BUSINESS ANALYTICS CLUB

Workshop Series 9.19

SQL and Supply Chain Management

Nicole Lee
Eddie Shim

BAC

Learning Objective

1. Identify potential operations issues of a large company
2. Understand what relational databases are
   - Distinguish between tables, records, fields, and field values
3. Be able to write queries in SQL to answer questions
4. Understand basic database concepts like normalization and entity relationships
5. Apply data to decisions in supply chain management

BAC

General Mills Supply Chain

You're a supply chain analyst at General Mills, one of the top manufacturers of consumer foods. General Mills sells its products to retail stores. You are given a dataset with consumers, products, shippers, and suppliers.

Although General Mills is a US Company, your boss wants you to find ways of lowering shipping and production costs within and outside the US.

You must figure out which suppliers and shippers the General Mills should keep.

BAC

Database Terms

1. What are databases?
   - Collections of info organized by logical structure of that info
2. Database management software
   - Software to create, store, organize, and retrieve data from a database
     - Proprietary: Oracle, Access, SQL Server
     - Open Source: PostgreSQL, MySQL
   - Database administrator (DBA)
     - Person responsible for dev. & ops of a database

BAC

Some Practical Industrial Advice

- Every company stores its data in some kind of relational database (for very good reason)
- As an analyst, you will want to pull data from that database constantly
  - Building and designing a database; inserting new values – not so much
- Knowing how to write a SQL query will put you at a huge advantage relative to your peers
  - Don't be beholden to the DBA (who is mainly focused on keeping the database up)

BAC

General Mills Dataset:

bit.ly/gmillsdata

BAC

Each of these is a table

Your Database:

Tablename	Records
Customers	91
Categories	8
Employees	10
OrderDetails	518
Orders	196
Products	77
Shippers	3
Suppliers	29

Restore Database

Each line shows # of records or "rows" in the table

BAC

What is in the `Customers` Table?

SQL Statement:

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

SQL keyword indicating a query in which we will be "selecting" data from a table.

Here we list which "fields" we want from the table – a '*' indicates that we want all of them. We can also list columns we want by name.

```
SELECT *
FROM
Customers;
```

A SQL keyword that tells us from which table we will be selecting our observations from.

The database table we are selecting observations from.

BAC

These are the "attributes" or "fields" of the table – also sometimes called "columns"

These are the "records" or "rows" from the Products table

Number of Records: 77

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	30
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	40
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	97
10	Ikura	4	8	12 - 200 ml jars	31
11	Queso Cabrales	5	4	1 kg pkg.	21
12	Queso Manchego La Pastora	5	4	10 - 500 g pkgs.	38

You try it:

Get all the rows from the `Products` table

BAC

```
SELECT *
FROM
Products;
```

BAC

What if we only want the price?

BAC

```
SELECT Price
FROM
Products;
```

BAC

What if we want Price and ProductID?

BAC

Note the inclusion of a comma separating the fields we are selecting

```
SELECT Price, ProductID
FROM
Products;
```

BAC

Let's find the cheap products
(say price < $10)

```
SELECT *
FROM
Products
WHERE Price < 10;
```

This is, appropriately enough,
called a "where clause"

BAC

You try it:
Get all rows in Products table where Price is greater than or equal to $30

BAC

```
SELECT *
FROM
Products
WHERE Price >= 30;
```

BAC

How about products *less than* $10 but *greater than* $5?

BAC

```
SELECT *
FROM
Products
WHERE Price <10
AND Price > 5;
```

This AND lets us string together more conditions

BAC

Any ideas? Read as "less than" OR "greater than" which just means "not equal to"

```
SELECT
FROM
Products
WHERE …
```

- `Price > 10`
- `Price < 10`
- `Price < 10 AND Price > 5`
- `Price < 10 OR Price > 25`
- `Price <> 100`
- `Price > 2 OR Price = 1`
- `(Price > 2 AND Price < 10) OR Price = 1`

BAC

How *many* products have a price less than $10?

