER AS C, BOOK AS B, STORED_IN AS S
    WHERE C.Cust_ID = B.Cust_ID AND B.ISBN = S.ISBN_num
    AND S.Warehouse_num = 2;
    ```

INSERT Syntax:

BOOK

- Required Attributes - ISBN, Year, Title, Price, Rating, Stock, Pub_Date, Pub_ID, Category
- Dependencies - There must be a Pub_ID in the PUBLISHER relation that matches the Pub_ID required for a book first. It's also good practice to add the associated Auth_ID and ISBN to the WRITTEN_BY relation and the associated ISBN and Warehouse_Num to STORED_IN relation.
- Sample Insert:

```
/*INSERT BOOK "ABC'S" WRITTEN BY AUTH_ID 1 AND
WAREHOUSE 1, PUBLISHER 1*/

INSERT INTO BOOK(ISBN, YEAR, TITLE, PRICE, RATING, STOCK, PUB_DATE,
PUB_ID, CATEGORY)
VALUES (757575757, 2018, "ABC'S", 10.00, 5, 3, "2016-4-23", 1, "Kids");

INSERT INTO WRITTEN_BY(AUTH_ID, ISBN)
VALUES (1, 757575757);

INSERT INTO STORED_IN(ISBN, WAREHOUSE_NUM)
VALUES (757575757,1);
```

AUTHOR

- Required Attributes - Auth_ID, Fname, Lname
- Dependencies - N/A
- Sample Insert:

```
/*INSERT author named Billy Bob, hasn't
written anything yet*/

INSERT INTO AUTHOR (AUTH_ID, FNAME, LNAME, BIO)
VALUES (22, "Billy", "Bob", "I like pizza");
```

PUBLISHER

- Required Attributes - Pub_ID, Phone, Co_Name
- Dependencies - N/A
- Sample Insert:

```
/*INSERT A NEW PUBLISHER*/
INSERT INTO PUBLISHER (PUB_ID,EMAIL,PHONE,CO_NAME)
VALUES (9, "OSU@publishing.com", "16149999999", "OSU")
```

CUSTOMER

- Required Attributes - Cust_ID, Fname, Lname, Address_ID, Email

- Dependencies - There must be a Address_ID in the ADDRESS relation that matches the Address_ID required for a customer first.
- Sample Insert

```
INSERT INTO ADDRESS (ADDRESS_ID,STREET,CITY,STATE,ZIP)
VALUES (21, "222 rock street.", "Cincinnati", "Ohio", 45216);

INSERT INTO CUSTOMER (CUST_ID, FNAME, LNAME, ADDRESS_ID, EMAIL)
VALUES (8, "Karen", "Jacobs", 21, "Jacobs@abc.com");

INSERT INTO PAYMENT_INFO(PAYMENT_INFO, CUST_ID)
VALUES ("7432798143978143", 8);
```

DELETE Syntax:

BOOK

- Dependencies - If a book is being removed then the tuples with corresponding ISBN in the ORDERS relation, WRITTEN_BY relation, and STORED_IN relation must be deleted.
- Sample Delete:
    - DELETE FROM book
      WHERE ISBN=72227710

AUTHOR

- Dependencies - If an author is being removed then the tuples with corresponding Auth_ID in the WRITTEN_BY relation must be deleted and the tuples with corresponding ISBN in the BOOK relation.
- Sample Delete:
    - DELETE FROM AUTHOR
      WHERE AUTH_ID = 4

PUBLISHER

- Dependencies - If a publisher is being removed then the tuples with corresponding Pub_ID in the BOOK relation must be deleted.
- Sample Delete:
    - DELETE FROM PUBLISHER
      WHERE PUB_ID = 4

CUSTOMER

- Dependencies - If a customer is being removed then the tuples with corresponding Cust_ID in the PAYMENT_INFO relation and ORDERS relation must be deleted.
- Sample Delete:
    - DELETE FROM CUSTOMER

WHERE CUST_ID = 4

Section III: Graded Checkpoint Documents

Checkpoint 1

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 - first name, last name, address, payment info, customer #
Employee - first name, last name, employee #, salary/ hourly rate, SSN, address
Administrator/Manager - first name, last name, salary
Book - ISBN #, year, category, price, # in stock, rating
Publisher - publication date, contact info - email, phone
Author - first name, last name, ID#

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 purchase BOOK entities
- BOOKS entities ordered from PUBLISHER
- ADMINISTRATORS review SALES
- ADMINISTRATORS are a subgroup of EMPLOYEE
- AUTHOR writes BOOKS
- PUBLISHER publishes BOOKS

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.

- Warehouse - address, warehouse number
- Orders - order #, date bought, quantity

A warehouse would be useful to the stakeholders because it allows them to keep track of how many books they have and where some of their employees may work. The orders entity would be useful to keep track customers placing orders for books. Not only would

customers and order have a relationship, but so would the orders and warehouse would have a relationship.

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 report of all sales that have occurred in the last month
- A report of a customer's purchases
- A report of books that may need to be restocked soon based on recent purchase histories
  4. A report of employees where they work and who manages them.
- 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.

