Class_4_Notes

JOINS-JOINS-JOINS

Getting data from Files and Tidying in the Verse

Recall from your homework - class 3, we used tidyverse inner_join:

```
library(tidyverse)
library(stringr)
library(lubridate)

OrdDetail = read_csv(str_c(locationString,"SalesOrderDetail.csv"))
Prod = read_csv(str_c(locationString,"Product.csv"))


ProdSales = OrdDetail %>%
  inner_join(Prod, by = "ProductID") %>%
  group_by(ProductID, Name) %>%
  summarise(TotalSales = round(sum(LineTotal),2)) %>%
  arrange(ProductID)
```

Description	Valuse
Total Sales =	708690.2

ProductID	Name	TotalSales
707	Sport-100 Helmet, Red	734.79
708	Sport-100 Helmet, Black	1032.00
711	Sport-100 Helmet, Blue	757.10
712	AWC Logo Cap	277.36
714	Long-Sleeve Logo Jersey, M	779.84
715	Long-Sleeve Logo Jersey, L	1463.83

Getting Data from the Server, and Tidying in the Verse

Data doesn't always come from text files. MOST data in Enterprises are stored in SQL Servers *(ALL of the ERP, and >90% of Line of Business Applications data is in SQL databases).*

Lets get data straight off the server:

```
# Select Data from Sales Order Detail
OrdDetailSQL <-  dbGetQuery(con1,"
SELECT
[SalesLT].[SalesOrderDetail].[SalesOrderID]
,[SalesLT].[SalesOrderDetail].[SalesOrderDetailID]
,[SalesLT].[SalesOrderDetail].[OrderQty]
,[SalesLT].[SalesOrderDetail].[ProductID]
,[SalesLT].[SalesOrderDetail].[UnitPrice]
,[SalesLT].[SalesOrderDetail].[UnitPriceDiscount]
,[SalesLT].[SalesOrderDetail].[LineTotal]
```

```
FROM
[SalesLT].[SalesOrderDetail]
")

# Get Data From Product

ProdSQL <- dbGetQuery(con1,"
SELECT
[SalesLT].[Product].[ProductCategoryID]
,[SalesLT].[Product].[ProductID]
,[SalesLT].[Product].[Name]
,[SalesLT].[Product].[ProductNumber]
,[SalesLT].[Product].[ListPrice]
,[SalesLT].[Product].[StandardCost]
FROM [SalesLT].[Product]
")


ProdSalesSQL = OrdDetail %>%
  inner_join(Prod, by = "ProductID") %>%
  group_by(ProductID, Name) %>%
  summarise(TotalSales = round(sum(LineTotal),2)) %>%
  arrange(ProductID)
```

Description	Valuse
Total Sales =	708690.2

ProductID	Name	TotalSales
707	Sport-100 Helmet, Red	734.79
708	Sport-100 Helmet, Black	1032.00
711	Sport-100 Helmet, Blue	757.10
712	AWC Logo Cap	277.36
714	Long-Sleeve Logo Jersey, M	779.84
715	Long-Sleeve Logo Jersey, L	1463.83

Getting Data and Tidying on the Server

Using SQL Server to do the work *(Note how Group By works like group_by R, and Order By works like arrange in R)*:

```
ProdSalesSQL2 <-  dbGetQuery(con1,"
SELECT
[SalesLT].[SalesOrderDetail].[ProductID]
,[SalesLT].[Product].[Name]
,sum([SalesLT].[SalesOrderDetail].[LineTotal]) AS TotalSales
FROM
[SalesLT].[SalesOrderDetail]
INNER JOIN
[SalesLT].[Product]
ON
[SalesLT].[SalesOrderDetail].[ProductID] = [SalesLT].[Product].[ProductID]
GROUP BY
[SalesLT].[SalesOrderDetail].[ProductID]
```

```
,[SalesLT].[Product].[Name]
ORDER BY
[SalesLT].[SalesOrderDetail].[ProductID]
")
```

