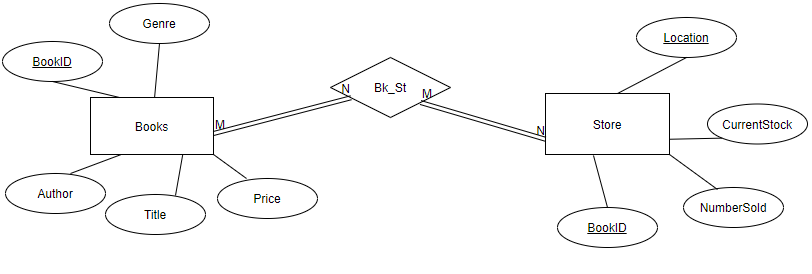
**Introduction**

The Williams & Smith bookstore chain is keeping track of the 1000 books that are currently being sold in their 100 book store locations. So far, the data that has been collected have been title, author, genre and the price of each book sold in the store. Each of these books are given a book ID and the same collection of books is being sold in every store location. We also have the data for the different locations, the current stock amount of the books as well as the number of books sold in each location.

**Complete ER Diagram:**



**Relational Algebra:**

**The following statement helps us see which author has the most titles on the shelves in the bookstores**

Author  Count Author (Books)

**This statement helps us see how many titles of each Genre is available in the book stores**

GenreCount Genre (Books)

**This statement helps us see the average price of the books being sold in the book stores**

Average Price(Books)

**We are able to see the total amount of books from each author across all the store locations**

R1 <- ( Books IXI Store)

Author SUM Number\_Sold(R1)

**We are able to see the total amount of money made from each store location with this statement**

R1 <- ( Books IXI Store)

R2 <- πLocation, Price, Number\_Sold, (Price\*Number\_Sold)

LocationLocation, SUM(Price\*Number\_Sold)(R2)

**DDL Statements:**

All the data is part of the table WSProjectData. I use the following DDL Statements to create the two above entities, Books and Store.

*Create Table Books*

*AS (Select \**

*From WSProjectData*

*WHERE Location = 'New York, NY')*

*Select \**

*From Books;*

*Alter Table Books Drop Column Location;*

*Alter Table Books Drop Column Current\_Stock;*

*Alter Table Books Drop Column Number\_Sold;*

*Alter Table Books ADD Primary Key(BookID);*

*Create Table Store As*

*Select Location, BookID, Current\_Stock,Number\_Sold*

*From WSProjectData;*

*Alter Table Store ADD Primary Key(Location,BookID);*

*Alter Table Store ADD Foreign Key(BookID) References Books(BookID);*

**DML Statements:**

**The following statement helps us see which author has the most titles on the shelves in the bookstores**

*Select Author, Count (\*)*

*From Books*

*Group by Author*

*Order by 2 Desc;*

**This statement helps us see how many titles of each Genre is available in the book stores**

*Select Genre, Count (\*)*

*From Books*

*Group By Genre*

*Order by 2 Desc;*

**This statement helps us see the average price of the books being sold in the book stores**

*Select Avg(Price)*

*from Books;*

**We are able to see the total amount of books from each author across all the store locations**

*Select Author, Sum (Number\_Sold) AS Total\_Number\_Of\_Books\_Sold*

*from Books*

*Natural Join Store*

*Group By Author*

*Order By 2 Desc;*

**We are able to see the total amount of money made from each store location with this statement**

*Select Location, Sum(Revenue\_Per\_Book) AS Total\_Revenue\_Per\_Store*

*From*

*(Select Location, Price,Number\_Sold, (Number\_Sold\*Price)AS Revenue\_Per\_Book*

*from Books*

*Natural Join Store)*

*Group By Location*

*Order By 2 Desc;*

**PL/SQL Statements:**

**This function is used to find the total book sales of the specific book in a certain book store location**

Create or Replace Function Book\_Revenue (storeLocation VarChar2, book VarChar2)

Return Number

IS book\_rev Number := 0;

Begin

Select Location, Title, Price, Number\_Sold,(Number\_Sold\*Price) Into book\_rev

From Books Natural Join Stores

Where Location = storeLocation and Title = book;

Return book\_rev;

End;

/

**This procedure is used to let us know if a certain book is out of stock in a specific store location**

Create or Replace PROCEDURE ReStock (storeLocation VarChar2, book Number)

Begin

Select Location, BookID, Current\_Stock

From Books Natural Join Stores

Where Location = storeLocation and BookID = book;

IF (Current\_Stock = 0)

dbms\_output.put\_line('Need to Restock this book');

End IF;

End;

**This gives us a trigger whenever we update the price of a book and it is greater than $20 since that is considered to be an expensive book**

Create Or Replace Trigger highPrice Before Update On Books For Each Row when (NEW.Price > 20)

Begin

dbms\_output.put\_line('Old Price: ' || :OLD.Price);

dbms\_output.put\_line('New Price: ' || :NEW.Price );

dbms\_output.put\_line('This is a considerably more expensive book');

End;

/