User Manual

DATABASE EXPLANATION

This section is intended to inform the user of the real world representation of each table in the database. Each of the tables below represents a table in the database. All attributes of each entity are displayed along with their real world meaning and corresponding data type. Underlined attributes represent the Primary Keys of the entity, and attributes in red text represent the Foreign Keys that are not part of the Primary Key. The constraint on each attribute in the database is assumed to be NOT NULL unless specified otherwise. Some other constraints will need to be handled through intermediary software, such as ensuring that AuthorID is exactly 9 numeric characters.

BOOK - a physical book, identified by its ISBN. It is important to note that ISBN does not uniquely identify a specific copy of the book. As such, there may be several physical instances of a particular ISBN stored in the database. (See the explanation for **INVENTORY**)

Attribute	Data Type	Real World Meaning
ISBN	VARCHAR(12)	A unique numeric commercial book identifier
Title	VARCHAR(128)	Name of book
Category	VARCHAR(32)	Book type (Computer, Romance, Horror, etc.)
Publisher	VARCHAR(64)	A company which sets the editorial and commercial direction to people who publish books
PublishDate	INT	The year in which the book was published

AUTHOR - an individual person who has written a book that is currently stored in **INVENTORY**

Attribute	Data Type	Real World Meaning
AuthorID	CHAR(9)	Author's unique Identification Number
Name	VARCHAR(48)	Author's full name

WRITTEN_BY - describes the connection between **AUTHOR** and **BOOK**. An author (AuthorID) contributed to the writing of a book (ISBN)

Attribute	Data Type	Real World Meaning
AuthorID	CHAR(9)	Author's unique Identification Number
ISBN	VARCHAR(12)	Author's full name

EMPLOYEE - an individual member of the staff

Attribute	Data Type	Real World Meaning
EmployeeID	CHAR(9)	Employee's unique Identification Number
Name	VARCHAR(48)	Employee's full name
Salary	INT	Employee's salary
Department	VARCHAR(32)	Employee's department

INVENTORY - keeps track of the quantity of each **BOOK** in the database as well as the purchase and selling price

Attribute	Data Type	Real World Meaning
ISBN	VARCHAR(12)	A unique numeric commercial book identifier
Quantity	INT	Total copies of the book
PurchasePrice	REAL	Price that Bits & Books paid for the book (in USD)
SellingPrice	REAL	Price that Bits & Books sells the book to customers (in USD)

CUSTOMER - an individual customer of Bits & Books. An individual enrolls to become a Bits & Books Member and is given a Membership Card. The only information necessary to apply for the membership is the customer's name (the CustomerID is automatically generated).

Attribute	Data Type	Real World Meaning
CustomerID	CHAR(9)	Customer's unique Identification Number
Name	VARCHAR(48)	Customer's full name
Email	VARCHAR(64) Can be NULL	Customer's E-Mail address
Phone	CHAR(10) Can be NULL	Customer's phone number
Address	VARCHAR(64) Can be NULL	Customer's street address
City	VARCHAR(32) Can be NULL	Customer's city
State	CHAR(2) Can be NULL	Customer's state
Zip	CHAR(5) Can be NULL	Customer's zip code
Sex	CHAR(1) Can be NULL	Customer's sex

COSTUMER_PURCHASE_ORDER - an individual instance of a **CUSTOMER** purchasing a **BOOK**. An individual transaction is defined as the purchase of one or more of a specific **BOOK**. A customer may purchase multiple different books, but each unique ISBN is assigned a corresponding TIN to identify the price and quantity of a particular ISBN purchased by a customer. CustomerID may be NULL to account for customers that are not enrolled in the Bits & Books Membership.

