SQL

Intermediate SQL and database functions: Grouping and Joining; multi-table operations

Author: W.P.G.Peterson

Assignment Contents

EXPECTED TIME: 2.5 HRS

Overview

This assignment is designed to pick up where the last assignment - "`Introduction To Relational Databases`" - left off. The last assignment focused on accessing data using SQL, and summarizing data to aid in data selection. That process of using "`SELECT`" was reviewed in Lecture 4-5 and 4-6

After this assignment, you should be comfortable with the commands needed to return data from multiple tables. After combining data from multiple tables, the data should be downloadable, and straightforward to analyze in Python, or another program.

Once again, although SQL does offer data munging, cleaning and analysis capabilities, the focus here is how to engage with a SQL database and return an n-by-p collection of data from one or more tables from a database, containing all the information you need.

Activities in this Assignment

- Review of `SELECT` commands
- Use `GROUP BY` statements
- Use `JOIN` statements
- Build complicated queries

`SELECT` Review

Let's return to the `<Customers>` table in the test database provided by w3schools.

Question 1

```
### GRADED
### What is the command that returns all the columns from the Customers table?

### Do not put square brackets around the table name.
### Do not end the statement with a semicolon.
### Use the "*" for selecting columns
### This is covered at the start of lecture 4-5

### Put the complete command in a string, and assign to ans1
### YOUR ANSWER BELOW

ans1 = "SELECT * FROM Customers"
```

Question 2

Suppose we want to see how many customers are from each country. From what we learned last lesson, we can return the count of records that feature a certain country name in the country column with:

`SELECT <func1>(*) FROM Customers <func2> Country = "<Country Name>"`

```
### GRADED
### Both functions were used / covered in Assignment 3

### func2 is covered near the start of lecture 4-5
### func1 is covered mentioned early on in lecture 4-6

### Assign the correct SQL function for "func1" to ans1 as a string
### Assign the correct SQL function for 'func2" to ans2 as a string

### YOUR ANSWER BELOW

ans1 = "COUNT"
ans2 = "WHERE"
```

GROUP BY

The above only gives us the count of records from one country. If we want to see the count for every country, we can use a "`GROUP BY`" statement.

"`GROUP BY`" statments are covered in lecture 4-6.

Try:
`SELECT COUNT(*), Country FROM Customers GROUP BY "Country"`

Question 3

```
### GRADED
### How many countries have more than 10 records in the Customers table?

### Assign int answer to ans1
### YOUR ANSWER BELOW

# `SELECT COUNT(*) AS total, Country FROM Customers GROUP BY Country HAVING COUNT(*) >9 ORDER BY COUNT(*) DESC `

ans1 = 3
```

Question 4

```
### GRADED
### If you wanted the results from:
### `SELECT COUNT(*), Country FROM Customers GROUP BY Country`
### to be *ordered by the count*, what could you add to the end of the query?
### Ordering is covered later in lecture 4-6.

### 'a') ORDER BY COUNT(*) Decending
### 'b') ORDER COUNT(*)
### 'c') ORDER BY COUNT(Country)
### 'd') SORT COUNT(*)
### 'e') ORDER BY "Country"
### Assign character associated with choice as string to ans1

### YOUR ANSWER BELOW

ans1 = 'c'
```

Question 5

```
### GRADED
### What price appears three and only three times in the "Products" table?
### All the information needed to answer this question is in lectures 4-5 and 4-6.
### Assign appropriate price - int or float - to ans1

### YOUR ANSWER BELOW

# `SELECT COUNT(*), Price FROM Products GROUP BY Price HAVING COUNT(*) = 3`

ans1 = 10
```

Question 6

```
### GRADED
### In the "OrderDetails" table, three products have been ordered more times than the other products
### This is a question about frequency of orders, NOT quantity of items in orders.

### What are the prices of those three products?

### Put the three prices into a list as numbers, (ints or floats), in any order.
### e.g. [price1, price2, price3]
### Assign list to ans1

### Hint: Answering this question can be accomplished one of two ways:
### A. With multiple queries -
### ### 1. What products were ordered the most; 2. Find the prices of products with those ProductIDs.
### B. Use JOINing logic introduced in lectures 4-7 and 4-8.

### Either way, Grouping and Counting ProductID's and cross-referencing with price is necessary

### YOUR ANSWER BELOW

# `SELECT COUNT(OD.ProductID), OD.ProductID, P.Price
# FROM OrderDetails AS OD
# JOIN Products AS P ON OD.ProductID = P.ProductID
# GROUP BY OD.ProductID
# ORDER BY COUNT(OD.ProductID) DESC

ans1 = [12.5, 55, 34.8]
```

JOIN Functions

Given what was covered thus far in this assignment, answering the above question probably required making two queries, one to find the most frequently ordered productIDs, from the "OrderDetails" table, then one requesting the products with those IDs from the "Products" table.

