Final Project Report

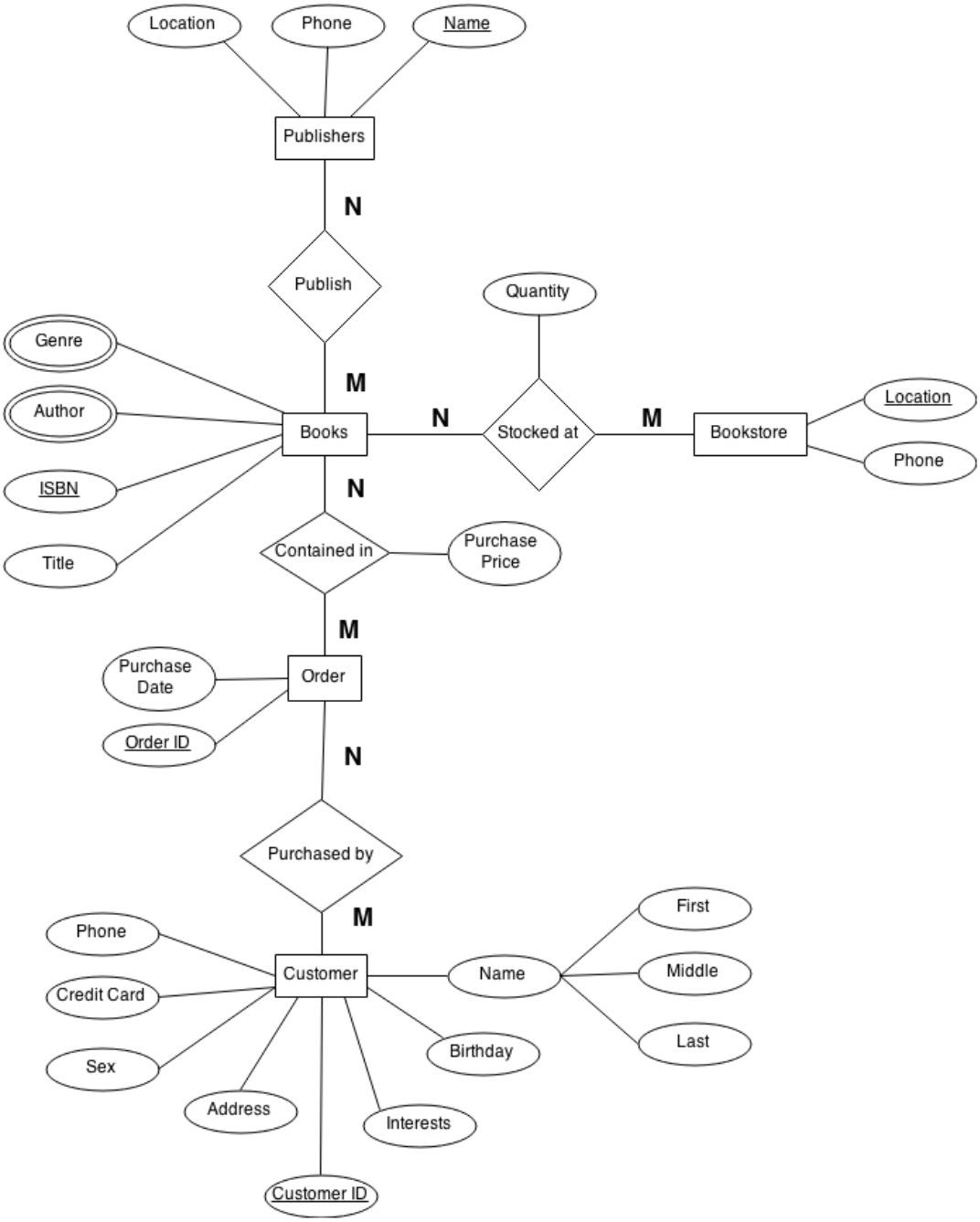
Database Description, User Manual, Checkpoints & SQLite Database

Zachary Schuller(.33), Olivia Whitman(.68), Emma Rastatter(.6), Sanny Kumar(.439)