Attribute	**Data Type**	**Real World Meaning**
TIN	CHAR(9)	Transaction Identification Number
Time	TIMESTAMP	Time that the customer completed the purchase
Price	REAL	Price of one book (in USD) Total Price = Price X Quantity
Quantity	INT	Quantity of books purchased
ISBN	VARCHAR(12)	A unique numeric commercial book identifier
CustomerID	CHAR(9) Can be NULL	Customer's unique Identification Number
EmployeeID	CHAR(9)	Employee's unique Identification Number

BOOKSTORE_PURCHASE_ORDER - an individual instance of an **EMPLOYEE** purchasing a **BOOK** (usually from a publisher). An individual transaction is defined as the purchase of one or more of a specific **BOOK**. An employee may purchase multiple different books, but each unique ISBN is assigned a corresponding TIN to identify the price and quantity of a particular ISBN purchased by the employee.

Attribute	**Data Type**	**Real World Meaning**
TIN	CHAR(9)	Transaction Identification Number
Time	TIMESTAMP	Time of the purchase order
Price	REAL	Price of one book (in USD) Total Price = Price X Quantity
Quantity	INT	Quantity of books purchased
ISBN	VARCHAR(12)	A unique numeric commercial book identifier
EmployeeID	CHAR(9)	Employee's unique Identification Number

SQL QUERY EXAMPLES

Query 1: Find the titles of all books by Pratchett that cost less than $10

Pratchett ← ($\sigma_{Name = Pratchett}$(**AUTHOR**)) $\bowtie_{AuthorID = AuthorID}$ (**WRITTEN_BY**)
PratchettBooks ← (Pratchett) $\bowtie_{ISBN = ISBN}$ (**BOOK**)
PratchettBooksPrices ← (PratchettBooks) $\bowtie_{ISBN = ISBN}$ (**INVENTORY**)
Result ← π_{Title}($\sigma_{Selling\_Price < 10}$ (PratchettBooksPrices))

```
SELECT B.TITLE
FROM BOOK AS B, AUTHOR AS A, WRITTEN_BY AS W, INVENTORY AS I
WHERE A.AUTHORID = W.AUTHORID AND B.ISBN = W.ISBN AND I.ISBN = B.ISBN AND A.NAME LIKE '% PRATCHETT' AND
I.SELLING_PRICE < 10;
```

Query 2: Give all the titles and their dates of purchase made by customer named Isaac Newton

Newton ← ($\sigma_{Name = Isaac\ Newton}$(**CUSTOMER**)) $\bowtie_{CustomerID = CustomerID}$ (**CUSTOMER_PURCHASE_ORDER**)
Result ← $\pi_{Title,\ Purchase\_Date}$ [(CustomerPurchases) $\bowtie_{ISBN = ISBN}$ (**BOOK**)]

```
SELECT B.TITLE, CPO.TIME
FROM BOOK AS B, CUSTOMER_PURCHASE_ORDER AS CPO, CUSTOMER AS C
WHERE C.CUSTOMERID = CPO.CUSTOMERID AND CPO.ISBN = B.ISBN AND C.NAME = 'ISAAC NEWTON';
```

Query 3: Find the titles and ISBNs for all books with less than 5 copies in stock

LessThanFive ← $\sigma_{Qty < 5}$ (**INVENTORY**)
Result ← $\pi_{Title,\ ISBN}$ [(LessThanFive) $\bowtie_{ISBN = ISBN}$ (**BOOK**)]

```
SELECT B.TITLE, B.ISBN
FROM BOOK AS B, INVENTORY AS I
WHERE B.ISBN = I.ISBN AND I.QUANTITY < 5;
```

Query 4: Give all the customers who purchased a book by Pratchett and the titles of Pratchett books they purchased