Description	Valuse
Total Sales =	708690.2

ProductID	Name	TotalSales
707	Sport-100 Helmet, Red	734.7900
708	Sport-100 Helmet, Black	1031.9951
711	Sport-100 Helmet, Blue	757.0961
712	AWC Logo Cap	277.3631
714	Long-Sleeve Logo Jersey, M	779.8440
715	Long-Sleeve Logo Jersey, L	1463.8322

Also, note that the SQL INNER JOIN is similar to R. For application purposes, you should plan to do as much tidying in SQL as possible - Enterprise data is usually **HUGE** in scale, and your desktop isn't going to handle it. and neither will *project* level servers and *marts* like Tableau and Alteryx. That said, you'll still need to do tidying in R too.

Adding Product Category and Tidying in the Verse

```
# Get Data From Product with Product Category
Prod <- dbGetQuery(con1,"
SELECT
[SalesLT].[Product].[ProductCategoryID]
,[SalesLT].[Product].[ProductID]
,[SalesLT].[Product].[Name]
,[SalesLT].[Product].[ProductNumber]
,[SalesLT].[Product].[ListPrice]
,[SalesLT].[Product].[StandardCost]
FROM [SalesLT].[Product]
")

ProdCat <- dbGetQuery(con1,"
SELECT
[SalesLT].[ProductCategory].[ProductCategoryID]
,[SalesLT].[ProductCategory].[Name]
FROM [SalesLT].[ProductCategory]
")

ProdCatSales = ProdCat %>%
  rename(CategoryName = Name) %>%
  # important to understand how dplyr will handle duplicate column names
  inner_join(Prod, by = "ProductCategoryID") %>%
  inner_join(OrdDetail, by = "ProductID") %>%
  group_by(ProductCategoryID, CategoryName) %>%
  summarise(TotalSales = round(sum(LineTotal),2)) %>%
  arrange(ProductCategoryID)
```

Description	Valuse
Total Sales =	708690.2

ProductCategoryID	CategoryName	TotalSales
5	Mountain Bikes	170825.89
6	Road Bikes	183130.30
7	Touring Bikes	220655.38
8	Handlebars	1192.97
9	Bottom Brackets	1320.17
10	Brakes	830.70

Note the *rename* here. R will not allow you to create a dataframe with 2 columns of the same name *(ProductCategory and Product both have "Name" columns)*. Neither will SQL, but R will rename the column for you *(then you can rename later)*. SQL will just give you an error if you want to store the results in a table - or temp table *(not runnning this below, but it will err out - run in beaver if you want to see)*. SQL won't give you an error if you're just running a report:

```
ProductCatSalesSQL = dbGetQuery(con1,"
/*

DROP TABLE IF EXISTS ##TempProductQuery
set nocount on

SELECT
 [SalesLT].[ProductCategory].[ProductCategoryID]
,[SalesLT].[ProductCategory].[Name]
,[SalesLT].[Product].[Name]
,sum([SalesLT].[SalesOrderDetail].[LineTotal]) AS TotalSales
INTO ##TempProductQuery
FROM
[SalesLT].[ProductCategory]
INNER JOIN
[SalesLT].[Product]
ON
[SalesLT].[ProductCategory].[ProductCategoryID] = [SalesLT].[Product].[ProductCategoryID]
INNER JOIN
[SalesLT].[SalesOrderDetail]
ON
[SalesLT].[Product].[ProductID] = [SalesLT].[SalesOrderDetail].[ProductID]
GROUP BY
 [SalesLT].[ProductCategory].[ProductCategoryID]
,[SalesLT].[ProductCategory].[Name]
,[SalesLT].[Product].[Name]


*/
")

# No output - just error
```

Note: You can create a temp table in SQL to store results and manipulate data. It's similar to putting a query result into your own dataframe - but it lives on the server for the duration of your session. That said - **don't use temp tables in the this class**. I put it here so you'd be aware of the functionality. You won't need them for classwork or exams.