4/26/2015

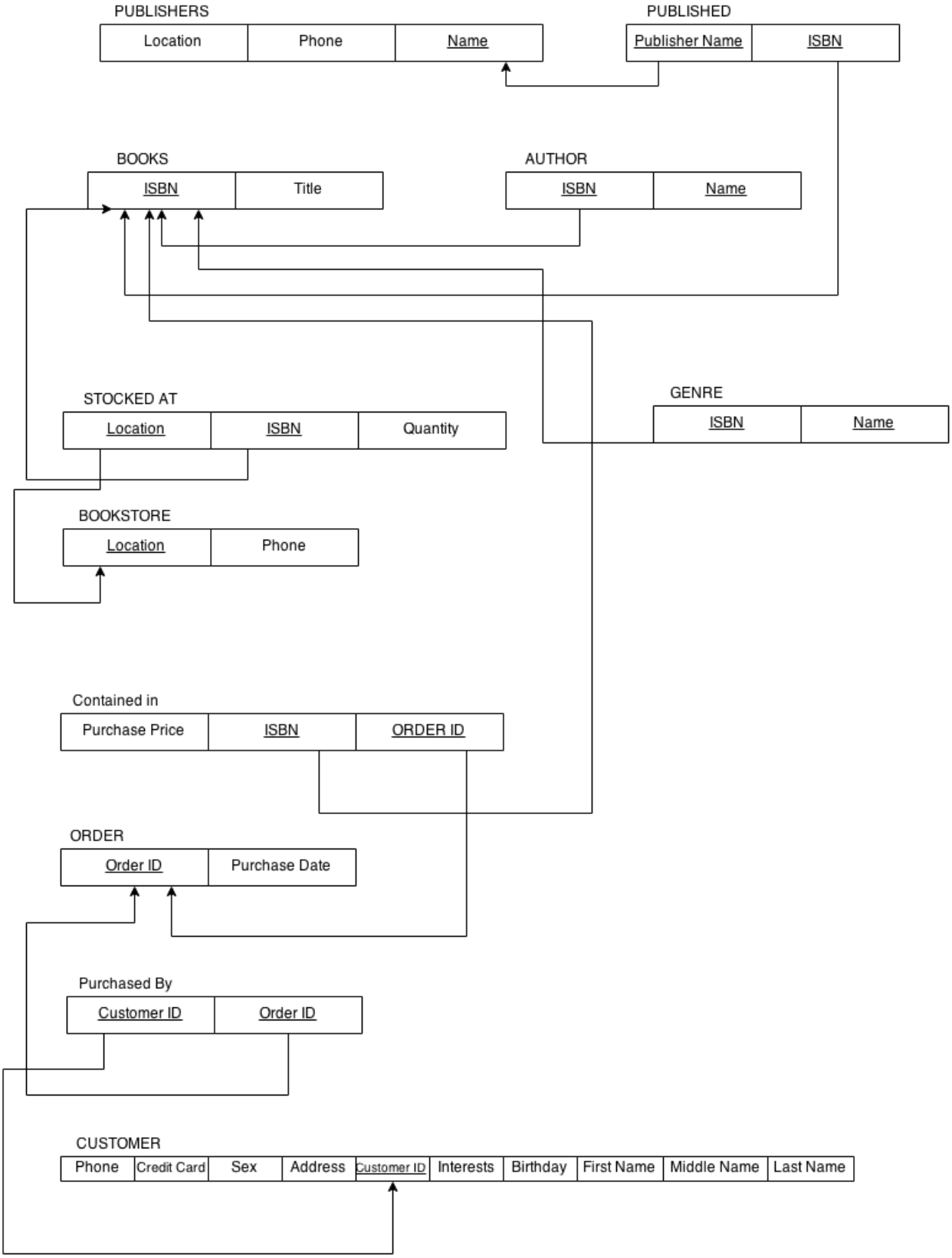
Database Description

ER-Model



We assume that ISBN's are unique. We allow multiple authors to a single book. We assume that the customer has only one Interests. We assume that all customers have a first and last name. We assume that Publishers are unique, but they may also publish multiple books. We assume that there is only one Bookstore at a location. We assume that a customer may have multiple orders, but and order has only one customer. We assume that books are stocked at Bookstores. We assume that a Book can have multiple genres. We assume that books are purchased on a date that exists.

Relational Schema



Normalized Database

The relations Publishers, Published, Books, Bookstore, Author, Stocked At, Genre, Contained In, Orders, Purchased By, Customer are all in 3-NF as you can see from the following functional dependencies:

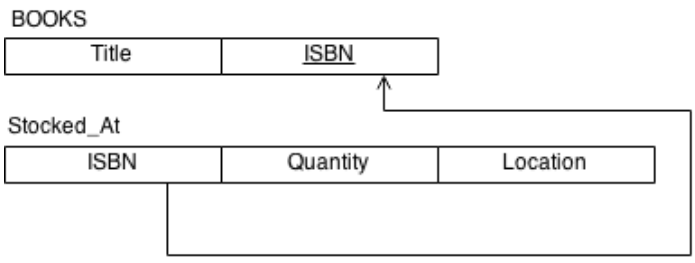
Publishers: $Name \rightarrow \{Location, Phone\}$
Published: $ISBN \rightarrow \{Publisher\_Name\}$
Books: $ISBN \rightarrow \{Title\}$
Author: $ISBN \rightarrow \{Name\}$
Stocked_At: $Location, ISBN \rightarrow \{Quantity\}$
Genre: $I \quad BN \rightarrow \{Name\}$
Bookstore: $Location \rightarrow \{Phone\}$
Contained_In: $Order\_ID, ISBN \rightarrow \{Purchased\_Price\}$
Orders: $Order\_ID \rightarrow \{Purchase\_Date\}$
Purchased_Price: $OrderID \rightarrow \{Customer\_ID\}$
Customer: $CustomerID \rightarrow \{Phone, Credit\_card, Sex, Address, interests, birthday, First\_name, Middle\_Name, Last\_Name, email\}$

The database shown above in the relational schema and functional dependencies already conforms to BCNF since all for all X → Y, X is a super key.

