# IS475/675 Agenda for 04/14/2025

- Present the use of Common Table Expresssions (CTE’s).
- Compare and contrast Views and CTE’s.
- Discuss applications for Views and CTE’s – a “group of a group.”

- While waiting for class to start, if you didn’t do SQL Lab Exercise 8 or attend class on Wednesday(04/10/2025) then execute SQL Server Management Studio and run this script file: K:\cob\is475\labfiles\SQLLab8.sql

# We are creating more complex queries

## Simple queries usually:

Generate large result tables with relatively simple filtering operations.

Tend to require only one transaction table.

Do not require significant changes to the structure of the data; one row in the transaction table produces one filtered row in the result table.

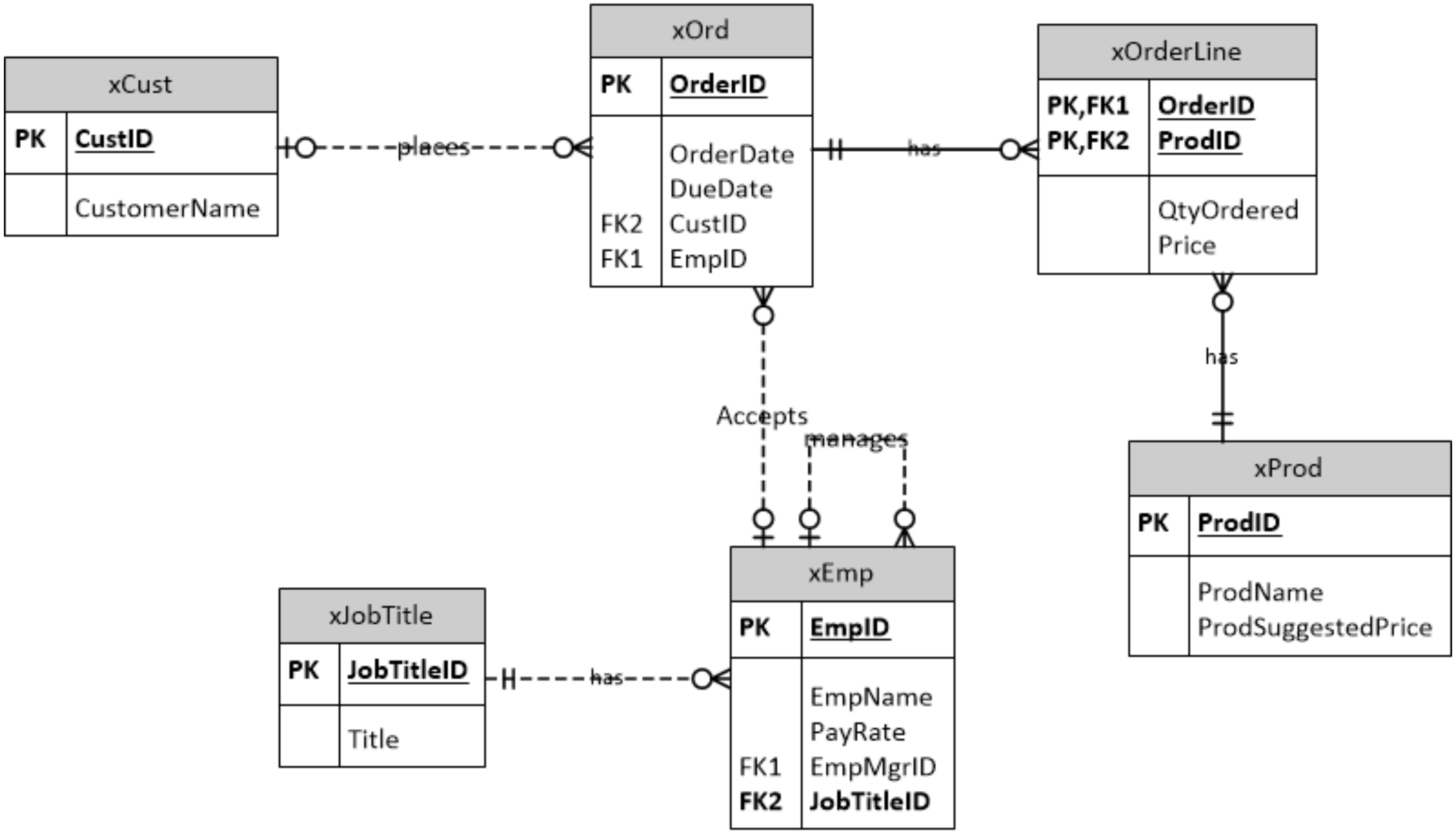
## More complex queries can:

Require a combination of joins, group functions, and sub-queries.

Return one row per group because they make greater use of group functions.

Provide direct information for decision makers.

# Remember the database design for exercises 7 & 8



xCust
PK **CustID**
CustomerName

xOrd
PK **OrderID**
OrderDate
DueDate
FK2 CustID
FK1 EmpID

xOrderLine
**PK,FK1** **OrderID**
**PK,FK2** **ProdID**
QtyOrdered
Price

xProd
PK **ProdID**
ProdName
ProdSuggestedPrice

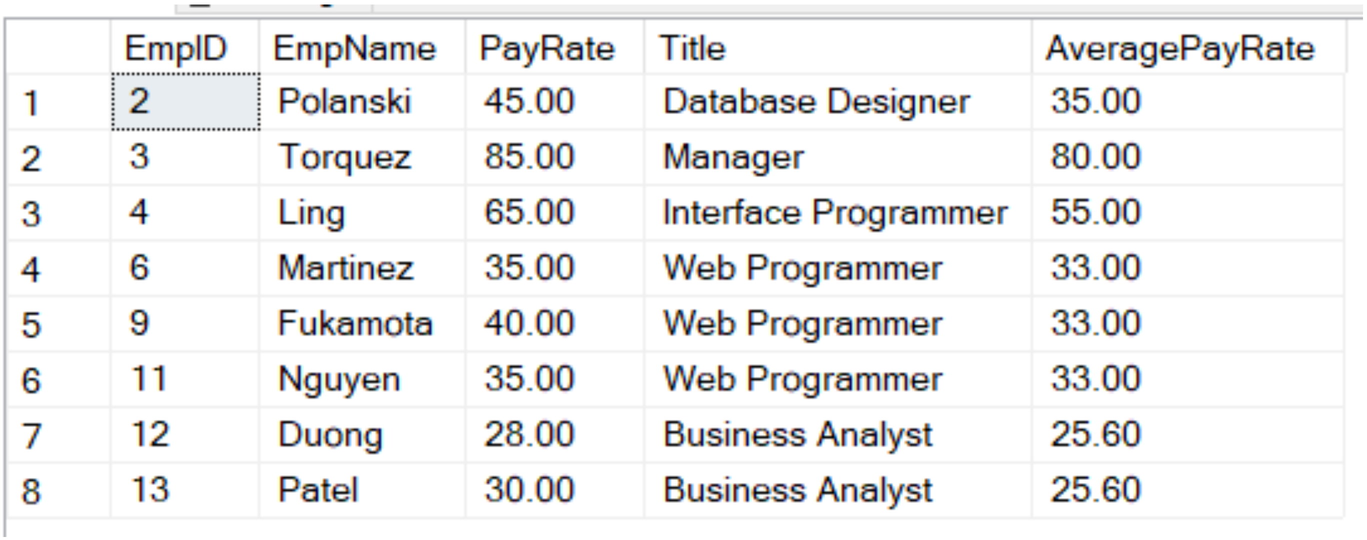
xJobTitle
PK **JobTitleID**
Title

xEmp
PK **EmpID**
EmpName
PayRate
FK1 EmpMgrID
**FK2** **JobTitleID**

# Look at the content of the tables

- SELECT * FROM xEmp;
- SELECT * FROM xProd;
- SELECT * FROM xOrd;
- SELECT * FROM xOrderLine;
- SELECT * FROM xCust
- SELECT * FROM xJobTitle;

# Recap: Which employees have a payrate than is higher than the average payrate for their job title?



| | EmpID | EmpName | PayRate | Title | AveragePayRate |
|---|---|---|---|---|---|
| 1 | 2 | Polanski | 45.00 | Database Designer | 35.00 |
| 2 | 3 | Torquez | 85.00 | Manager | 80.00 |
| 3 | 4 | Ling | 65.00 | Interface Programmer | 55.00 |
| 4 | 6 | Martinez | 35.00 | Web Programmer | 33.00 |
| 5 | 9 | Fukamota | 40.00 | Web Programmer | 33.00 |
| 6 | 11 | Nguyen | 35.00 | Web Programmer | 33.00 |
| 7 | 12 | Duong | 28.00 | Business Analyst | 25.60 |
| 8 | 13 | Patel | 30.00 | Business Analyst | 25.60 |

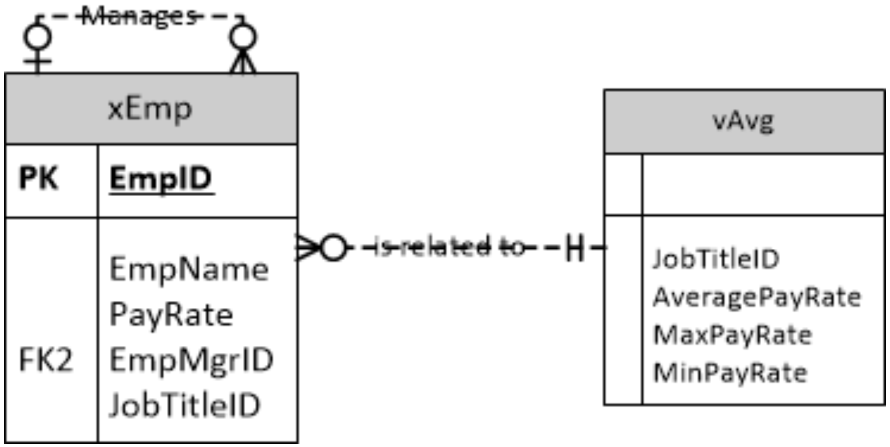
# Recap - we can simplify queries with the use of SQL Views. ***What is a SQL view?***

- A “virtual” table.
  - A set of SQL statements that creates a result table which can be accessed by other SQL statements.
- A database object.
  - The code for a view is stored in the database.
  - A view contains no data of its own.
  - A view relies on the data in the base tables used to create the view.
- A set of stored SQL code.
  - Stores code; not data.

# We created a SQL View to solve the problem (from SQL Lab Exercise 8, Task 4, pg. 8

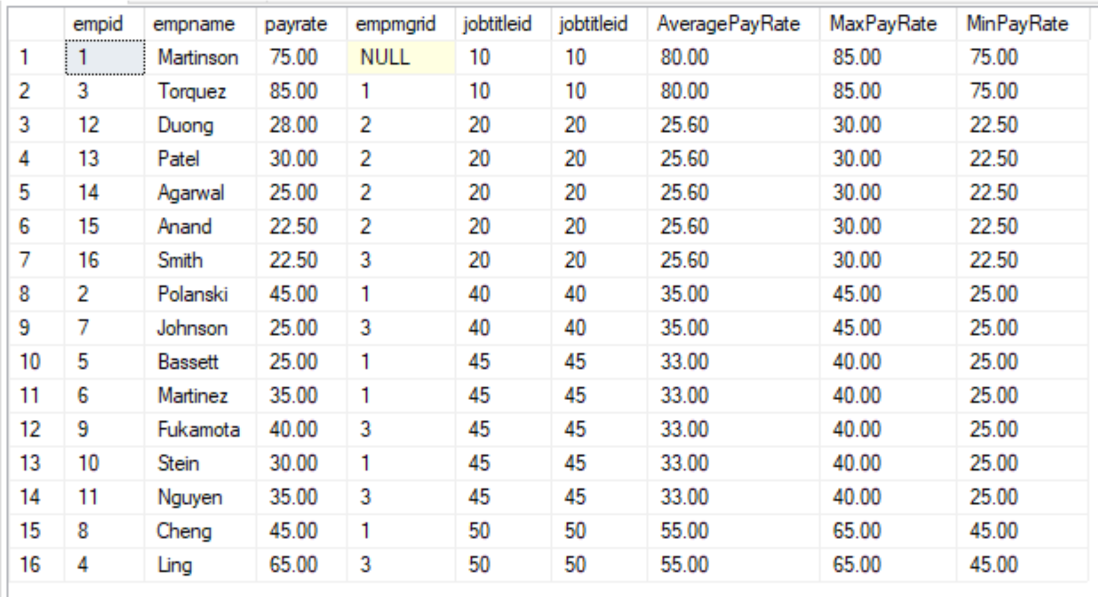
```
CREATE VIEW vAvg AS
SELECT         jobtitleID,
               AVG(payrate) AS AveragePayRate,
               MAX(payrate) AS MaxPayRate,
               MIN(payrate) AS MinPayRate
FROM           xemp
GROUP BY       jobtitleID;
```

# Join the view to the xemp table



The view does not have a primary key because it does not contain data. As long as you include a field that can be used to join (JobTitleID in this situation), then the view can be joined to other tables or other views.

