BOOKSTORE PROJECT

Noname– 100000000

Noname– 100000000

About

We aim to design a well organized, user friendly e-bookstore database which can be used in phone applications or web applications. To establish this goal we have to keep our database simple but in the same time it must be detailed and effective. Some features are searching a book by genre, searching for best sellers, searching for top trending books. Users will be able to order books and share their experience by voting them.

Depending on the feedback a book gets its rate will change. The average of all votings will be the rate which is displayed on the corresponding book details.

Also on every order of that book the system will remember the total number of orders. Which will make it able to the user to know the top sellers.

Data and Requirement Analysis

The database will consist of tables book table, user table, order table, payment type table, publisher table and author table.

Books are identified by an unique book id and their title, isbn, price, rate, sold quantity, publish date and genre must be recorded. A book may belongs to more than one genre. Rate can take values between 0-5 which is

0 for default and isbn must be a 13 digit number. Sold quantitiy is 0 as default.

Orders are identified by an unique order id and their order date, feedback must be recorded. Feedback can take values between 0-5.

Users are identified by an unique user id and first name, last name, phone number, address, email, birthdate and age must be recorded. Address is composed of city, state, and zipcode. Phone number must consist of 11 digits.

Authors are identified by an unique author id and first name, last name, birthdate must be recorded.

Every order has a payment type which includes a unique payment id. Also payment types have payment infos (cash, card etc.). Default value for cash is ‘Nakit’

Publishers are identified by an unique publisher id and their name, phone number, adress must be recorded. Address is composed of city, state, and zipcode. Phone number must consist of 11 digits.

A author can have more then one book but atleast one. A book can have only one author.

A book may be ordered many times. But every order can contain one book.

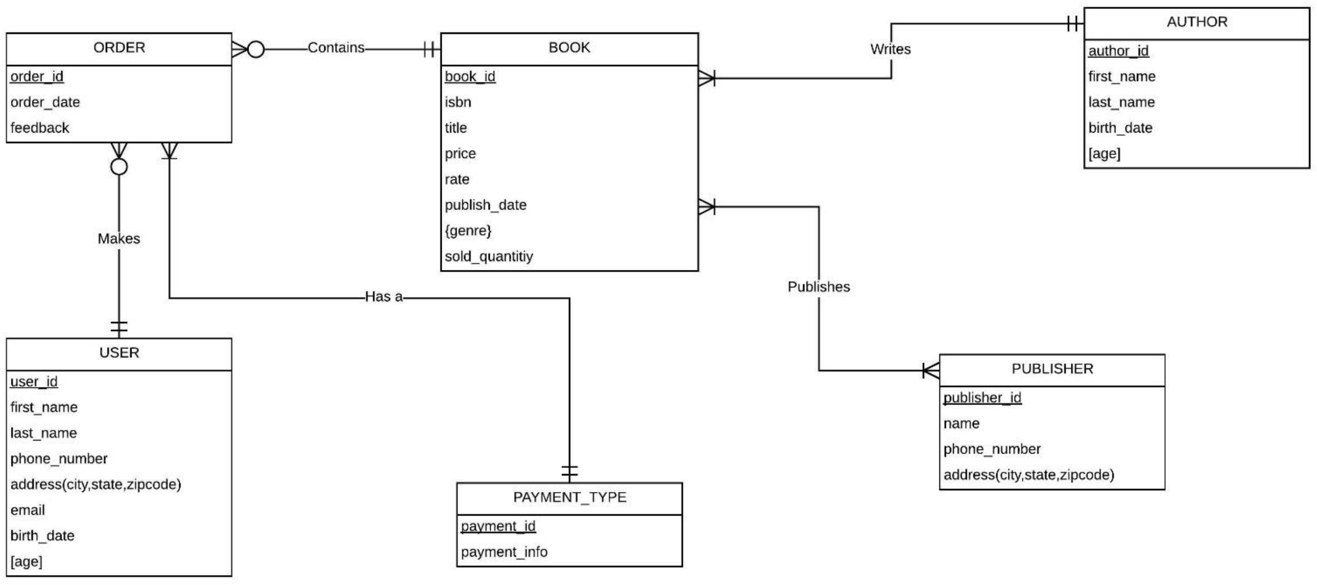
A user may have many orders. But every order must belong to a single user.

A order must be ordered with a single payment type.

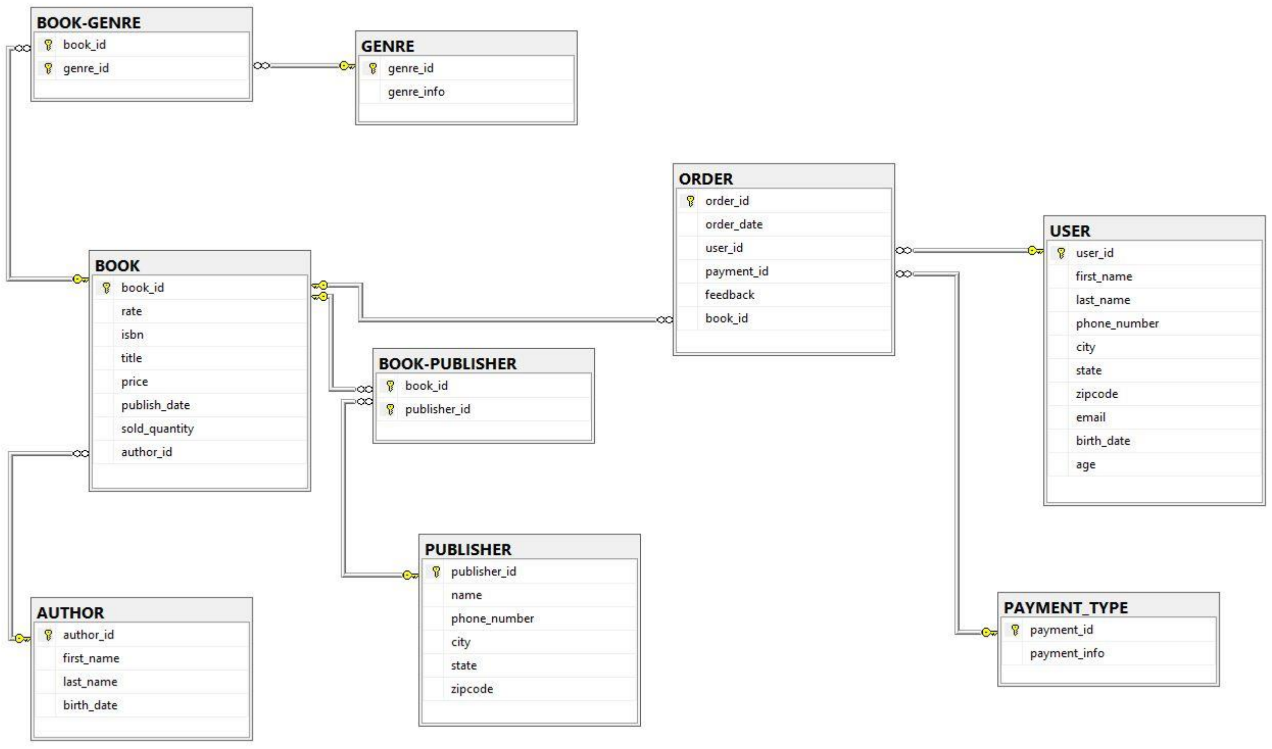
A book may published by many publishers. A publisher may publish many books.

For this project we have colloborated with the book store Mephisto kitabevi. Because of business privacy they didnt shared informations like the number of sells for every book but we could get a list of book names and their prices. We couldnt get a list of books for the project but they let us take photos and notes about the books informations.

E-R Diagram



Database Diagram



Information About the Tables & Attiributes

We created 9 tables for msql. These are Author, Book, Publisher,Payment_Type, Order, Genre, User, Book-Publisher, Book-Genre. Now we can explain these tables :

- BOOK

We created Book table which contains all books registered in the database. The Book table has 8 attributes :

book_id is primary key and unique. Every book has one book_id.

Rate is avarage of all feedbacks and rate can take values between 1-5.

ISBN is an international standart book number. Every book has a unique isbn number.

Title is the title of books.

Price is the price of the books (decimal(19,4)). We have used this data type because a price can take values like 19.90.

Publish_date shows date of publish for that book.

Sold quantity shows the number of orders for a corresponding book.

author_id points to an author data in the author table who wrote the corresponding book.

- AUTHOR

We created Author Table which contains 4 attributes :

author_id is a primary key and is unique. Every author has one unique author_id.

Author also has first name , last name and birth_date attributes. Birth date is stored in a special format date. (yy-mm-dd).

- PUBLISHER

We created Publisher table which contains 6 attributes:

publisher_id is primary key and unique.Every publisher has one unique publisher_id.

publisher also has name, city, state, zipcode.

phone number is unique for publishers.

- GENRE

We created Genre table. Genre shows the category of that book. Genre table contains two attiribute:

genre_id is primary key and unique.

genre_info is that takes nvarchar(50).(adventure,horror etc.)

- USER

We created User table because we will use these datas to create the UI for users. User table has 10 attributes:

first_name, last_name, city, state, zipcode .

email and phone_number are unique.

age calculated from birth_date. We have implemented a formula in the database. (CONVERT([smallint],datediff(year,[birth_date],getdate()))).

- ORDER

We created Order table because we need store the orders. This table has 6 attributes:

order_id is primary key and unique.

order date shows date of order. Which has a datatype date.

user_id, book_id and payment_id are foreign keys.

feedback takes integer value between 1-5 for rate.

- PAYMENT_TYPE

We created payment_type table because every order can be ordered by two types of payment(cash or credit card). This table has 2 attributes:

payment_id is primary key and unique.

payment info shows type of payment.

- BOOK-GENRE

We created Book-Genre table because a book can belong many genres and a genre can have many books.

This table has 2 foreign key :

book_id and genre_id.