BAC

```
SELECT
COUNT(*)
FROM
Products
WHERE Price < 10;
```

This COUNT is a function
counts up the # of rows
that satisfy the criteria

BAC

SQL Statement:

```
select count(*) from Products where Price < 10;
```

Run SQL »

Result:

Number of Records: 1

count(*)
11

The answer is 11

We can rename using AS – we made up the name "NumProducts"

```
SELECT
COUNT(*) AS NumProducts
FROM
Products
WHERE Price < 10;
```

Number of Records: 1

NumProducts
11

BAC

What's the total dollar value of merchandise?

```
SELECT
SUM(Price)
FROM
Products;
```

SQL Statement:

```
SELECT SUM(PRICE) FROM Products;
```

Run SQL »

Result:

Number of Records: 1

SUM(PRICE)
2222.71

Here's something interesting – the query results *look like* tables themselves

SQL Statement:

```
select count(*) from Products where Price < 10;
```

Run SQL »

Result:

Number of Records: 1

count(*)
11

SQL Statement:

```
select * from Products where Price < 10;
```

Run SQL »

Result:

Number of Records: 11

ProductID	ProductName
13	Konbu
19	Teatime Chocolate Biscuits
23	Tunnbröd
24	Guaraná Fantástica
33	Geitost
41	Jack's New England Clam Chowder

BAC

```
CREATE TABLE cheapstuff AS
SELECT *
FROM
Products
WHERE Price < 10;
```

We can create a new table in our database on the fly, based on our query!

BAC

Your Database:

Tablename	Records
Customers	91
Categories	8
Employees	10
OrderDetails	518
Orders	196
Products	77
Shippers	3
Suppliers	29
cheapstuff	11

Restore Database

You try it:
Make a table of all expensive products (say Price > $50)

BAC

```
CREATE TABLE DearStuff
AS
SELECT *
FROM
Products
WHERE Price > 50;
```

BAC

Can we list products in price order?

BAC

```
SELECT *
FROM Products
ORDER BY Price DESC;
```

DESCending order – can also do ASCending order

This ORDER BY will give the results in some specific ordering

The field to order the results by

DESCending means highest value first

BAC

Can we create a new field based on price that lists whether a product is cheap or not?

BAC

This ensures that our resulting table will list the price

```
SELECT
Price,
CASE WHEN Price < 10
    THEN 't'
    ELSE 'f' END AS Cheap
FROM Products
```

This is conditional. In words, "if the price is less than $10, set the variable "Cheap" to 't'; if it's not less than $10, set "Cheap" to 'f'.

SQL Statement: Edit th

```
SELECT Price,
CASE WHEN Price < 10 then 't' ELSE 'f' END as Cheap
FROM Products;
```

Run SQL »

Result:

Number of Records: 77

Price	Cheap
18	f
19	f
10	f
22	f

You try it:

Make a table with a field showing whether a price is between $5 and $10, restricted to products with a price less than $15.

BAC

```
SELECT Price,
CASE WHEN Price > 5
AND Price < 10
   THEN 't'
   ELSE 'f' END AS midrange
FROM Products
WHERE Price < 15;
```

What's the breakdown between cheap and expensive products?

BAC

First, let's create a table with the cheap classification

```
CREATE TABLE ProductPriceClass AS
SELECT Price,
CASE WHEN Price < 10
    THEN 't'
    ELSE 'f' END AS Cheap
FROM Products;
```

We now have a table that looks like this:

Column "Cheap" has two kinds of values; 't' and 'f'

Price	Cheap
9.2	t
9	t
9.65	t
9.5	t
9.5	t
10	f
10	f
14	f
12.5	f
14	f
14	f
12	f
12.75	f
13.25	f
14	f
12.5	f
10	f
13	f

If we "group" rows together by the value in "Cheap" and then count, we get:

5 rows where Cheap = 't'
13 rows where Cheap = 'f'

Cheap	NumProducts
t	5
f	13

BAC

```
SELECT
Cheap,
COUNT(*) AS ProductCount
FROM ProductPriceClass
GROUP BY Cheap
```

This COUNT(*) is now counting all observations *within* the groups generated by "Cheap"

This GROUP BY keyword combo is critically important – it is setting up a very different kind of query than we have seen so far

This "Cheap" field (which we created) is what we are going to "group" results by

BAC

SQL Statement:

```
SELECT
Cheap
, COUNT(*) as ProductCount
FROM ProductPriceClass
GROUP BY Cheap;
```

Run SQL »

Result:

Number of Records: 2

Cheap	ProductCount
f	66
t	11

You try it:
How many products at each price?

BAC

```
SELECT Price,
COUNT(*) as NumProducts
FROM Products
GROUP BY Price;
```

Related Question:
How *many* different prices do we have in the inventory?

BAC

```
SELECT DISTINCT(Price)
FROM Products;
```

This keyword DISTINCT gets all prices, then eliminates duplicates

BAC

We have 77 products

SQL Statement:

```
SELECT Price FROM Products;
```

Run SQL »

Result:

Number of Records: 77

Price
18
19
10
22

But only 62 distinct prices

SQL Statement:

```
SELECT DISTINCT(Price) FROM Products;
```

Run SQL »

Result:

Number of Records: 62

Price
2.5
4.5
6
7

Another way to count uniques

Another way to count uniques

BAC

This is called a "sub-query" – we can create tables on the fly and treat as if they already existed

```
SELECT
COUNT(*)
FROM  (
        SELECT
        DISTINCT(PRICE)
        FROM Products
        )
```

BAC

Now for something quite different

BAC

We actually have *multiple* tables

Your Database: ?

Tablename	Records
Customers	91
Categories	8
Employees	10
OrderDetails	518
Orders	196
Products	77
Shippers	3
Suppliers	29

Restore Database

BAC

How are they related?

- Each customer can have multiple orders
- Each order can have multiple products
- But each customer, order, and product should be *unique*
- In a database, keys serve to:
  - Uniquely identify entities (like orders and customers)
  - Document relationships between entities

BAC

This is a special field – it is called the table's "Primary Key" – it uniquely identifies a row

This is a special field – it is a "Foreign Key" – it is the primary key to *some other table*.

Number of Records: 77

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	30
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	40
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	97
10	Ikura	4	8	12 - 200 ml jars	31
11	Queso Cabrales	5	4	1 kg pkg.	21
12	Queso Manchego La Pastora	5	4	10 - 500 g pkgs.	38

Let's look at the relationship between customers and orders.

BAC

One-to-many relationship?

- Some customers have more than one order
  - How can we tell?
    - ```
      SELECT
      COUNT(DISTINCT(CustomerID))
      FROM Orders;
      ```
    - `SELECT COUNT(*) FROM Orders;`
  - What's the distribution of orders per customer?
      ```

```sql
SELECT
    CustomerID,
    COUNT(*) AS NumOrders
FROM Orders
GROUP BY CustomerId
ORDER BY NumOrders DESC;
```

What if we want to see orders by Customer *Country?*

- Problem: "Country" isn't in the Order table. If it were, we could just do a "GROUP BY"
- Solution: We can do a JOIN. A join lets us combine columns from different tables, as needed.
  - Great discussion / visualization of different kinds of joins: bit.ly/allthejoins

BAC

Join

```
… FROM A
{LEFT, RIGHT, INNER, OUTER} JOIN B
ON A.primary_key = B.foreign_key
```

LEFT JOIN: everything in A, only rec'd in B that match
RIGHT JOIN: everything in B, only rec'd in A that match
INNER JOIN: has to be in both
OUTER JOIN: everything in both tables, regardless

What do we want?