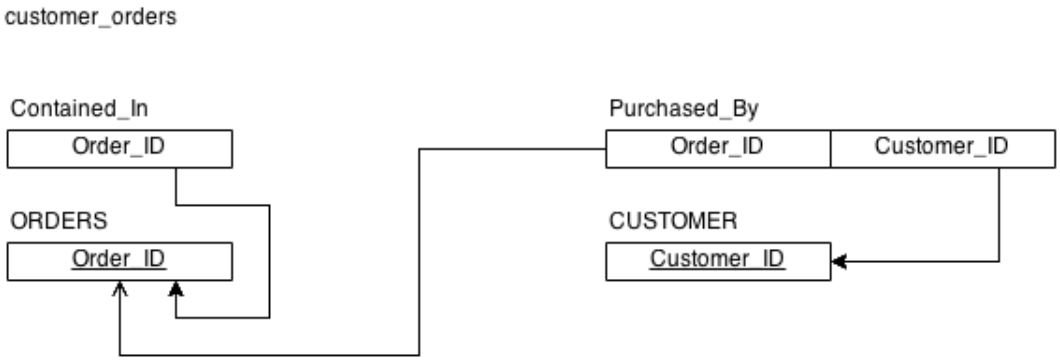
Views with Relational Schema and SQL for each

This view is for a person stocking the shelves who wants to know which books have a quantity less than the average quantity.
CREATE VIEW low_stock AS SELECT Title, Books.ISBN, quantity, location FROM Stocked_At, Books WHERE books.ISBN = Stocked_At.ISBN HAVING Stocked_At.quantity < avg(SELECT sum(Stocked_At.ISBN) FROM Stocked_At GROUP BY Stocked_at.ISBN) ORDER BY Title;

low_stock



This view is for customers viewing the total amount they have spent on their orders.
CREATE VIEW customer_orders AS SELECT sum(Purchased_Price) FROM Contained_in, Orders, Purchased_By, Customer WHERE Contained_in.Order_ID = Orders.Order_ID AND Purchased_by.Order_ID = Orders.Order_ID AND Purchased_By.Customer_ID = Customer.Customer_ID; GROUP BY Customer_ID;



Indexes

The database for the bookstore uses SQLite3, thus the only indexing option available to use for indexing is the index tree. To prevent run time from increasing for certain searches, special care was taken for considering attributes for indexing. It was decided that only the attributes that were foreseen to have equality checks would be indexed. In other words if an attribute was predicted to have numerous less than or greater than SQL queries, it was not indexed. The naming scheme for the indices is Tablename_Index, where Tablename is the format used in creating the table. The indices are sorted in ascending order, which is supposed to be compatible with most versions of SQLite. The attributes indexed for each table is as follows:

Table	**Indexed Attributes**
Publishers	Name, Location
Books	ISBN, Title
Bookstore	Location
Orders	Order_ID
Customer	Customer_ID, Interests, Last_Name
Purchased_By	Customer_ID, Order_ID
Contained_In	Order_ID, ISBN
Stocked_At	ISBN, Location
Genre	ISBN, Name
Author	ISBN, Name
Published	ISBN, Publisher_Name

Sample Transactions

Below are three sample transactions that can be run on our database which include an English description of the transaction and its corresponding SQL. Please note that if you wish to run the example transactions with the SQLite Firefox extension, you should remove the keywords "BEGIN TRANSACTION;" and "COMMIT;". Should you be using the SQLite console, these keywords would not need to be removed.

This transaction is useful for a user placing an order. The user wishes that this transaction be completed fully or, if it is unable to be completely processed, aborted.

BEGIN TRANSACTION;
insert into Orders(Order_ID,Purchase_Date) values (123,'2015-10-11');
insert into Contained_In (Purchase_Price, ISBN,Order_ID) values (18.00,'72227710',123);
insert into Purchased_By(Customer_ID,Order_ID) values(1,123);
COMMIT;

This transaction is useful for an employee entering a new book that a publisher now publishes.