Using a JOIN function, we can request data from multiple table with a single query.

A Note on Query Styles:

The lectures (4-7/8) introduce "`JOIN`"s and aliases implicitly. For example, from early in lecture 4-8:

"`SELECT f_name, l_name, class FROM Student a, Enrolls_in b WHERE a.ssn = b.ssn`"

Later in lecture 4-8, a more explicit style is used. e.g:

"`SELECT a.f_name, a.l_name, b.class FROM Student AS a INNER JOIN Enrolls_in AS b WHERE a.ssn = b.ssn`"

Note how in the column selection, the columns are explicitly associated with their table, and the aliases are written "`<Table> AS <Alias>`" instead of "`<Table> <Alias>`".

The explicit style will be used through the rest of this assignment as a way to more thoroughly demonstrate the function of each query.

Question 7

Run:

```
SELECT Orders.OrderID, Employees.FirstName FROM Orders LEFT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
```

```
### GRADED
### After Running the above Query, answer the following:

### What is the FirstName of the employee responsible for the Order with ID 10255?

### Assign name as string to ans1
### YOUR ANSWER BELOW

# `SELECT Orders.OrderID, Employees.FirstName FROM Orders
# LEFT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
# WHERE Orders.OrderID = 10255

ans1 = "Anne"
```

Aliases

As should be evident from the simple query above, JOIN queries can become typing heavy.
To cut down on the typing, aliases can be used. Try the below query:

```
SELECT o.OrderID as OrderNumber, e.FirstName as First FROM Orders AS o, Employees AS e WHERE e.EmployeeID = o.EmployeeID
```

You should see functionally the same output as:
```
SELECT Orders.OrderID, Employees.FirstName FROM Orders LEFT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
```

BUT, with the column headers renamed.

In summary, when referring to tables, we are able to use the abbreviations defined in the `FROM` part of our query, defined in the form `<tableName> AS <TableAlias>`.

A More Complicated Query

Before asking for the results of some complicated queries, we will build a query statement piece-by-piece.

Step 1: Determine the total price of each order.
- Join Order Detail with Products table
- Multiply Quantity (from OrderDetail) with price (from Products)
- Sum the above for each order.

Step 2: See which Employee is reponsible for the most money spent by customers.
- Join OrderDetails with Orders; Order with Employees for OrderID-> EmployeeID -> FirstName
- Group by table created in step 1 by Employee First Name

Step 1.

Joining Order Detail with Products Table (and start to use aliases).

```
SELECT od.OrderID, od.Quantity, p.Price FROM OrderDetails AS od JOIN Products AS p ON od.ProductID = p.ProductID
```

We did not choose to show "ProductID" because although that is what we are joining on, after that join, it has served its purpose.

Result:

Number of Records: 518

OrderID	Quantity	Price
10248	12	21
10248	10	14
10248	5	34.8

Next we want to multiply our quantity by our price: So in our `SELECT` statement we will add: `od.Quantity * p.Price AS TotalCost`, for a query of:

```
SELECT od.OrderID, od.Quantity, p.Price, od.Quantity * p.Price AS TotalCost FROM OrderDetails AS od JOIN Products AS p ON od.ProductID = p.ProductID
```

Number of Records: 518

OrderID	Quantity	Price	TotalCost
10248	12	21	252
10248	10	14	140
10248	5	34.8	174

Finally, to find the total per order, we need to sum the total cost while grouping by OrderID.

Adding a `GROUP BY`:

```
SELECT od.OrderID, od.Quantity, p.Price, od.Quantity * p.Price AS TotalCost FROM OrderDetails AS od JOIN Products AS p ON od.ProductID = p.ProductID GROUP BY od.OrderID
```

Yeilds:

Number of Records: 196

OrderID	Quantity	Price	TotalCost
10248	5	34.8	174
10249	40	53	2120
10250	15	21.05	315.75

So above, although some aggregation has occurred, what is displayed is the last record for "Quantity", "Price", and "TotalCost." Thus, some aggregation function in the `SELECT` statement is needed. I will wrap the "Quantity * Price in a "SUM".