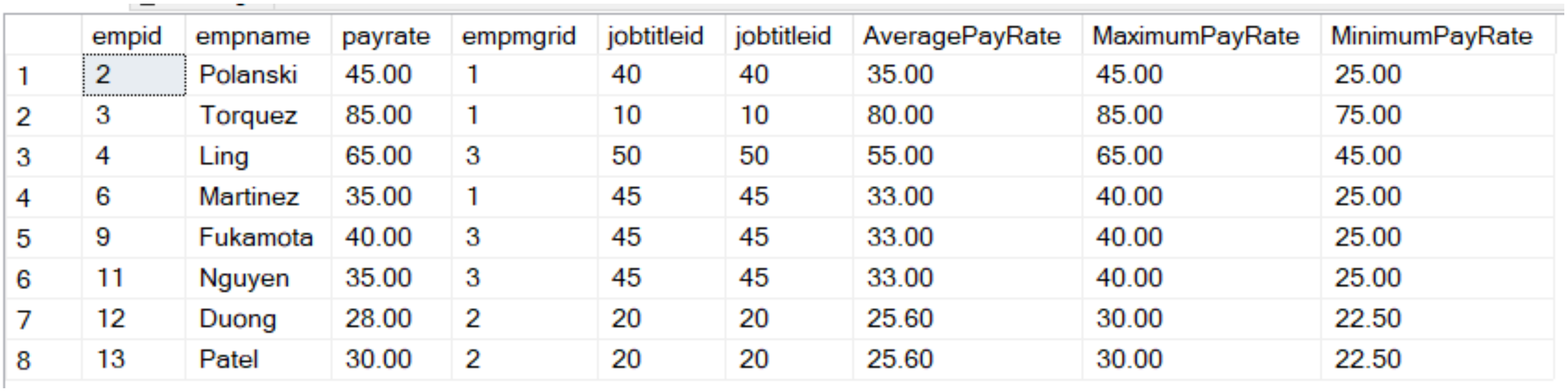
```
SELECT *
FROM   xemp
INNER JOIN vAvg
ON xemp.jobtitleid =
vAvg.jobtitleid
```



|  | empid | empname | payrate | empmgrid | jobtitleid | jobtitleid | AveragePayRate | MaxPayRate | MinPayRate |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Martinson | 75.00 | NULL | 10 | 10 | 80.00 | 85.00 | 75.00 |
| 2 | 3 | Torquez | 85.00 | 1 | 10 | 10 | 80.00 | 85.00 | 75.00 |
| 3 | 12 | Duong | 28.00 | 2 | 20 | 20 | 25.60 | 30.00 | 22.50 |
| 4 | 13 | Patel | 30.00 | 2 | 20 | 20 | 25.60 | 30.00 | 22.50 |
| 5 | 14 | Agarwal | 25.00 | 2 | 20 | 20 | 25.60 | 30.00 | 22.50 |
| 6 | 15 | Anand | 22.50 | 2 | 20 | 20 | 25.60 | 30.00 | 22.50 |
| 7 | 16 | Smith | 22.50 | 3 | 20 | 20 | 25.60 | 30.00 | 22.50 |
| 8 | 2 | Polanski | 45.00 | 1 | 40 | 40 | 35.00 | 45.00 | 25.00 |
| 9 | 7 | Johnson | 25.00 | 3 | 40 | 40 | 35.00 | 45.00 | 25.00 |
| 10 | 5 | Bassett | 25.00 | 1 | 45 | 45 | 33.00 | 40.00 | 25.00 |
| 11 | 6 | Martinez | 35.00 | 1 | 45 | 45 | 33.00 | 40.00 | 25.00 |
| 12 | 9 | Fukamota | 40.00 | 3 | 45 | 45 | 33.00 | 40.00 | 25.00 |
| 13 | 10 | Stein | 30.00 | 1 | 45 | 45 | 33.00 | 40.00 | 25.00 |
| 14 | 11 | Nguyen | 35.00 | 3 | 45 | 45 | 33.00 | 40.00 | 25.00 |
| 15 | 8 | Cheng | 45.00 | 1 | 50 | 50 | 55.00 | 65.00 | 45.00 |
| 16 | 4 | Ling | 65.00 | 3 | 50 | 50 | 55.00 | 65.00 | 45.00 |

# Filter the rows and sort the result table

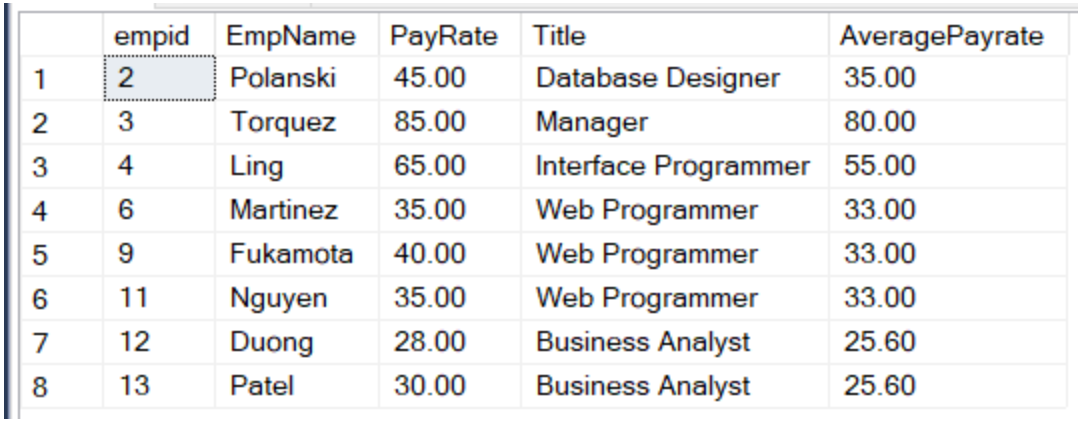
```
SELECT *
FROM   xemp
INNER JOIN vAvg
ON xemp.jobtitleid = vAvg.jobtitleid
WHERE Payrate > AveragePayrate
ORDER BY empid
```



| | empid | empname | payrate | empmgrid | jobtitleid | jobtitleid | AveragePayRate | MaximumPayRate | MinimumPayRate |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | Polanski | 45.00 | 1 | 40 | 40 | 35.00 | 45.00 | 25.00 |
| 2 | 3 | Torquez | 85.00 | 1 | 10 | 10 | 80.00 | 85.00 | 75.00 |
| 3 | 4 | Ling | 65.00 | 3 | 50 | 50 | 55.00 | 65.00 | 45.00 |
| 4 | 6 | Martinez | 35.00 | 1 | 45 | 45 | 33.00 | 40.00 | 25.00 |
| 5 | 9 | Fukamota | 40.00 | 3 | 45 | 45 | 33.00 | 40.00 | 25.00 |
| 6 | 11 | Nguyen | 35.00 | 3 | 45 | 45 | 33.00 | 40.00 | 25.00 |
| 7 | 12 | Duong | 28.00 | 2 | 20 | 20 | 25.60 | 30.00 | 22.50 |
| 8 | 13 | Patel | 30.00 | 2 | 20 | 20 | 25.60 | 30.00 | 22.50 |

# Add the job title and SELECT the columns

```
SELECT              xemp.empid, xemp.EmpName, xemp.PayRate,
                    jt.Title,
                    vAvg.AveragePayrate
FROM                xemp
INNER JOIN vAvg
ON xemp.jobtitleid = vAvg.jobtitleid
INNER JOIN xJobTitle jt
on xemp.jobtitleid = jt.jobtitleid
WHERE Payrate > AveragePayrate
ORDER BY empid
```



|  | empid | EmpName | PayRate | Title | AveragePayrate |
|---|---|---|---|---|---|
| 1 | 2 | Polanski | 45.00 | Database Designer | 35.00 |
| 2 | 3 | Torquez | 85.00 | Manager | 80.00 |
| 3 | 4 | Ling | 65.00 | Interface Programmer | 55.00 |
| 4 | 6 | Martinez | 35.00 | Web Programmer | 33.00 |
| 5 | 9 | Fukamota | 40.00 | Web Programmer | 33.00 |
| 6 | 11 | Nguyen | 35.00 | Web Programmer | 33.00 |
| 7 | 12 | Duong | 28.00 | Business Analyst | 25.60 |
| 8 | 13 | Patel | 30.00 | Business Analyst | 25.60 |

# What is a Common Table Expression (CTE)?

- A CTE is much like a view.
- A CTE creates a named virtual result table, just like a view.
- A CTE, however, is not a database object – it is only available in the session that is actively using the code.
- It is a temporary virtual result table, while a view is a more permanent virtual result table.
- A CTE is not ANSI-standard.  It is available in MS SQL Server T-SQL.

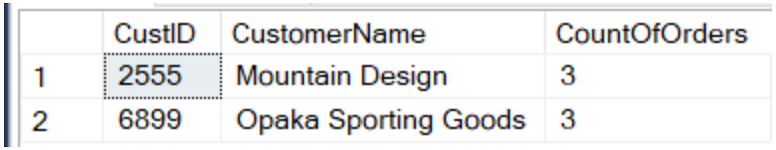
# Do the same thing with a CTE

```sql
WITH cteAvgRate AS
(SELECT jobtitleid,
    avg(payrate) AveragePayRate
 FROM  xemp
 GROUP BY jobtitleid)

SELECT emp.empid,
       emp.empname,
       emp.payrate,
       Title,
       cteAvgRate.AveragepayRate
FROM   xemp emp
inner join    cteAvgRate
ON     emp.jobtitleid = cteAvgRate.jobtitleid
inner join    xJobTitle jt
ON     emp.jobtitleid = jt.jobtitleid
WHERE  payrate > averagepayrate
ORDER BY 1
```

# View vs. CTE

| | View | Common Table Expression |
|---|---|---|
| **Create/Store** | Stored as a database object. | Not stored as a database object.  Local to a single query. |
| **Extent of Use** | Use when the result table will be used in more than one query. | Use when the result table is local to single query. |
| **Portability** | Can be accessed by programs other than SQL. | Can only be used by SQL. |

# Moving on! Which customer(s) placed the most orders with our company based on a count of the orders?



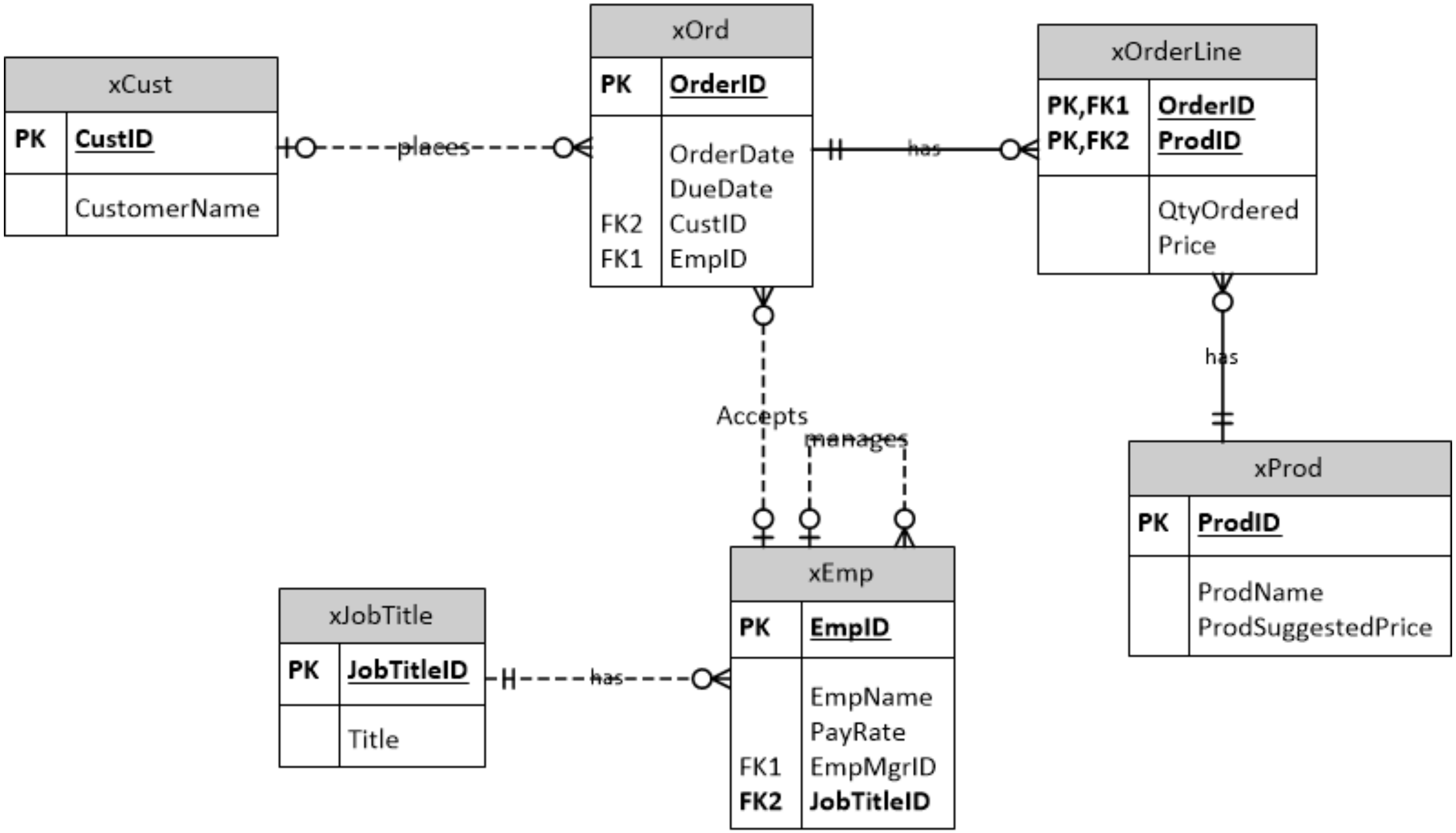
| | CustID | CustomerName | CountOfOrders |
|---|---|---|---|
| 1 | 2555 | Mountain Design | 3 |
| 2 | 6899 | Opaka Sporting Goods | 3 |