Pratchett ← ($\sigma_{Name = Pratchett}$(**AUTHOR**)) $\bowtie_{AuthorID = AuthorID}$ (**WRITTEN_BY**)
PratchettBooks ← (Pratchett) $\bowtie_{ISBN = ISBN}$ (**BOOK**)
PratchettBookPurchases← (PratchettBooks) $\bowtie_{ISBN = ISBN}$ (**CUSTOMER_PURCHASE_ORDER**)
CustomersPurchasingPratchett ← (**CUSTOMER**) $\bowtie_{CustomerID = CustomerID}$ (PratchettBookPurchases)
Result ← $\pi_{Name,\ Title}$ (CustomersPurchasingPratchett)

```
SELECT C.NAME, B.TITLE
FROM CUSTOMER AS C, BOOK AS B, CUSTOMER_PURCHASE_ORDER AS CPO, AUTHOR AS A, WRITTEN_BY AS W
```

```
WHERE C.CUSTOMERID = CPO.CUSTOMERID AND B.ISBN = CPO.ISBN AND B.ISBN = W.ISBN AND A.AUTHORID
=  W.AUTHORID AND A.NAME LIKE '% PRATCHETT';
```

Query 5: Find the total number of books purchased by a customer named Isaac Newton

Newton ← ($\sigma_{Name = Isaac\ Newton}$(**CUSTOMER**)) $\bowtie_{CustomerID = CustomerID}$ (**CUSTOMER_PURCHASE_ORDER**)
Result ← $\sigma_{Quantity}$ [$\mathcal{F}_{SUM(Quantity)}$ (Newton)]

```
SELECT SUM(CPO.QUANTITY) AS BOOKS_PURCHASED
FROM CUSTOMER AS C, CUSTOMER_PURCHASE_ORDER AS CPO
WHERE CPO.CUSTOMERID = C.CUSTOMERID AND C.NAME = 'ISAAC NEWTON';
```

Query 6: Find the customer who has purchased the most books and the total number of books they have purchased

CustomerPurchases ← (**CUSTOMER**) $\bowtie_{CustomerID = CustomerID}$ (**CUSTOMER_PURCHASE_ORDER**)
TotalForEachCustomer ← $\rho_{(TotalQty)}[_{CustomerID}\mathcal{F}_{SUM(Quantity)}$ (CustomerPurchases)]
Result ← $\pi_{Fname,Lname,TotalQty}$ [$\mathcal{F}_{MAX(TotalQty)}$ (TotalForEachCustomer)]

```
SELECT NAME, MAX(BOOKS_PURCHASED)
FROM
        (SELECT C.NAME, SUM(CPO.QUANTITY) AS BOOKS_PURCHASED
        FROM CUSTOMER AS C, CUSTOMER_PURCHASE_ORDER AS CPO
        WHERE C.CUSTOMERID = CPO.CUSTOMERID
        GROUP BY C.CUSTOMERID);
```

Query 7: Find the salesperson who generated the most revenue (Books Sold * Price of Books)

EmployeeSales ← (**EMPLOYEE**) $\bowtie_{EmployeeID = EmployeeID}$ (**CUSTOMER_PURCHASE_ORDER**)
TotalForEachSalesperson ← $\rho_{(TotalSales)}[_{EmployeeID}\mathcal{F}_{SUM(Quantity * Price)}$ (EmployeeSales)]
Result ← $\pi_{Name,\ TotalSales}$ [$\mathcal{F}_{MAX(TotalSales)}$ (TotalForEachSalesperson)]

```
SELECT NAME, MAX(TOTAL_SALES) AS TOTAL_SALES
  FROM
    (SELECT E.NAME, SUM(CPO.PRICE * CPO.QUANTITY) AS TOTAL_SALES
    FROM EMPLOYEE AS E, CUSTOMER_PURCHASE_ORDER AS CPO
    WHERE E.EMPLOYEEID = CPO.EMPLOYEEID
    GROUP BY E.EMPLOYEEID);
```

Query 8: Find the total amount of sales revenue earned

Result ← $\pi_{TotalRevenue}\{(\rho_{(TotalSales)}[(\mathcal{F}_{SUM(Quantity * Price)}(\textbf{CUSTOMER\_PURCHASE\_ORDER})]\}$

```
SELECT SUM(CPO.PRICE * CPO.QUANTITY) AS TOTAL_REVENUE
FROM CUSTOMER_PURCHASE_ORDER AS CPO;
```