"Price" is no longer relevant, because it cannot be summed our counted in a meaningful way. Thus, its removal from the `SELECT` statement.

Finally, "Quantity" will also be summed so that average price per item might be calculated later.
OrderID will be kept, as it is needed to relate this information to the employee table:

```
SELECT od.OrderID, SUM(od.Quantity) AS TotalItems, SUM(od.Quantity * p.Price) AS TotalCost FROM OrderDetails AS od JOIN Products AS p ON od.ProductID = p.ProductID GROUP BY od.OrderID
```

Number of Records: 196

OrderID	TotalItems	TotalCost
10248	27	566
10249	49	2329.25
10250	60	2267.25

Step 2:

With all order-price information aggregated, employee aggregation can start.

First add a join to the "Orders" Table, which provides EmployeeID:

```
SELECT o.EmployeeID, od.OrderID, SUM(od.Quantity) AS TotalItems, SUM(od.Quantity * p.Price) AS TotalCost FROM OrderDetails AS od JOIN Products AS p ON od.ProductID = p.ProductID JOIN Orders AS o ON od.OrderID = o.OrderID GROUP BY od.OrderID
```

Number of Records: 196

EmployeeID	OrderID	TotalItems	TotalCost
5	10248	27	566
6	10249	49	2329.25
4	10250	60	2267.25
3	10251	41	839.5

Then add a join with Employees table. Note removal of "o.EmployeeID" because that column is only needed for the Join:

```
SELECT e.FirstName, od.OrderID, SUM(od.Quantity) AS TotalItems, SUM(od.Quantity * p.Price) AS TotalCost FROM OrderDetails AS od JOIN Products AS p ON od.ProductID = p.ProductID JOIN Orders AS o ON od.OrderID = o.OrderID JOIN Employees As e ON o.EmployeeID = e.EmployeeID GROUP BY od.OrderID
```

Number of Records: 196

FirstName	OrderID	TotalItems	TotalCost
Steven	10248	27	566
Michael	10249	49	2329.25
Margaret	10250	60	2267.25

Finally modify the `GROUP BY` to find the total sales of each employee.

Note removal of `GROUP BY od.OrderID`. This is for two reasons: First, SQL will not tolerate two `GROUP BY` statements. Second: Because we are now interested in totalitems / revenue for an employee, the particular orderID is irrelevant, other than for joining `OrderDetails` with `Orders`.

```
SELECT e.FirstName, SUM(od.Quantity) AS TotalItems, SUM(od.Quantity * p.Price) AS TotalCost FROM OrderDetails AS od JOIN Products AS p ON od.ProductID = p.ProductID JOIN Orders AS o ON od.OrderID = o.OrderID JOIN Employees As e ON o.EmployeeID = e.EmployeeID GROUP BY e.FirstName
```

Number of Records: 9

FirstName	TotalItems	TotalCost
Andrew	1315	32503.16
Anne	649	15734.0999999
Janet	1725	42838.3500000

Question 8

```
### GRADED
### How many orders is the employee with first name "Laura" responsible for?

### HINT: Try an INNER JOIN  Between Orders and Employees
### HINT: Remeber quotation marks for strings in "WHERE" statements.
### HINT: Individual orders in the "Orders" Table are defined by OrderID

### YOUR ANSWER BELOW

# SELECT COUNT(Orders.OrderID), Employees.FirstName FROM Employees
# LEFT JOIN Orders ON Orders.EmployeeID = Employees.EmployeeID
# WHERE Employees.FirstName = "Laura

ans1 = 27
```

Question 9

```
### GRADED
### How many total items is "Speedy Express" responsible for shipping?
```

```
### The SUM function is never explicitly demonstrated in lecture, but will be useful here.
### For example, try:
### `SELECT SUM(Quantity) FROM OrderDetails`

### HINT: "Shippers" can be joined to "OrderDetails" via "Orders" With the "ShipperID" and "OrderID" columns
### HINT: All Joins must be accomplished BEFORE any "GROUP BY"

### Assign int answer to ans1

### YOUR ANSWER BELOW

# SELECT S.ShipperName, SUM(OD.Quantity) FROM OrderDetails AS OD
# JOIN Orders AS O ON O.OrderID = OD.OrderID
# JOIN Shippers AS S ON O.ShipperID = S.ShipperID
# GROUP BY S.ShipperName HAVING S.ShipperName = "Speedy Express"

ans1 = 3575
```