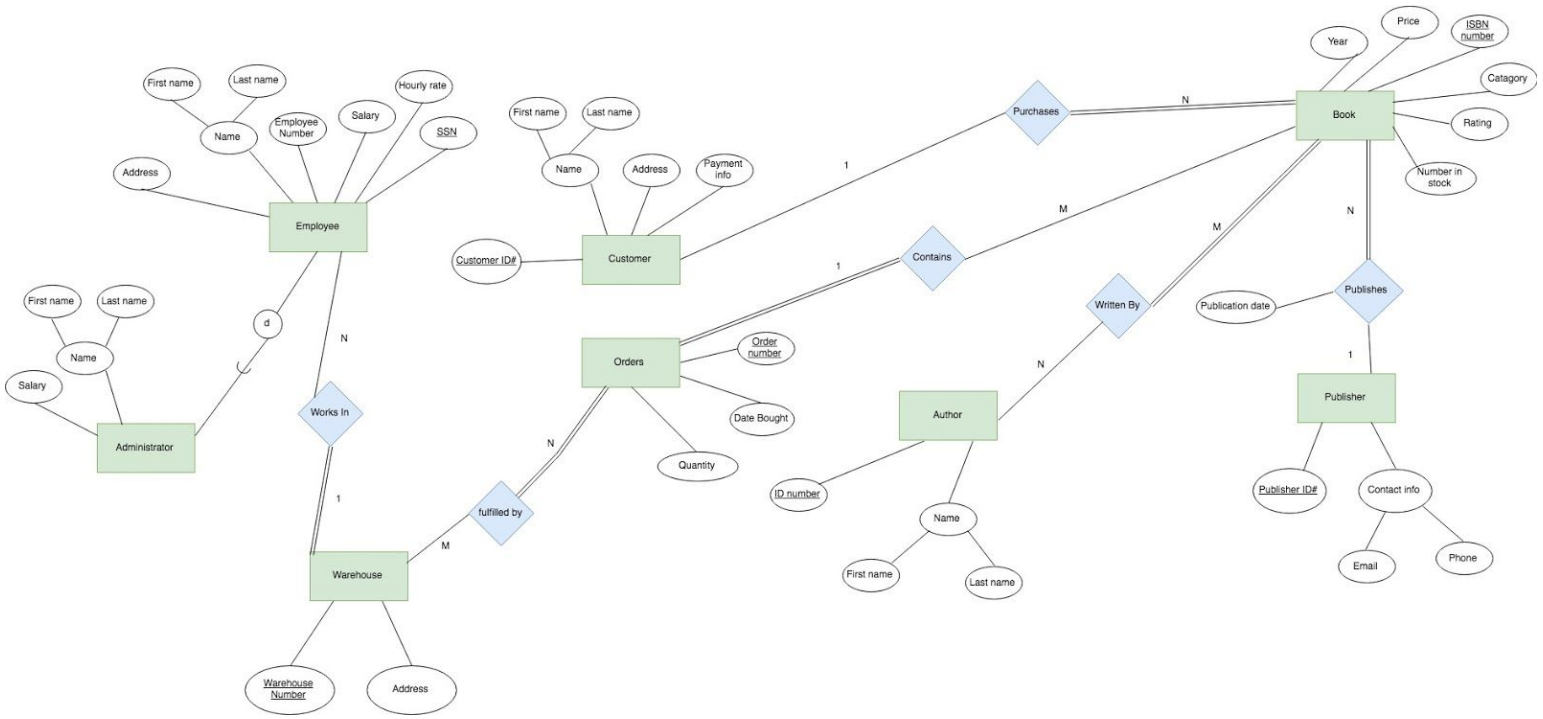
5. Suppose we want to add a new publisher to the database. How would we do that given the entities and relationships you've outlined above? Given your above description, is it possible to add a new publisher to your database without knowing the title of any books they have published? If not, revise your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

Yes. Since the publishers are separate from the book entities, you can add a publisher without having to first create a new book entry. The new publisher can have a relationship with books published, but the title isn't needed to just add a new publisher.

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. A customer wants to leave a review online for a book.
    a. We would have to add a review attribute for the book entity.
2. A customer wants to know more details about book dimensions and amount of pages.
    a. We could have to add dimensions and number of pages for the book entity.
3. We want to add a rewards system where every 10th book a customer buys they receive a free book.
    a. We would have to add a customer information attribute (how many books they've purchased) under the orders attribute.

7.



Comments from grader:

AUTHOR
- must have bio attribute
CUSTOMER
-make sure customers can have multiple forms of payment too
PUBLISHER
- No publisher name?
- How is "contact info - email, phone" a cohesive attribute?
---> ok looks like on the schema it makes sense, but that's incorrect form in part 1 in describing those attributes
Overall fantastic job on entity/attributes, got most of the necessary stuff
2. All the relationships look solid
3. Great proposed entities, those should work
4. Looks good
5.You never explained how by just saying yes, nbd tho.
6. This isn't really the intention of the question. Update operations are adding, deleting, or modifying current entities. Like modifying a datarow, not modifying the actual entity itself via new attributes. So for future checkpoints a correct answer for this is like "add a book, change a books publisher, create a order" I can 100% see how the question could be misinterpreted though so it's all good
ER Diagram
-make sure all primary keys are underlined, some are missing
-some should allow for multivalued attributes, like allow for multiple payment information under CUSTOMER
-I don't know if your warehouse to book setup makes sense, books could be at different warehouses but be within the same order and your schema wouldn't allow for that. Instead have warehouse be associated with books directly and order with books directly, but no direct link between warehouse and order. Unless you have a good case for otherwise and explain it