This is a “group of a group” – in this example it is a MAX of a COUNT

Pseudocode:

```
SELECT customer data
FROM cust, ord
WHERE COUNT(orderID) = MAX(COUNT(orderID))
```

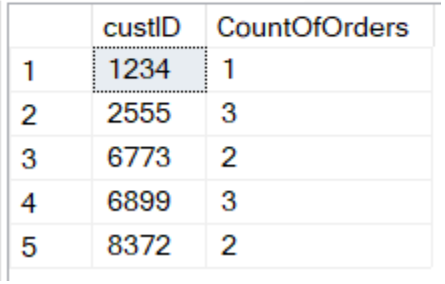
Please note that this is not possible. First, a group function cannot be included in the WHERE clause. Second, it is not possible to nest GROUP functions.


xCust
PK CustID
CustomerName
places
xOrd
PK OrderID
OrderDate
DueDate
FK2 CustID
FK1 EmpID
has
xOrderLine
PK,FK1 OrderID
PK,FK2 ProdID
QtyOrdered
Price
has
xProd
PK ProdID
ProdName
ProdSuggestedPrice
Accepts
manages
xEmp
PK EmpID
EmpName
PayRate
FK1 EmpMgrID
FK2 JobTitleID
has
xJobTitle
PK JobTitleID
Title

# Let's explore the problem – what are we counting?

Example of "playing with code" to get an idea of the logic:

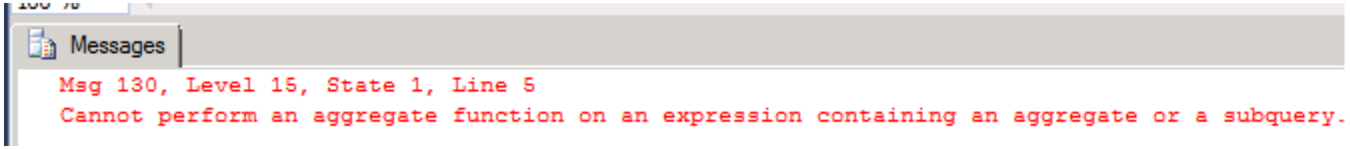
```
SELECT  custID,
        count(*)   CountOfOrders
FROM    xOrd
GROUP BY custID
```



|   | custID | CountOfOrders |
|---|---|---|
| 1 | 1234 | 1 |
| 2 | 2555 | 3 |
| 3 | 6773 | 2 |
| 4 | 6899 | 3 |
| 5 | 8372 | 2 |

How do you see just the customers with the most orders? Especially if you don't know how many there are? Not this way!

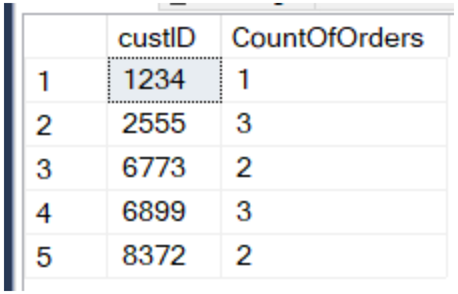
```
SELECT  custID,
        count(*)   CountOfOrders
FROM    xOrd
GROUP BY custID
HAVING count(*) = max(count(*))
```



```
Msg 130, Level 15, State 1, Line 5
Cannot perform an aggregate function on an expression containing an aggregate or a subquery.
```

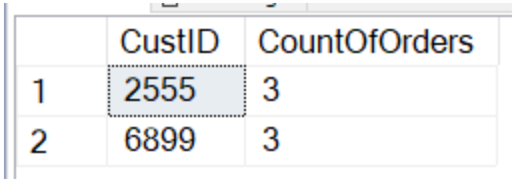
# Let’s use a View and a sub-query for the basic logic

```
CREATE VIEW vCountOrders AS
SELECT          custID,
                count(*)  CountOfOrders
FROM            xOrd
GROUP BY        custID;
```



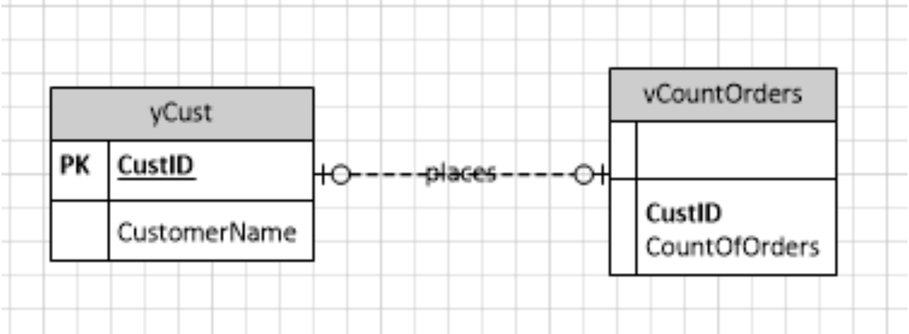
|   | custID | CountOfOrders |
|---|---|---|
| 1 | 1234 | 1 |
| 2 | 2555 | 3 |
| 3 | 6773 | 2 |
| 4 | 6899 | 3 |
| 5 | 8372 | 2 |

```
SELECT          *
FROM            vCountOrders vCount
WHERE           countoforders =
                (SELECT MAX(CountOfOrders)
                 FROM   vCountOrders);
```

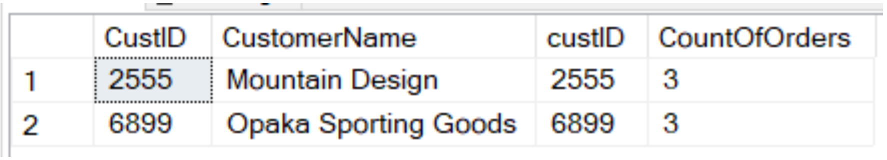


|   | CustID | CountOfOrders |
|---|---|---|
| 1 | 2555 | 3 |
| 2 | 6899 | 3 |

# Join in the customer table to see the customer name



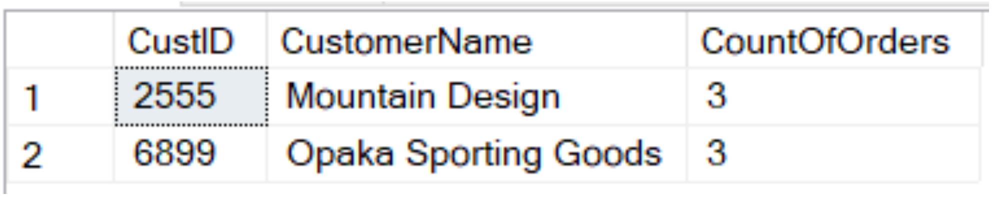
```
SELECT          *
FROM            xCust cust
INNER JOIN      vCountOrders vCount
ON              cust.custid = vCount.custID
WHERE           countoforders =
                (SELECT MAX(CountOfOrders)
                 FROM   vCountOrders);
```



| | CustID | CustomerName | custID | CountOfOrders |
|---|---|---|---|---|
| 1 | 2555 | Mountain Design | 2555 | 3 |
| 2 | 6899 | Opaka Sporting Goods | 6899 | 3 |

# Choose the columns you want to display

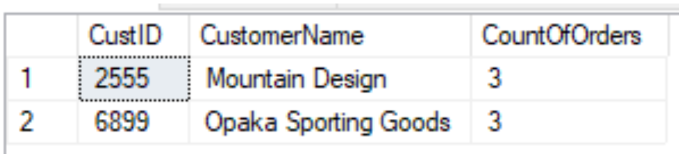
```
SELECT          vCount.CustID,
                cust.CustomerName,
                vCount.CountOfOrders
FROM            xCust cust
INNER JOIN      vCountOrders vCount
ON              cust.custid = vCount.custID
WHERE           countoforders =
                (SELECT MAX(CountOfOrders)
                 FROM   vCountOrders);
```



|   | CustID | CustomerName | CountOfOrders |
|---|---|---|---|
| 1 | 2555 | Mountain Design | 3 |
| 2 | 6899 | Opaka Sporting Goods | 3 |

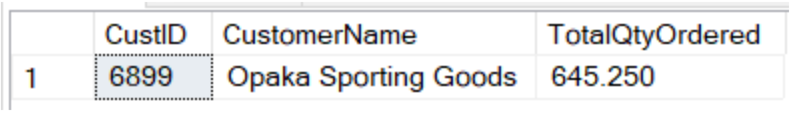
Must separate the group function of a COUNT from the group function of a MAX. Let’s use a CTE and a sub-query to accomplish the same goal as the VIEW in the prior slide.

```
WITH cteCountOrders AS
(SELECT        custID,
               count(*)  CountOfOrders
 FROM          xOrd
 GROUP BY      custID)

 SELECT        cust.CustID,
               CustomerName,
               CountOfOrders
FROM           xCust cust
INNER JOIN     cteCountOrders cteCount
ON             cust.custid = ctecount.custID
WHERE          countoforders =
               (SELECT MAX(CountOfOrders)
                FROM   cteCountOrders);
```



|  | CustID | CustomerName | CountOfOrders |
|---|---|---|---|
| 1 | 2555 | Mountain Design | 3 |
| 2 | 6899 | Opaka Sporting Goods | 3 |

# New query: Which customer bought the most items from us based on the quantity of items purchased?

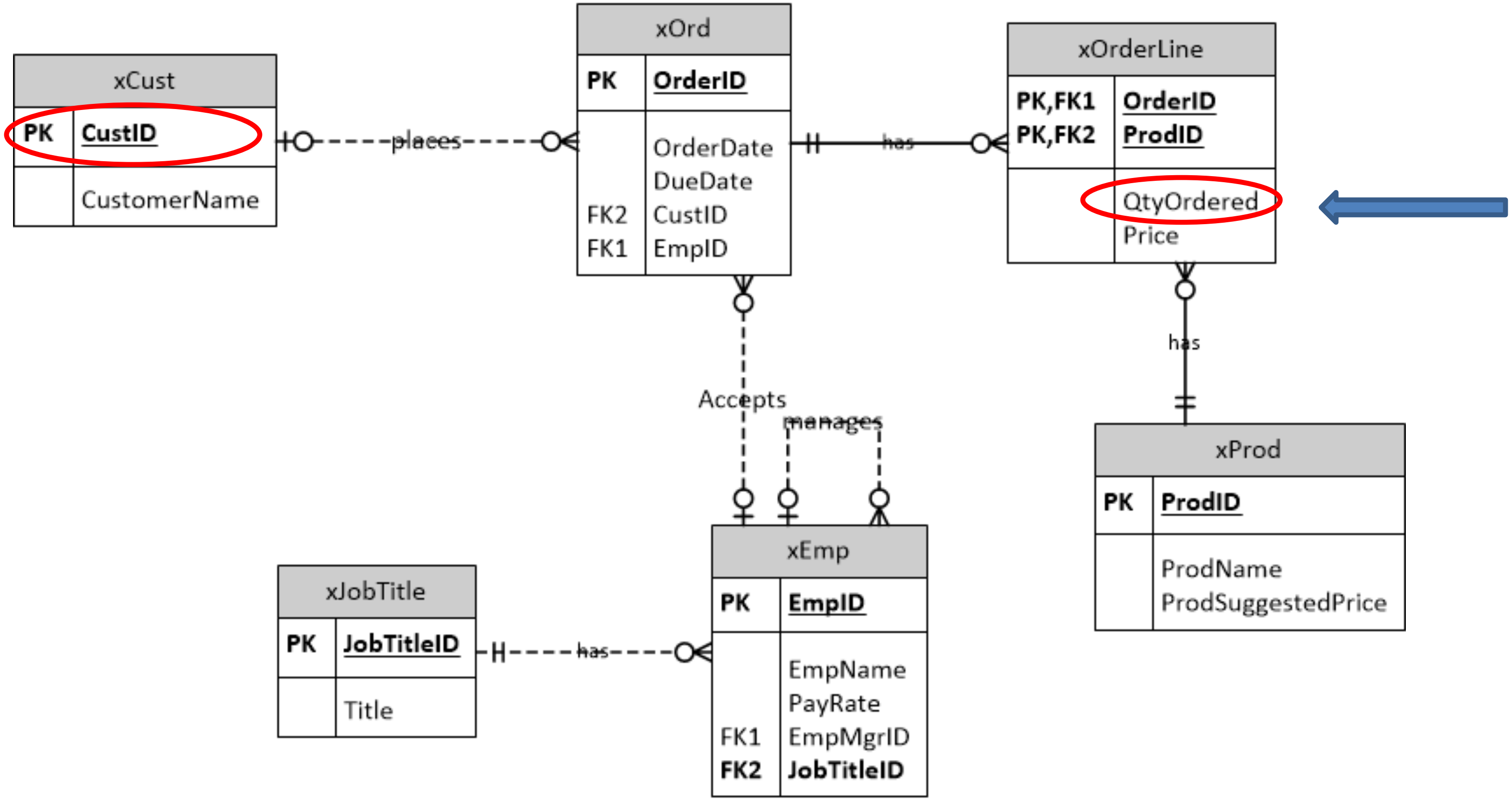


|  | CustID | CustomerName | TotalQtyOrdered |
|---|---|---|---|
| 1 | 6899 | Opaka Sporting Goods | 645.250 |

Where do the columns come from (which tables)?

What is the basic logic of the query (which table or tables are necessary to find the required rows in the result table)?