BEGIN TRANSACTION;
insert into Books (ISBN,Title) values('44444444','Your Computer and You');
insert into Published (Publisher_Name,ISBN,Year) values('McGraw-Hill Osborne Media','44444444',2007);
insert into Genre(ISBN,Name) values ('44444444','Computer');
COMMIT;

This transaction is useful for a new user creating a profile in order to purchase books.
BEGIN TRANSACTION; insert into Customer(Phone,Credit_Card,Sex,Address,Customer_ID,Interests,Birthday,First_Name,Middle_Name,Last_Name) values ('614-444-4444','1234567890','M','160 East Lane Rd.',123,'Science','2015-10-10','Billy','Joe','Junior'); COMMIT;

User Manual

Description of Database

Our database is a representation of the entities and business processes at the bookstore Bits and Books. The database is designed to allow employees and managers to more efficiently manage the bookstore. Our database contains tables that allow employees to track information and indices that quicken the process of finding information for employees. The database stores information about Publishers, Books, Bookstore locations, Orders, and Customers, as well as information relating these entities to each other. The database is implemented in SQLite, and all of the relations are in 3-NF. For each table

we will describe its real-world counterpart and a description of each attribute for that table including its data type and any built-in constraints. You can see the relational schema and ER diagrams for this database at the beginning of this report.

First the table Publishers represents the Publishers that the Bits & Books bookstore purchases from. The attribute Location is a varchar with the maximum size of 50 characters to store the physical location of the publisher's headquarters. The attribute Phone is used to store the phone number provided by the publisher and is stored as a varchar with a maximum size of 12 characters. Finally, the attribute Name is the provided name of the publisher; it must be unique and cannot be NULL. Name is the primary key of this table.

The table Books represents a physical book which was purchased and can be stack on the shelves at a bookstore location. The attribute Title is the title of the book stored as a varchar with maximum length of 50 characters and cannot be NULL. The attribute ISBN is the ISBN of the book stored as a varchar of maximum length 13 characters, must be unique and cannot be NULL. ISBN is the primary key of this table.

The table Bookstore represents the physical bookstore location that stocks books. The attribute Phone is the given phone number of the store location and is stored as a varchar with maximum length of 12 characters. The attribute Location is the physical location of the bookstore, is stored as a varchar with a maximum length of 50 characters, must be unique, and cannot be NULL. The location attribute is the primary key of this table.

The table Orders represents an order placed by a customer that purchased multiple books. The attribute Purchase_Date stores the date of the order's purchase stored as a Date datatype. The attribute Order_ID is a unique integer that identifies the order and cannot be NULL. Order_ID is the primary key of this table.

The table Customer represents a customer who wishes to purchase books and their related data. The attribute Phone is their given phone number stored as a varchar with a maximum of 12 characters. The attribute Credit_Card is the customer's credit card number stored as a varchar with maximum length of 16 characters. The attribute Sex is the customer's provided Sex stored as a varchar with a length of 1 character. The attribute Address is the customer's provided home address stored as a varchar with maximum length of 50 characters. The customer can provided one Interests which is stored as a varchar with maximum length of 50 characters. The attribute Birthday is the customer's provided birthdate which is stored as a Date datatype. The attribute First_Name is the customer's first name stored as a varchar with maximum length of 20 characters and cannot be NULL. The attribute Middle_Name is the customer's middle name stored as a varchar with maximum length of 20 characters. The attribute Last_Name is the customer's last name stored as a varchar with maximum length of 20 characters and cannot be NULL. The attribute Customer_ID is a unique integer that identifies the customer and their data. Customer_ID is the primary key of this table.

The table Purchased_By links together Orders and the Customers who have placed said Orders. The foreign key attribute Customer_ID is an integer that uniquely identifies an existing customer who placed the order and cannot be NULL. The foreign key attribute Order_ID is an integer uniquely identifies an existing Order that has been placed and cannot be NULL.

The table Contained_In represents a book that has been purchased as part of an Order. The attribute Purchase_Price stores the price of the book at the time of purchase and is stored as a Real datatype

which cannot be NULL. The foreign key Order_ID uniquely identifies an order that has been placed and cannot be NULL. The foreign key ISBN uniquely identifies an existing book and cannot be NULL.

The table Stocked_At represents a book that has been stocked at a particular bookstore location. The foreign key Location is a varchar with maximum length of 50 characters that uniquely identifies the bookstore location at which the book has been stocked that cannot be NULL. The foreign key ISBN is a varchar with maximum length of 13 characters that cannot be NULL which uniquely identifies the existing book that has been stocked at the given location. The attribute Quantity is an integer storing the amount of the provided book at the given location which cannot be NULL.

The table Genre represents the genre categorization of a book. The foreign key ISBN is a varchar with maximum length of 13 characters which cannot be NULL that uniquely identifies an existing book. The attribute Name is the name of the category stored as a varchar with maximum length of 50 characters which cannot be NULL.

