Bits & Books Project

User Manual

Date: 17 April 2024

Contributors:

Sean Kelley

Jack Kern

Tyra Shin

Justin Sparacino

Trey Thomas

Table Descriptions:

Membership(Mem_Id(FK), Email, Password)

This represents a membership that a customer can join. They have an integer Member_Id, which is a foreign key to their Customer_Id. They have a text email to keep track of their account and they also have a text password for their membership.

Person(Id, First_Name, Middle_Name, Last_Name)

This represents a person in the bookstore and is a superclass to customers and authors. We save an integer Id to uniquely identify a person. We also save their First_Name, Middle_Name and Last_name as text.

Author(Auth_Id(FK))

This is where we save authors and represent a person that is also an author. This is a subclass of a Person and the Auth_Id is a foreign key to the Person Id.

Writes(Auth_Id(FK), ISBN(FK))

This is a relationship between a book and the author(s) that write it. We save the integer Auth_Id as a foreign key. We also save the text ISBN as a foreign key. If a book has multiple authors, it will be included in multiple Writes rows.

Purchase(Order_No, Cust_Id(FK), Book(FK), Price, Purchase_Date, Store_No(FK))

This table holds all the purchases that a customer makes. Each purchase only has one book and the quantity. We save the Order_No as an integer to uniquely identify them. We save the Cust_Id as an integer foreign key so we know who made the purchase. We also save the book as a text foreign key to the book's ISBN, price as an integer, the Data of the purchase as text, and the store number that the purchase was made at as an integer foreign key.

Publisher(Pub_Id, Pub_Name)

This represents a book publisher. It has an integer Pub_Id that allows us to uniquely identify a publisher. It also has a text Pub_Name where we store the name of the publisher.

Customer(Cust_Id(FK), Phone_No)

This represents a customer at the bookstore. This is a subclass of a Person. We have an integer Cust_Id that is a foreign key to a person. We also have a Phone_No that we save for contact information for a customer.

Book(ISBN, Pub_Id(FK), Year, Price, Title)

This represents a book that the bookstore is selling. We save the text ISBN as the primary key to uniquely identify a book. We save an integer foreign key to a publisher Id. We also save an integer year for the year the book was published, a real Price for the book, a text title for the book and a text category for the book.

Bookstore(Store_No, Store_Loc)

This represents a location of Bits and Books. We have an integer primary key Store_no that uniquely identifies the store and a text store_loc that saves the location of the store. We use this to allow us to have different locations where customers can buy books.

Stores(ISBN(FK), Store_Id(FK), Quantity)

This is a relationship between a Book and a Bookstore. This is how we save the inventory of a book in a bookstore. We have a text primary key ISBN that refers to a book's ISBN, an integer primary key Store_no that refers to the store that the inventory is at. We also store an integer quantity

Order_Books (ISBN(FK), Order_No(FK), Quantity)

This is a relationship for books in a purchase. It has an integer primary key for Order_No which relates to the order number of a purchase. It also stores a text primary key ISBN that relates to a book's ISBN. Lastly, we store an integer quantity of books that the customer is purchasing.

Categories (ISBN(FK), Category)

This is a relationship for a book that saves the text foreign key of ISBN for a Book and saves the text category for the book. This is important because we use it to handle books that have multiple categories.

Sample Queries:

I: This query finds all the books in the database written by Terry Pratchett and costs less than $10.00.

$$R1 \leftarrow \sigma(Last\_Name = 'pratchett')(PERSON)$$

$$R2 \leftarrow (AUTHOR \bowtie (Auth\_Id=ID)R1)$$

$$R3 \leftarrow (Writes \bowtie (R2.Auth\_Id = Writes.Auth\_Id)R2)$$

$$R4 \leftarrow (BOOK \bowtie (BOOK.ISBN = R1.ISBN)R3)$$

$$R5 \leftarrow \sigma(Price<10.00)(R4)$$

$$R6 \leftarrow \pi(Title)(R5)$$

```
SELECT Title
FROM BOOK
JOIN Writes ON BOOK.ISBN = Writes.ISBN
JOIN AUTHOR ON AUTHOR.Auth_Id = writes.Auth_ID
JOIN PERSON ON AUTHOR.Auth_Id = PERSON.Id
WHERE Last_Name = 'pratchett' AND Price < 10.0;
```

II: This SQL query finds the total amount of books purchased by a singular customer, in this sample, the customer is Jack Orange.