What are the simplest requirements necessary to accomplish the basic logic?



| xCust | |
|---|---|
| PK | CustID |
| | CustomerName |

places

| xOrd | |
|---|---|
| PK | OrderID |
| | OrderDate<br>DueDate |
| FK2 | CustID |
| FK1 | EmpID |

has

| xOrderLine | |
|---|---|
| PK,FK1 | OrderID |
| PK,FK2 | ProdID |
| | QtyOrdered<br>Price |

Accepts

manages

has

| xProd | |
|---|---|
| PK | ProdID |
| | ProdName<br>ProdSuggestedPrice |

| xJobTitle | |
|---|---|
| PK | JobTitleID |
| | Title |

has

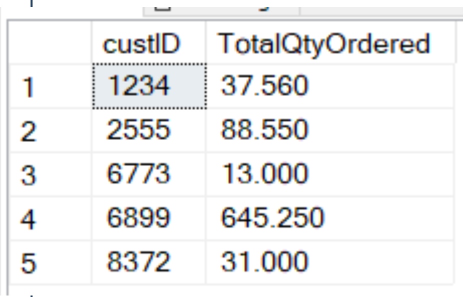
| xEmp | |
|---|---|
| PK | EmpID |
| | EmpName<br>PayRate |
| FK1 | EmpMgrID |
| FK2 | JobTitleID |

# Let’s explore the problem – what are we adding up?

```
Example of “playing with code” to get an idea of
the logic:

SELECT *
FROM xOrderline
INNER JOIN xOrd
ON xOrd.orderid = xOrderline.orderid
```

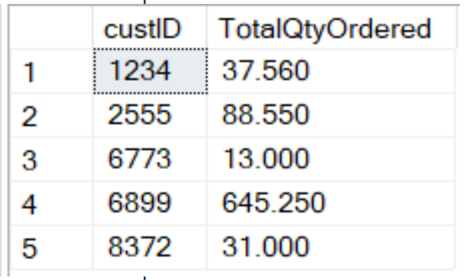
This requires the data stored in two different tables – the Orderline table for the qtyOrdered by product, and then the ord table to access the customer who purchased the product.

```
How do you sum up the qtyOrdered by customer?
SELECT custID,
        sum(qtyOrdered) TotalQtyOrdered
FROM xOrderline
INNER JOIN xOrd
ON xOrd.orderid = xOrderline.orderid
GROUP BY custid
```



| | custID | TotalQtyOrdered |
|---|---|---|
| 1 | 1234 | 37.560 |
| 2 | 2555 | 88.550 |
| 3 | 6773 | 13.000 |
| 4 | 6899 | 645.250 |
| 5 | 8372 | 31.000 |

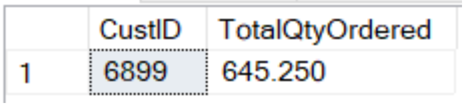
```
“Replace” the xOrderLine and xOrd tables with a VIEW

CREATE VIEW vCustQty AS
SELECT custID,
       sum(qtyOrdered) TotalQtyOrdered
FROM xOrderline
INNER JOIN xOrd
ON xOrd.orderid = xOrderline.orderid
GROUP BY custid
```



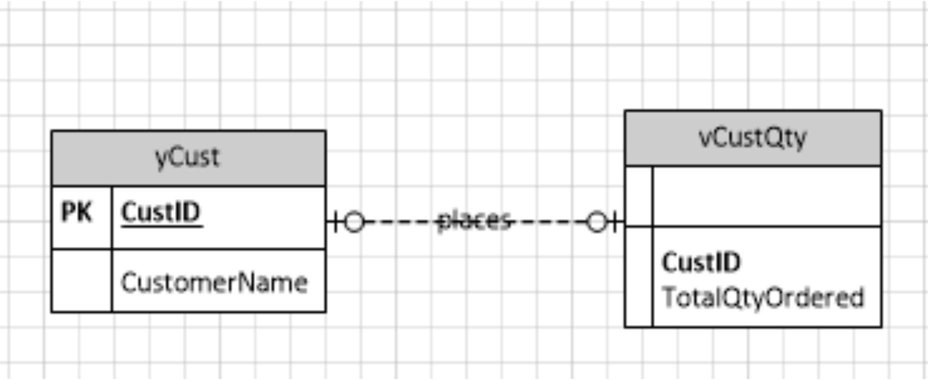
|   | custID | TotalQtyOrdered |
|---|---|---|
| 1 | 1234 | 37.560 |
| 2 | 2555 | 88.550 |
| 3 | 6773 | 13.000 |
| 4 | 6899 | 645.250 |
| 5 | 8372 | 31.000 |

Test out the basic logic

```
SELECT         *
FROM           vCustQty
WHERE          TotalqtyOrdered =
               (SELECT MAX(TotalqtyOrdered)
                FROM  vCustQty);
```



|   | CustID | TotalQtyOrdered |
|---|---|---|
| 1 | 6899 | 645.250 |



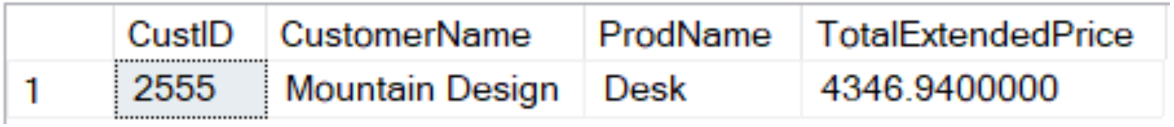
```
SELECT        vCustQty.CustID,
              cust.CustomerName,
              vCustQty.TotalqtyOrdered
FROM          xCust Cust
INNER JOIN    vCustQty
ON            cust.custid = vCustQty.custID
WHERE         TotalqtyOrdered =
              (SELECT MAX(TotalqtyOrdered)
               FROM   vCustQty);
```

Join the view with the Cust table to access the customer name

# Do the same thing, except with a CTE instead of a VIEW

```
WITH cteSumqtyOrdered AS
(SELECT custID,
       sum(qtyOrdered) TotalqtyOrdered
FROM xOrderline
INNER JOIN xOrd
ON xOrd.orderid = xOrderline.orderid
GROUP BY custid
)

 SELECT        ctesq.CustID,
               cust.CustomerName,
               ctesq.TotalqtyOrdered
FROM           xCust Cust
INNER JOIN cteSumqtyOrdered as ctesq
ON             cust.custid = ctesq.custID
WHERE          TotalqtyOrdered =
               (SELECT MAX(TotalqtyOrdered)
                FROM  ctesumqtyOrdered);
```

# New query:  Which customer spent the most for desks?

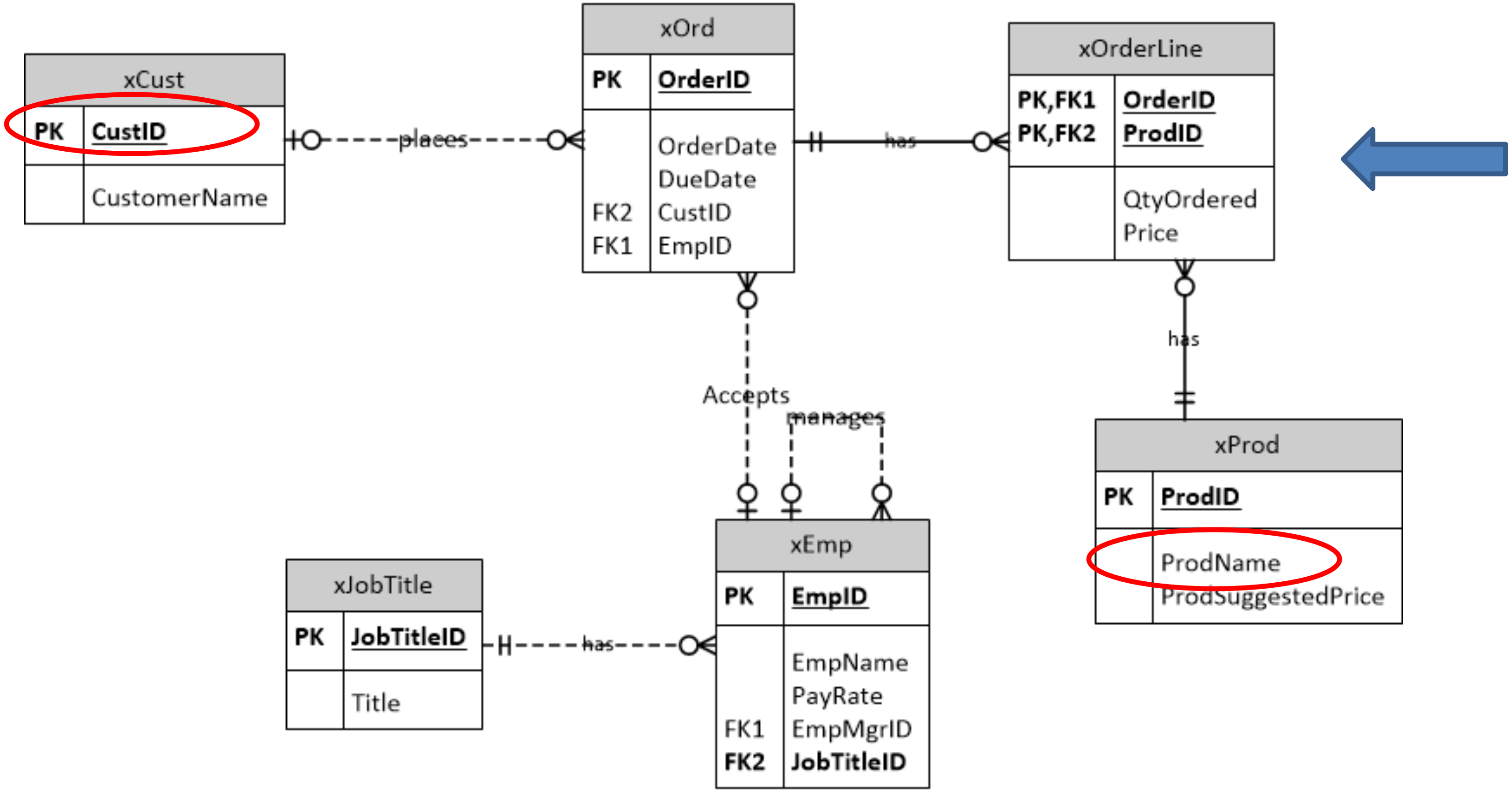


|   | CustID | CustomerName | ProdName | TotalExtendedPrice |
|---|---|---|---|---|
| 1 | 2555 | Mountain Design | Desk | 4346.9400000 |

Where do the columns come from (which tables)?

What is the basic logic of the query (which table or tables are necessary to find the required rows in the result table)?

What are the simplest requirements necessary to accomplish the basic logic?



**xCust**

| | |
|---|---|
| **PK** | **CustID** |
| | CustomerName |

**xOrd**

| | |
|---|---|
| **PK** | **OrderID** |
| | OrderDate<br>DueDate |
| FK2 | CustID |
| FK1 | EmpID |

**xOrderLine**

| | |
|---|---|
| **PK,FK1** | **OrderID** |
| **PK,FK2** | **ProdID** |
| | QtyOrdered<br>Price |

**xProd**

| | |
|---|---|
| **PK** | **ProdID** |
| | ProdName<br>ProdSuggestedPrice |

**xEmp**

| | |
|---|---|
| **PK** | **EmpID** |
| | EmpName<br>PayRate |
| FK1 | EmpMgrID |
| **FK2** | **JobTitleID** |

**xJobTitle**

| | |
|---|---|
| **PK** | **JobTitleID** |
| | Title |

# Write the basic logic in pseudocode

```
SELECT    customer data
FROM      cust, ord, orderline and prod
WHERE     SUM(qtyOrdered*price for desks by customer) =
                    MAX(SUM(qtyOrdered*price for desks by customer))
```

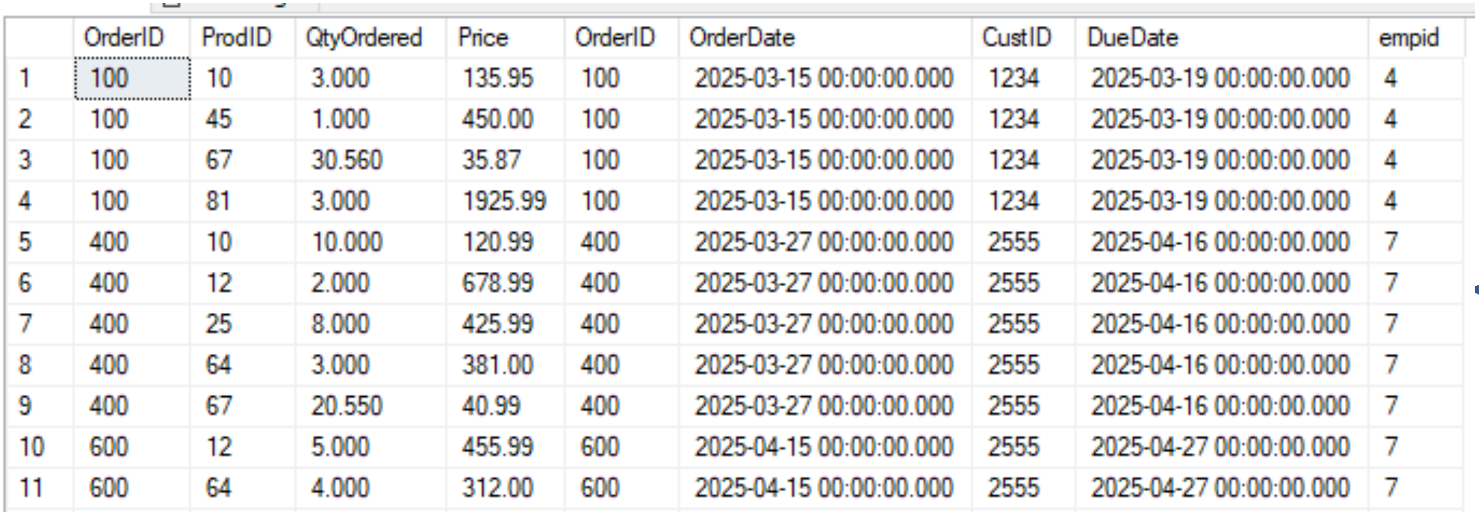
This is a “group of a group” – it is a MAX of a SUM

***This code isn’t designed to actually work as a SQL query.  It is just written to get an understanding of the basic logic necessary to accomplish the query.***

# Separate the two group functions – focus on the first GROUP function – the SUM of qtyOrdered*price for a product for a customer

```
SELECT *
FROM xOrderline
INNER JOIN xOrd
ON xOrderline.orderid = xOrd.orderid
ORDER BY custid
```

This requires the data stored in two different tables – the Orderline table for the qtyOrdered by product, and then the ord table to access the customer who purchased the product.