- BOOK-PUBLISHER

We created Book-Publisher table because a book can be published by many publishers and obviously a Publisher publishes many boks. This table has 2 foreign key:

book_id and Publisher_id.

CONSTRAINTS -- INDICES -- DEFAULTS :

Indices

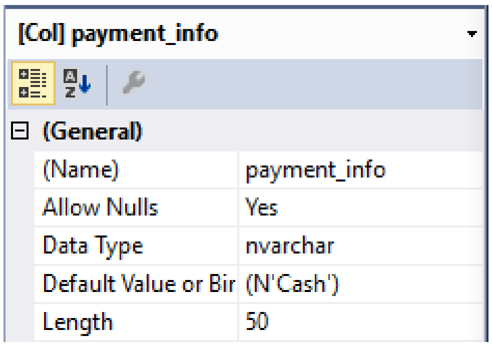
- We created indices for both uniqueness and primary keys : order_id, book_id, user_id, Publisher_id

Constraints

- We created check constraints for feedback , phone_number and isbn.
  We checked feedback from order table with this formula
  ([feedback]>(0) AND [feedback]<(6))
  We checked phone numbers from Publisher and user tables with this formula (len([phone_number])=(11)). This length of phone number is implemented for Turkey
  We checked isbn from book table with this formula
  (len([isbn])=(13))

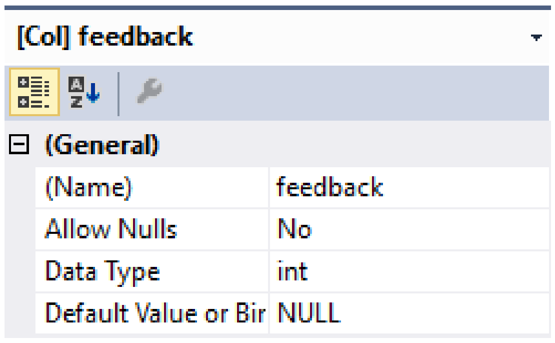
Defaults

- We implemented defaults value for payment_info, sold_quantity, rate, feedback.


[Col] payment_info

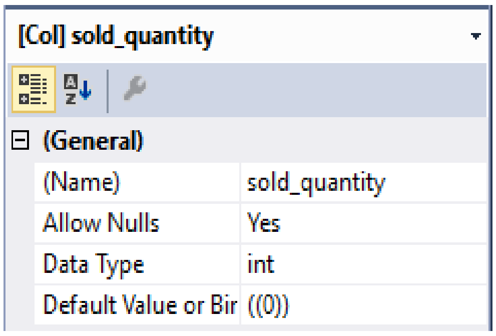
(General)

(Name)	payment_info
Allow Nulls	Yes
Data Type	nvarchar
Default Value or Bir	(N'Cash')
Length	50


[Col] feedback

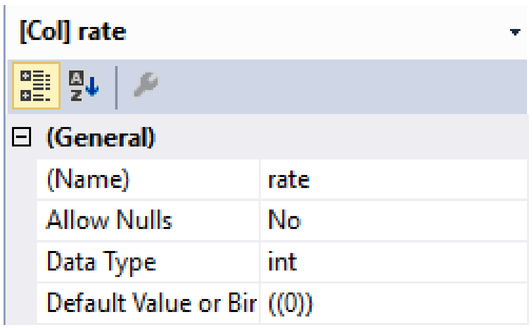
(General)

(Name)	feedback
Allow Nulls	No
Data Type	int
Default Value or Bir	NULL


[Col] sold_quantity

(General)

(Name)	sold_quantity
Allow Nulls	Yes
Data Type	int
Default Value or Bir	((0))

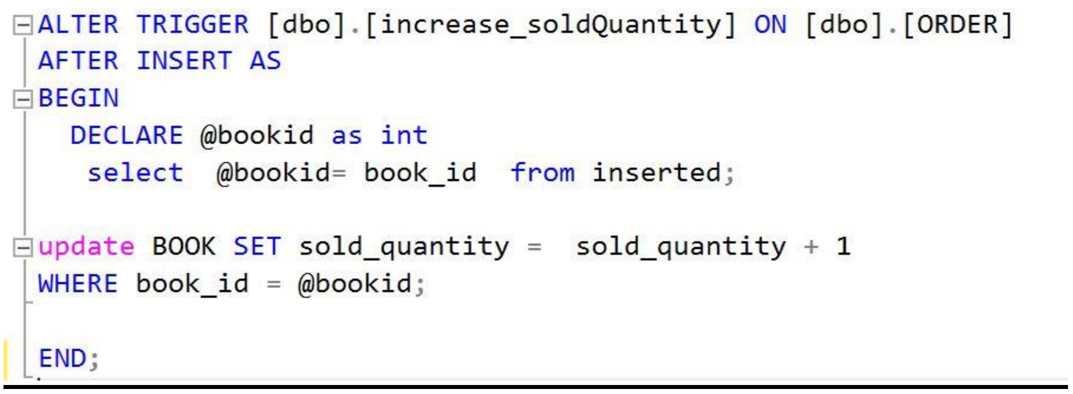

[Col] rate

(General)

(Name)	rate
Allow Nulls	No
Data Type	int
Default Value or Bir	((0))

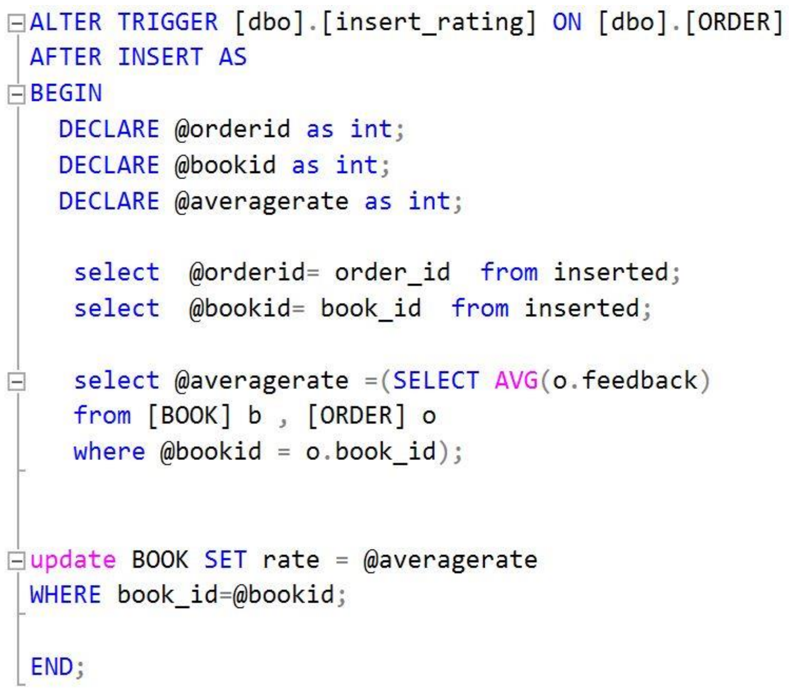
Triggers

Increase Sold Quantity Trigger


```
ALTER TRIGGER [dbo].[increase_soldQuantity] ON [dbo].[ORDER]
AFTER INSERT AS
BEGIN
   DECLARE @bookid as int
     select  @bookid= book_id  from inserted;

update BOOK SET sold_quantity =  sold_quantity + 1
WHERE book_id = @bookid;

END;
```

We have used this trigger to increase the sold quantity of a book by one every time the corresponding book is sold.

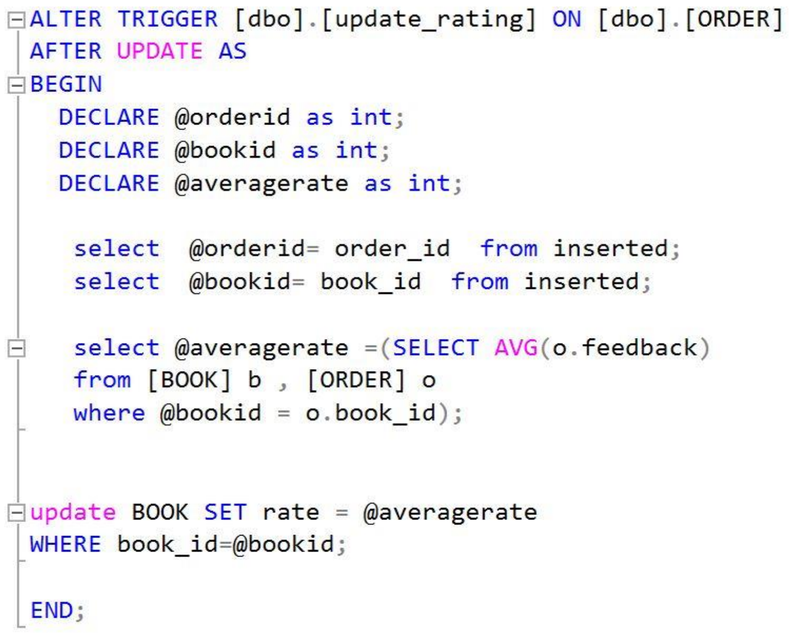
Insert Rating Trigger



```sql
ALTER TRIGGER [dbo].[insert_rating] ON [dbo].[ORDER]
AFTER INSERT AS
BEGIN
  DECLARE @orderid as int;
  DECLARE @bookid as int;
  DECLARE @averagerate as int;

   select  @orderid= order_id  from inserted;
   select  @bookid= book_id  from inserted;

   select @averagerate =(SELECT AVG(o.feedback)
   from [BOOK] b , [ORDER] o
   where @bookid = o.book_id);


update BOOK SET rate = @averagerate
WHERE book_id=@bookid;

END;
```

We have created this trigger on order table. It works as follows. Every time a new order is created the users feedback is calculated with all other orders of that book and the rate of that corresponding book changes. Every time a new voting is catched the rate changes (average of all votes). Votes are stored in the feedback variable of order.