The table Author represents an Author who has written at least one book that can be found in the database. The foreign key ISBN is a varchar with maximum length of 13 characters that uniquely identifies a book and cannot be NULL. The attribute Name stores the name of the author as a varchar with maximum length 50 characters which cannot be NULL.

The table Published links together books and their publishers. The foreign key ISBN is a varchar with maximum length of 13 characters that uniquely identifies an existing book and cannot be NULL. The foreign key Publisher_Name is a varchar with maximum length of 50 characters that uniquely identifies an existing Publisher and cannot be NULL. The attribute Year is an integer which stored the year in which the book was published by the given publisher.

Sample SQL Queries

The following are sample queries accompanied by their expected output (some of which may be truncated for space purposes) that can be run on our database. Above the queries are an English description and the corresponding relational algebra.

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

$A \leftarrow Books * Author$
$B \leftarrow A * Contained\_In$
$\pi_{Title}[\sigma_{Purchased\_price < 10 \text{ and } name="Terry\ Pratchett"} B]$

```
SELECT BOOKS.Title
FROM BOOKS, AUTHOR, Contained_In
WHERE BOOKS.ISBN = AUTHOR.ISBN
AND BOOKS.ISBN  = Contained_in.ISBN
AND AUTHOR.Name  =  'Terry Pratchett'
AND Contained_In.Purchase_Price<10;
```

Example Expected Output:

"Going Postal"
"Guards! Guards!"
"Small Gods"
"Small Gods"
"Small Gods"

Gives all the titles and their dates of purchase made by a single customer.

$\pi_{purchased\_date,Title}[\sigma_{customer\_ID=1}(((((Purchased\_By * Contained\_In) * Orders) * Books) * Customers) * Author)]$

```
SELECT BOOKS.title, ORDERS.Purchase_Date
FROM BOOKS, AUTHOR, Contained_in, CUSTOMER, ORDERS, Purchased_By
WHERE BOOKS.ISBN = AUTHOR.ISBN
AND BOOKS.ISBN = Contained_in.ISBN
AND Purchased_By.Order_ID = Contained_In.Order_ID
AND ORDERS.Order_ID = Contained_In.Order_ID
AND ORDERS.Order_ID = Contained_In.Order_ID
AND Purchased_By.Customer_ID = CUSTOMER.Customer_ID
AND CUSTOMER.Customer_ID =1;
```

Example Expected Output:

"OCP: Oracle9i Certification Kit","2015-10-10"
"OCP: Oracle9i Certification Kit","2015-10-10"
"OCP: Oracle9i Certification Kit","2015-10-10"
"OCP: Oracle9i Certification Kit","2015-10-10"
"OCP: Oracle9i Certification Kit","2015-10-10"

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

$ISBN\_quant \leftarrow {}_{ISBN}\Im_{sum(quantity)}(Books * Stocked\_At)$

$\pi_{Title,\ ISBN}(\sigma_{ISBN\_quant.sum<5}((Books * Stocked\_At) * ISBN\_quant))$

```
SELECT BOOKS.title, BOOKS.ISBN
FROM BOOKS, STOCKED_AT
WHERE BOOKS.ISBN=STOCKED_AT.ISBN
GROUP BY STOCKED_AT.ISBN
HAVING sum(STOCKED_AT.Quantity) < 5;
```

Example Expected Output:

"Persuasive Technology: Using Computers to Change What We Think and Do" "1558606432"

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

$\pi_{Title,Cutsomer\_ID}[\sigma_{Author.name=\text{"Terry Pratchett"}}((((Books * Author) * Contained\_In) * Purchased\_By) * Customer)]$

```
SELECT BOOKS.title, CUSTOMER.Customer_ID
FROM BOOKS, AUTHOR, Contained_In, CUSTOMER, Purchased_By
WHERE BOOKS.ISBN = AUTHOR.ISBN
AND BOOKS.ISBN = Contained_in.ISBN
AND Purchased_By.Order_ID = Contained_In.Order_ID
AND Purchased_By.Customer_ID = CUSTOMER.Customer_ID
AND AUTHOR.Name= ‘Terry Pratchett’
GROUP BY CUSTOMER.Customer_ID;
```

Example Expected Output:

NULL

Finds the total number of books purchased by a single customer.
$A \leftarrow (Customer * Purchased\_By)$ $B \leftarrow A * Contained\_In$ $C \leftarrow {}_{ISBN}\Im_{count(ISBN)}(\sigma_{customer\_ID=1}B)$ $\pi_{Customer\_ID,\ count}\ [(\sigma_{customer\_ID=1}B) \times C]$
SELECT CUSTOMER.Customer_ID, count(Contained_in.ISBN) FROM Contained_in, CUSTOMER, Purchased_By WHERE Purchased_By.Order_ID = Contained_In.Order_ID AND Purchased_By.Customer_ID = CUSTOMER.Customer_ID AND CUSTOMER.Customer_ID =1;
Example Expected Output:
"1" "1"

