Bits & Books Project

Database Description

Date: 17 April 2024

Contributors:

Sean Kelley

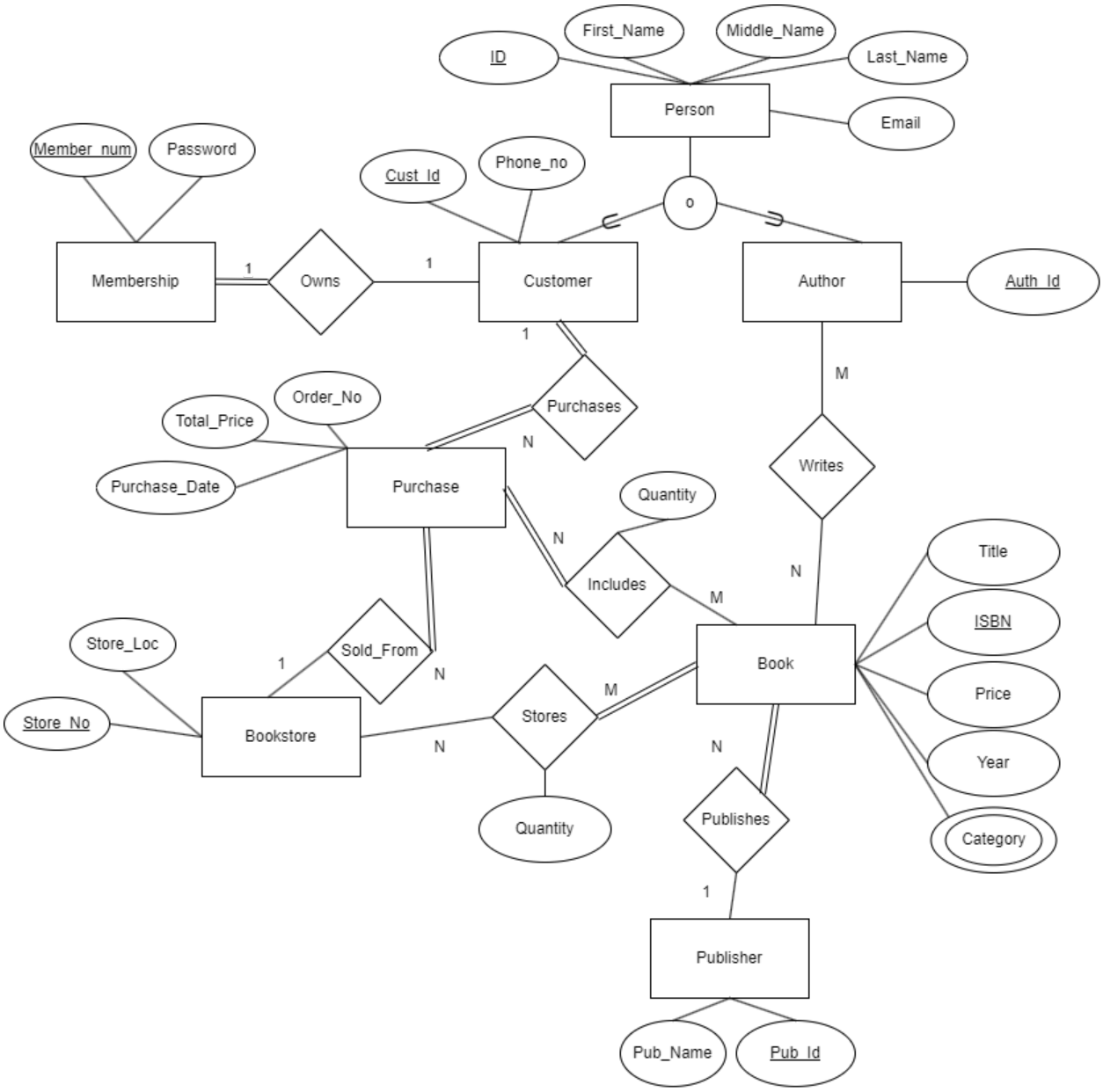
Jack Kern

Tyra Shin

Justin Sparacino

Trey Thomas

EER Diagram:


First_Name
Middle_Name
ID
Last_Name
Person
Email
Member_num
Password
Cust_Id
Phone_no
o
Membership
1
Owns
1
Customer
Author
Auth_Id
1
M
Purchases
Order_No
Total_Price
N
Writes
Purchase_Date
Purchase
Quantity
N
Includes
M
N
Title
ISBN
Store_Loc
Sold_From
1
N
Book
Price
Store_No
Bookstore
Stores
M
Year
N
N
Quantity
Publishes
Category
1
Publisher
Pub_Name
Pub_Id

Relational Schema:

Membership(Mem_Id(FK), Email, Password)

Foreign Key Mem_Id refers to Customer Id

Person(Id, First_Name, Middle_Name, Last_Name)

Author(Auth_Id(FK))

Foreign Key Auth_Id refers to Person Id

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

Foreign Key Auth_Id refers to Author Auth_Id

Foreign Key ISBN refers to Book ISBN

Purchase(Order_No, Cust_Id(FK), Total_Price, Date, Store_No(FK))

Foreign Key Cust_Id refers to Customer Id

Foreign Key Store_no refer s to BookStore Store_No

Customer(Cust_Id(FK), Phone_no)

Foreign Key Cust_Id refers to Person Id

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

Foreign Key ISBN refers to Book ISBN

Foreign Key Order_No refers to Purchase Order_No

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

Foreign Key Pub_Id refers to Publisher Pub_Id

Bookstore(Store_No, Store_Loc)

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

Foreign Key ISBN refers to Book ISBN

Foreign Key Store_Id refers to Bookstore Store_No

Category(ISBN(FK), Category)

Foreign Key ISBN refers to Book ISBN

Publisher(Pub_Id, Pub_name)

Functional Dependencies:

Membership:

Mem_Id -> Password

Person:

Id -> (First_Name, Middle_Name, Last_Name, Email)

Author:

Auth_Id -> None (since there are no other attributes)

Writes:

(Author_Id, ISBN) -> None (since there are no other attributes)

Purchase:

Order_no -> (Cust_Id, Store_No, Total_Price, Date)

Customer:

Cust_Id -> Phone_No

Order_Books:

(ISBN, Order_No) -> None (since there are no other attributes)

Book:

ISBN -> (Pub_Id, Year, Price, Title, Category)

Bookstore:

Store_No -> Store_Loc

Stores:

(ISBN, Store_Id) -> Quantity

Category:

(ISBN, Category) -> None (No other attributes)

Publisher:

Pub_Id -> Pub_Name

Normalized Forms:

Person: BCNF
Book: BCNF
Bookstore: BCNF
Customer: BCNF
Membership: BCNF
Author: BCNF
Writes: BCNF
Order_Books: BCNF
Publisher: BCNF
Stored: BCNF
Category: BCNF

Indices:

I: Index for the table BOOK, indexed based off of the publisher ID. There will be a large number of books populating this table, and publisher is likely one of the more common attributes of a Book for sorting.

```
CREATE INDEX BOOK_PUB

ON BOOK(PUB_ID);
```

II: Index for the table PERSON, indexed based off of first and last name. There are likely going to be many people added to this database, which will be referenced regularly in queries, and aside from ID (the key), name seems the second most likely identifier to be used in such queries.

```
CREATE INDEX PERSON_NAME

ON PERSON(First_Name, Last_Name);
```

III: Index added for the table PURCHASE, indexed off of the purchaser customer's ID. Many purchases are expected to be made, and it seems likely that sorting/filtering these purchases by customer will happen often.

```
CREATE INDEX PURCHASE_CUSTOMER

ON PURCHASE(CUST_ID);
```

Views:

View 1: Customer Purchases. This view could show each customer's total expenditure on books. It would involve joining the Customer and Purchase tables and aggregating on Total_Price.

```
CREATE VIEW Customer_Purchases AS

SELECT Customer.Cust_Id, SUM(Book.Price) as Total_Expenditure

FROM Customer

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

GROUP BY Customer.Cust_Id;
```