Getting Data from the Server, and Tidying in the Server

Without using a temp table, the query would look like this:

```
ProdCatSalesSQL = dbGetQuery(con1,"

SELECT
 [SalesLT].[ProductCategory].[ProductCategoryID]
,[SalesLT].[ProductCategory].[Name] AS CategoryName
,[SalesLT].[Product].[Name]
,sum([SalesLT].[SalesOrderDetail].[LineTotal]) AS TotalSales
FROM
[SalesLT].[ProductCategory]
INNER JOIN
[SalesLT].[Product]
ON
[SalesLT].[ProductCategory].[ProductCategoryID] = [SalesLT].[Product].[ProductCategoryID]
INNER JOIN
[SalesLT].[SalesOrderDetail]
ON
[SalesLT].[Product].[ProductID] = [SalesLT].[SalesOrderDetail].[ProductID]
GROUP BY
 [SalesLT].[ProductCategory].[ProductCategoryID]
,[SalesLT].[ProductCategory].[Name]
,[SalesLT].[Product].[Name]

")
```

Description	Valuse
Total Sales =	708690.2

ProductCategoryID	CategoryName	Name	TotalSales
23	Caps	AWC Logo Cap	277.3631
33	Cleaners	Bike Wash - Dissolver	251.8759
11	Chains	Chain	97.1520
29	Vests	Classic Vest, M	1295.4000
29	Vests	Classic Vest, S	3014.5037
10	Brakes	Front Brakes	766.8000

LEFT JOINS

What if management says "I want to see the products that didn't sell too - just as interested in those". A left join includes everything on the *Left* side of the join *(or the right side with a RIGHT JOIN - this works the same in R)*:

Single Left Join

```
AllProdSalesSQL = dbGetQuery(con1,"
SELECT
 [SalesLT].[Product].[ProductID]
,[SalesLT].[Product].[Name] AS Product_Name
,SUM([SalesLT].[SalesOrderDetail].[LineTotal]) AS TotalSales
FROM
[SalesLT].[Product]
```

```
LEFT JOIN
[SalesLT].[SalesOrderDetail]
ON
[SalesLT].[Product].[ProductID] = [SalesLT].[SalesOrderDetail].[ProductID]
GROUP BY
 [SalesLT].[Product].[ProductID]
,[SalesLT].[Product].[Name]

")
```

Description	Valuse
Total Sales =	708690.2

ProductID	Product_Name	TotalSales
879	All-Purpose Bike Stand	NA
712	AWC Logo Cap	277.3631
877	Bike Wash - Dissolver	251.8759
843	Cable Lock	NA
952	Chain	97.1520
866	Classic Vest, L	NA

LEFT JOIN with INNER JOIN

Now let's create a LEFT JOIN between Product and SalesOrderDetail with an INNER JOIN between Product Category and Product:

```
AllProdCatSalesSQL = dbGetQuery(con1,"

SELECT
 [SalesLT].[ProductCategory].[Name]
,[SalesLT].[Product].[ProductID]
,[SalesLT].[Product].[Name] AS Product_Name
,SUM([SalesLT].[SalesOrderDetail].[LineTotal]) AS TotalSales
FROM
[SalesLT].[ProductCategory]
INNER JOIN
[SalesLT].[Product]
ON
[SalesLT].[ProductCategory].[ProductCategoryID] = [SalesLT].[Product].[ProductCategoryID]
LEFT JOIN
[SalesLT].[SalesOrderDetail]
ON
[SalesLT].[Product].[ProductID] = [SalesLT].[SalesOrderDetail].[ProductID]
GROUP BY
 [SalesLT].[ProductCategory].[Name]
,[SalesLT].[Product].[ProductID]
,[SalesLT].[Product].[Name]

")

## BE SURE AND USE na.rm = T when summing a column with NAs
```