$$R1 \leftarrow \sigma(First\_Name = \text{‘Jesse’ AND } Last\_Name = \text{‘Teal’})(PERSON)$$

$$R2 \leftarrow (CUSTOMER \bowtie Cust\_Id = IDR1)$$

$$R3 \leftarrow (PURCHASE \bowtie R2.Cust\_Id = BOOKS.Cust\_IDR2)$$

$$R4 \leftarrow (BOOKS \bowtie R3.Order\_No = BOOKS.Order\_NoR3)$$

$$R5 \leftarrow Cust\_Id\Im SUM\ Quantity(R4)$$

```
SELECT SUM(ORDER_BOOKS.Quantity)
FROM PURCHASE
JOIN CUSTOMER ON CUSTOMER.Cust_Id = PURCHASE.Cust_Id
JOIN PERSON ON PERSON.Id = CUSTOMER.Cust_Id
JOIN ORDER_BOOKS ON ORDER_BOOKS.Order_No = PURCHASE.Order_No
WHERE PERSON.First_Name = 'Jack' AND PERSON.Last_Name = 'Orange';
```

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

f MAX (num_books) (ρ (Cust_Id, Phone_no, Order_num, ISBN, Price, num_books) (Cust_Id f SUM(Quantity)(CUSTOMER ⋈ Cust_Id = Customer_ID PURCHASES)))

```
SELECT Cust_Id, SUM(Quantity) as Total_Books_Purchased
FROM PURCHASE
JOIN ORDER_BOOKS ON PURCHASE.Order_No = ORDER_BOOKS.Order_no
GROUP BY Cust_Id
ORDER BY Total_Books_Purchased DESC
LIMIT 1;
```

IV: Find the total number of books the publisher Pratchett has published.

fCOUNT(Pub_Id)((σName = “Pratchett”(PUBLISHER)) ⋈ Pub_Id = Publisher_ID BOOK)

$$R1 \leftarrow (AUTHOR \bowtie (Auth\_Id=ID)PERSON)$$

$$R2 \leftarrow (Writes \bowtie (R1.Auth\_Id = Writes.Auth\_Id)R1)$$

$$R3 \leftarrow (BOOK \bowtie (BOOK.ISBN = R1.ISBN)R2)$$

$$R4 \leftarrow \sigma Last\_Name= \text{‘pratchett’}(R3)$$

$$R5 \leftarrow fCOUNT(BOOk.ISBN)(R4)$$

```
SELECT COUNT(BOOK.ISBN)
FROM BOOK
JOIN PERSON ON AUTHOR.Auth_Id = PERSON.Id
JOIN Writes ON BOOK.ISBN = Writes.ISBN
JOIN AUTHOR ON AUTHOR.Auth_Id = writes.Auth_Id
WHERE Last_Name = 'pratchett';
```

V: Find the number of purchases made by customers with memberships.

R1

f COUNT(Member_num)(MEMBERSHIP ⋈ Cust_Id = Cust_Id PURCHASE)

```
SELECT COUNT(Mem_Id)
FROM MEMBERSHIP
JOIN CUSTOMER ON MEMBERSHIP.Mem_Id = CUSTOMER.Cust_ID
JOIN PURCHASE ON CUSTOMER.Cust_Id = PURCHASE.Cust_Id
JOIN PERSON ON PERSON.Id = CUSTOMER.Cust_Id
JOIN ORDER_BOOKS ON ORDER_BOOKS.Order_No = PURCHASE.Order_No
WHERE MEMBERSHIP.Mem_Id = CUSTOMER.Cust_Id;
```

VI: Find and list the titles and ISBNs of books that <a certain author> has written that are published by Pratchett.

AUTH_BOOK ← (((σAuth_Id = author_idAUTHOR)⋈Auth_Id = Author_ID”WRITES) * BOOK)

πtitle, ISBN(σName = “Pratchett”(AUTH_BOOK ⋈Publisher_ID = Pub_ID PUBLISHER))

```
SELECT title, ISBN
FROM (
    SELECT *
    FROM AUTHOR
    JOIN WRITES ON AUTHOR.Auth_Id = WRITES.Auth_ID
    JOIN BOOK ON WRITES.ISBN = BOOK.ISBN
    WHERE WRITES.Auth_Id = 'author_id'
) AS AUTH_BOOK
JOIN PUBLISHER ON AUTH_BOOK.Pub_ID = PUBLISHER.Pub_ID
WHERE Pub_Name = 'Pratchett';
```

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

π(title,ISBN)(σQuantity<5(BOOK⋈(BOOK.ISBN=STORES.ISBN)STORES))

```
SELECT Title, BOOK.ISBN
FROM BOOK
JOIN STORES ON BOOK.ISBN = STORES.ISBN
GROUP BY Title, BOOK.ISBN
HAVING SUM(STORES.Quantity) < 5;
```

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

CUSTOMER_BOOK_PUB ← (PURCHASE ⋈ Book = ISBN (BOOK ⋈ Publisher = Pub_Id PUBLISHER)) π Customer, Title (σ Name = “Pratchett” (CUSTOMER_BOOK_PUB))

```
SELECT First_Name, Last_Name, Title

FROM PERSON

JOIN CUSTOMER ON PERSON.ID = CUSTOMER.Cust_Id

JOIN PURCHASE ON CUSTOMER.Cust_Id = PURCHASE.Cust_Id

JOIN ORDER_BOOKS ON PURCHASE.Order_No = ORDER_BOOKS.Order_No

JOIN BOOK ON ORDER_BOOKS.ISBN = BOOK.ISBN

JOIN PUBLISHER ON BOOK.Pub_Id = PUBLISHER.Pub_Id

WHERE Pub_Name = 'Pratchett';
```