Customer_Purchases = π Cust_Id, Total_Expenditure (γ Cust_Id; SUM(Total_Price)->Total_Expenditure (σ Customer.Cust_Id = Purchase.Cust_Id (Customer x Purchase)))

3128	199.33
3132	67.8
3135	70.39
3137	184.88
3140	194.1

View 2: Author Book Inventory. This view would show the total quantity of all authors' books across all bookstores. It would involve joining the Book, Stored, and Writes tables and aggregating on Quantity.

```
CREATE VIEW Book_Inventory AS

SELECT Title, SUM(Quantity) as Total_Quantity

FROM Writes JOIN (Book JOIN Stores ON Book.ISBN = Stores.ISBN) ON Writes.ISBN =
Book. ISBN

GROUP BY Title;
```

Book_Inventory = π Title, Total_Quantity (γ Title; SUM(Quantity)->Total_Quantity (σ Writes.Auth_Id = Book.Auth_Id AND Book.ISBN = Stored.ISBN AND Auth_Id = <author Id> (Writes x (Book x Stored))))

A Field Guide to American Houses	1198852315
Amsterdam	2700553433
Analysis for Financial Management	325843906
Architecture: Form, Space, and Order	1231968515
Atonement	213600342

Sample Transactions:

I: This transaction represents the changes made to the database when a purchase is made. In this example, the book, “on writing,” by Stephen King, is purchased by a customer, so the database needs to add the purchase details to the Purchase relation and needs to update the quantity of the book in the store’s relation.

```
BEGIN TRANSACTION purchase
INSERT INTO PURCHASE(Order_No, Cust_ID, Book, Price,
Purchase_Date, Store_no) VALUES (12345, 4005,
'On Writing', 7.99, ‘Apr 15 2024’, 11);
IF error THEN GO TO UNDO; END IF;

UPDATE STORES
SET quantity = quantity - 1
JOIN BOOKSTORE ON BOOKSTORE.Store_no =
PURCHASE.Store_no
JOIN STORES ON BOOKSTORE.Store_no = Stores.Store_no
WHERE BOOKS.ISBN = STORES.ISBN;
IF error THEN GO TO UNDO; END IF;

COMMIT;
GO TO FINISH;

UNDO:
ROLLBACK;

FINISH:
END TRANSACTION;
```

II: This transaction registers a new customer into the database. This is useful as it allows us to maintain meaningful information about our customers in case they want to return products or want to be notified of sales and products in the future.

```
BEGIN TRANSACTION register_customer
INSERT INTO PERSON(ID, First_Name, Middle_Name,
Last_Name, Email) VALUES (12345, ‘John’,
‘W’, ‘Smith’, ‘smith1@email);
IF error THEN GO TO UNDO; END IF;
```

```
INSERT INTO CUSTOMER(Cust_Id, Phone_no) VALUES
(12345, 1234567890);
IF error THEN GO TO UNDO; END IF;

COMMIT;
GO TO FINISH;

UNDO:
ROLLBACK;

FINISH:
END TRANSACTION;
```

III: This transaction is useful as new books will be added to the database, so it is convenient to have a transaction that can easily update the database for us.
BEGIN TRANSACTION add_book

```
INSERT INTO PERSON(ID, First_Name, Middle_Name,
Last_Name, Email) VALUES (12350, 'Janice',
'B', 'Thomas', 'Thomas1@email);
IF error THEN GO TO UNDO; END IF;

INSERT INTO AUTHOR(Auth_Id) VALUES (12345);
IF error THEN GO TO UNDO; END IF;

INSERT INTO BOOK(ISBN, Pub_Id, Year, Price, Title, Category)
VALUES (123456789, 2015, 45.99, 'Intro To
Databases', Computer)
IF error THEN GO TO UNDO; END IF;

COMMIT;
GO TO FINISH;

UNDO:
ROLLBACK;

FINISH:
END TRANSACTION;
```