Description	Valuse
Total Sales =	708690.2

Name	ProductID	Product_Name	TotalSales
Road Frames	680	HL Road Frame - Black, 58	NA
Road Frames	706	HL Road Frame - Red, 58	NA
Helmets	707	Sport-100 Helmet, Red	734.790
Helmets	708	Sport-100 Helmet, Black	1031.995
Socks	709	Mountain Bike Socks, M	NA
Socks	710	Mountain Bike Socks, L	NA

2 LEFT JOINs

Extending this example, let's create a LEFT JOIN between Product and SalesOrderDetail with an LEFT JOIN between Product Category and Product:

```
AllProdCatandProdSalesSQL = dbGetQuery(con1,"
SELECT
 [SalesLT].[ProductCategory].[Name]
,[SalesLT].[Product].[ProductID]
,[SalesLT].[Product].[Name] AS Product_Name
,SUM([SalesLT].[SalesOrderDetail].[LineTotal]) AS TotalSales
FROM
[SalesLT].[ProductCategory]
LEFT JOIN
[SalesLT].[Product]
ON
[SalesLT].[ProductCategory].[ProductCategoryID] = [SalesLT].[Product].[ProductCategoryID]
LEFT JOIN
[SalesLT].[SalesOrderDetail]
ON
[SalesLT].[Product].[ProductID] = [SalesLT].[SalesOrderDetail].[ProductID]
GROUP BY
 [SalesLT].[ProductCategory].[Name]
,[SalesLT].[Product].[ProductID]
,[SalesLT].[Product].[Name]

")
```

Description	Valuse
Total Sales =	708690.2

Name	ProductID	Product_Name	TotalSales
Accessories	NA	NA	NA
Bikes	NA	NA	NA
Clothing	NA	NA	NA
Components	NA	NA	NA
Road Frames	680	HL Road Frame - Black, 58	NA
Road Frames	706	HL Road Frame - Red, 58	NA

SQL Filtering

One last thing for today. SQL has very sophisticated filtering and aggregation functions, but now for the basics: the **WHERE** clause is roughly equivalent to R's "filter". SQL can also filter aggregates using the **HAVING** clause. See below:

```
FilterExample = dbGetQuery(con1,"
SELECT
[SalesLT].[ProductCategory].[ProductCategoryID]
,[SalesLT].[ProductCategory].[Name]
,[SalesLT].[Product].[ProductID]
,[SalesLT].[Product].[Name] AS Product_Name
,SUM([SalesLT].[SalesOrderDetail].[LineTotal]) AS TotalSales
FROM
[SalesLT].[ProductCategory]
LEFT JOIN
[SalesLT].[Product]
ON
[SalesLT].[ProductCategory].[ProductCategoryID] = [SalesLT].[Product].[ProductCategoryID]
LEFT JOIN
[SalesLT].[SalesOrderDetail]
ON
[SalesLT].[Product].[ProductID] = [SalesLT].[SalesOrderDetail].[ProductID]
WHERE
[SalesLT].[ProductCategory].[ProductCategoryID] = 5
GROUP BY
[SalesLT].[ProductCategory].[ProductCategoryID]
,[SalesLT].[ProductCategory].[Name]
,[SalesLT].[Product].[ProductID]
,[SalesLT].[Product].[Name]
HAVING
SUM([SalesLT].[SalesOrderDetail].[LineTotal])  < 1000.00

")
```

ProductCategoryID	Name	ProductID	Product_Name	TotalSales
5	Mountain Bikes	984	Mountain-500 Silver, 40	745.7868
5	Mountain Bikes	985	Mountain-500 Silver, 42	813.5856
5	Mountain Bikes	986	Mountain-500 Silver, 44	271.1952
5	Mountain Bikes	987	Mountain-500 Silver, 48	406.7928
5	Mountain Bikes	989	Mountain-500 Black, 40	971.9820
5	Mountain Bikes	990	Mountain-500 Black, 42	971.9820
5	Mountain Bikes	993	Mountain-500 Black, 52	971.9820