- For each order, get the Country of the associated Customer

BAC

Note that we are using 'o' and 'c' as *aliases* for Orders and Customers table – this is because if same field name appears in two different tables, SQL doesn't know which one you are talking about

```
SELECT
o.OrderID,
c.Country
FROM Orders AS o
LEFT JOIN Customers AS c
ON o.CustomerID = c.CustomerID
```

BAC

Take every record from "Orders" and then match it up with "Customers" on the basis of CustomerID

```
SELECT
o.OrderID, c.Country
FROM Orders AS o
LEFT JOIN Customers AS c
ON o.CustomerID =
  c.CustomerID;
```

BAC

Note same # of records in Orders as in our LEFT JOIN query

Reason: Every order has 1 and only 1 associated customer, so every order has a "match"

SQL Statement:

```
SELECT
OrderID,
Country
FROM Orders AS o
LEFT JOIN Customers AS c
ON o.CustomerID = c.CustomerID;
```

Run SQL »

Result:

Number of Records: 196

OrderID	Country
10248	Finland
10249	Brazil
10250	Brazil
10251	France

Your Database: ?

Tablename	Records
Customers	91
Categories	8
Employees	10
OrderDetails	518
Orders	196
Products	77
Shippers	3
Suppliers	29

Restore Database

BAC

What if I don't match CustomerID? (don't run this at home)

When does 1 = 1 ?
Always. So SQL matches up every record of Orders with every record in Customers

```
SELECT
o.OrderID,
c.Country
FROM Orders AS o
LEFT JOIN Customers AS c
ON 1 = 1;
```

BAC

SQL Statement:

```
SELECT
OrderID,
Country
FROM Orders AS o
LEFT JOIN Customers AS c
ON 1 = 1;
```

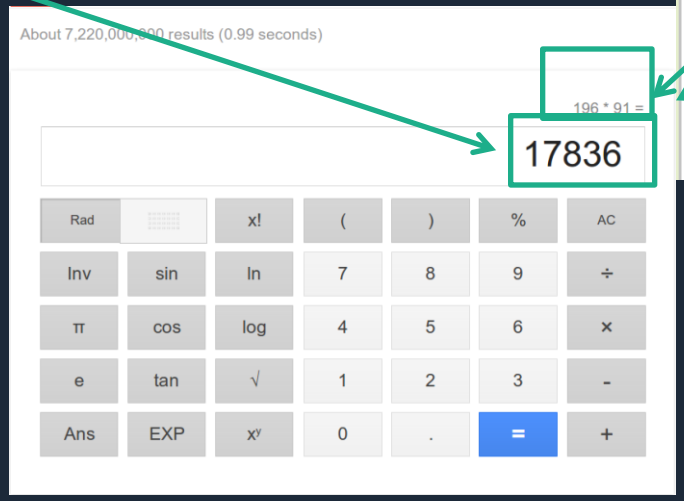
Run SQL »

Result:

Number of Records: 17836

OrderID
10248
10248
10248
10248

The 1 = 1 query matches all to all – the number of records is just the (records in 1) x (record in 2)



Your Database:

Tablename	Records
Customers	91
Categories	8
Employees	10
OrderDetails	518
Orders	196
Products	77
Shippers	3
Suppliers	29

Restore Database

What if a customer record is missing? Let's delete the customer with the lowest ID that made an order

```
DELTE FROM Customers
WHERE CustomerID = 2;
```

BAC

Now let's re-run our LEFT JOIN slightly modified

'null' value for Country – this is because no longer a matching record (we deleted it)

```
SELECT
    o.OrderID,
    c.Country,
    o.CustomerID
FROM Orders AS o
LEFT JOIN Customers AS c
On o.CustomerID = c.CustomerID
ORDER BY o.CustomerID ASC;
```

SQL Statement: Edit the SQL Statement, a

```
SELECT
o.OrderID, c.Country, o.CustomerId
FROM Orders AS o
LEFT JOIN Customers AS c
ON o.CustomerID = c.CustomerID
ORDER BY o.CustomerID ASC;
```

Run SQL »

Result:

Number of Records: 196

OrderID	Country	CustomerID
10308	null	2
10365	Mexico	3
10355	UK	4
10383	UK	4
10278	Sweden	5
10280	Sweden	5

What if we only want records where we have a customer?