IX: Update the price of all books published by a certain publisher.

```
UPDATE BOOK

SET Price = Price + 10

WHERE Pub_Id IN (

   SELECT Pub_Id

   FROM PUBLISHER

   WHERE Pub_Name = ‘Pratchett’

);
```

X: Updates the last name of a Customer using their Cust_Id:

```
UPDATE PERSON
SET Last_Name = Buckeye
WHERE ID IN (
   SELECT Cust_Id
   FROM CUSTOMER
   WHERE Cust_Id = 3584
);
```

Insert Commands:

To insert a book, you must have already inserted the publisher. You will also want to insert a Writes

Insert Book:

INSERT INTO Book (ISBN, Title, Pub_Id, Year, Price) VALUES (@isbn, @title, @pubId, @year, @price);

To insert a category, you must have a book already that you want to assign the category to.

Insert Categories:

INSERT INTO Categories (ISBN, Category) VALUES (@isbn, @category);

Insert Publisher:

INSERT INTO Publisher (Pub_Name) VALUES (@name)

Insert Person:

INSERT INTO Person (First_Name, Middle_Name, Last_Name) VALUES (@firstName, @middleName, @lastName);

To insert an author, you must first create them as a person.

Insert Author:

INSERT INTO Author (Auth_Id) VALUES (@personId);

To insert a customer, you must first create them as a person.

Insert Customer:

INSERT INTO Customer (Cust_Id, Phone_No) VALUES (@personId, @phone);

To insert a membership, you must first have a customer created already for the membership.

Insert Membership:

INSERT INTO Membership (Mem_Id, Email, Password) VALUES (@memberId, @email, @password);

To insert into stores, you must first has a book created and a bookstore to store it in created.

Insert Stores:

INSERT INTO Stores(ISBN, Store_Id, Quantity) VALUES (@isbn, @storeNumber, @quantity);

Insert Bookstore:

INSERT INTO Bookstore (Store_Loc) VALUES (@storeLoc);

To insert into writes, you must first have a book created and an author created.

Insert Writes:

INSERT INTO Writes (ISBN, Auth_ID) VALUES (@isbn, @authId);

To insert into Order_Books, you must first have a book and a purchase to add it to.

Insert Order_Books:

INSERT INTO Order_Books (ISBN, Quantity, Order_No) VALUES (@isbn, @quantity, @orderNumber);

To insert into Purchase you must have a Customer, BookStore, and Book(s), which are inserted into the Books table.

Insert Purchase:

INSERT INTO Purchase (Cust_Id, Store_No, Price, Date) VALUES (@custId, @storeNumber, @price, @date);

Delete Commands:

To delete a Book, you must also delete all: WRITES with that book, BOOKS with that book, STORES with that book, and CATEGORIES with that book.

Delete Book:

```
DELETE FROM Book WHERE ISBN = @isbn;
```

Delete Categories:

```
DELETE FROM Categories WHERE ISBN = @isbn AND Category = @category
```

To delete a Publisher, you must also delete all books that are published by that publisher.

Delete Publisher:

```
DELETE FROM Publisher WHERE Pub_Name = @name; Or DELETE FROM Publisher WHERE
Pub_Id= @pubId;
```

To delete a Person, you must also delete all Customers and Authors that reference the person being deleted.

Delete Person:

```
DELETE FROM Person WHERE First_Name = @firstName AND Middle_Name = @middleName
AND Last_Name = @lastName;
Or
DELETE FROM Person where Id = @id;
```

To delete an Author, you must first delete the Person associated with them.

Delete Author:

```
DELETE FROM Author WHERE Auth_Id = @authId;
```

To delete a Customer, you must first delete the Person associated with them.

Delete Customer:

```
DELETE FROM Customer WHERE Cust_Id = @id;
```

To delete a Membership, you must first delete the Customer associated with it.

Delete Membership:

```
DELETE FROM Membership WHERE Mem_Id = @id;
```

Delete Writes:

```
DELETE FROM Writes WHERE Auth_Id = @id AND ISBN = @isbn;
```

Delete Stores:

```
DELETE FROM Stores WHERE ISBN = @isbn AND Store_Id = @storeId;
```

Delete Order_Books:

```
DELETE FROM Order_Books WHERE ISBN = @isbn AND Order_No = @orderNumber;
```

To delete a bookstore, you must also delete all Purchases from that bookstore and all Stores for that Bookstore.

Delete Bookstore:

```
DELETE FROM Bookstore WHERE Store_Id = @storeId;
```

To delete a Purchase you must delete all Books for that purchase.

Delete Purchase:

```
DELETE FROM Purchase WHERE Order_No = @orderNumber;
```