Finds the customer who has purchased the most books and the total number of books they have purchased.
$Count \leftarrow {}_{ISBN}\Im_{count(ISBN)}(Contained\_In)$ $CustTotals \leftarrow \pi_{Count.count,customer\_ID}((Purchased\_By * Contained\_In) * Count)$ $max \leftarrow {}_{Customer\_ID}\Im_{\max(CustTotals.count)}(CustTotals)$ $\pi_{Customer\_ID,max.max}(max \bowtie (max.max = CustTotals.count)CustTotals)$
SELECT CUSTOMER_TOTALS.Customer_ID, max(Customer_count) FROM (SELECT CUSTOMER.Customer_ID AS Customer_ID, count(Contained_in.ISBN) AS Customer_count FROM Contained_in, CUSTOMER, Purchased_By WHERE Purchased_By.Order_ID = Contained_In.Order_ID AND Purchased_By.Customer_ID = CUSTOMER.Customer_ID GROUP BY CUSTOMER.Customer_ID) AS CUSTOMER_TOTALS;
Example Expected Output:
"3" "2"

Lists all books in stock at a particular location.
Store ←[σ(Location = '20 Lost rd., Florida, OH')[$Bookstore$]] StoreStock ←$Store \bowtie (Store.Location= Stocked\_At.Location)\ Stocked\_At$ BookTitles ←$StoreStock \bowtie (StoreStock.ISBN= Books.ISBN)\ Books\ \pi\ Titles$[$BookTitles$]
Select Title from BookStore, Stocked_At, Books where Bookstore.Location='20 Lost rd., Florida, OH' and Stocked_At.ISBN=Books.ISBN and Stocked_At.Location=Bookstore.Location;
Example Expected Output:
"OCP: Oracle9i Certification Kit " "SQL Server 2000 for Experienced DBA's" "Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations "

Lists all books in Computer genre under $10.
Genie← $Books$⋈($Books.ISBN= Genre.ISBN$) $Genre$ price←$Genie$⋈($Genie.ISBN= Contained\ In.ISBN$)$ContainedIn\ \pi Titles$[$\sigma$(purchase_price<10 and $name$="$Computer$")[$price$]]
Select Title from Books, Genre, Contained_In where Books.ISBN=Genre.ISBN and Genre.ISBN= Contained_In.ISBN and purchase_price<10 and Genre.Name='Computer';
Example Expected Output:
NULL

Lists customers who purchased books on a given day.
custpurch ← $Customer$⋈(*Customer.Customer ID= Purchased By.CustomerID*) $Purchased\_By$ date ← $Order$⋈(*custpurch.Order ID= Order.Order ID*) $custpurch\ \pi_{fname,lna\ \ e}$[$\sigma$(*Purchase Date=given*)[$date$]]
Select First_Name, Last_Name from Customer, Purchased_By, Orders Where Customer.Customer_ID=Purchased_By.Customer_ID and Purchased_By.Order_ID= Orders.Order_ID and Purchase_Date='1995';
Example Expected Output:
NULL

The following queries are some more advanced requests that can be performed on the database.

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

$spent \leftarrow {}_{Customer\_ID}\Im_{\text{total}(purchase\_price)}(Orders)$

$\pi_{First\_Name,Middle\_Name,Last\_Name,total}((((Customer * Purchased\_By) * Orders) * Contained\_In) * spent)$

```
SELECT First_Name, Middle_Name, Last_Name, total(Purchase_Price)
FROM Customer, Purchased_By, Orders, Contained_In
WHERE Contained_In.Order_ID = Orders.Order_ID AND Orders.Order_ID =
Purchased_By.Order_ID
        AND Purchased_By.Customer_ID = Customer.Customer_ID
GROUP BY Customer.Customer_ID;
```

Example Expected Output:

```
"Bob","Joe","Junior","104.97"
"Billy","Sameul","Bob","34.99"
"Lloyd","","Irving","100"
"Yuri","","Lowell","17.49"
"Sheena","","Fujibayashi","49.95"
"Estellise","Sidos","Heurrassein","2"
```

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

$avg \leftarrow \Im_{\text{average}(purchase\_price)}(((Orders * Contained\_In) * Purchased\_by) * Customer)$

$sum \leftarrow {}_{Customer\_ID}\Im_{\text{sum}(purchase\_price)}(((Orders * Contained\_In) * Purchased\_by) * Customer)$