Replacement of LEFT JOIN with just JOIN causes the null record to be dropped

```
SELECT
    o.OrderID,
    c.Country,
    o.CustomerID
FROM Orders AS o
JOIN Customers AS c
On o.CustomerID = c.CustomerID
ORDER BY o.CustomerID ASC;
```

SQL Statement: Edit the SQL Statement, a

```
SELECT
o.OrderID, c.Country, o.CustomerId
FROM Orders AS o
JOIN Customers AS c
ON o.CustomerID = c.CustomerID
ORDER BY o.CustomerID ASC;
```

Run SQL »

Result:

Number of Records: 195

OrderID	Country	CustomerID
10365	Mexico	3
10355	UK	4
10383	UK	4
10278	Sweden	5
10280	Sweden	5
10384	Sweden	5
10265	France	7

Let's go back to our original question: Geographic breakdown of orders

Your Database:

Tablename	Records
Customers	91
Categories	8
Employees	10
OrderDetails	518
Orders	196
Products	77
Shippers	3
Suppliers	29

Restore Database

First, let's bring back Customer #2

Have outer-query that counts up countries with a GROUP BY

Make a sub-query that returns each customer Country

```
SELECT Country,
       COUNT(*) AS NumOrders
FROM
       (SELECT
       Country
       FROM Orders AS o
       JOIN Customers AS c
       On o.CustomerID = c.CustomerID)
GROUP BY Country
ORDER BY NumOrders DESC;
```

BAC

SQL Statement: Edit the SQL Statement, and click "Run SQL" to see the result.

```
SELECT Country, COUNT(*) AS num_orders
FROM
(SELECT
Country
FROM Orders AS o
JOIN Customers AS c
```

Run SQL »

Result:

Number of Records: 21

Country	num_orders
USA	29
Germany	25
Brazil	19
France	18
Austria	13
UK	12
Canada	9
Mexico	9
Venezuela	9
Finland	8
Italy	7
Spain	7
Sweden	7
Ireland	6
Portugal	5
Denmark	4
Switzerland	4
Belgium	2
Argentina	1
Norway	1
Poland	1

BAC

Questions

1. What distinct countries are the suppliers from?

2. For each supplier, what is the average price of the products they sell?

3. Who is the best shipper by number of orders shipped?

4. What is the quantity of each product sold?

5. Give us the name of each product as well as the quantity sold of the product.

BAC

Answer 1

1. What distinct countries are the suppliers from?

```
SELECT
DISTINCT(Country)
FROM Suppliers
```

BAC

Answer 2

2. For each supplier, what is the average price of the products they sell?

```
SELECT
SupplierID
,AVG(Price)
FROM Products
GROUP BY SupplierID
```

BAC

Answer 3

3. Who is the best shipper by number of orders shipped?

```
SELECT ShipperID
,COUNT(*) AS NumOrdersByShipperID
FROM ORDERS
GROUP BY ShipperID
```

BAC

Answer 4

4. What is the quantity of each product sold?

```
CREATE TABLE ProductQuant AS
SELECT
ProductID
,SUM(Quantity) AS TotalQuantSold
FROM OrderDetails
GROUP BY ProductID
```

BAC

Answer 5

4. Give us the name of each product as well as the quantity sold of the product?

```
SELECT
b.ProductName
, a.TotalQuantSold
FROM TEST AS a
LEFT JOIN Products AS b
ON a.ProductID = b.ProductID
```

BAC

Acknowledgement

John Horton

Assistant Professor of Information,
Operations and Management Sciences

BAC