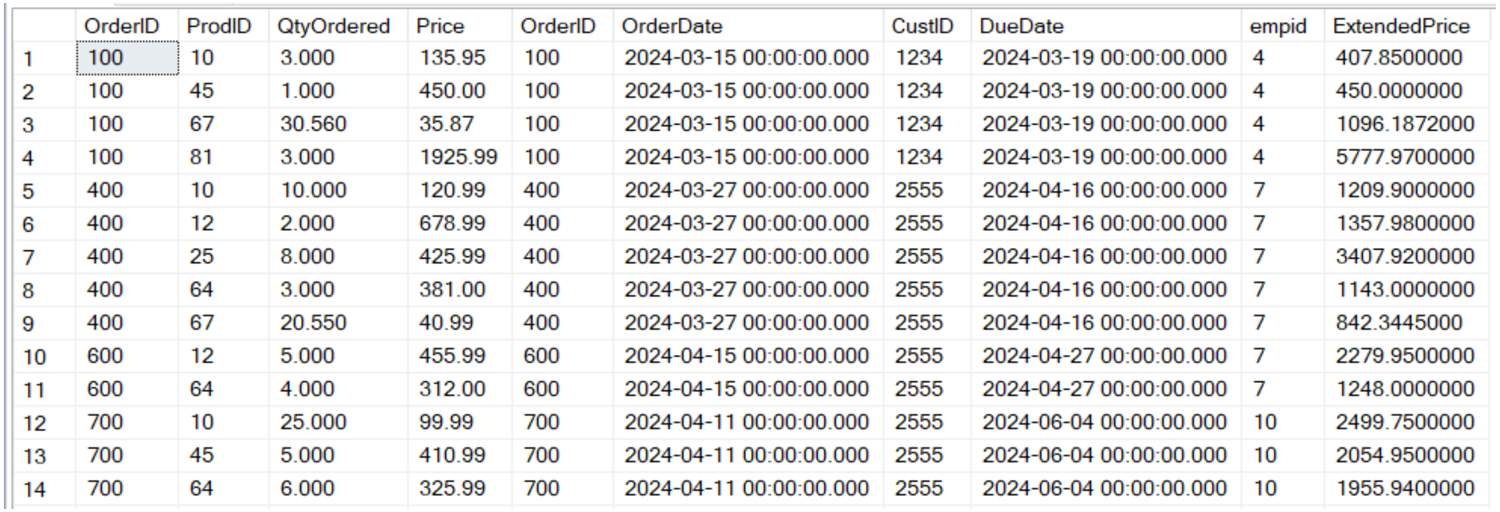
| | OrderID | ProdID | QtyOrdered | Price | OrderID | OrderDate | CustID | DueDate | empid |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 100 | 10 | 3.000 | 135.95 | 100 | 2025-03-15 00:00:00.000 | 1234 | 2025-03-19 00:00:00.000 | 4 |
| 2 | 100 | 45 | 1.000 | 450.00 | 100 | 2025-03-15 00:00:00.000 | 1234 | 2025-03-19 00:00:00.000 | 4 |
| 3 | 100 | 67 | 30.560 | 35.87 | 100 | 2025-03-15 00:00:00.000 | 1234 | 2025-03-19 00:00:00.000 | 4 |
| 4 | 100 | 81 | 3.000 | 1925.99 | 100 | 2025-03-15 00:00:00.000 | 1234 | 2025-03-19 00:00:00.000 | 4 |
| 5 | 400 | 10 | 10.000 | 120.99 | 400 | 2025-03-27 00:00:00.000 | 2555 | 2025-04-16 00:00:00.000 | 7 |
| 6 | 400 | 12 | 2.000 | 678.99 | 400 | 2025-03-27 00:00:00.000 | 2555 | 2025-04-16 00:00:00.000 | 7 |
| 7 | 400 | 25 | 8.000 | 425.99 | 400 | 2025-03-27 00:00:00.000 | 2555 | 2025-04-16 00:00:00.000 | 7 |
| 8 | 400 | 64 | 3.000 | 381.00 | 400 | 2025-03-27 00:00:00.000 | 2555 | 2025-04-16 00:00:00.000 | 7 |
| 9 | 400 | 67 | 20.550 | 40.99 | 400 | 2025-03-27 00:00:00.000 | 2555 | 2025-04-16 00:00:00.000 | 7 |
| 10 | 600 | 12 | 5.000 | 455.99 | 600 | 2025-04-15 00:00:00.000 | 2555 | 2025-04-27 00:00:00.000 | 7 |
| 11 | 600 | 64 | 4.000 | 312.00 | 600 | 2025-04-15 00:00:00.000 | 2555 | 2025-04-27 00:00:00.000 | 7 |

First 11 rows of the result table

# Let’s add a calculation to the SELECT list.

```
SELECT    *,
          qtyOrdered * price ExtendedPrice
FROM xOrderline
INNER JOIN xOrd
ON xOrderline.orderid = xOrd.orderid
ORDER BY custid
```

First 14 rows of the result table

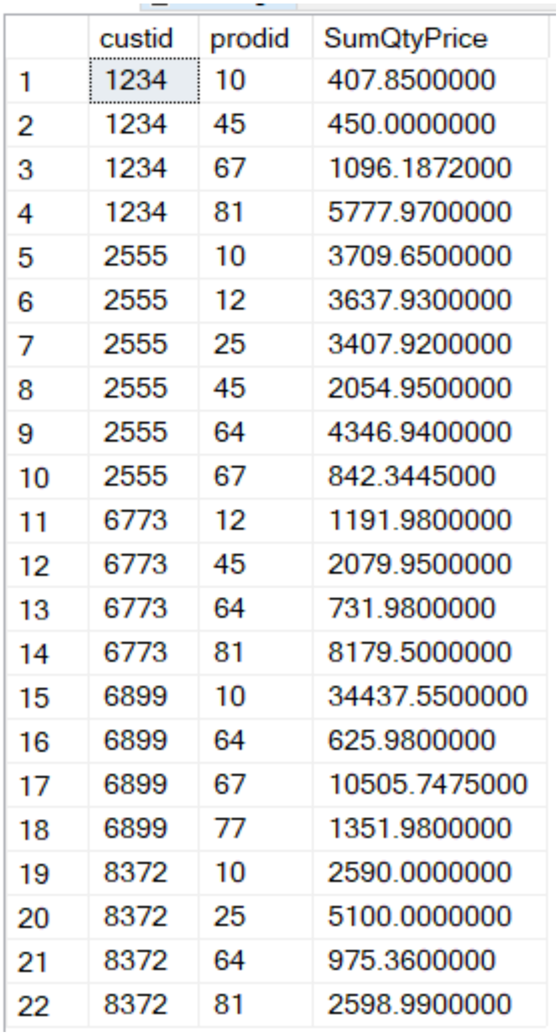


| | OrderID | ProdID | QtyOrdered | Price | OrderID | OrderDate | CustID | DueDate | empid | ExtendedPrice |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 100 | 10 | 3.000 | 135.95 | 100 | 2024-03-15 00:00:00.000 | 1234 | 2024-03-19 00:00:00.000 | 4 | 407.8500000 |
| 2 | 100 | 45 | 1.000 | 450.00 | 100 | 2024-03-15 00:00:00.000 | 1234 | 2024-03-19 00:00:00.000 | 4 | 450.0000000 |
| 3 | 100 | 67 | 30.560 | 35.87 | 100 | 2024-03-15 00:00:00.000 | 1234 | 2024-03-19 00:00:00.000 | 4 | 1096.1872000 |
| 4 | 100 | 81 | 3.000 | 1925.99 | 100 | 2024-03-15 00:00:00.000 | 1234 | 2024-03-19 00:00:00.000 | 4 | 5777.9700000 |
| 5 | 400 | 10 | 10.000 | 120.99 | 400 | 2024-03-27 00:00:00.000 | 2555 | 2024-04-16 00:00:00.000 | 7 | 1209.9000000 |
| 6 | 400 | 12 | 2.000 | 678.99 | 400 | 2024-03-27 00:00:00.000 | 2555 | 2024-04-16 00:00:00.000 | 7 | 1357.9800000 |
| 7 | 400 | 25 | 8.000 | 425.99 | 400 | 2024-03-27 00:00:00.000 | 2555 | 2024-04-16 00:00:00.000 | 7 | 3407.9200000 |
| 8 | 400 | 64 | 3.000 | 381.00 | 400 | 2024-03-27 00:00:00.000 | 2555 | 2024-04-16 00:00:00.000 | 7 | 1143.0000000 |
| 9 | 400 | 67 | 20.550 | 40.99 | 400 | 2024-03-27 00:00:00.000 | 2555 | 2024-04-16 00:00:00.000 | 7 | 842.3445000 |
| 10 | 600 | 12 | 5.000 | 455.99 | 600 | 2024-04-15 00:00:00.000 | 2555 | 2024-04-27 00:00:00.000 | 7 | 2279.9500000 |
| 11 | 600 | 64 | 4.000 | 312.00 | 600 | 2024-04-15 00:00:00.000 | 2555 | 2024-04-27 00:00:00.000 | 7 | 1248.0000000 |
| 12 | 700 | 10 | 25.000 | 99.99 | 700 | 2024-04-11 00:00:00.000 | 2555 | 2024-06-04 00:00:00.000 | 10 | 2499.7500000 |
| 13 | 700 | 45 | 5.000 | 410.99 | 700 | 2024-04-11 00:00:00.000 | 2555 | 2024-06-04 00:00:00.000 | 10 | 2054.9500000 |
| 14 | 700 | 64 | 6.000 | 325.99 | 700 | 2024-04-11 00:00:00.000 | 2555 | 2024-06-04 00:00:00.000 | 10 | 1955.9400000 |

Must now decide what to group on to create the SUM of the ExtendedPrice

# Now group it!

```sql
SELECT  custid,
        prodid,
        SUM(qtyOrdered*Price) SumQtyPrice
FROM xOrderline
INNER JOIN xOrd
ON xOrderline.orderid = xOrd.orderid
GROUP BY custid, prodid
ORDER BY custid, prodID
```



| | custid | prodid | SumQtyPrice |
|---|---|---|---|
| 1 | 1234 | 10 | 407.8500000 |
| 2 | 1234 | 45 | 450.0000000 |
| 3 | 1234 | 67 | 1096.1872000 |
| 4 | 1234 | 81 | 5777.9700000 |
| 5 | 2555 | 10 | 3709.6500000 |
| 6 | 2555 | 12 | 3637.9300000 |
| 7 | 2555 | 25 | 3407.9200000 |
| 8 | 2555 | 45 | 2054.9500000 |
| 9 | 2555 | 64 | 4346.9400000 |
| 10 | 2555 | 67 | 842.3445000 |
| 11 | 6773 | 12 | 1191.9800000 |
| 12 | 6773 | 45 | 2079.9500000 |
| 13 | 6773 | 64 | 731.9800000 |
| 14 | 6773 | 81 | 8179.5000000 |
| 15 | 6899 | 10 | 34437.5500000 |
| 16 | 6899 | 64 | 625.9800000 |
| 17 | 6899 | 67 | 10505.7475000 |
| 18 | 6899 | 77 | 1351.9800000 |
| 19 | 8372 | 10 | 2590.0000000 |
| 20 | 8372 | 25 | 5100.0000000 |
| 21 | 8372 | 64 | 975.3600000 |
| 22 | 8372 | 81 | 2598.9900000 |

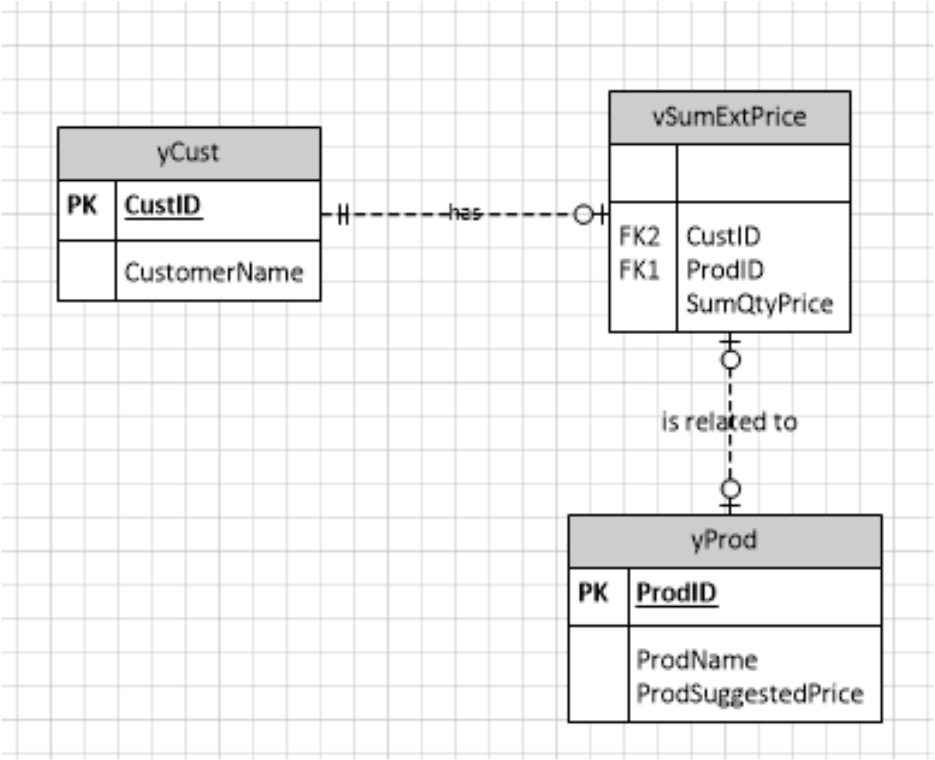
We don’t know which prodID represents a desk yet, but that is OK.  We now know which customer bought what product and the extended price for that product.  This is assuming that a customer is capable of buying the same product more than once – which is likely quite true!