Update Rating Trigger

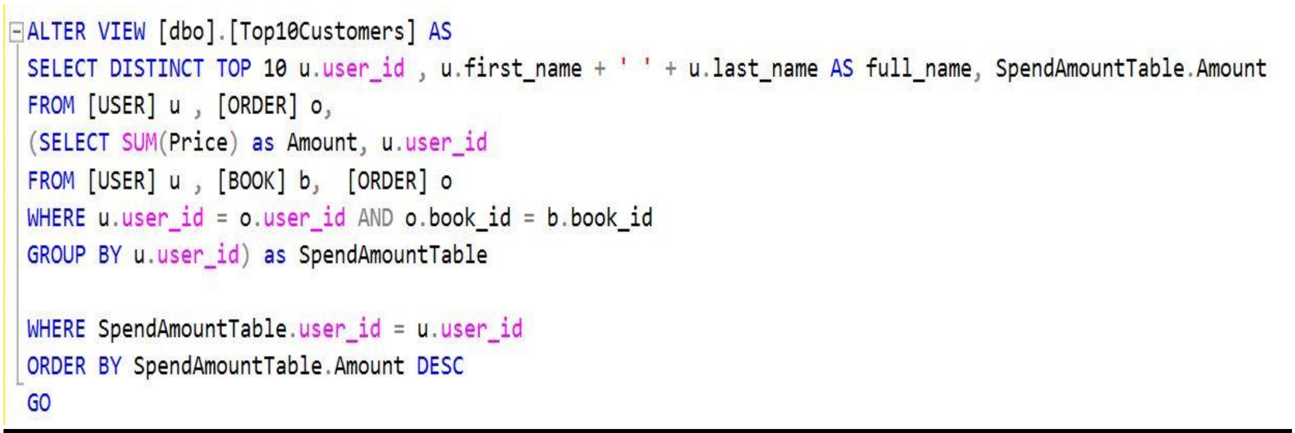


```sql
ALTER TRIGGER [dbo].[update_rating] ON [dbo].[ORDER]
AFTER UPDATE AS
BEGIN
  DECLARE @orderid as int;
  DECLARE @bookid as int;
  DECLARE @averagerate as int;

   select  @orderid= order_id  from inserted;
   select  @bookid= book_id  from inserted;

   select @averagerate =(SELECT AVG(o.feedback)
   from [BOOK] b , [ORDER] o
   where @bookid = o.book_id);


update BOOK SET rate = @averagerate
WHERE book_id=@bookid;

END;
```

This trigger is almost same as the insert trigger but it is triggered every time the user changes his/her mind and updates his/her vote.

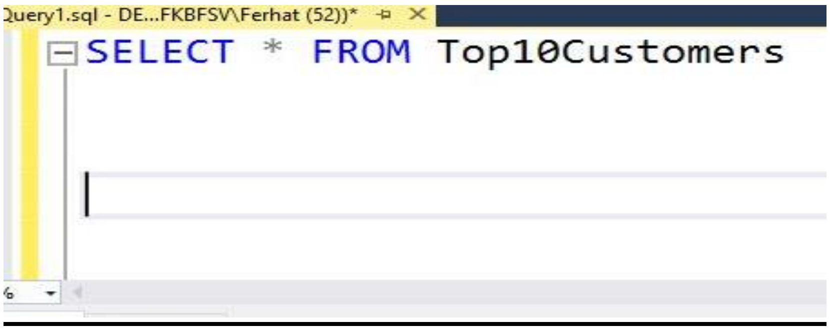
<u>Views</u>

<u>Top10Customers View</u>



```
ALTER VIEW [dbo].[Top10Customers] AS
SELECT DISTINCT TOP 10 u.user_id , u.first_name + ' ' + u.last_name AS full_name, SpendAmountTable.Amount
FROM [USER] u , [ORDER] o,
(SELECT SUM(Price) as Amount, u.user_id
FROM [USER] u , [BOOK] b,  [ORDER] o
WHERE u.user_id = o.user_id AND o.book_id = b.book_id
GROUP BY u.user_id) as SpendAmountTable

WHERE SpendAmountTable.user_id = u.user_id
ORDER BY SpendAmountTable.Amount DESC
GO
```

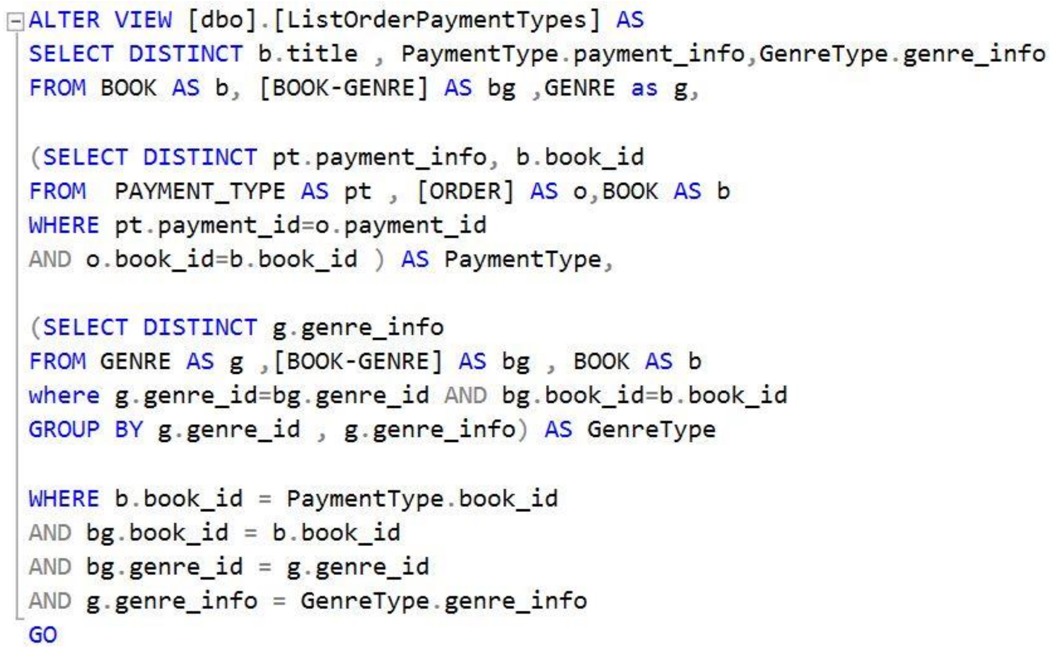
In this view we displayed the users that have spend the most money as top 10. We have combined first name and last name as full name. We have created a inner select query which returns the SpendAmountTable. The inner join returns the users with their total amount that they have spend in the system. From these rows we have used the top 10 with amount descending.



Query1.sql - DE...FKBFSV\Ferhat (52))*

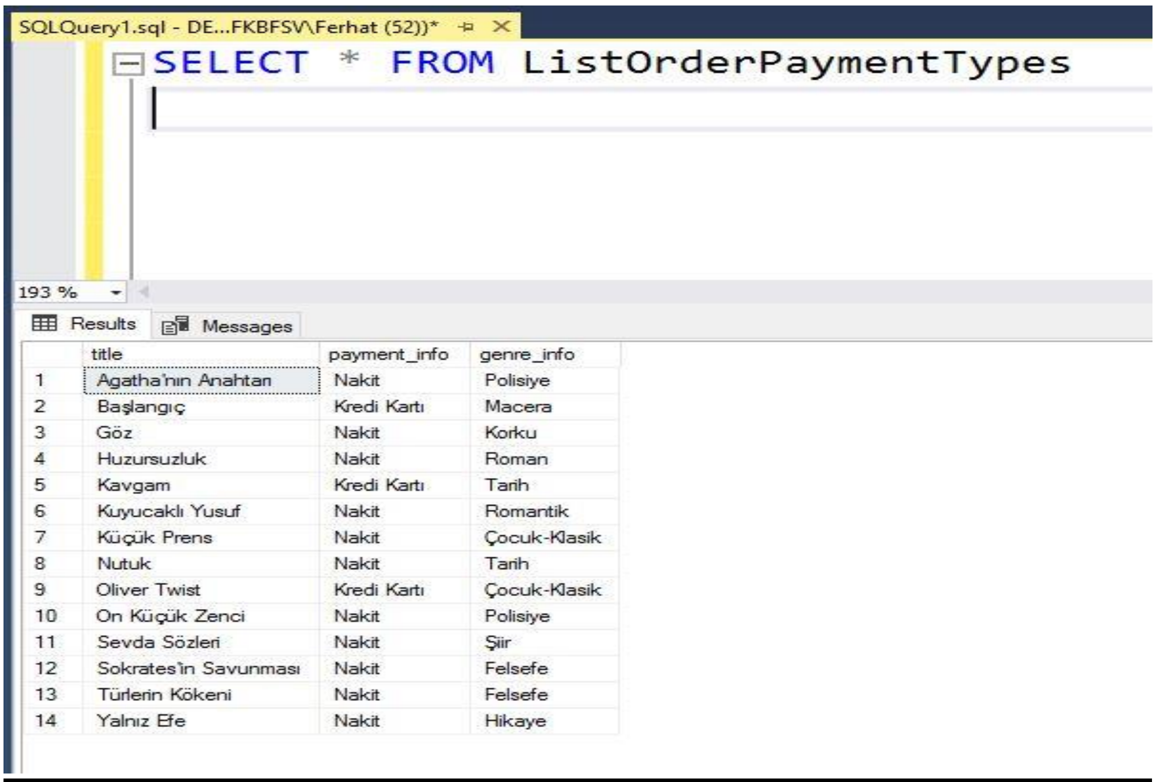
```
SELECT * FROM Top10Customers
```

<u>ListOrderPaymentTypes View</u>



```sql
ALTER VIEW [dbo].[ListOrderPaymentTypes] AS
SELECT DISTINCT b.title , PaymentType.payment_info,GenreType.genre_info
FROM BOOK AS b, [BOOK-GENRE] AS bg ,GENRE as g,

(SELECT DISTINCT pt.payment_info, b.book_id
FROM  PAYMENT_TYPE AS pt , [ORDER] AS o,BOOK AS b
WHERE pt.payment_id=o.payment_id
AND o.book_id=b.book_id ) AS PaymentType,

(SELECT DISTINCT g.genre_info
FROM GENRE AS g ,[BOOK-GENRE] AS bg , BOOK AS b
where g.genre_id=bg.genre_id AND bg.book_id=b.book_id
GROUP BY g.genre_id , g.genre_info) AS GenreType

WHERE b.book_id = PaymentType.book_id
AND bg.book_id = b.book_id
AND bg.genre_id = g.genre_id
AND g.genre_info = GenreType.genre_info
GO
```