Query 9: Find the total amount of expense on inventory

Result ← $\pi_{TotalExpenses}\{(\rho_{(TotalExpenses)}[(\mathcal{F}_{SUM(Quantity * Purchase\_Price)}(\textbf{INVENTORY})]\}$

```
SELECT SUM(I.PURCHASE_PRICE * I.QUANTITY) AS TOTAL_COST_OF_INVENTORY
FROM INVENTORY AS I;
```

Query 10: Find the total cost of goods sold

BooksPurchased ← (**BOOK**)$\bowtie_{ISBN = ISBN}$ (**CUSTOMER_PURCHASE_ORDER**)
Result ← $\mathcal{F}_{SUM(Quantity * Purchase\_Price)}$[(BooksPurchased) $\bowtie_{ISBN = ISBN}$ (**INVENTORY**)]

```
SELECT SUM(I.PURCHASE_PRICE * CPO.QUANTITY) AS COST_OF_GOODS_SOLD
FROM CUSTOMER_PURCHASE_ORDER AS CPO, BOOK AS B, INVENTORY AS I
WHERE B.ISBN = CPO.ISBN AND I.ISBN = CPO.ISBN;
```

Query 11: Provide a list of customer names, along with the total dollar amount each customer has spent

CustomerPurchases ← (**CUSTOMER**)$\bowtie_{CustomerID = CustomerID}$ (**CUSTOMER_PURCHASE_ORDER**)
PurchasePrices ← $\rho_{(Dollars\_Spent)}[_{CustomerID}\mathcal{F}_{SUM(Quantity * PurchaPrice)}$(CustomerPurchases)]
Result ← $\pi_{(Name, Dollars\_Spent)}$(PurchasePrices)

```
SELECT NAME, DOLLARS_SPENT
FROM
        (SELECT C.NAME, SUM(CPO.PRICE * CPO.QUANTITY) AS DOLLARS_SPENT
        FROM CUSTOMER AS C, CUSTOMER_PURCHASE_ORDER AS CPO
        WHERE C.CUSTOMERID = CPO.CUSTOMERID
        GROUP BY C.CUSTOMERID);
```

Query 12: Provide a list of customer names and e-mail addresses for customers who have spent more than the average customer

CustomerPurchases ← (**CUSTOMER**)$\bowtie_{CustomerID = CustomerID}$ (**CUSTOMER_PURCHASE_ORDER**)
PurchasePrices ← $\rho_{(Dollars\_Spent)}[_{CustomerID}\mathcal{F}_{SUM(Quantity * PurchaPrice)}$(CustomerPurchases)]
Average ←$\mathcal{F}_{AVG(Dollars\_Spent)}\{\rho_{(Dollars\_Spent)}\ [_{CustomerID}\mathcal{F}_{SUM(Quantity * PurchaPrice)}$(CustomerPurchases)]}

AboveAvgCustomers ← $\pi_{\textbf{PurchasePrices > Average}}$**(CUSTOMER)**
Result ← $\pi_{(\text{Name, Email})}$(AboveAvgCustomers)

```
SELECT C.NAME, C.EMAIL
FROM CUSTOMER AS C, CUSTOMER_PURCHASE_ORDER AS CPO
WHERE C.CUSTOMERID = CPO.CUSTOMERID
GROUP BY C.CUSTOMERID
HAVING SUM(CPO.PRICE * CPO.QUANTITY) >
        (SELECT AVG(DOLLARS_SPENT)
        FROM
                (SELECT SUM(CPO.PRICE * CPO.QUANTITY) AS DOLLARS_SPENT
                FROM CUSTOMER AS C, CUSTOMER_PURCHASE_ORDER AS CPO
                WHERE C.CUSTOMERID = CPO.CUSTOMERID
                GROUP BY C.CUSTOMERID));
```