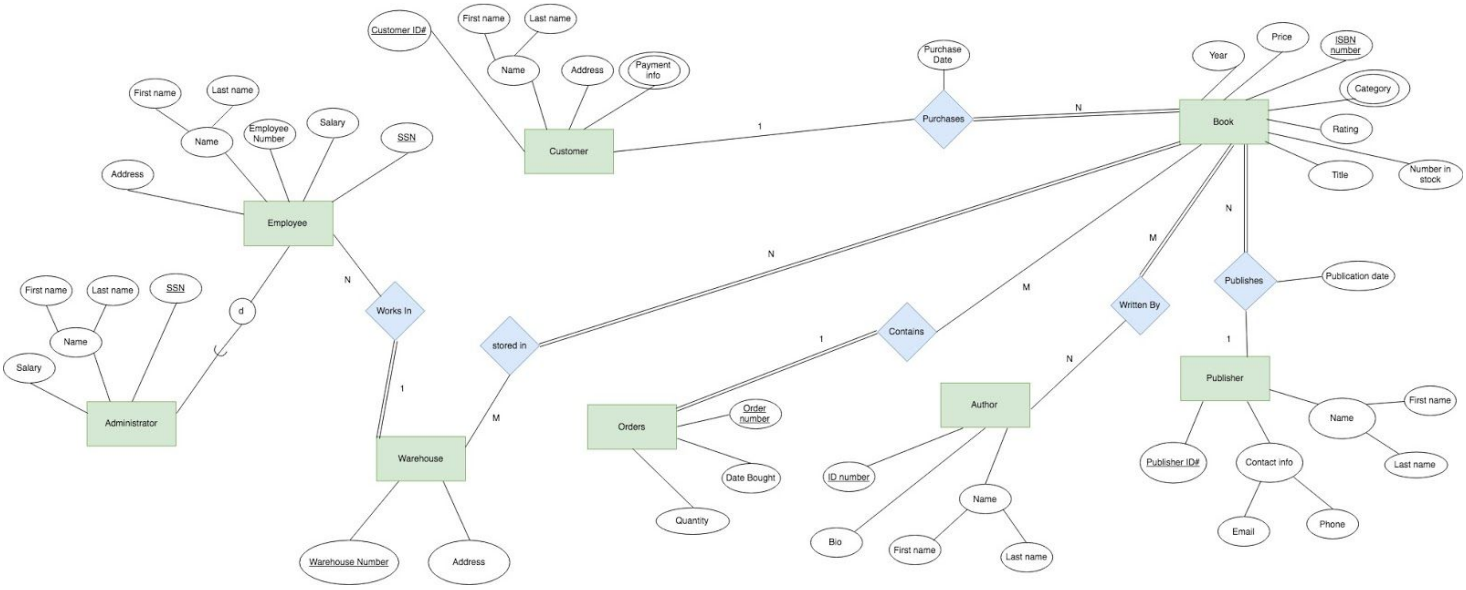
Overall super great project. Your schema is very clean and implementation should be easy. As I overlooked your comments initially, if any offered suggestions take undue time to change, just resubmit your checkpoint two a couple days later or whenever you get around to updating it.

Revisions:

- ***See next checkpoint for revised ER Diagram, or page 1, section 1, for final version. Includes all revisions about entities, attributes, and ER diagram.***

Checkpoint 2

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	Year	Title	Price	Rating	Stock	Pub_ date	Pub_ ID (FK)	Cust_ ID (FK)	O_ num (FK)	Purchase_ Date

WRITTEN BY

Auth_ID (FK)	ISBN (FK)

AUTHOR

Auth_ID	Fname	Lname	Bio

PUBLISHER

Pub_ID	Email	Phone	FName	LName

CUSTOMER

Cust_ID	Fname	Lname	Address	Email

ORDER

Order_num	Date_Bought	Quantity

STORED IN

ISBN_num	Warehouse_num

WAREHOUSE

Warehouse_num	Address

EMPLOYEE

SSN	Address	Fname	Lname	Emp_Num	Salary	WNum (FK)

ADMINISTRATOR

Salary	Fname	Lname	SSN (FK)

PAYMENT INFO

Payment_Info	Cust_ID (FK)

Category

ISBN_num (FK)	Category

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

$$\pi_{\text{Title}}(\sigma_{\text{Price} < 10}(\sigma_{\text{Lname} = \text{'Pratchett'}}(\text{BOOK} * (\text{AUTHOR* WRITTEN BY}))))$$

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

$$\pi_{\text{Title}}(\pi_{\text{Purchase\_Date}}(\sigma_{\text{CustID} = \text{'00001'}}(\text{BOOK}))$$

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

$$\pi_{\text{Title, ISBN}}(\sigma_{\text{Stock} < 5}(\text{BOOK}))$$

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

$$\text{Book\_AuthorID} \leftarrow \pi_{\text{Cust\_ID, Title, Auth\_ID}}(\text{BOOK*WRITTEN BY})$$

$$\text{Author\_Name} \leftarrow \pi_{\text{Cust\_ID, Title, ALname}}(\text{BOOK\_AUTHORID}*\rho(\text{Lname, ALName})\text{AUTHOR})$$

$$\pi_{\text{Fname, Lname, Title}}(\text{CUSTOMER} *(\sigma_{\text{ALname} = \text{"Pratchett"}}(\text{AUTHOR\_NAME}))$$

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

$$\mathbf{F}_{\text{COUNT ISBN}} (\sigma_{\text{cust\_id} = \text{"123456"}}(\text{CUSTOMER} \bowtie_{\text{CUSTOMER.cust\_id = BOOK.cust.id}} \text{BOOK}))$$

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

$$\text{PURCHASED} \leftarrow \text{CUSTOMER} \bowtie_{\text{CUSTOMER.cust\_id = BOOK.cust.id}} \text{BOOK}$$

$$\text{MAX\_TABLE} \leftarrow {}_{\text{cust\_ID}}\mathbf{F}\ \text{COUNT Isbn}(\text{PURCHASED})$$

$${}_{\text{cust\_ID}}\mathbf{F}\ \text{Max count\_isbn}(\text{MAX\_TABLE})$$

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.