$\pi_{First\_Name,Middle\_Name,Last\_Name,email,Purchase\_Price}[\sigma_{avg.average>sum.sum}(((((Orders * Contained\_In) * Purchased\_by) * Customer) * sum) \times avg)]$

```
SELECT First_Name, Middle_Name, Last_Name, email, Contained_In.Purchase_Price
FROM Customer, Purchased_By, Orders, Contained_In
WHERE Contained_In.Order_ID = Orders.Order_ID AND Orders.Order_ID =
Purchased_By.Order_ID
        AND Purchased_By.Customer_ID = Customer.Customer_ID
GROUP BY Customer.Customer_ID
HAVING sum(Contained_In.Purchase_Price) > (SELECT  avg(Contained_In.Purchase_Price)
FROM Customer, Purchased_By, Orders, Contained_In
WHERE Contained_In.Order_ID = Orders.Order_ID AND Orders.Order_ID =
Purchased_By.Order_ID
        AND Purchased_By.Customer_ID = Customer.Customer_ID);
```

Example Expected Output:

```
"Bob","Joe","Junior",,"104.97"
"Lloyd","","Irving",,"50"
"Sheena","","Fujibayashi",,"49.95"
"Kratos","","Aurion",,"38.49"
```

Provides 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.

$count \leftarrow {}_{ISBN}\Im_{\text{count}(ISBN)}(Contained\_In)$
$\pi_{Title,count}(count * Books)$

```
SELECT COUNT(Contained_In.ISBN) AS Total_Sold, Books.Title
FROM Contained_In, Books
WHERE Contained_In.ISBN = Books.ISBN
GROUP BY Books.ISBN
ORDER BY Total_Sold desc;
```

Example Expected Output:

"3","Small Gods"
"2","Choosing & Using Hand Tools"
"1","Atonement"
"1","On Human Nature"
"1","Consilience: The Unity of Knowledge"

Provides 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.

$total \leftarrow {}_{ISBN}\Im_{\text{sum}(purchase\_price)}(Contained\_In)$
$\pi_{Title,total}(total * Books)$

```
SELECT SUM(Contained_In.Purchase_Price) AS Total, Books.Title
FROM Contained_In, Books
WHERE Contained_In.ISBN = Books.ISBN
GROUP BY Books.ISBN
ORDER BY Total desc;
```

Example Expected Output:

"139.2","Intermediate Accounting"
"126.67","Financial Reporting and Analysis"
"122.5","Financial Markets & Corporate Strategy"
"112.95","Advanced Accounting"

Finds the most popular author in the database (i.e. the one who has sold the most books).
$count \leftarrow {}_{ISBN}\Im_{\text{count}(purchase\_price)}(Contained\_In)$ $maximum \leftarrow \Im_{\max(count)}(Count)$ $\pi_{max.max,name}(((max \bowtie (max.max = count.count)count) * Books) * Author)$
SELECT MAX(Total_sold), Name FROM (SELECT COUNT(Contained_In.ISBN) AS Total_Sold, Name FROM Contained_In, Books, Author WHERE Contained_In.ISBN = Books.ISBN AND Author.ISBN = Books.ISBN GROUP BY Author.ISBN);
Example Expected Output:
"5","Matthew Weishan"

Finds the most profitable author in the database for this store (i.e. the one who has brought in the most money).
$sum \leftarrow {}_{ISBN}\Im_{\text{sum}(purchase\_price)}(Contained\_In)$ $max \leftarrow \Im_{\max(sum)}(sum)$ $\pi_{name,max}(((max \bowtie (max.max = sum.sum)sum) * Author) * Books)$
SELECT MAX(Total), Name FROM (SELECT SUM(Contained_In.Purchase_Price) AS Total, Name FROM Contained_In, Books, Author WHERE Contained_In.ISBN = Books.ISBN AND Author.ISBN = Books.ISBN GROUP BY Author.ISBN);
Example Expected Output:
"524.85","Matthew Weishan"

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

$$total \leftarrow {}_{ISBN}\Im_{\text{total}(purchase\_price)}(Contained\_In)$$
$$maximum \leftarrow \Im_{\max(total)}(total)$$
$$maxName \leftarrow \pi_{name}((Author * Contained\_In * Books) \bowtie (maximum.max = contained\_In.purchase\_price)\, maximum)$$
$$author \leftarrow \sigma_{author.name=maxName.name}(Books * Author * Contained\_In * Orders)$$
$$\pi_{phone,credit\_card,sex,address,customer\_ID,interests,Birthday,First\_Name,Middle\_Name,Last\_Name,email}(author)$$

```
SELECT Phone, Credit_Card, Sex, Address, Customer.Customer_ID, Interests, Birthday, First_Name,
Middle_Name, Last_Name, email
FROM Author, Books, Contained_in, Orders, Customer, Purchased_By
WHERE Author.ISBN = Books.ISBN AND Contained_in.ISBN = Books.ISBN AND
Contained_in.Order_ID = Orders.Order_ID
 AND Purchased_By.Order_ID = Orders.Order_ID AND Purchased_By.Customer_ID =
Customer.Customer_ID AND Author.Name = (SELECT Name FROM (SELECT MAX(total), Name
FROM (SELECT Total(Contained_In.Purchase_Price) AS Total, Name
FROM Contained_In, Books, Author
WHERE Contained_In.ISBN = Books.ISBN AND Author.ISBN = Books.ISBN
GROUP BY Author.ISBN)));
```