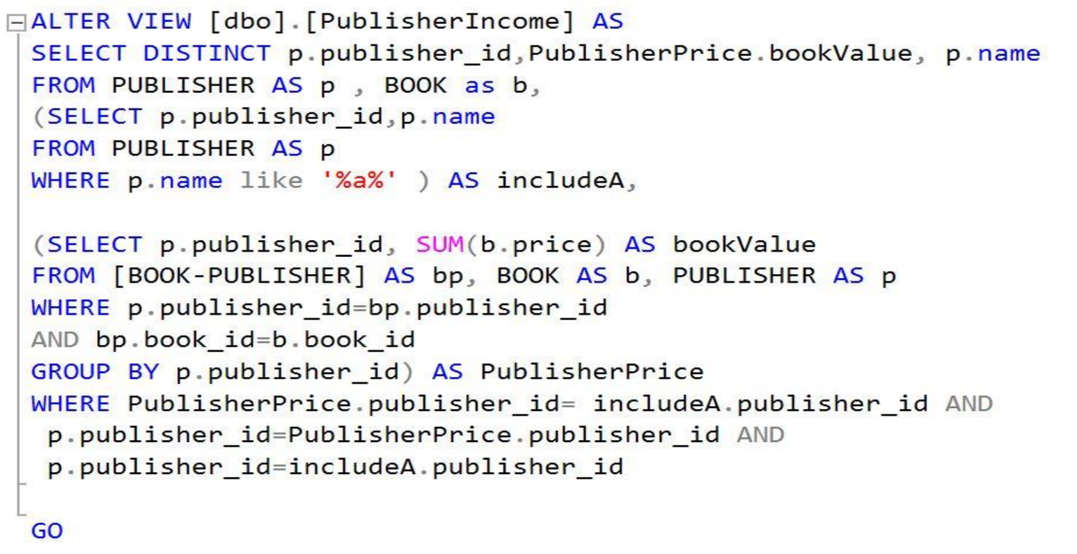
In this view we have returned the genre info of ordered books and the payment type of these orders. We have used two inner select queries one for payment type one for genre type. In the main query we combine these two inner queries.



```sql
SELECT * FROM ListOrderPaymentTypes
```

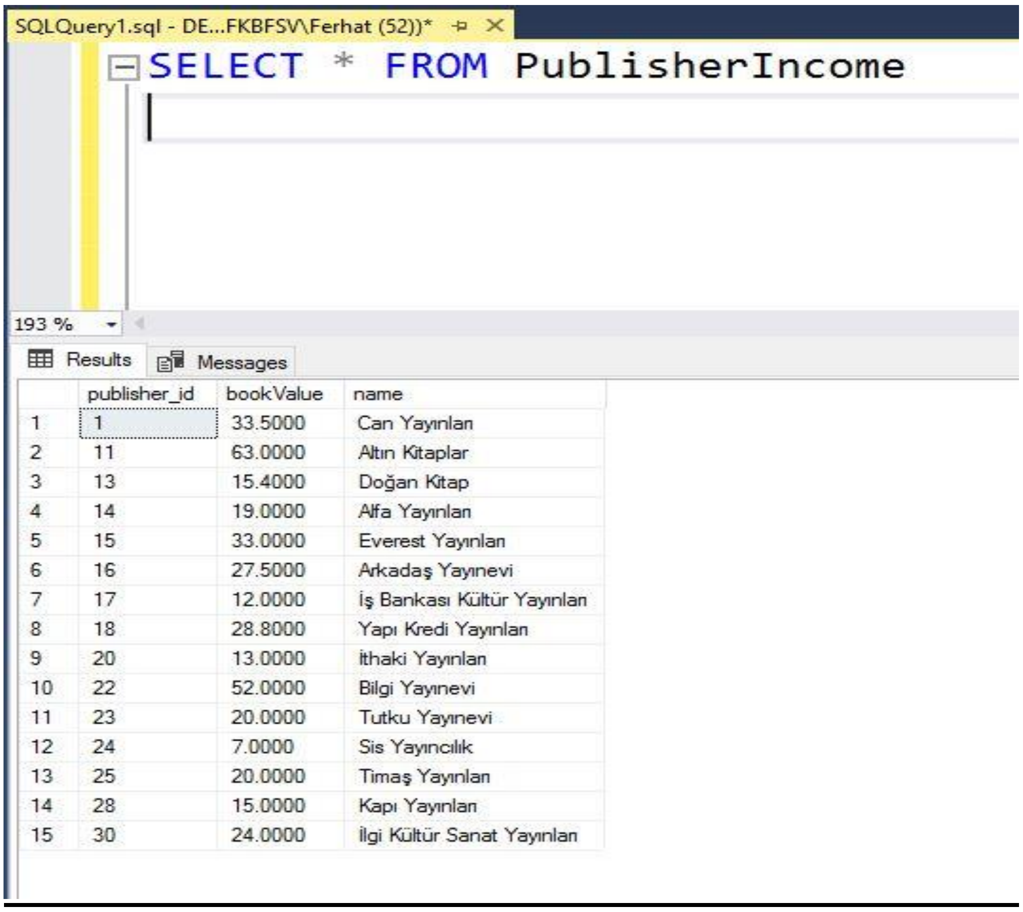
	title	payment_info	genre_info
1	Agatha'nın Anahtarı	Nakit	Polisiye
2	Başlangıç	Kredi Kartı	Macera
3	Göz	Nakit	Korku
4	Huzursuzluk	Nakit	Roman
5	Kavgam	Kredi Kartı	Tarih
6	Kuyucaklı Yusuf	Nakit	Romantik
7	Küçük Prens	Nakit	Çocuk-Klasik
8	Nutuk	Nakit	Tarih
9	Oliver Twist	Kredi Kartı	Çocuk-Klasik
10	On Küçük Zenci	Nakit	Polisiye
11	Sevda Sözleri	Nakit	Şiir
12	Sokrates'in Savunması	Nakit	Felsefe
13	Türlerin Kökeni	Nakit	Felsefe
14	Yalnız Efe	Nakit	Hikaye

<u>PublisherIncome View</u>



```
ALTER VIEW [dbo].[PublisherIncome] AS
SELECT DISTINCT p.publisher_id,PublisherPrice.bookValue, p.name
FROM PUBLISHER AS p , BOOK as b,
(SELECT p.publisher_id,p.name
FROM PUBLISHER AS p
WHERE p.name like '%a%' ) AS includeA,

(SELECT p.publisher_id, SUM(b.price) AS bookValue
FROM [BOOK-PUBLISHER] AS bp, BOOK AS b, PUBLISHER AS p
WHERE p.publisher_id=bp.publisher_id
AND bp.book_id=b.book_id
GROUP BY p.publisher_id) AS PublisherPrice
WHERE PublisherPrice.publisher_id= includeA.publisher_id AND
 p.publisher_id=PublisherPrice.publisher_id AND
 p.publisher_id=includeA.publisher_id

GO
```

In this view we have displayed the sum of all books prices that the corresponding publisher with a name containing the letter “a” publishes as bookValue and extra informations about the publisher such as publisher id and publisher name. In the inner query PublisherPrice we have calculated the total sum of all books prices by SUM function. And the other inner query includeA returns the publishers with a name containing the letter “a”.