Query 13: 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

PurchasedBooks ← (**BOOK**)⋈$_{\text{ISBN = ISBN}}$ (**CUSTOMER_PURCHASE_ORDER**)
Result ← $\pi_{(\text{Title, SUM(Quantity)})}$ ($_{\text{ISBN}}\mathcal{F}_{\text{SUM(Quantity)}}$ (PurchasedBooks)

```
SELECT B.TITLE, SUM(CPO.QUANTITY) AS TOTAL_SOLD
FROM BOOK AS B, CUSTOMER_PURCHASE_ORDER AS CPO
WHERE B.ISBN = CPO.ISBN
GROUP BY B.ISBN
ORDER BY TOTAL_SOLD DESC;
```

Query 14: 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

PurchasedBooks ← (**BOOK**)⋈$_{\text{ISBN = ISBN}}$ (**CUSTOMER_PURCHASE_ORDER**)
Result ← $\pi_{(\text{Title, SUM(Quantity * Price)})}$ ($_{\text{ISBN}}\mathcal{F}_{\text{SUM(Quantity * Price)}}$ (PurchasedBooks)

```
SELECT B.TITLE, SUM(CPO.QUANTITY * CPO.PRICE) AS TOTAL_SALES
FROM BOOK AS B, CUSTOMER_PURCHASE_ORDER AS CPO
WHERE B.ISBN = CPO.ISBN
GROUP BY B.ISBN
ORDER BY TOTAL_SALES DESC;
```

Query 15: Find the most popular author in the database (i.e. the one who has sold the most books)

PurchasedBooks ← (**WRITTEN_BY**)$\bowtie_{ISBN = ISBN}$ (**CUSTOMER_PURCHASE_ORDER**)
AuthorPurchases ← $\rho_{(TotalSales)}$ [$_{AuthorID}\mathcal{F}_{SUM(Quantity)}$(**AUTHOR**)$\bowtie_{ISBN = ISBN}$ (PurchasedBooks)]
Result ← $\pi_{(Title,\ SUM(Quantity))}$ $\mathcal{F}_{MAX(TotalSales)}$(AuthorPurchases)

```
SELECT NAME, MAX(TOTAL_BOOKS_SOLD)
FROM
        (SELECT A.NAME, SUM(CPO.QUANTITY) AS TOTAL_BOOKS_SOLD
        FROM AUTHOR AS A, CUSTOMER_PURCHASE_ORDER AS CPO, WRITTEN_BY AS W
        WHERE A.AUTHORID = W.AUTHORID AND W.ISBN = CPO.ISBN
        GROUP BY A.AUTHORID);
```

Query 16: Find the most profitable author in the database for this store (i.e. the one who has brought in the most money)

PurchasedBooks ← (**WRITTEN_BY**)$\bowtie_{ISBN = ISBN}$ (**CUSTOMER_PURCHASE_ORDER**)
AuthorPurchases ← $\rho_{(TotalSales)}$ [$_{AuthorID}\mathcal{F}_{SUM(Quantity * Price)}$(**AUTHOR**)$\bowtie_{ISBN = ISBN}$ (PurchasedBooks)]
Result ← $\pi_{(Title,\ MAX(TotalSales))}$ $\mathcal{F}_{MAX(TotalSales)}$(AuthorPurchases)

```
SELECT NAME, MAX(TOTAL_SALES)
FROM
        (SELECT A.NAME, SUM(CPO.QUANTITY * CPO.PRICE) AS TOTAL_SALES
        FROM AUTHOR AS A, CUSTOMER_PURCHASE_ORDER AS CPO, WRITTEN_BY AS W
        WHERE A.AUTHORID = W.AUTHORID AND W.ISBN = CPO.ISBN
        GROUP BY A.AUTHORID);
```

Query 17: Provide a list of customer information for customers who purchased anything written by the most profitable author in the database