Example Expected Output:

```
"","","M","","1","Science","1995","Bob","Joe","Junior",""
```

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

$$average \leftarrow \Im_{\text{total}(purchase\_price)}(Contained\_In)$$
$$summ \leftarrow {}_{Customer\_ID}\Im_{\text{sum}(purchase\_price)}(Contained\_In * Purchased\_By * Customer)$$
$$authors \leftarrow \sigma_{summ.sum>average.avg}(Author * Books * Contained\_In * Orders * Purchased\_By * Customer)$$
$$\pi_{name,purchase_{price}}(authors)$$

```
SELECT Author.Name, Contained_In.Purchase_Price
FROM Author, Books, Contained_In, Orders, Purchased_By, Customer
WHERE Author.ISBN = Books.ISBN AND Contained_in.ISBN = Books.ISBN AND
Contained_in.Order_ID = Orders.Order_ID
 AND Purchased_By.Order_ID = Orders.Order_ID AND Purchased_By.Customer_ID =
Customer.Customer_ID
GROUP BY Customer.Customer_ID
HAVING SUM(Contained_In.Purchase_Price) > (SELECT  avg(Contained_In.Purchase_Price)
FROM Customer, Purchased_By, Orders, Contained_In
WHERE Contained_In.Order_ID = Orders.Order_ID AND Orders.Order_ID =
Purchased_By.Order_ID
        AND Purchased_By.Customer_ID = Customer.Customer_ID);
```

Example Expected Output:

```
"Matthew Weishan","104.97"
"Ralph Kimball","50"
"Ian H. Witten","49.95"
"Ken Henderson","38.49"
```

Insert Samples

Below are some examples of inserting values into the various relations of the database. Please note that apostrophes used for possession should be double quotes (") so that the insert statements execute properly.

1. insert into Books (ISBN, Title) values ('0000000001254', 'The origin of Lifer');
2. insert into Author (ISBN, Name) values ('0000000001254', 'M Knight Shamalan');
3. insert into Publishers (Location, Phone, Name) values ('Happy Orchard Brewery','1234567890','Reptilian Overlord Shapeshifter');
4. insert into Published ('Reptilian Overlord Shapeshifter','0000000001254', 1999);
5. insert into Genre(ISBN, Name) values ('0000000001254', 'Cults');
6. insert into Orders('Order_ID,Purchase_Date) values (60, 1987-01-01);
7. insert into Contained_In (Purchase_Price, ISBN,Order_ID) values(12.00,'0000000001254',60);

8. insert into Customer(Phone,Credit_Card,Sex,Address,Customer_ID,Interests,Birthday,First_Name,Middle_Name,Last_Name) values ('','','M','41 Curl Drive, Columbus OH',9002,'',1999-10-10,'Abraham','Honest','Lincoln');
9. insert into Purchased_By(Customer_ID,Order_ID) values(9002,60);
10. insert into Bookstore(Location,Phone) values('Willy Wonka''s Chocolate Factory','8675309');
11. insert into Stocked_At(Location,ISBN,Quantity) values ('Willy Wonka''s Chocolate Factory','0000000001254',3);

Delete Samples

Below are examples of deleting values from the various relations of the database. Please note that apostrophes used for possession should be double quotes (") so that the delete statements execute properly.

1. delete from Books where ISBN='0000000001254';
2. delete from Author where ISBN='0000000001254';
3. delete from Publishers where Location='Happy Orchard Brewery';
4. delete from Published where ISBN= '0000000001254';
5. delete from Genre where ISBN='0000000001254';
6. delete from Orders where Order_ID=120;
7. delete from Contained_In where ISBN='0000000001254' and Order_ID=120;
8. delete from Customer where Customer_ID=9002;
9. delete from Purchased_By where Customer_ID=9002 and Order_ID=120;
10. delete from Bookstore where Location = 'Willy Wonka''s Chocolate Factory';
11. delete from Stocked_At where Location = 'Willy Wonka''s Chocolate Factory' and ISBN='0000000001254';

Graded Checkpoint Documents

For the graded checkpoints, please see the previously uploaded and submitted documents. As per the suggestions provided by the grader for the checkpoints, we have updated the ER model, relational schema, views, and parts of the example queries so that they better fit the requirements of the project.

The SQLite Database

See the attached Bookstore.db file for the SQLite database documented in the previous sections of this report. The commands used to create this database along with its indices and insertions for default data have been placed in the attached text file named 'Database_Creation_SQL.txt'.