Question 10

```
### GRADED
### What PostalCode (from Customers table) recieved the most expensive order?
### ### Assign PostalCode as string to ans1.

### NB: Order Total should be calculated as follows:
### For all Products in an OrderID, Sum the
### OrderDetails.Quantity * Products.Price.

### HINT: If no aliases are used, the start of the select might read:
### `SELECT Customers.PostalCode, SUM(OrderDetails.Quantity * Products.Price) AS OrderTotal`
### The "SUM" Will require a "GROUP BY" to be useful

### And the the end of the query might read:
### `ORDER BY OrderTotal`

### Customers can be connected to Products via Orders and OrderDetails with
### "CustomerID," "OrderID," and "ProductID."
### Three JOIN statments will be required.


### YOUR ANSWER BELOW

# SELECT C.PostalCode, SUM(OD.Quantity*P.Price) AS OrderTotal
# FROM Customers AS C
# JOIN Orders AS O ON C.CustomerID = O.CustomerID
# JOIN OrderDetails AS OD ON OD.OrderID = O.OrderID
# JOIN Products AS P ON OD.ProductID = P.ProductID
# GROUP BY OD.OrderID
# ORDER BY OrderTotal

ans1 = '05487-020'
```

Question 11

```
### GRADED
### In which city is the supplier located that supplies the most expensive product?
### ### Assign answer as string to ans1

### HINT: Suppliers and Products can be connected via "SupplierID"

### YOUR ANSWER BELOW

# SELECT S.City, P.Price FROM Suppliers as S
# JOIN Products AS P ON S.SupplierID = P.SupplierID
# ORDER BY P.Price

ans1= 'Paris'
```

Question 12

```
### GRADED
### Which country supplied the most items found in the OrdersDetails table?

### e.g. OrderDetails describes # of items in an order, and productID
### Each product ID is associated with a supplier.
### Each supplier is in a country.
### Total the number of items supplied by each country

### Hint: OrderDetails may be connected to Suppliers via Products.
### Hint: This will require two Joins, and one Group By

### Which country supplies the most items? Assign as string to ans1
### How many items did that country supply? Assign as int to ans2

### YOUR ANSWER BELOW

# SELECT S.Country, SUM(OD.Quantity) AS TotProds FROM OrderDetails AS OD
# JOIN Products AS P ON P.ProductID = OD.ProductID
# JOIN Suppliers AS S ON S.SupplierID = P.SupplierID
# GROUP BY S.Country
# ORDER BY TotProds DESC
# LIMIT 1

ans1 = 'Australia'
ans2 = 1610
```

JOINS pt.II

The joins specified above have been trivial: Whichever columns were joined had all values in each of the two joined table.
Test the following two commands:

```
SELECT * FROM Employees as E LEFT JOIN Orders as o ON e.EmployeeID = o.EmployeeID Group BY e.EmployeeID
```

And:

```
SELECT * FROM Employees as E INNER JOIN Orders as o ON e.EmployeeID = o.EmployeeID Group BY e.EmployeeID
```

Question 13

```
### GRADED
### What is the last name of the employee present in the "left" join but missing from the "Inner" Join above?
### Assign string to ans1

### YOUR ANSWER BELOW

ans1 = "West"
```

Try:

```
SELECT e.LastName, o.OrderID FROM Employees as e LEFT JOIN Orders as o ON e.EmployeeID = o.EmployeeID WHERE o.EmployeeID IS NULL
```

Note that the "`IS NULL`" is how to test whether or not a value is null. e.g., `o.EmployeeID = NULL` will not work.

The default join in SQL is an "INNER" Join, which simply means that only records present in both tables are included.

A left join includes every record from the first table being Joined (Employees above), but only records from the second table (Orders above) where the key being joined upon is present in the left table.

In that left join, as seen above, the data absent from the "right" table is simply marked as NULL.

While theoretically there is a fair amount that can be learned about joins, "full" and "left" joins are the most frequently used. The w3schools site has more on joins if you are interested.

A Couple Final Notes:

A Note on the Data Science Process:

This joining process is possible in SQL, however, it is also possible in Python. The lecture and this assignment introduce joins and the logic of joins, however, there is neither an explicit nor implicit recommendation that you should accomplish joins in SQL, or accomplish them in Python. Ultimately, whichever process is easier and/or quicker for you is recommended.

SQL Fluency

Like most skills, SQL fluency will atrophy without consistent use. SQL expertise is a prerequisite for many roles and activities. After these assignments, you will hardly be expert, but you should have enough knowledge to decipher many SQL queries, and have a solid foundation upon which you can further develop your SQL skill.