- Select all books written by authors with the last name “Smith”

$$\text{TABLE1} \leftarrow \sigma_{\text{last\_name} = \text{“Smith”}}(\text{AUTHOR})$$

$$\text{TABLE1} \bowtie_{\text{table1.auth\_ID} = \text{written\_by.auth\_ID}} \text{WRITTEN\_BY}$$

- Find the average price of all books

$$\mathbf{F}_{\text{AVERAGE price}}(\text{BOOK})$$

- Customers who bought books that were stored in Warehouse “2”

$$\text{TABLE1} \leftarrow \text{BOOK} \bowtie_{\text{ISBN} = \text{ISBN\_num}} \text{STORED\_IN}$$

$$\text{TABLE2} \leftarrow \text{TABLE1} \bowtie_{\text{Table1.Warehouse\_num} = \text{Warehouse.warehouse\_num}} \text{WAREHOUSE}$$

$$\text{TABLE3} \leftarrow \text{TABLE2} \bowtie_{\text{Table1.Cust\_ID} = \text{Customer.cust\_ID}} \text{CUSTOMER}$$

$$\pi_{\text{Fname, Lname}}(\sigma_{\text{Warehouse\_num} = \text{“2”}}(\text{TABLE3}))$$

Comments from grader:

ER Diagram:
Attributes should be one word or joined by an underline
What's going on with your administrator, employee relationship? If there's a subclass/superclass why are there common attributes still in both and not only in the superclass?
Address attribute is insufficient, think about all the info you fill out for address when you have something shipped. Either make it its own entity with many attributes and link it to warehouse and customer (perhaps publisher too) or just expand out it into sub attributes. The former is highly recommended for the final project

Relational Schema:
Arrows aren't shown linking the foreign keys, make sure you have that for the final project
Otherwise good table notation

Algebra:
Logic needs split into many lines for readability. Assign smaller segments to variables that are well named. This allows for an easier flow of logic
Make formatting consistent with size and font and whatnot
Base logic does seem good! Just needs to be split out

Extra queries: These fulfill requirements!

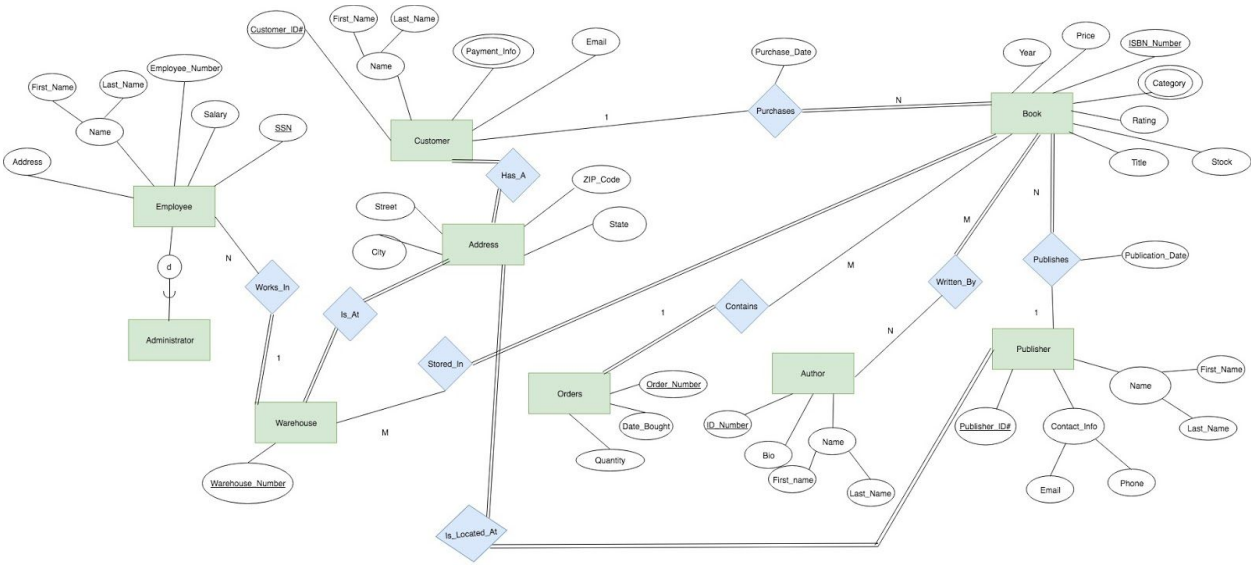
Overall good job, just some tweaks before turning in the final project at the end of the semester, follow those and you'll get get full credit on the final project, good job!!