SQLQuery1.sql - DE...FKBFSV\Ferhat (52))*

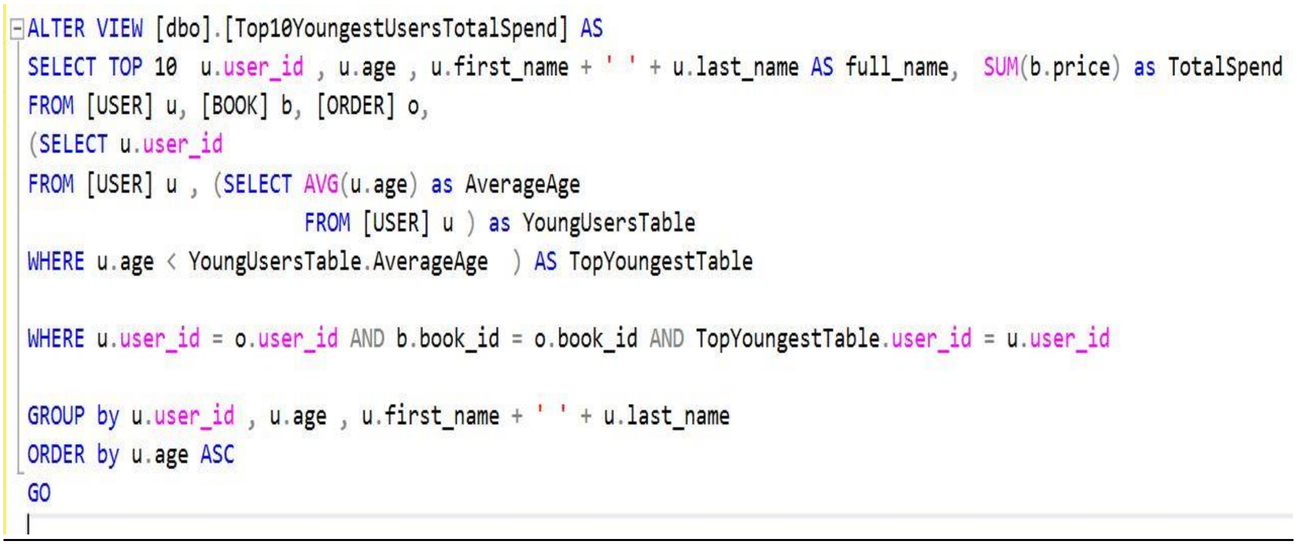
```
SELECT * FROM PublisherIncome
```

193 %

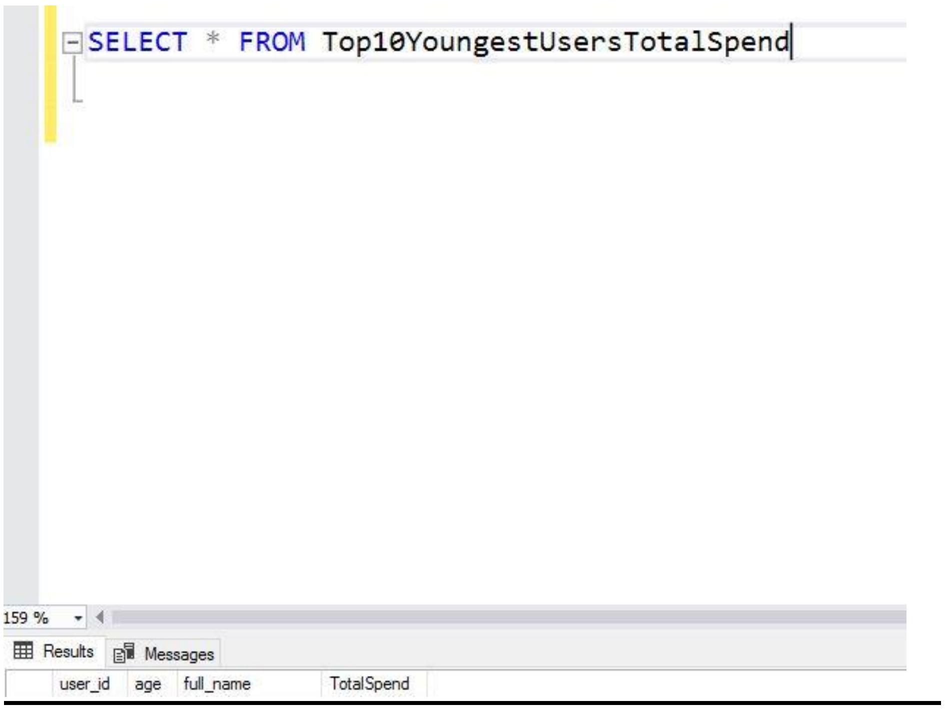
Results | Messages

	publisher_id	bookValue	name
1	1	33.5000	Can Yayınları
2	11	63.0000	Altın Kitaplar
3	13	15.4000	Doğan Kitap
4	14	19.0000	Alfa Yayınları
5	15	33.0000	Everest Yayınları
6	16	27.5000	Arkadaş Yayınevi
7	17	12.0000	İş Bankası Kültür Yayınları
8	18	28.8000	Yapı Kredi Yayınları
9	20	13.0000	İthaki Yayınları
10	22	52.0000	Bilgi Yayınevi
11	23	20.0000	Tutku Yayınevi
12	24	7.0000	Sis Yayıncılık
13	25	20.0000	Timaş Yayınları
14	28	15.0000	Kapı Yayınları
15	30	24.0000	İlgi Kültür Sanat Yayınları

Top10YoungestUsersTotalSpend View


```
ALTER VIEW [dbo].[Top10YoungestUsersTotalSpend] AS
SELECT TOP 10  u.user_id , u.age , u.first_name + ' ' + u.last_name AS full_name,  SUM(b.price) as TotalSpend
FROM [USER] u, [BOOK] b, [ORDER] o,
(SELECT u.user_id
FROM [USER] u , (SELECT AVG(u.age) as AverageAge
                 FROM [USER] u ) as YoungUsersTable
WHERE u.age < YoungUsersTable.AverageAge  ) AS TopYoungestTable

WHERE u.user_id = o.user_id AND b.book_id = o.book_id AND TopYoungestTable.user_id = u.user_id

GROUP by u.user_id , u.age , u.first_name + ' ' + u.last_name
ORDER by u.age ASC
GO
```

In this view we have displayed the users that are younger than the average of all users and their total sum of all books prices that they have ordered. This is not the total spend money of the user it just the sum of all ordered books different unit prices. (we wanted to display something different than total spend money because we have calculated that already in a procedure). Users are ordered in age ascending order. Extra informations about the user is displayed also such as user id and age.



```
SELECT * FROM Top10YoungestUsersTotalSpend
```