Let’s make a view out of that code so that we don’t have to deal with the GROUP function SUM in other queries that need to use the total data.

```sql
CREATE VIEW vSumExtPrice AS
SELECT  custid,
        prodid,
        SUM(qtyOrdered*Price) SumQtyPrice
FROM xOrderline
INNER JOIN xOrd
ON xOrderline.orderid = xOrd.orderid
GROUP BY custid, prodid
```

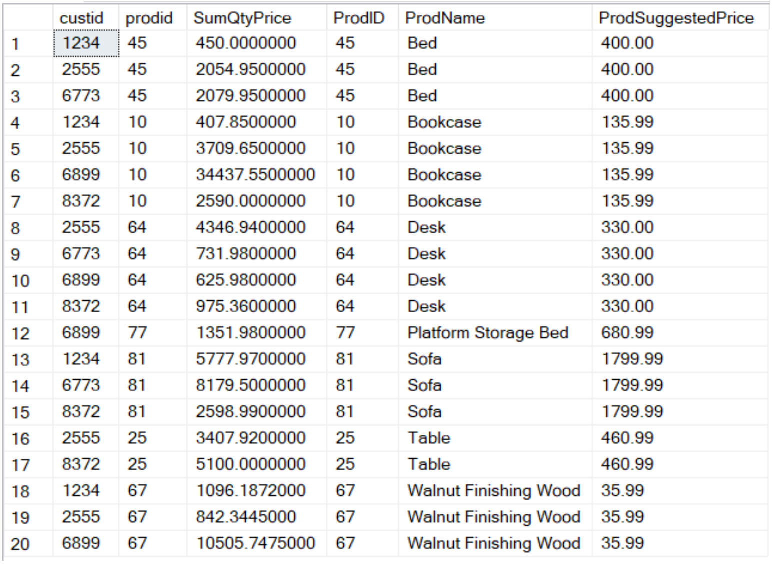
Notice that the ORDER BY statement is gone. A view cannot include an ORDER BY statement.

# The view relates to both the Prod and Cust tables because it contains two potential FK's – A CustID and a ProdID



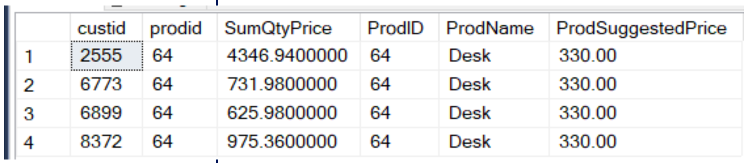
Look at the orders placed for a product name of a desk. The product name is stored in the xProd table, so let's join those two tables to get an idea of what data would be available.

```
SELECT  *
FROM    vsumextprice
INNER JOIN xProd
ON vsumextprice.prodid = xProd.prodid
ORDER BY 5
```



| | custid | prodid | SumQtyPrice | ProdID | ProdName | ProdSuggestedPrice |
|---|---|---|---|---|---|---|
| 1 | 1234 | 45 | 450.0000000 | 45 | Bed | 400.00 |
| 2 | 2555 | 45 | 2054.9500000 | 45 | Bed | 400.00 |
| 3 | 6773 | 45 | 2079.9500000 | 45 | Bed | 400.00 |
| 4 | 1234 | 10 | 407.8500000 | 10 | Bookcase | 135.99 |
| 5 | 2555 | 10 | 3709.6500000 | 10 | Bookcase | 135.99 |
| 6 | 6899 | 10 | 34437.5500000 | 10 | Bookcase | 135.99 |
| 7 | 8372 | 10 | 2590.0000000 | 10 | Bookcase | 135.99 |
| 8 | 2555 | 64 | 4346.9400000 | 64 | Desk | 330.00 |
| 9 | 6773 | 64 | 731.9800000 | 64 | Desk | 330.00 |
| 10 | 6899 | 64 | 625.9800000 | 64 | Desk | 330.00 |
| 11 | 8372 | 64 | 975.3600000 | 64 | Desk | 330.00 |
| 12 | 6899 | 77 | 1351.9800000 | 77 | Platform Storage Bed | 680.99 |
| 13 | 1234 | 81 | 5777.9700000 | 81 | Sofa | 1799.99 |
| 14 | 6773 | 81 | 8179.5000000 | 81 | Sofa | 1799.99 |
| 15 | 8372 | 81 | 2598.9900000 | 81 | Sofa | 1799.99 |
| 16 | 2555 | 25 | 3407.9200000 | 25 | Table | 460.99 |
| 17 | 8372 | 25 | 5100.0000000 | 25 | Table | 460.99 |
| 18 | 1234 | 67 | 1096.1872000 | 67 | Walnut Finishing Wood | 35.99 |
| 19 | 2555 | 67 | 842.3445000 | 67 | Walnut Finishing Wood | 35.99 |
| 20 | 6899 | 67 | 10505.7475000 | 67 | Walnut Finishing Wood | 35.99 |

```
SELECT  *
FROM    vsumextprice
INNER JOIN xProd
ON vsumextprice.prodid = xProd.prodid
WHERE prodname = 'desk'
```



| | custid | prodid | SumQtyPrice | ProdID | ProdName | ProdSuggestedPrice |
|---|---|---|---|---|---|---|
| 1 | 2555 | 64 | 4346.9400000 | 64 | Desk | 330.00 |
| 2 | 6773 | 64 | 731.9800000 | 64 | Desk | 330.00 |
| 3 | 6899 | 64 | 625.9800000 | 64 | Desk | 330.00 |
| 4 | 8372 | 64 | 975.3600000 | 64 | Desk | 330.00 |

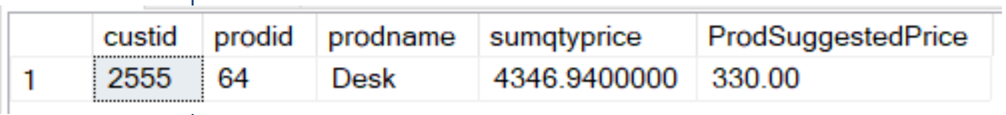
Turn the query into a view.  We are creating a view of a table joined with a view.

Will need to specify the columns for the View because it isn’t possible to create a view when the columns have the same name (like ProdID in the query on the previous page) – so can’t use the asterisk to declare the columns for a view.

```
create view vDesk as
SELECT custid,
       yprod.prodid,
       prodname,
       sumqtyprice,
       ProdSuggestedPrice
FROM   vsumextprice
INNER JOIN xProd
ON vsumextprice.prodid = xProd.prodid
WHERE prodname = 'desk'
```

# Find the correct row or rows for the goal of the query

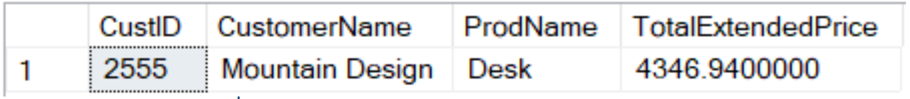
```
SELECT *
FROM vDesk
WHERE sumqtyprice =
      (SELECT MAX(sumqtyprice)
        FROM vDesk)
```



|   | custid | prodid | prodname | sumqtyprice | ProdSuggestedPrice |
|---|---|---|---|---|---|
| 1 | 2555 | 64 | Desk | 4346.9400000 | 330.00 |

# Join the table(s) and additional columns desired for the result table.

```
SELECT  vDesk.CustID,
        CustomerName,
         ProdName,
         SumQtyPrice TotalExtendedPrice
FROM vDesk
INNER JOIN xCust
ON vDesk.custID = xCust.custid
WHERE sumqtyprice =
      (SELECT MAX(sumqtyprice)
       FROM vDesk)
```



|   | CustID | CustomerName | ProdName | TotalExtendedPrice |
|---|---|---|---|---|
| 1 | 2555 | Mountain Design | Desk | 4346.9400000 |

```sql
WITH cteSumExtPrice AS
(SELECT  custid,
         prodid,
        SUM(qtyOrdered*Price) SumQtyPrice
FROM xOrderline
INNER JOIN xOrd
ON xOrderline.orderid = xOrd.orderid
GROUP BY custid, prodid),

cteDesk as
(SELECT  custid,
         yprod.prodid,
         prodname,
         sumqtyprice,
         ProdSuggestedPrice
FROM     cteSumExtPrice
INNER JOIN xProd
ON ctesumextprice.prodid = xProd.prodid
WHERE prodname = 'desk')

SELECT   cteDesk.CustID,
         CustomerName,
         ProdName,
         SumQtyPrice TotalExtendedPrice
FROM cteDesk
INNER JOIN xCust
ON cteDesk.custID = xCust.custid
WHERE  sumqtyprice =
      (SELECT MAX(sumqtyprice)
        FROM cteDesk)
```

Can accomplish the same goal with CTE

```sql
WITH cteSumExtPrice AS
(SELECT  custid,
        xOrderline.prodid,
        prodname,
     SUM(qtyOrdered*price) AS SumQtyPrice
FROM xOrderline
INNER JOIN xOrd
ON xOrderline.orderid = xOrd.orderid
INNER JOIN xProd
ON xOrderline.prodid = xProd.prodid
WHERE prodname = 'desk'
GROUP BY custid, xOrderline.prodid, prodname)

SELECT  cteSumExtPrice.CustID,
        CustomerName,
        ProdName,
        SumQtyPrice TotalExtendedPrice
FROM cteSumExtPrice
INNER JOIN xCust
ON cteSumExtPrice.custID = xCust.custid
WHERE  sumqtyprice =
      (SELECT MAX(sumqtyprice)
        FROM cteSumExtPrice)
```

Can accomplish the same goal with a single CTE

Or a single VIEW…

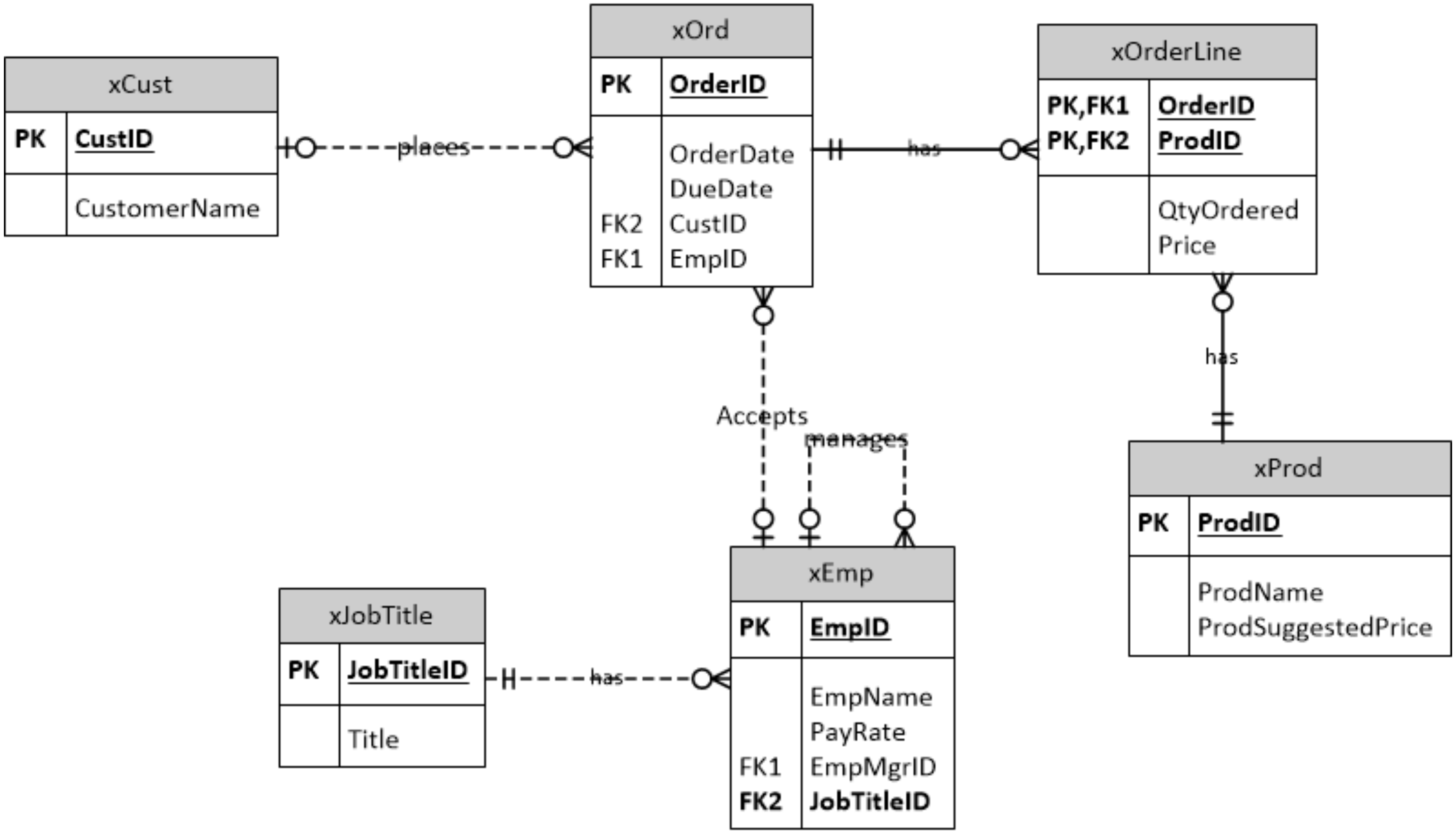
# Why bother using Views or CTEs?