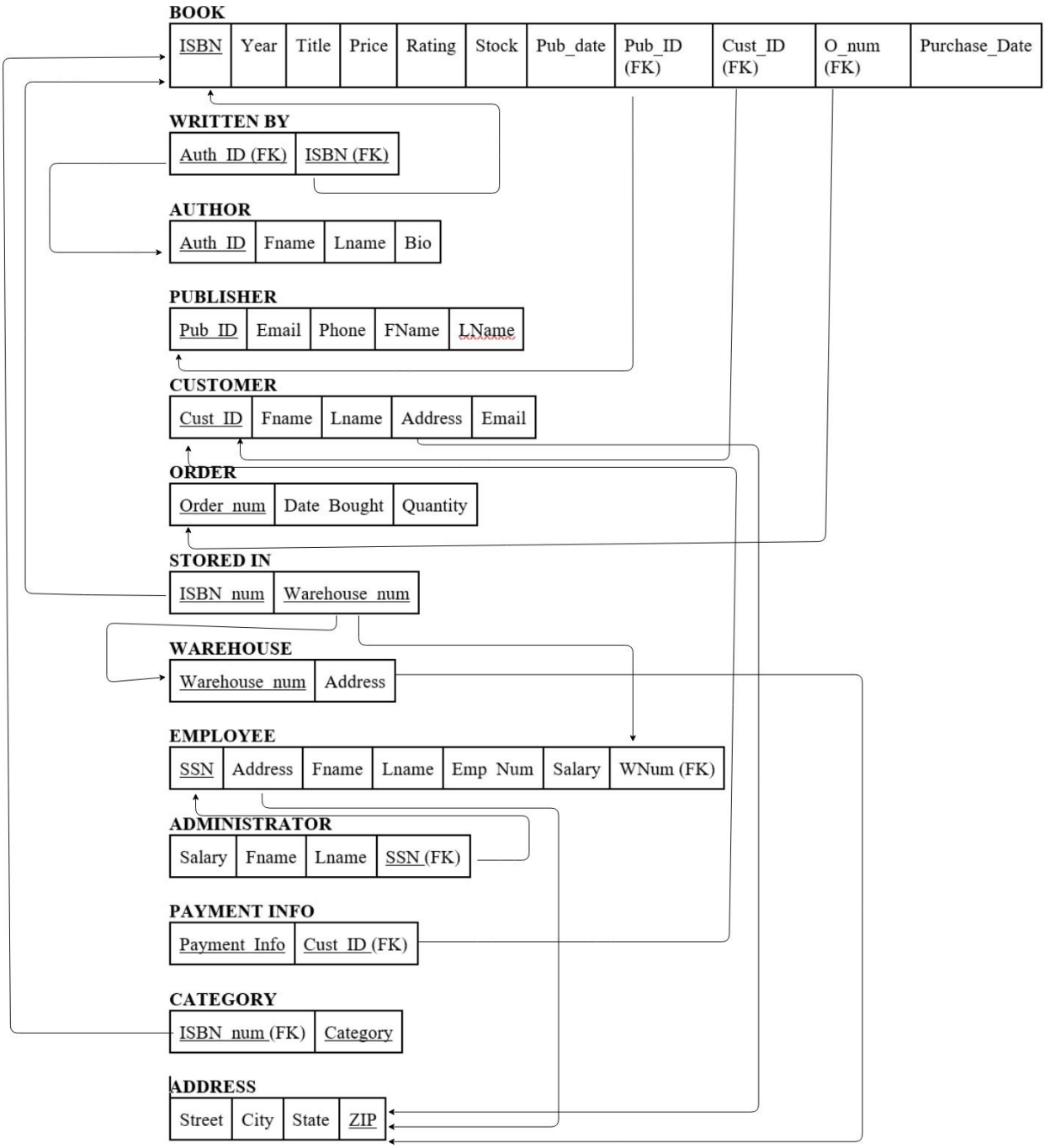
Revisions:

- ***See next checkpoint for revised ER Diagram and Relational Model.***
- ***See Part 1, Section 2, for Relational Algebra revisions***

Checkpoint 3

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.**





2. Given your relational schema, create a text file containing the SQL code to create your database. Use this SQL to create a database in MySQL within XAMPP.[1] The name of the database should be **bitsbooks**. 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.

In separate file called Database Creation Code.

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 B.Title
FROM BOOK AS B, AUTHOR AS A, WRITTEN_BY AS W
WHERE B.Price <10 AND B.ISBN = W.ISBN AND W.Auth_ID = A.Auth_ID AND A.Lname =
'Pratchett';
```

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

```
SELECT B.Title, B.Purchase_Date
FROM BOOK AS B, CUSTOMER AS C
WHERE B.Cust_ID = C.Cust_ID AND C.Cust_ID = 123456;
```

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

```
SELECT  B.Title, B.ISBN
FROM BOOK AS B
WHERE B.stock < 5;
```

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

```
SELECT C.Fname, C.Lname, B.Title
FROM CUSTOMER AS C, BOOK AS B, WRITTEN_BY AS W
WHERE C.Cust_ID = B.Cust_ID
AND B.ISBN= W.ISBN
AND W.Auth_ID = A.Auth_ID
AND A.Lname = 'Pratchett';
```

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

```
SELECT SUM(O.quantity)
FROM ORDER AS O, CUSTOMER AS C, BOOK AS B
WHERE C.Cust_ID = B.Cust_ID
AND B.O_num = O.Order_num
AND C.Cust_ID = 123456;
```

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

```
SELECT Lname, MAX(total)
FROM (SELECT C.LName AS LName, SUM(O.Quantity) AS total
FROM CUSTOMER AS C, BOOK AS B, ORDER AS O
WHERE C.Cust_ID = B.Cust_ID
AND B.O_num = O.Order_num
GROUP BY C.LName);
```

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".

- Select all books written by authors with the last name "Smith"

  $$\text{TABLE1} \leftarrow \sigma_{\text{Last\_Name = ``Smith''}}(\text{AUTHOR})$$

  $$\text{ANSWER} \leftarrow \text{TABLE1} \bowtie_{\text{TABLE1.Auth\_ID = WRITTEN\_BY..Auth\_ID}} \text{WRITTEN\_BY}$$

  ```
  SELECT B.ISBN
  FROM BOOK AS B, WRITTEN_BY AS W, AUTHOR AS A
  WHERE B.ISBN = W.ISBN
  AND A.AUTH_ID = W.AUTH_ID
  AND A.Lname = 'Smith';
  ```

- Find the average price of all books

  $$\mathbf{F}_{\text{AVERAGE price}}(\text{BOOK})$$