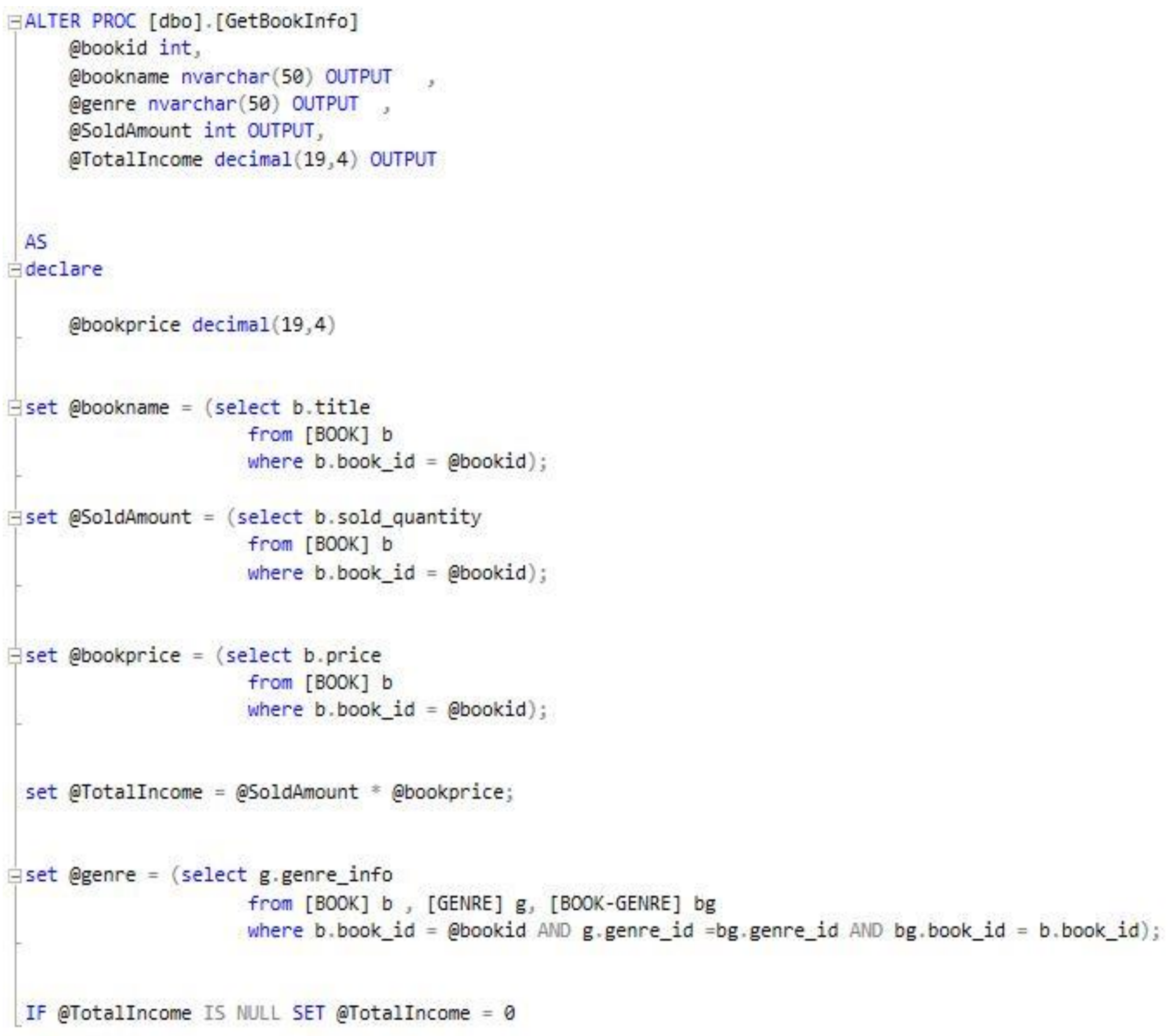
159 %

Results | Messages

	user_id	age	full_name	TotalSpend

Stored Procedures

GetBookInfo Procedure

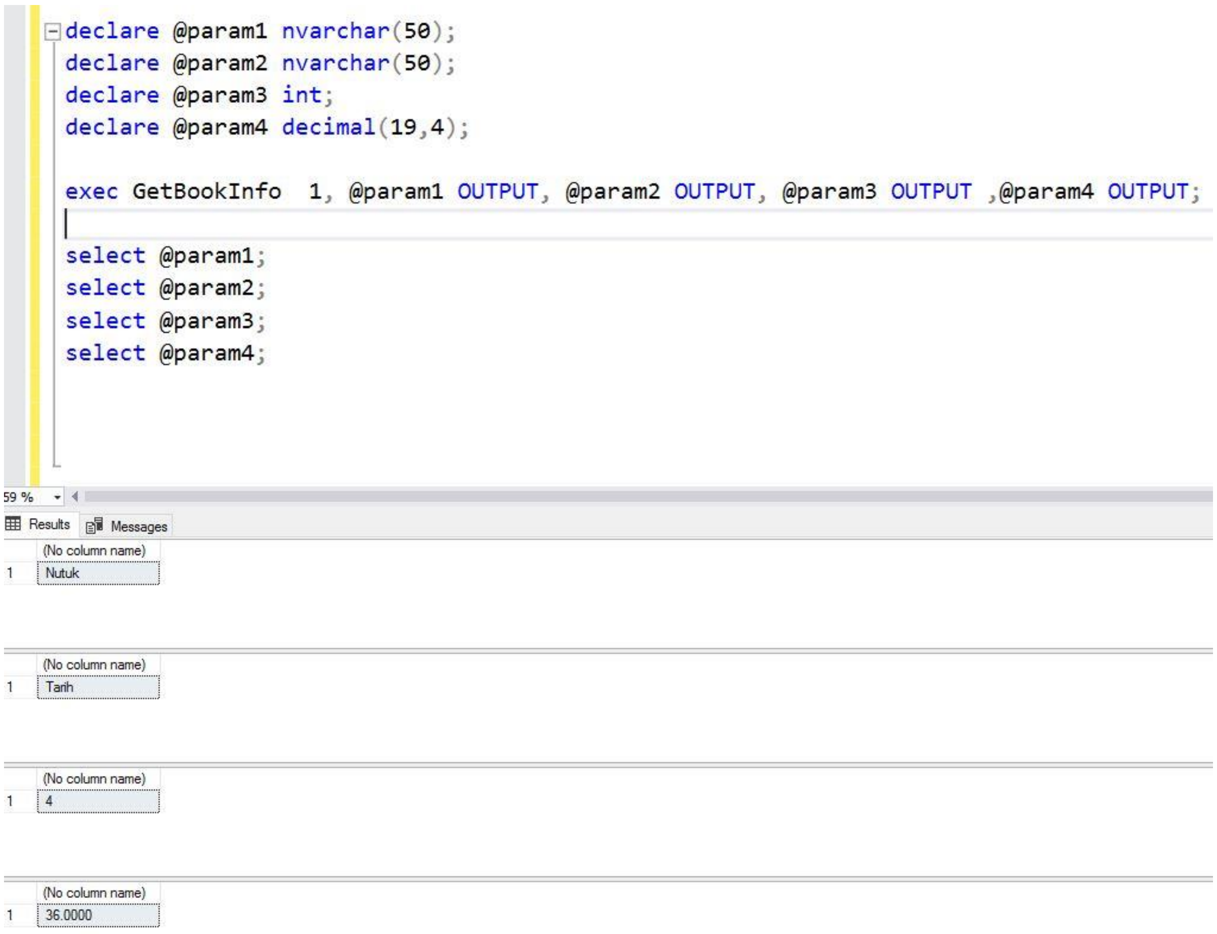


```
ALTER PROC [dbo].[GetBookInfo]
    @bookid int,
    @bookname nvarchar(50) OUTPUT   ,
    @genre nvarchar(50) OUTPUT  ,
    @SoldAmount int OUTPUT,
    @TotalIncome decimal(19,4) OUTPUT


AS
declare

    @bookprice decimal(19,4)


set @bookname = (select b.title
                    from [BOOK] b
                    where b.book_id = @bookid);

set @SoldAmount = (select b.sold_quantity
                    from [BOOK] b
                    where b.book_id = @bookid);


set @bookprice = (select b.price
                    from [BOOK] b
                    where b.book_id = @bookid);


set @TotalIncome = @SoldAmount * @bookprice;


set @genre = (select g.genre_info
                    from [BOOK] b , [GENRE] g, [BOOK-GENRE] bg
                    where b.book_id = @bookid AND g.genre_id =bg.genre_id AND bg.book_id = b.book_id);


IF @TotalIncome IS NULL SET @TotalIncome = 0
```

This procedure simply returns informations about the book that is searched with bookid. The ouputs are bookname, genre, SoldAmount and totalincome.

Total income is calculated with soldQuantity*unitprice.

If there is no order of the book id given from the user the totalincome value will be set to 0.



```
declare @param1 nvarchar(50);
declare @param2 nvarchar(50);
declare @param3 int;
declare @param4 decimal(19,4);

exec GetBookInfo  1, @param1 OUTPUT, @param2 OUTPUT, @param3 OUTPUT ,@param4 OUTPUT;

select @param1;
select @param2;
select @param3;
select @param4;
```

59 %

Results | Messages

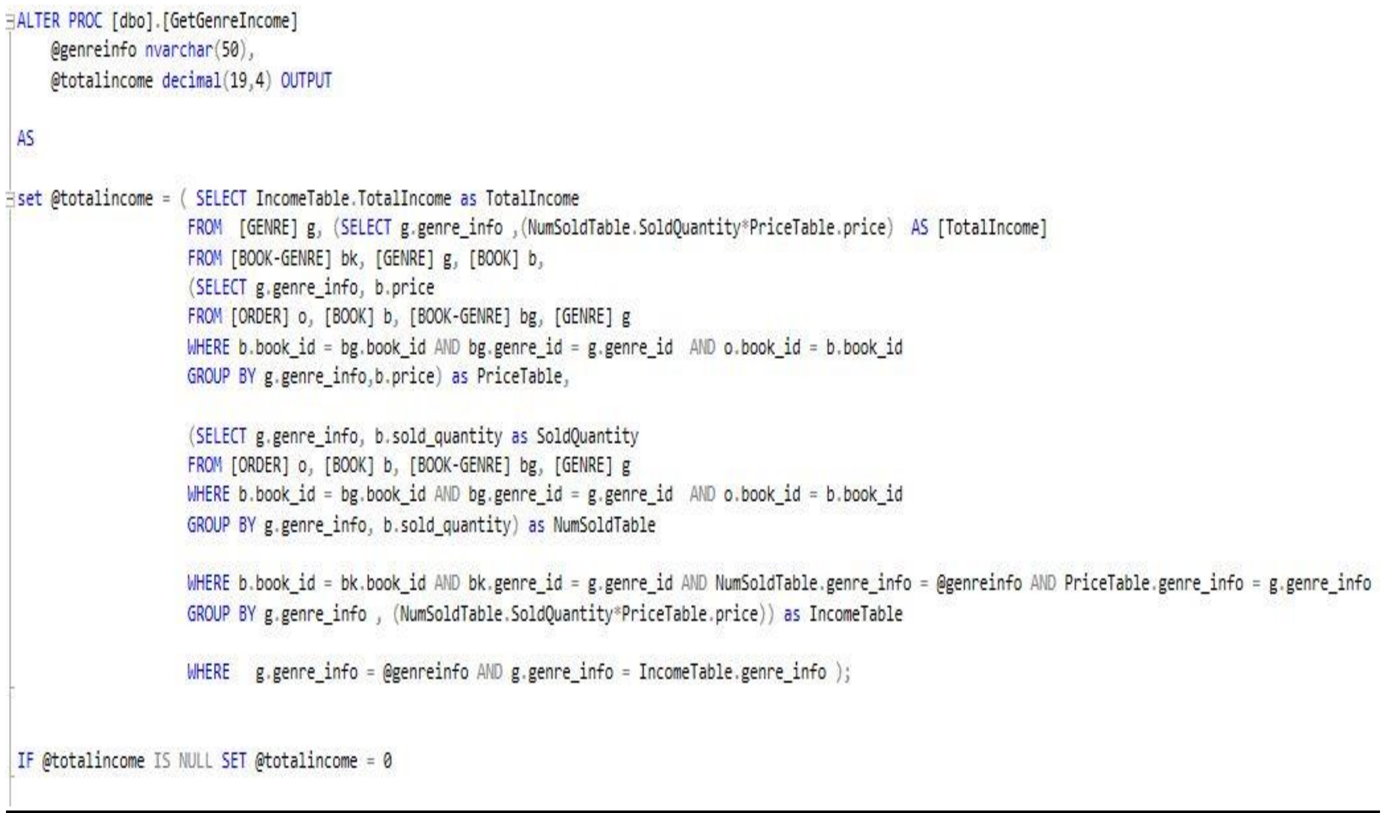
	(No column name)
1	Nutuk

	(No column name)
1	Tarih

	(No column name)
1	4

	(No column name)
1	36.0000

<u>GetGenreIncome Procedure</u>



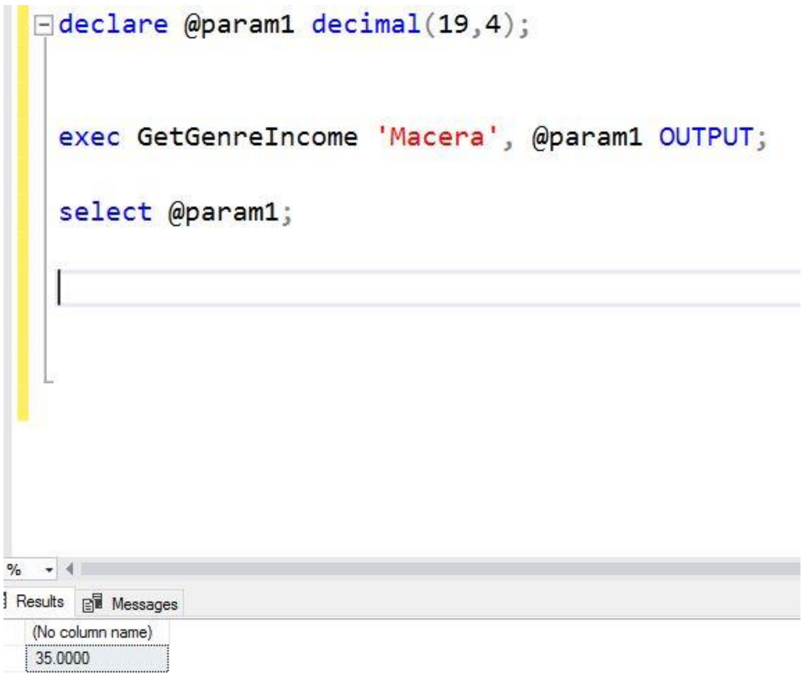
```sql
ALTER PROC [dbo].[GetGenreIncome]
    @genreinfo nvarchar(50),
    @totalincome decimal(19,4) OUTPUT

AS

set @totalincome = ( SELECT IncomeTable.TotalIncome as TotalIncome
                   FROM  [GENRE] g, (SELECT g.genre_info ,(NumSoldTable.SoldQuantity*PriceTable.price)  AS [TotalIncome]
                   FROM [BOOK-GENRE] bk, [GENRE] g, [BOOK] b,
                   (SELECT g.genre_info, b.price
                   FROM [ORDER] o, [BOOK] b, [BOOK-GENRE] bg, [GENRE] g
                   WHERE b.book_id = bg.book_id AND bg.genre_id = g.genre_id  AND o.book_id = b.book_id
                   GROUP BY g.genre_info,b.price) as PriceTable,

                   (SELECT g.genre_info, b.sold_quantity as SoldQuantity
                   FROM [ORDER] o, [BOOK] b, [BOOK-GENRE] bg, [GENRE] g
                   WHERE b.book_id = bg.book_id AND bg.genre_id = g.genre_id  AND o.book_id = b.book_id
                   GROUP BY g.genre_info, b.sold_quantity) as NumSoldTable

                   WHERE b.book_id = bk.book_id AND bk.genre_id = g.genre_id AND NumSoldTable.genre_info = @genreinfo AND PriceTable.genre_info = g.genre_info
                   GROUP BY g.genre_info , (NumSoldTable.SoldQuantity*PriceTable.price)) as IncomeTable

                   WHERE   g.genre_info = @genreinfo AND g.genre_info = IncomeTable.genre_info );


IF @totalincome IS NULL SET @totalincome = 0
```