PurchasedBooks ← (**WRITTEN_BY**)$\bowtie_{ISBN = ISBN}$ (**CUSTOMER_PURCHASE_ORDER**)
AuthorPurchases ← $\rho_{(TotalSales)}$ [$_{AuthorID}\mathcal{F}_{SUM(Quantity * Price)}$(**AUTHOR**)$\bowtie_{ISBN = ISBN}$ (PurchasedBooks)]
MostPopularAuthor ← $\pi_{(Title)}$ $\mathcal{F}_{MAX(TotalSales)}$(AuthorPurchases)
CustomerPurchases ← (**CUSTOMER**)$\bowtie_{CustomerID = CustomerID}$ (**CUSTOMER_PURCHASE_ORDER**)
Result ← $\pi_{(*)}$(CustomerPurchases) $\bowtie_{AuthorID = AuthorID}$ (MostPopularAuthor)

```
SELECT C.*
FROM CUSTOMER AS C, WRITTEN_BY AS W, CUSTOMER_PURCHASE_ORDER AS CPO,
        (SELECT A.AUTHORID
        FROM
                (
```

```
            SELECT A.AUTHORID, MAX(TOTAL_SALES)
            FROM
                  (SELECT A.AUTHORID, SUM(CPO.QUANTITY * CPO.PRICE) AS TOTAL_SALES
                  FROM AUTHOR AS A, CUSTOMER_PURCHASE_ORDER AS CPO, WRITTEN_BY AS W
                  WHERE A.AUTHORID = W.AUTHORID AND W.ISBN = CPO.ISBN
                  GROUP BY A.AUTHORID) AS A)
      AS X
WHERE X.AUTHORID = W.AUTHORID AND W.ISBN = CPO.ISBN AND C.CUSTOMERID = CPO.CUSTOMERID;
```

Query 18: Provide the list of authors who wrote the books purchased by the customers who have spent more than the average customer

CustomerPurchases ← (**CUSTOMER**)$\bowtie_{CustomerID = CustomerID}$ (**CUSTOMER_PURCHASE_ORDER**)
PurchasePrices ← $\rho_{(Dollars\_Spent)}[_{CustomerID}\mathcal{F}_{SUM(Quantity * PurchaPrice)}$(CustomerPurchases)]
Average ←$\mathcal{F}_{AVG(Dollars\_Spent)}\{\rho_{(Dollars\_Spent)}\ [_{CustomerID}\mathcal{F}_{SUM(Quantity * PurchaPrice)}$(CustomerPurchases)]}
AboveAvgCustomers ← $\pi_{(Name, Email)}[\ \pi_{\mathbf{PurchasePrices > Average}}$**(CUSTOMER)]**
X ← (AboveAverageCustomers) $\bowtie_{CustomerID = CustomerID}$ (**CUSTOMER_PURCHASE_ORDER**)
Y ← (**AUTHOR**) $\bowtie_{AuthorID = AuthorID}$ [(X) $\bowtie_{ISBN = ISBN}$ (**WRITTEN_BY**)]
Result ← $\pi_{(Name)}$ (Y)

```
SELECT DISTINCT(A.NAME)
FROM CUSTOMER_PURCHASE_ORDER AS CPO, AUTHOR AS A, WRITTEN_BY AS W,
      (
      SELECT C.CUSTOMERID
      FROM CUSTOMER AS C, CUSTOMER_PURCHASE_ORDER AS CPO
      WHERE C.CUSTOMERID = CPO.CUSTOMERID
      GROUP BY C.CUSTOMERID
      HAVING SUM(CPO.PRICE * CPO.QUANTITY) >
            (
            SELECT AVG(DOLLARS_SPENT)
            FROM
                  (
                  SELECT SUM(CPO.PRICE * CPO.QUANTITY) AS DOLLARS_SPENT
                  FROM CUSTOMER AS C, CUSTOMER_PURCHASE_ORDER AS CPO
                  WHERE C.CUSTOMERID = CPO.CUSTOMERID
                  GROUP BY C.CUSTOMERID
                  )
            )
      ) AS X
WHERE X.CUSTOMERID = CPO.CUSTOMERID AND A.AUTHORID = W.AUTHORID AND W.ISBN = CPO.ISBN
```

INSERT SYNTAX

This section describes the syntax necessary to insert into each table in the database. Insertions that involve dependencies on other tables will be specified. The following insertion examples display all of the attributes regardless of whether or not they are required. Underlined attributes represent the Primary Key of the relation. Attributes in black text represent attributes that can NOT be NULL, and attributes in red text represent attributes that can be NULL.