- Group functions and joins are complex.
- GROUP BY should only be used with group functions (AVG, MAX, MIN, SUM, COUNT).
- Should not use a GROUP BY just to suppress rows in the result set. Use the GROUP BY only with a group function!
- Must have all non-group attributes that are in the SELECT list also in the GROUP BY statement.
- Difficult to do a group function of a group function. Examples:
  - The maximum of the sum of hours.
  - The minimum of a count of products.
- Joining multiple tables can yield full or partial cross joins making it difficult to trouble-shoot the SQL code.

# IS475/675 Agenda:  April 16, 2025

- Complete one more example of a “group of a group.”
- Do a bulk load of data to copy a table.
- Connect SQL Server to Excel (if we have time).
- Answer any questions.

# Tables used in this example – from SQL Lab Exercise 8



xCust
PK CustID
CustomerName

xOrd
PK OrderID
OrderDate
DueDate
FK2 CustID
FK1 EmpID

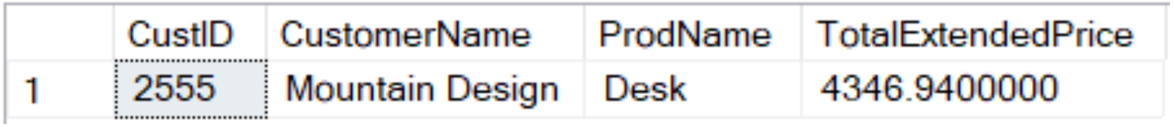
xOrderLine
PK,FK1 OrderID
PK,FK2 ProdID
QtyOrdered
Price

xProd
PK ProdID
ProdName
ProdSuggestedPrice

xEmp
PK EmpID
EmpName
PayRate
FK1 EmpMgrID
FK2 JobTitleID

xJobTitle
PK JobTitleID
Title

# New query:  Which customer spent the most for desks?

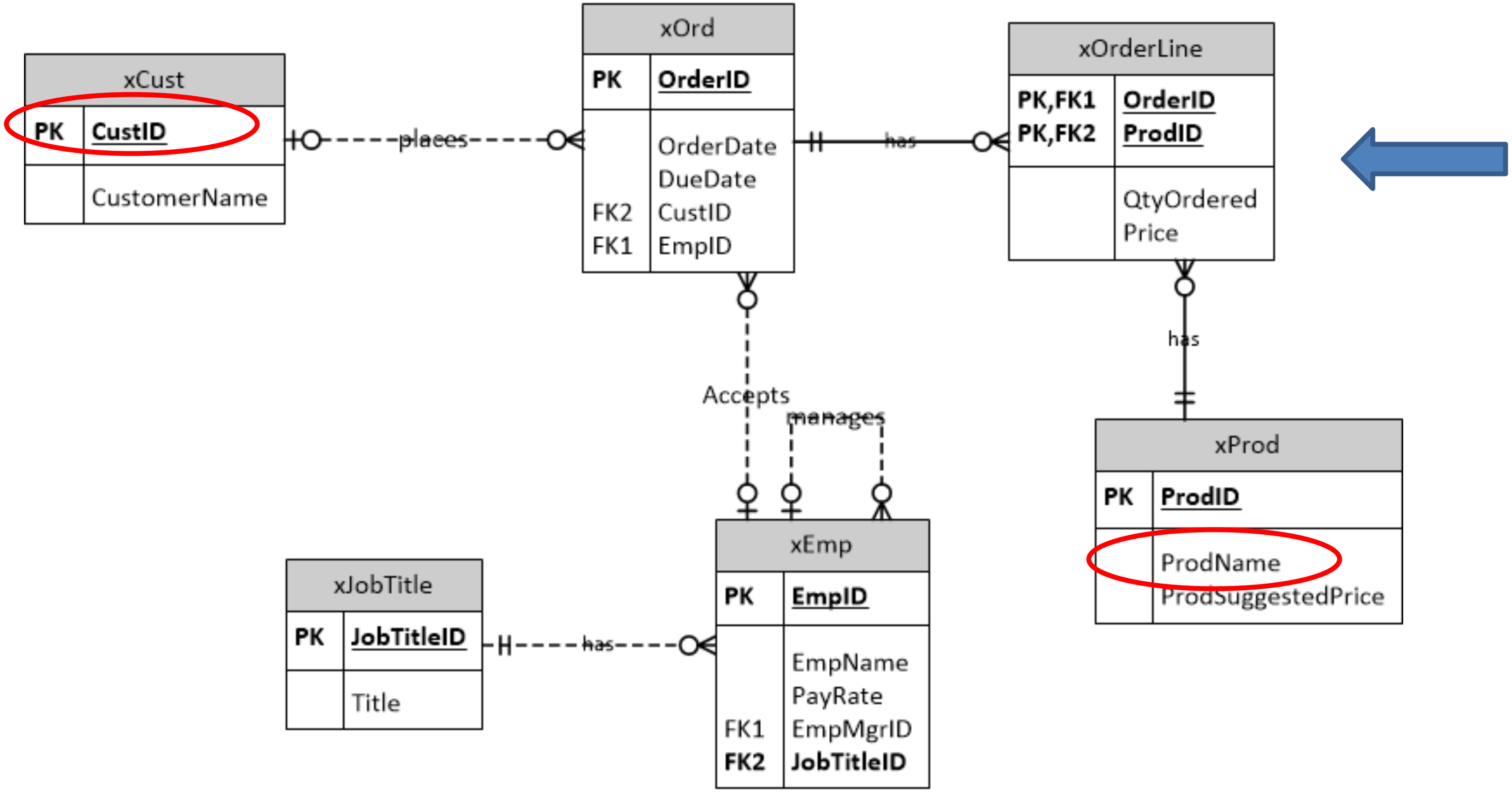


|  | CustID | CustomerName | ProdName | TotalExtendedPrice |
|---|---|---|---|---|
| 1 | 2555 | Mountain Design | Desk | 4346.9400000 |

Where do the columns come from (which tables)?

What is the basic logic of the query (which table or tables are necessary to find the required rows in the result table)?

What are the simplest requirements necessary to accomplish the basic logic?



| xCust | |
|---|---|
| PK | **CustID** |
| | CustomerName |

| xOrd | |
|---|---|
| PK | **OrderID** |
| | OrderDate<br>DueDate |
| FK2 | CustID |
| FK1 | EmpID |

| xOrderLine | |
|---|---|
| PK,FK1 | **OrderID** |
| PK,FK2 | **ProdID** |
| | QtyOrdered<br>Price |

| xProd | |
|---|---|
| PK | **ProdID** |
| | ProdName<br>ProdSuggestedPrice |

| xEmp | |
|---|---|
| PK | **EmpID** |
| | EmpName<br>PayRate |
| FK1 | EmpMgrID |
| **FK2** | **JobTitleID** |

| xJobTitle | |
|---|---|
| PK | **JobTitleID** |
| | Title |

# Write the basic logic in pseudocode

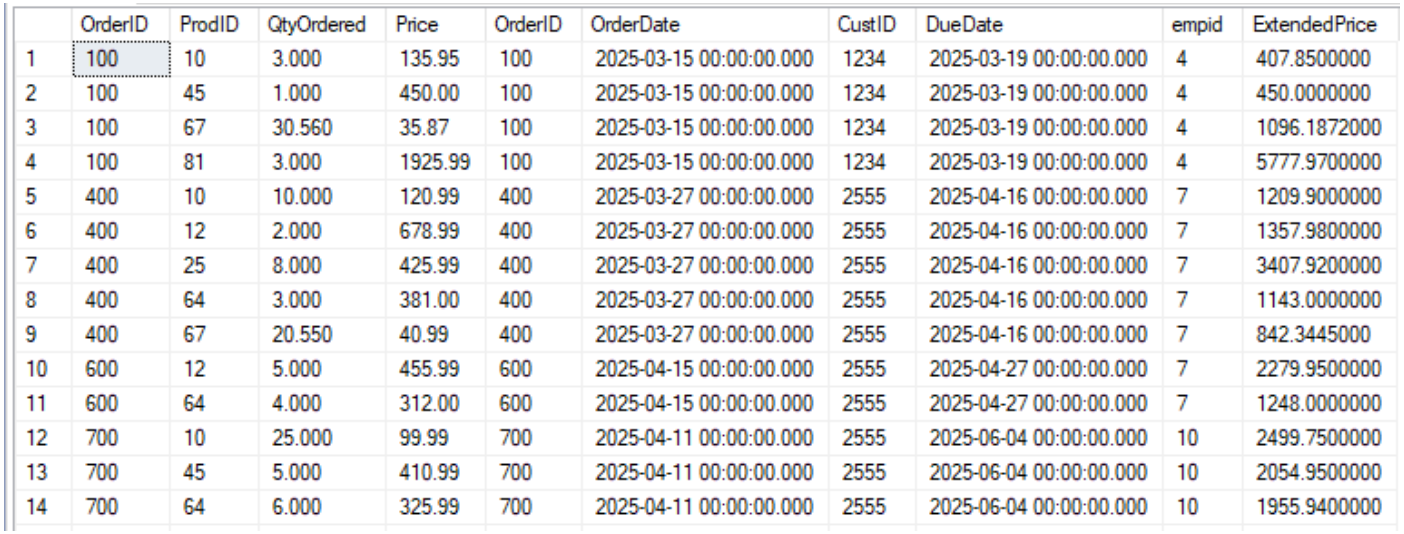
```
SELECT    customer data
FROM      cust, ord, orderline and prod
WHERE     SUM(qtyOrdered*price for desks by customer) =
                    MAX(SUM(qtyOrdered*price for desks by customer))
```

This is a “group of a group” – it is a MAX of a SUM

***This code isn’t designed to actually work as a SQL query.  It is just written to get an understanding of the basic logic necessary to accomplish the query.***

# Let’s add a calculation to the SELECT list.

```
SELECT   *,
         qtyOrdered * price ExtendedPrice
FROM xOrderline
INNER JOIN xOrd
ON xOrderline.orderid = xOrd.orderid
ORDER BY custid
```



| | OrderID | ProdID | QtyOrdered | Price | OrderID | OrderDate | CustID | DueDate | empid | ExtendedPrice |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 100 | 10 | 3.000 | 135.95 | 100 | 2025-03-15 00:00:00.000 | 1234 | 2025-03-19 00:00:00.000 | 4 | 407.8500000 |
| 2 | 100 | 45 | 1.000 | 450.00 | 100 | 2025-03-15 00:00:00.000 | 1234 | 2025-03-19 00:00:00.000 | 4 | 450.0000000 |
| 3 | 100 | 67 | 30.560 | 35.87 | 100 | 2025-03-15 00:00:00.000 | 1234 | 2025-03-19 00:00:00.000 | 4 | 1096.1872000 |
| 4 | 100 | 81 | 3.000 | 1925.99 | 100 | 2025-03-15 00:00:00.000 | 1234 | 2025-03-19 00:00:00.000 | 4 | 5777.9700000 |
| 5 | 400 | 10 | 10.000 | 120.99 | 400 | 2025-03-27 00:00:00.000 | 2555 | 2025-04-16 00:00:00.000 | 7 | 1209.9000000 |
| 6 | 400 | 12 | 2.000 | 678.99 | 400 | 2025-03-27 00:00:00.000 | 2555 | 2025-04-16 00:00:00.000 | 7 | 1357.9800000 |
| 7 | 400 | 25 | 8.000 | 425.99 | 400 | 2025-03-27 00:00:00.000 | 2555 | 2025-04-16 00:00:00.000 | 7 | 3407.9200000 |
| 8 | 400 | 64 | 3.000 | 381.00 | 400 | 2025-03-27 00:00:00.000 | 2555 | 2025-04-16 00:00:00.000 | 7 | 1143.0000000 |
| 9 | 400 | 67 | 20.550 | 40.99 | 400 | 2025-03-27 00:00:00.000 | 2555 | 2025-04-16 00:00:00.000 | 7 | 842.3445000 |
| 10 | 600 | 12 | 5.000 | 455.99 | 600 | 2025-04-15 00:00:00.000 | 2555 | 2025-04-27 00:00:00.000 | 7 | 2279.9500000 |
| 11 | 600 | 64 | 4.000 | 312.00 | 600 | 2025-04-15 00:00:00.000 | 2555 | 2025-04-27 00:00:00.000 | 7 | 1248.0000000 |
| 12 | 700 | 10 | 25.000 | 99.99 | 700 | 2025-04-11 00:00:00.000 | 2555 | 2025-06-04 00:00:00.000 | 10 | 2499.7500000 |
| 13 | 700 | 45 | 5.000 | 410.99 | 700 | 2025-04-11 00:00:00.000 | 2555 | 2025-06-04 00:00:00.000 | 10 | 2054.9500000 |
| 14 | 700 | 64 | 6.000 | 325.99 | 700 | 2025-04-11 00:00:00.000 | 2555 | 2025-06-04 00:00:00.000 | 10 | 1955.9400000 |