  ```
  SELECT AVG(B.Price)
  FROM BOOK AS B;
  ```

- Customers who bought books that were stored in Warehouse "2"

  $$\text{TABLE1} \leftarrow \text{BOOK} \bowtie_{\text{ISBN = ISBN\_num}} \text{STORED\_IN}$$

  $$\text{TABLE2} \leftarrow \text{TABLE1} \bowtie_{\text{Table1.Warehouse\_num = Warehouse.warehouse\_num}} \text{WAREHOUSE}$$

  $$\text{TABLE3} \leftarrow \text{TABLE2} \bowtie_{\text{Table1.Cust\_ID = Customer.cust\_ID}} \text{CUSTOMER}$$

$$ANSWER \leftarrow \pi_{Fname,\ Lname}(\sigma_{Warehouse\_num = \text{“2”}}(TABLE3))$$

```
SELECT C.Fname, C.Lname
FROM CUSTOMER AS C, BOOK AS B, STORED_IN AS S
WHERE C.Cust_ID = B.Cust_ID AND B.ISBN = S.ISBN_num AND S.Warehouse_num = 2;
```

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 C.Fname, C.Lname, SUM(B.Price)
FROM CUSTOMER AS C, BOOK AS B
WHERE C.Cust_ID = B.Cust_ID
GROUP BY C.Lname;
```

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

```
SELECT C.Fname, C.Lname, C.Email
FROM CUSTOMER AS C, BOOK AS B
WHERE(SELECT AVG(B.Price)
        FROM CUSTOMER AS C, BOOK AS B
        WHERE C.Cust_ID = B.Cust_ID GROUP BY C.Lname))>SUM(B.Price)
        AND C.Cust_ID = B.Cust_ID
GROUP BY C.Lname
```

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 B.title, SUM(O.Quantity)
FROM BOOK AS B, ORDER AS O
WHERE B.O_num = O.Order_num
GROUP BY B.title
ORDER BY SUM(O.Quantity) 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 B.title, dollars
FROM AUTHOR AS A, WRITTEN_BY AS WB, BOOKS AS B, ORDER AS O
WHERE A.Auth_ID = WB.Auth_ID AND WB.ISBN = B.ISBN AND B.O_num = O.Order_num AND
MAX(dollars = SUM(B.price *
(SELECT O.quantity
FROM BOOKS AS B, ORDER AS O
WHERE B.O_num = O.Order_num
GROUP BY(B.ISBN))
ORDER BY dollars
```

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

```
SELECT A.Fname, A.Lname
FROM AUTHOR AS A, WRITTEN_BY AS WB, BOOKS AS B, ORDER AS O
WHERE A.Auth_ID = WB.Auth_ID AND WB.ISBN = B.ISBN AND B.O_num = O.Order_num AND
MAX(SUM(
SELECT O.quantity
FROM BOOKS AS B, ORDER AS O
WHERE B.O_num = O.Order_num
GROUP BY B.ISBN))
```

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

```
SELECT A.Fname, A.Lname
FROM AUTHOR AS A, WRITTEN_BY AS WB, BOOKS AS B, ORDER AS O
WHERE A.Auth_ID = WB.Auth_ID AND WB.ISBN = B.ISBN AND B.O_num = O.Order_num AND
MAX(SUM(
SELECT O.quantity
FROM BOOKS AS B, ORDER AS O
WHERE B.O_num = O.Order_num
GROUP BY B.ISBN)) * B.Price)
```

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

```
SELECT C.Fname, C.Lname
FROM CUSTOMER AS C, AUTHOR AS A, WRITTEN_BY AS WB, BOOKS AS B, ORDERS AS O
WHERE O.Order_num = B.O_num AND B.ISBN = WB.ISBN AND WB.Auth_ID = A.Auth_ID AND
A.Fname = (SELECT A.Fname, A.Lname
FROM AUTHOR AS A, WRITTEN_BY AS WB, BOOKS AS B, ORDER AS O
```

WHERE A.Auth_ID = WB.Auth_ID AND WB.ISBN = B.ISBN AND B.O_num = O.Order_num AND MAX(SUM(O.quantity) * B.Price))

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