As a general rule, the database is set up such that any manipulations to inventory will begin with a purchase order. This can either be a CUSTOMER_PURCHASE_ORDER, which decreases INVENTORY, or a BOOKSTORE_PURCHASE_ORDER, which increases INVENTORY. The purchase orders incur more dependencies than any other tables in the database.

BOOK

INSERT INTO BOOK VALUES (ISBN, TITLE, CATEGORY, PUBLISHER, PUBLISHDATE);

Inserting a new BOOK into the database triggers the following insertion transaction in order:

BOOK → *INVENTORY → **AUTHOR → WRITTEN_BY
*INVENTORY does not necessarily need to be updated. The insertion of a BOOK entity does not mean the physical book is in INVENTORY
**INSERT into AUTHOR can be skipped if the author already exists in the database

AUTHOR

INSERT INTO AUTHOR VALUES (AUTHORID, NAME);

Inserting a new AUTHOR does not require insertion into any other tables.

WRITTEN_BY

INSERT INTO WRITTEN_BY VALUES (AUTHORID, ISBN);

Inserting a new instance of WRITTEN_BY does not require insertion into any other tables.

CUSTOMER

INSERT INTO CUSTOMER VALUES
(CUSTOMERID, NAME, EMAIL, PHONE, ADDRESS, CITY, STATE, ZIP, SEX);

Inserting a new CUSTOMER does not require insertion into any other tables.

EMPLOYEE

INSERT INTO EMPLOYEE VALUES (EMPLOYEEID, NAME, SALARY, DEPARTMENT);

Inserting a new EMPLOYEE does not require insertion into any other tables.

INVENTORY

INSERT INTO INVENTORY VALUES
(ISBN, QUANTITY, PURCHASE_PRICE, SELLING_PRICE);

Inserting a new instance of INVENTORY will trigger one of two possible insertion transactions.

(1) The book being added to inventory does not yet exist in the database. In this case INVENTORY triggers the following insertions in order:

INVENTORY → BOOK → *AUTHOR → WRITTEN_BY
*INSERT into AUTHOR can be skipped if the author already exists in the database

(2) The book being added already exists in inventory. In this case the insertion transaction only involves INVENTORY.

INVENTORY

CUSTOMER_PURCHASE_ORDER

INSERT INTO CUSTOEMER_PURCHASE_ORDER VALUES
(TIN, TIME, PRICE, QUANTITY, ISBN, CUSTOMERID, EMPLOYEEID);

Inserting a new instance of CUSTOMER_PURCHASE_ORDER triggers the following transaction in order:

CUSTOMER_PURCHASE_ORDER → *INVENTORY
*INVENTORY must be updated to account for the change in the quantity of books

BOOKSTORE_PURCHASE_ORDER

INSERT INTO BOOKSTORE_PURCHASE_ORDER VALUES
(TIN, TIME, PRICE, QUANTITY, ISBN, EMPLOYEEID);

An instance of BOOKSTORE_PURCHASE_ORDER represents the purchase of a book by Bits & Books. There are two possible transactions that may take place when a new instance of BOOKSTORE_PURCHASE_ORDER is created.