This procedure takes the genre name as input and gives its total income as output.

The first inner select query PriceTable returns the books genre info and their prices.

The second inner select query NumSoldTable return the soldquantity attiribute of the corresponding book.

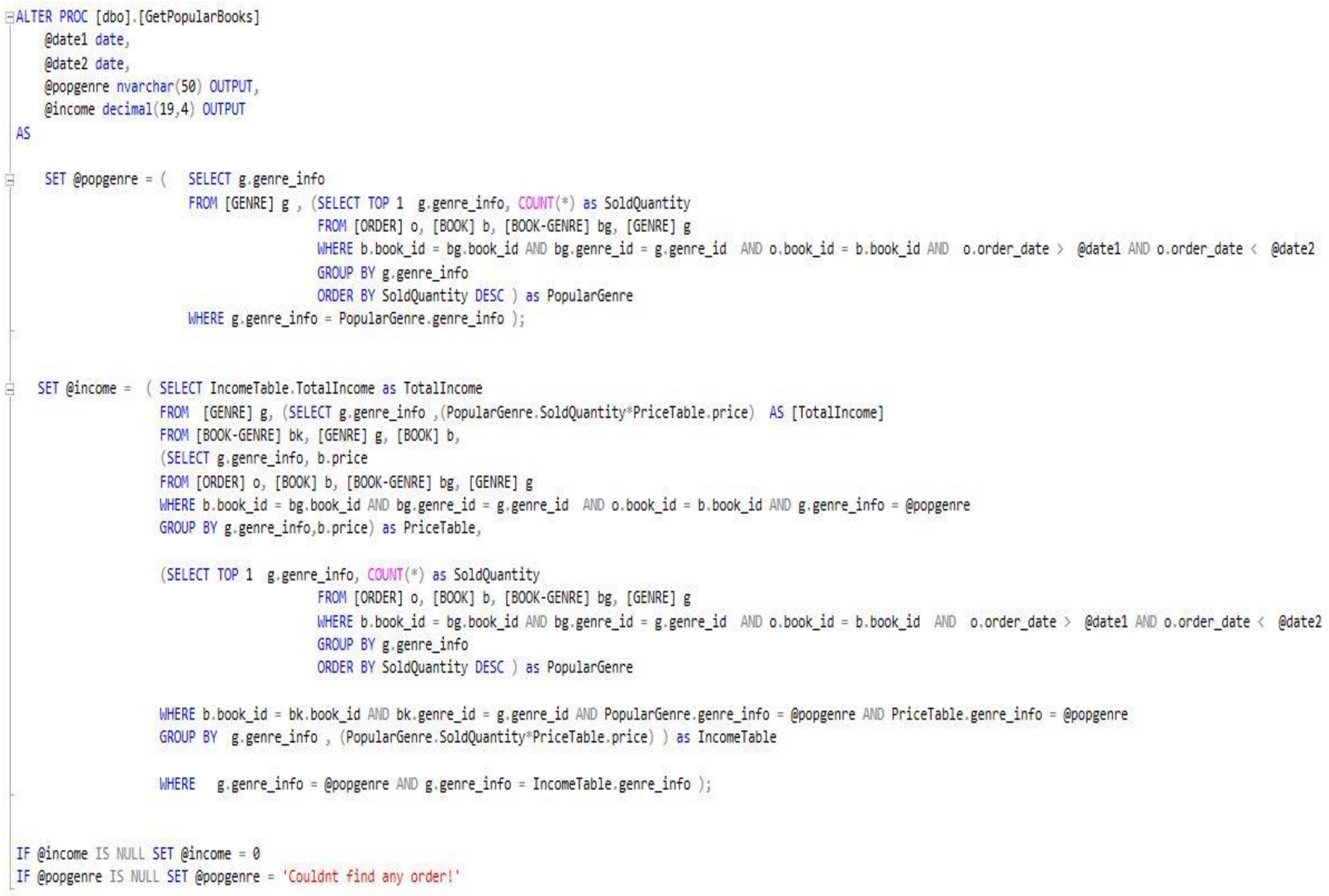
The main select query does the calculation SoldQuantity*price which is the total income. By the group by command we group these genre infos by this we also sum all the total incomes which gives us a final sum of all income from the taken genre info.



```sql
declare @param1 decimal(19,4);


exec GetGenreIncome 'Macera', @param1 OUTPUT;

select @param1;
```

```
Results   Messages
(No column name)
35.0000
```

GetPopularBooks Procedure



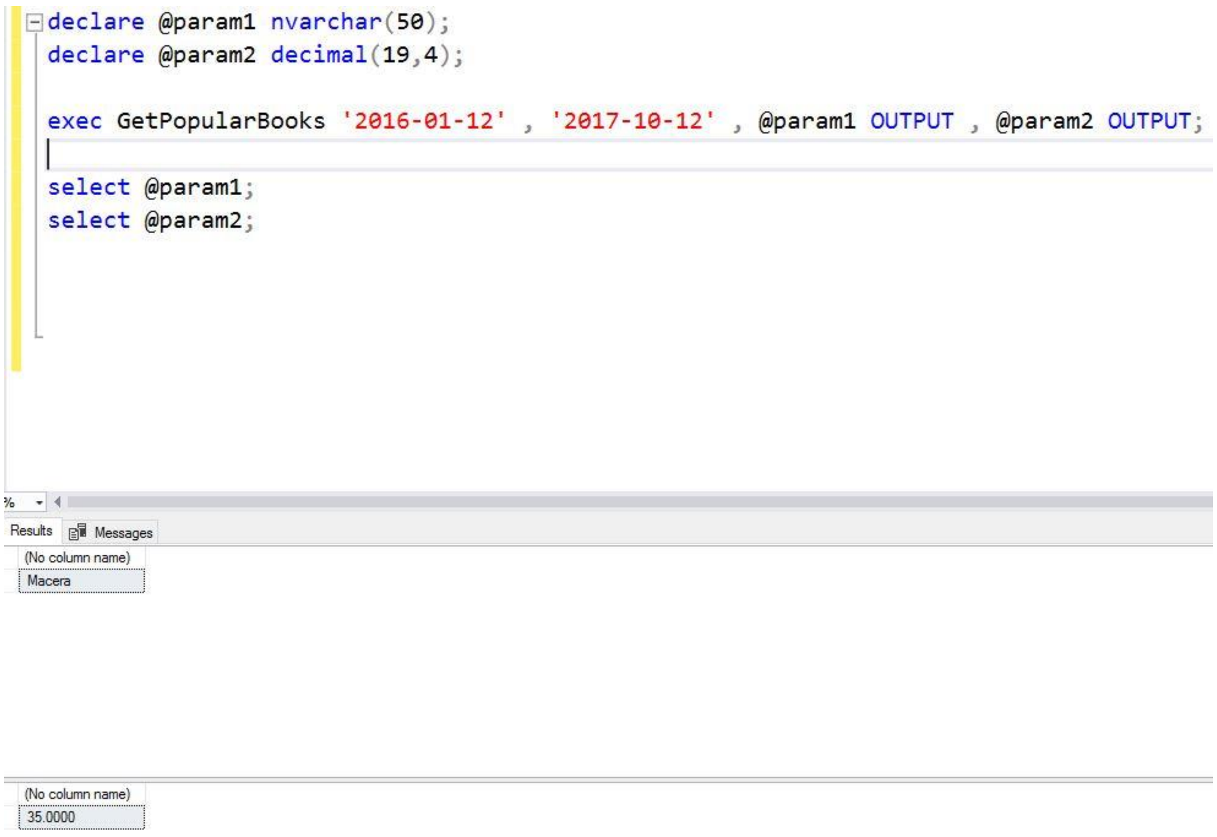
```sql
ALTER PROC [dbo].[GetPopularBooks]
    @date1 date,
    @date2 date,
    @popgenre nvarchar(50) OUTPUT,
    @income decimal(19,4) OUTPUT
AS

    SET @popgenre = (   SELECT g.genre_info
                        FROM [GENRE] g , (SELECT TOP 1  g.genre_info, COUNT(*) as SoldQuantity
                                    FROM [ORDER] o, [BOOK] b, [BOOK-GENRE] bg, [GENRE] g
                                    WHERE b.book_id = bg.book_id AND bg.genre_id = g.genre_id  AND o.book_id = b.book_id AND  o.order_date >  @date1 AND o.order_date <  @date2
                                    GROUP BY g.genre_info
                                    ORDER BY SoldQuantity DESC ) as PopularGenre
                        WHERE g.genre_info = PopularGenre.genre_info );


   SET @income =  ( SELECT IncomeTable.TotalIncome as TotalIncome
                    FROM  [GENRE] g, (SELECT g.genre_info ,(PopularGenre.SoldQuantity*PriceTable.price)  AS [TotalIncome]
                    FROM [BOOK-GENRE] bk, [GENRE] g, [BOOK] b,
                    (SELECT g.genre_info, b.price
                    FROM [ORDER] o, [BOOK] b, [BOOK-GENRE] bg, [GENRE] g
                    WHERE b.book_id = bg.book_id AND bg.genre_id = g.genre_id  AND o.book_id = b.book_id AND g.genre_info = @popgenre
                    GROUP BY g.genre_info,b.price) as PriceTable,

                    (SELECT TOP 1  g.genre_info, COUNT(*) as SoldQuantity
                                    FROM [ORDER] o, [BOOK] b, [BOOK-GENRE] bg, [GENRE] g
                                    WHERE b.book_id = bg.book_id AND bg.genre_id = g.genre_id  AND o.book_id = b.book_id  AND  o.order_date >  @date1 AND o.order_date <  @date2
                                    GROUP BY g.genre_info
                                    ORDER BY SoldQuantity DESC ) as PopularGenre

                    WHERE b.book_id = bk.book_id AND bk.genre_id = g.genre_id AND PopularGenre.genre_info = @popgenre AND PriceTable.genre_info = @popgenre
                    GROUP BY  g.genre_info , (PopularGenre.SoldQuantity*PriceTable.price) ) as IncomeTable

                    WHERE   g.genre_info = @popgenre AND g.genre_info = IncomeTable.genre_info );


IF @income IS NULL SET @income = 0
IF @popgenre IS NULL SET @popgenre = 'Couldnt find any order!'
```