```
SELECT A.Fname, A.Lname
FROM AUTHOR AS A, WRITTEN_BY AS WB, BOOKS AS B
WHERE A.Auth_ID = WB.Auth_ID AND WB.ISBN = B.ISBN AND B.Cust_ID = (SELECT B.Cust_ID
FROM BOOK AS B
GROUP BY B.Cust_ID
HAVING B.Price >= (SELECT AVG(B.Price)
FROM BOOK AS B
GROUP BY B.ISBN))
```

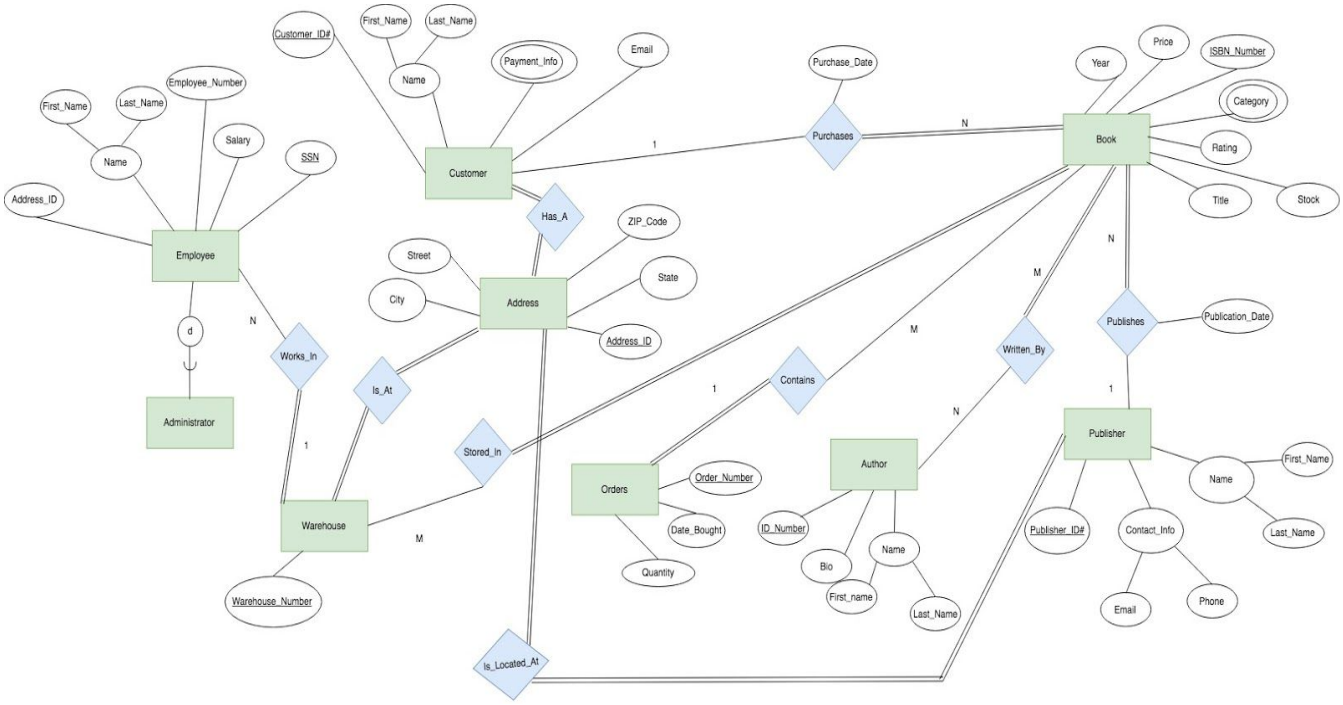
No comments from grader.

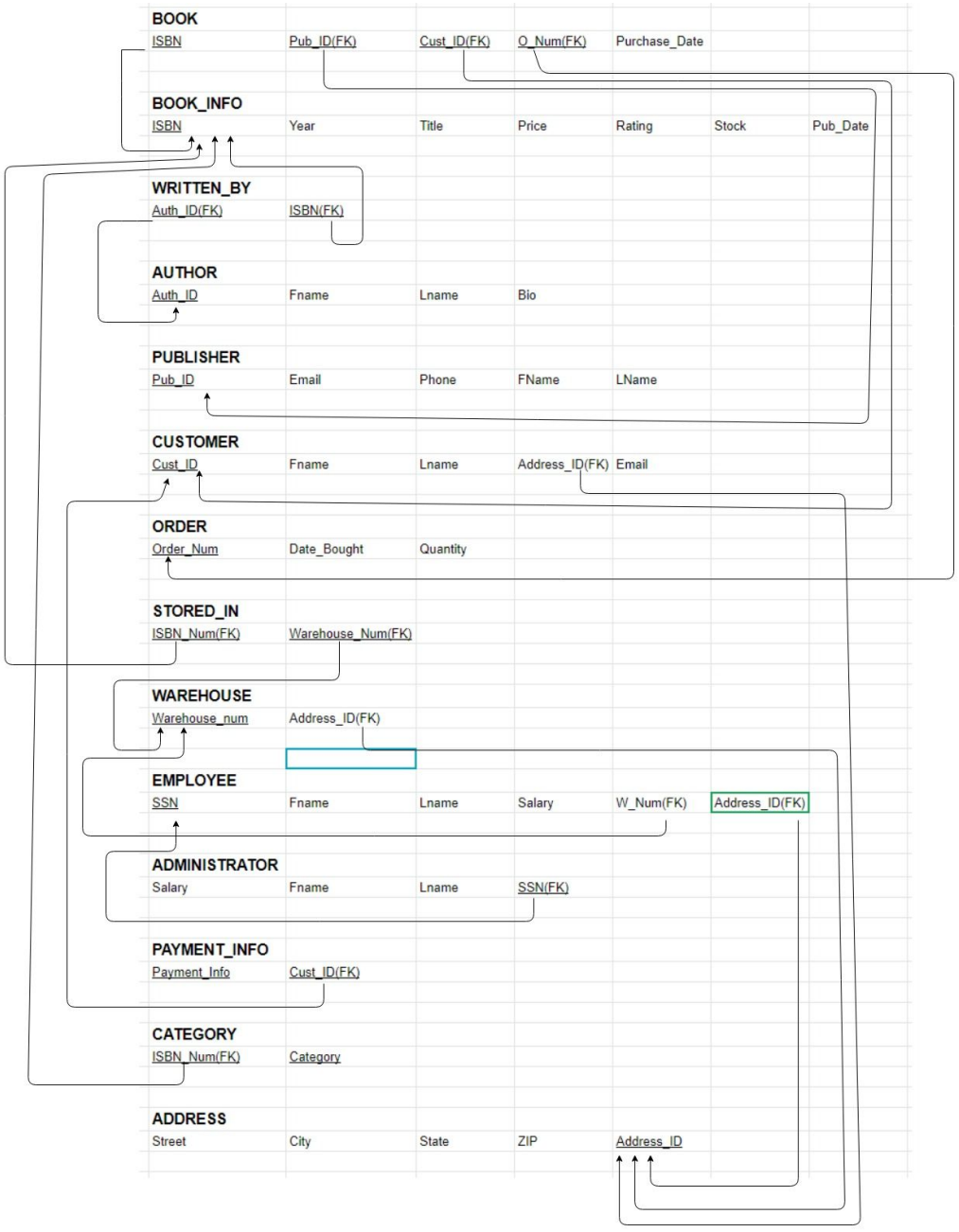
Revisions:

- ***See UPDATED Database Creation Code in folder.***
- ***Revised SQL code is in Part II, Section 3.***

Checkpoint 4

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.


Customer_ID#
First_Name
Last_Name
Name
Payment_Info
Email
Customer
Purchase_Date
Purchases
N
1
Book
Year
Price
ISBN_Number
Category
Rating
Title
Stock
Employee_Number
First_Name
Last_Name
Name
Salary
SSN
Address_ID
Employee
Has_A
ZIP_Code
Street
City
Address
State
Address_ID
N
Works_In
d
Administrator
Is_At
1
Warehouse
Warehouse_Number
M
Stored_In
Is_Located_At
1
Contains
M
Orders
Order_Number
Date_Bought
Quantity
M
Written_By
N
Author
ID_Number
Bio
Name
First_name
Last_Name
N
Publishes
Publication_Date
1
Publisher
Publisher_ID#
Contact_Info
Email
Phone
Name
First_Name
Last_Name



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:
ISBN → (Year, Title, Price, Rating, Stock, Pub_Date)

BOOK:
PUB_ID, Cust_ID, O_num → (Purchase_date, ISBN)

WRITTEN_BY:
Auth_ID, ISBN → (Year, Title, Price, Rating, Stock, Pub_Date, Fname, Lname, Bio)

AUTHOR:
Auth_ID → (Fname, Lname, Bio)

PUBLISHER:
Pub_ID → (Email, Phone, Fname, Lname)

CUSTOMER:
Cust_ID → (Fname, Lname, Address_ID, Email)

ORDER:
Order_num → (Date_Bought, Quantity)

STORED_IN:
ISBN_num, Warehouse_num→ (Year, Title, Price, Rating, Stock, Pub_Date, Address_ID)

WAREHOUSE:
Warehouse_num, Address_ID → (ZIP, State, City, Street)

EMPLOYEE:
SSN → (Address_ID, Fname, Lname, Salary, Wnum, Emp_Num)

ADMINISTRATOR:
SSN → (Fname, Lname, Salary)

PAYMENT_INFO:
Cust_ID → (Payment_Info)

CATEGORY:

ISBN_num → (Category)

ADDRESS:
Address_ID → (ZIP, State, City, Street)
ZIP, Street → (City, State)
Street, City, State → (ZIP)

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.

- **BOOKS:** BCNF
- **WRITTEN_BY:** BCNF
- **AUTHOR:** BCNF
- **PUBLISHER:** BCNF
- **CUSTOMER**: BCNF
- **ORDER:** BCNF
- **STORED_IN:** BCNF
- **WAREHOUSE:** BCNF
- **EMPLOYEE:** BCNF
- **ADMINISTRATOR:** BCNF
- **PAYMENT_INFO:** BCNF
- **CATEGORY:** BCNF
- **ADDRESS:** BCNF

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.

We revised our relation schemas to make sure each relation was in 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.

We could create a view that shows how many books a customer has purchased from each category of books. The code for this view is as follows:

```
CREATE VIEW CUST_CATEGORY (Fname, Lname, Category, Total)

AS SELECT C.Fname, C.Lname, CA.Category, SUM(B.ISBN)
```

```
FROM CUSTOMER AS C, CATEGORY AS CA, BOOK AS B

WHERE C.Cust_ID = B.Cust_ID AND CA.ISBN_num = B.ISBN

GROUP BY CA.Category;
```

We could create a view that shows the total salary and number of employees at each warehouse. The code for this view is as follows:

```
CREATE VIEW WAREHOUSE_INFO (Warehouse_Address, Total_Employees,
Total_Salary)

AS SELECT W.Address, E.COUNT(*), SUM(Salary)

FROM WAREHOUSE AS W, EMPLOYEE E

WHERE W.Warehous_num = E.WNum

GROUP BY W.Address;
```

Comments from grader:

ERD
Good job with implementing the location entity
Would a person hierarchy work well? Consider that so you can have both disjoint and overlapping hierarchy setup.

Relational Schema
Looks good

Functional Dependencies:
I think you're missing some transitive functional dependencies. For example {zip} -> {city, state}
Also some missing secondary keys, for example {email} -> {all other info}. Secondary keys are listed in functional dependencies

BCNF
Include a brief justification for BCNF, I realize it doesn't say that here, but for the final the rubric asks me for a one sentence precise justification

Views
Great!

Overall great job. Keep in mind for the final report, it's akin to a user manual. Ensure it is consistently formatted with regard to how you present headers, code, etc and professional.

Revisions:

- ***See final ER diagram in section 1, part 1.***
- ***See functional dependency revisions in section 1, part 1***
- ***BCNF description revisions in section 1, part 1.***

Part II: The SQL Database

Section 1: SQL CREATE
In the text document SQL_Code.txt

Section 2: SQL INSERTS
In the text document SQL_Insert.txt

Section 3: SQL QUERIES
In the text document SQL_Queries.txt

Section 4: SQL INSERT/ DELETE
In the text document SQL_DeleteAndInsertSample.txt