(1) The book that has been purchased does not yet exist in the database. In this case BOOKSTORE_PURCHASE_ORDER triggers the following transaction in order:

BOOKSTORE_PURCHASE_ORDER → BOOK → *AUTHOR → WRITTEN_BY → **INVENTORY
*INSERT into AUTHOR can be skipped if the author already exists in the database
**INVENTORY must be updated to account for the change in the quantity of books

(2) The book that has been purchased already exists in the database. In this case BOOKSTORE_PURCHASE_ORDER triggers the following transaction in order:

BOOKSTORE_PURCHASE_ORDER → *INVENTORY
**INVENTORY must be updated to account for the change in the quantity of books

DELETE SYNTAX

This section describes the syntax necessary to delete data from each of the tables in the database. In each example, only one tuple is being removed at a time. It is also worth noting that the tuple being removed in all of the following examples is identified by its Primary Key. The Primary Key of the relation will be used to identify a specific tuple in the table. Performing a DELETE that includes an attribute other than the Primary Key is unsafe and could result in unintended results. The '?' in each of the DELETE statements represents the Primary Key of the tuple to be deleted.

Many of the tables include dependencies or references to other tables in the database. In these situations, the relevant tables will be listed.

BOOK

```
DELETE FROM BOOK
WHERE ISBN = ?;
```

The above command should be issued only when the intention is to remove all records of a BOOK in the database. In order to do so, any tuple containing the desired ISBN would need to be deleted from the following tables:

BOOKSTORE_PURCHASE_ORDER, CUSTOMER_PURCHASE_ORDER, WRITTEN_BY, INVENTORY

AUTHOR

```
DELETE FROM AUTHOR
WHERE AUTHORID = ?;
```

The above command should only be issued when the intention is to remove all records of an AUTHOR in the database. In order to do so, any tuple containing the desired AuthorID would need to be deleted from the following tables:

WRITTEN_BY

WRITTEN_BY

```
DELETE FROM WRITTEN_BY
WHERE AUTHORID = ? AND ISBN = ?;
```

The above command should only be issued when the intention is to remove an instance of WRITTEN_BY in the database. This deletion is essentially removing the connection that an AUTHOR has with a BOOK. Before completing this deletion it is necessary to check two things. Is the book, identified by the ISBN, written by more than one author? If the answer is yes, then the ISBN does not necessarily need to be removed from the BOOK table. Has the author, identified by the AuthorID, written any other books that are currently stored in the database? If the answer is yes, then the AuthorID does not necessarily need to be removed from the AUTHOR table. After considering these possible dependencies, any tuple with the desired AuthorID or ISBN may need to be removed from the following tables:

BOOK, AUTHOR, INVENTORY

CUSTOMER

```
DELETE FROM CUSTOMER
WHERE CUSTOMERID = ?;
```

The above command should be issued only when the intention is to remove all records of a particular CUSTOMER in the database.

EMPLOYEE

```
DELETE FROM EMPLOYEE
WHERE EMPLOYEEID = ?;
```

The above command should be issued only when the intention is to remove all records of a particular EMPLOYEE in the database.

INVENTORY

```
DELETE FROM INVENTORY
WHERE ISBN = ?;
```

The above command should be issued only when the intention is to remove all records of a BOOK in the INVENTORY. This may happen when the quantity of a particular book in stock reaches zero, and Bits & Books has no intention of purchasing additional copies of the book. In order to perform this deletion, any tuple containing the desired ISBN would need to be deleted from the following tables:

WRITTEN_BY, BOOK, *AUTHOR
*AUTHOR would not need to be deleted if the author has written other books in the database

CUSTOMER_PURCHASE_ORDER

```
DELETE FROM CUSTOMER_PURCHASE_ORDER
WHERE TIN = ?;
```

The above command removes a specific transaction from the table. When deleting a transaction, it is important to update INVENTORY to reflect any associated change in quantity. Other than that, there are no other direct relations.

BOOKSTORE_PURCHASE_ORDER

```
DELETE FROM BOOKSTORE_PURCHASE_ORDER
WHERE TIN = ?;
```

The above command removes a specific transaction from the table. When deleting a transaction, it is important to update INVENTORY to reflect any associated change in quantity. Other than that, there are no other direct relations.