This procedure takes two dates and gives the popgenre which is the genre that sold the most between these dates and the total income from this genre.

The first query is for calculating the popular genre. For this we wrote a select query that lists all orders between date1 and date2. Then we ordered them with soldQuantity descending and toke the first row. By this we have calculated the order between date1 and date2 with the highest soldQuantity value.

The second query is for calculating the total income of that genre. Which first takes the prices of sold books between these dates and then multiplies the soldQuantity attiributes of these books. And finally adds up all sub totals.

If income is NULL which means no books of that genre was sold the income returns 0 also pop genre returns ‘Couldnt find any order!’.

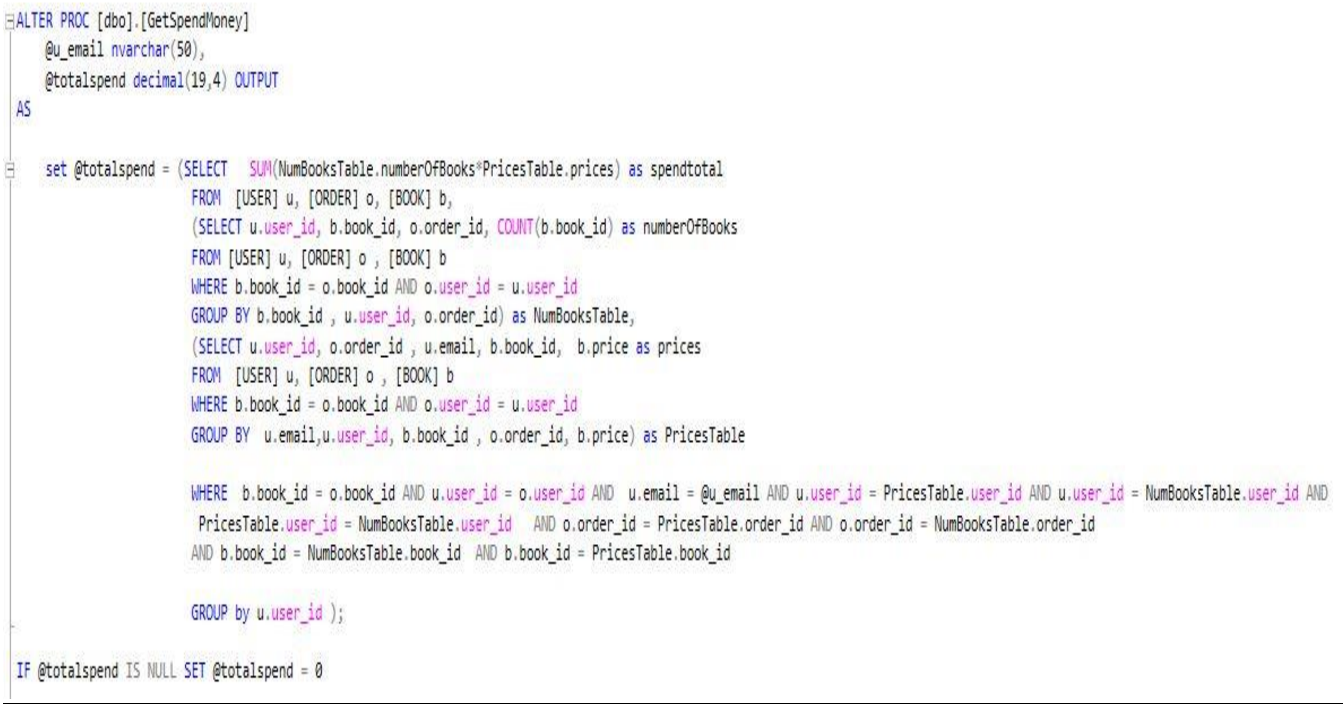


```
declare @param1 nvarchar(50);
declare @param2 decimal(19,4);

exec GetPopularBooks '2016-01-12' , '2017-10-12' , @param1 OUTPUT , @param2 OUTPUT;

select @param1;
select @param2;
```

```
(No column name)
Macera
```

```
(No column name)
35.0000
```

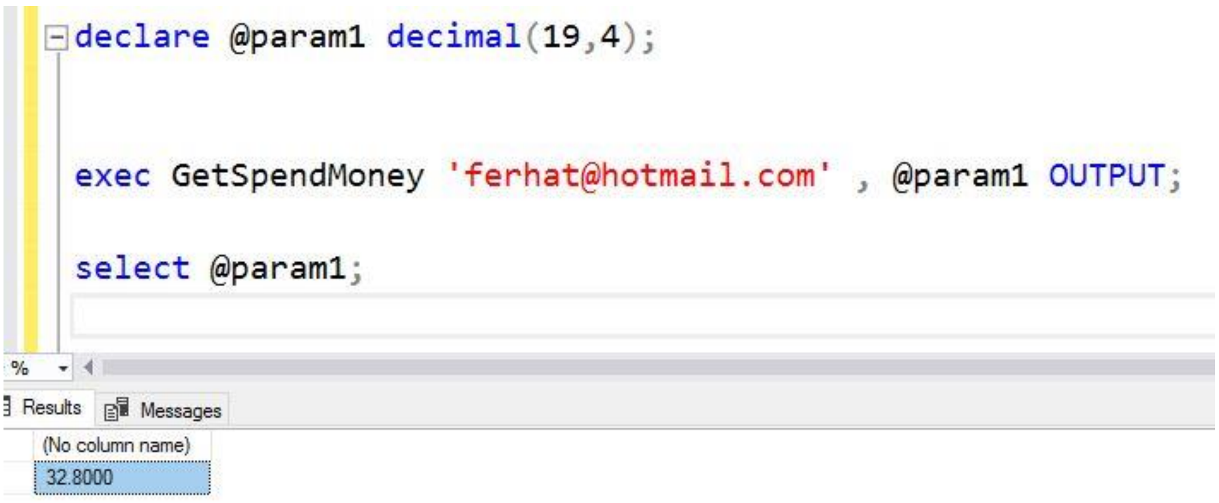
GetSpendMoney Procedure



```sql
ALTER PROC [dbo].[GetSpendMoney]
    @u_email nvarchar(50),
    @totalspend decimal(19,4) OUTPUT
AS

    set @totalspend = (SELECT   SUM(NumBooksTable.numberOfBooks*PricesTable.prices) as spendtotal
                    FROM  [USER] u, [ORDER] o, [BOOK] b,
                    (SELECT u.user_id, b.book_id, o.order_id, COUNT(b.book_id) as numberOfBooks
                    FROM [USER] u, [ORDER] o , [BOOK] b
                    WHERE b.book_id = o.book_id AND o.user_id = u.user_id
                    GROUP BY b.book_id , u.user_id, o.order_id) as NumBooksTable,
                    (SELECT u.user_id, o.order_id , u.email, b.book_id,  b.price as prices
                    FROM  [USER] u, [ORDER] o , [BOOK] b
                    WHERE b.book_id = o.book_id AND o.user_id = u.user_id
                    GROUP BY  u.email,u.user_id, b.book_id , o.order_id, b.price) as PricesTable

                    WHERE  b.book_id = o.book_id AND u.user_id = o.user_id AND  u.email = @u_email AND u.user_id = PricesTable.user_id AND u.user_id = NumBooksTable.user_id AND
                     PricesTable.user_id = NumBooksTable.user_id   AND o.order_id = PricesTable.order_id AND o.order_id = NumBooksTable.order_id
                    AND b.book_id = NumBooksTable.book_id  AND b.book_id = PricesTable.book_id

                    GROUP by u.user_id );

IF @totalspend IS NULL SET @totalspend = 0
```

This procedure takes user email as input and returns totalspend value which is the total amount that user has spend in the system.

Totalspend is calculated with a select query that has two inner select queries. The first query returns the numofbooks that user has ordered. The second query lists the prices of these books. The main query multiplies these two lists and sums up the subtotal and finally shows us the total amout the user has spend in the system.



```sql
declare @param1 decimal(19,4);


exec GetSpendMoney 'ferhat@hotmail.com' , @param1 OUTPUT;

select @param1;
```

```
(No column name)
32.8000
```