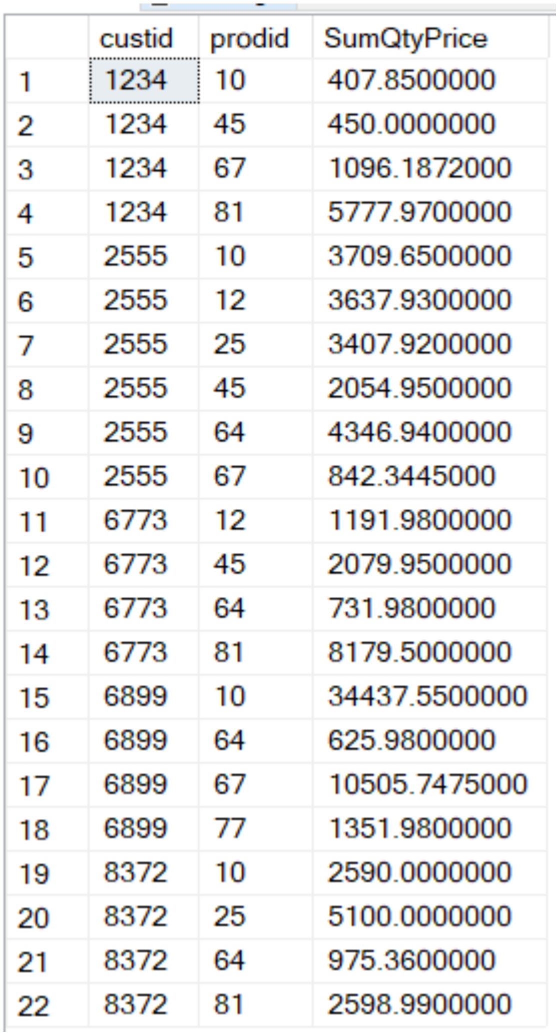
First 14 rows of the result table

Must now decide what to group on to create the SUM of the ExtendedPrice

# Now group it!

```
SELECT  custid,
        prodid,
        SUM(qtyOrdered*Price) SumQtyPrice
FROM xOrderline
INNER JOIN xOrd
ON xOrderline.orderid = xOrd.orderid
GROUP BY custid, prodid
ORDER BY custid, prodID
```

We don’t know which prodID represents a desk yet, but that is OK. We now know which customer bought what product and the extended price for that product. This is assuming that a customer is capable of buying the same product more than once – which is likely quite true!



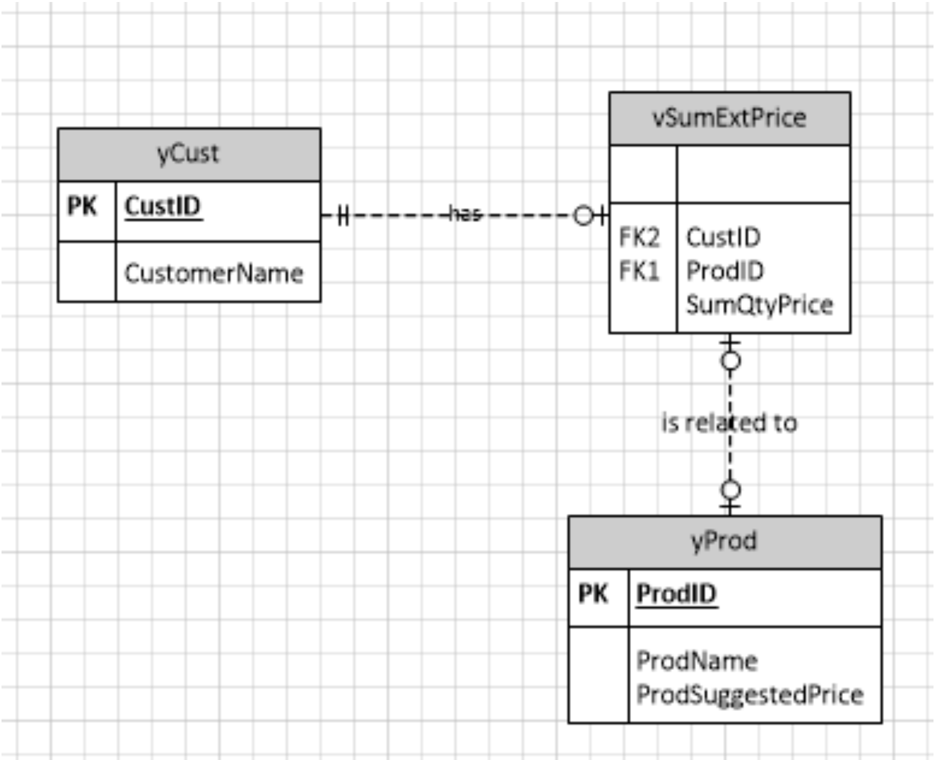
| | custid | prodid | SumQtyPrice |
|---|---|---|---|
| 1 | 1234 | 10 | 407.8500000 |
| 2 | 1234 | 45 | 450.0000000 |
| 3 | 1234 | 67 | 1096.1872000 |
| 4 | 1234 | 81 | 5777.9700000 |
| 5 | 2555 | 10 | 3709.6500000 |
| 6 | 2555 | 12 | 3637.9300000 |
| 7 | 2555 | 25 | 3407.9200000 |
| 8 | 2555 | 45 | 2054.9500000 |
| 9 | 2555 | 64 | 4346.9400000 |
| 10 | 2555 | 67 | 842.3445000 |
| 11 | 6773 | 12 | 1191.9800000 |
| 12 | 6773 | 45 | 2079.9500000 |
| 13 | 6773 | 64 | 731.9800000 |
| 14 | 6773 | 81 | 8179.5000000 |
| 15 | 6899 | 10 | 34437.5500000 |
| 16 | 6899 | 64 | 625.9800000 |
| 17 | 6899 | 67 | 10505.7475000 |
| 18 | 6899 | 77 | 1351.9800000 |
| 19 | 8372 | 10 | 2590.0000000 |
| 20 | 8372 | 25 | 5100.0000000 |
| 21 | 8372 | 64 | 975.3600000 |
| 22 | 8372 | 81 | 2598.9900000 |

Let’s make a view out of that code so that we don’t have to deal with the GROUP function SUM in other queries that need to use the total data.

```
CREATE VIEW vSumExtPrice AS
SELECT  custid,
        prodid,
        SUM(qtyOrdered*Price) SumQtyPrice
FROM xOrderline
INNER JOIN xOrd
ON xOrderline.orderid = xOrd.orderid
GROUP BY custid, prodid
```

Notice that the ORDER BY statement is gone. A view cannot include an ORDER BY statement.

# The view relates to both the Prod and Cust tables because it contains two potential FK's – A CustID and a ProdID



yCust

PK CustID

CustomerName

has

vSumExtPrice

FK2 CustID
FK1 ProdID
SumQtyPrice

is related to

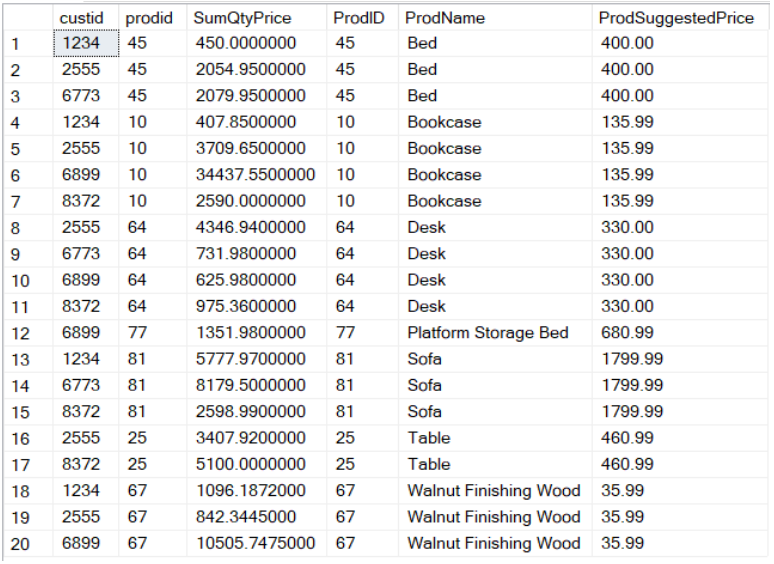
yProd

PK ProdID

ProdName
ProdSuggestedPrice

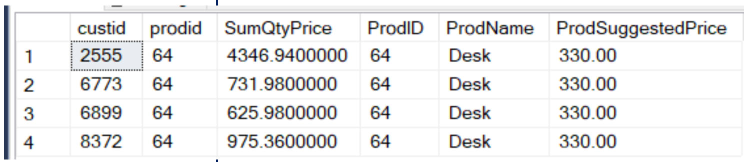
Look at the orders placed for a product name of a desk.  The product name is stored in the xProd table, so let's join those two tables to get an idea of what data would be available.

```
SELECT  *
FROM    vsumextprice
INNER JOIN xProd
ON vsumextprice.prodid = xProd.prodid
ORDER BY 5
```



| | custid | prodid | SumQtyPrice | ProdID | ProdName | ProdSuggestedPrice |
|---|---|---|---|---|---|---|
| 1 | 1234 | 45 | 450.0000000 | 45 | Bed | 400.00 |
| 2 | 2555 | 45 | 2054.9500000 | 45 | Bed | 400.00 |
| 3 | 6773 | 45 | 2079.9500000 | 45 | Bed | 400.00 |
| 4 | 1234 | 10 | 407.8500000 | 10 | Bookcase | 135.99 |
| 5 | 2555 | 10 | 3709.6500000 | 10 | Bookcase | 135.99 |
| 6 | 6899 | 10 | 34437.5500000 | 10 | Bookcase | 135.99 |
| 7 | 8372 | 10 | 2590.0000000 | 10 | Bookcase | 135.99 |
| 8 | 2555 | 64 | 4346.9400000 | 64 | Desk | 330.00 |
| 9 | 6773 | 64 | 731.9800000 | 64 | Desk | 330.00 |
| 10 | 6899 | 64 | 625.9800000 | 64 | Desk | 330.00 |
| 11 | 8372 | 64 | 975.3600000 | 64 | Desk | 330.00 |
| 12 | 6899 | 77 | 1351.9800000 | 77 | Platform Storage Bed | 680.99 |
| 13 | 1234 | 81 | 5777.9700000 | 81 | Sofa | 1799.99 |
| 14 | 6773 | 81 | 8179.5000000 | 81 | Sofa | 1799.99 |
| 15 | 8372 | 81 | 2598.9900000 | 81 | Sofa | 1799.99 |
| 16 | 2555 | 25 | 3407.9200000 | 25 | Table | 460.99 |
| 17 | 8372 | 25 | 5100.0000000 | 25 | Table | 460.99 |
| 18 | 1234 | 67 | 1096.1872000 | 67 | Walnut Finishing Wood | 35.99 |
| 19 | 2555 | 67 | 842.3445000 | 67 | Walnut Finishing Wood | 35.99 |
| 20 | 6899 | 67 | 10505.7475000 | 67 | Walnut Finishing Wood | 35.99 |

```
SELECT  *
FROM    vsumextprice
INNER JOIN xProd
ON vsumextprice.prodid = xProd.prodid
WHERE prodname = 'desk'
```



| | custid | prodid | SumQtyPrice | ProdID | ProdName | ProdSuggestedPrice |
|---|---|---|---|---|---|---|
| 1 | 2555 | 64 | 4346.9400000 | 64 | Desk | 330.00 |
| 2 | 6773 | 64 | 731.9800000 | 64 | Desk | 330.00 |
| 3 | 6899 | 64 | 625.9800000 | 64 | Desk | 330.00 |
| 4 | 8372 | 64 | 975.3600000 | 64 | Desk | 330.00 |

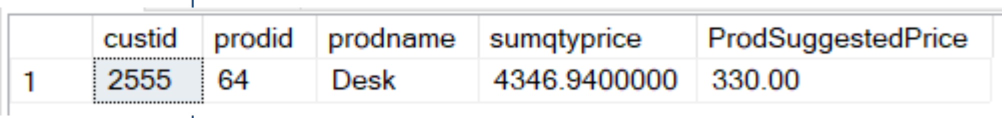
Turn the query into a view. We are creating a view of a table joined with a view.

Will need to specify the columns for the View because it isn't possible to create a view when the columns have the same name (like ProdID in the query on the previous page) – so can't use the asterisk to declare the columns for a view.

```
create view vDesk as
SELECT custid,
       xprod.prodid,
       prodname,
       sumqtyprice,
       ProdSuggestedPrice
FROM   vsumextprice
INNER JOIN xProd
ON vsumextprice.prodid = xProd.prodid
WHERE prodname = 'desk'
```

## Find the correct row or rows for the goal of the query

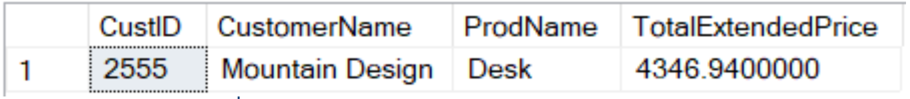
```
SELECT *
FROM vDesk
WHERE sumqtyprice =
      (SELECT MAX(sumqtyprice)
        FROM vDesk)
```



| | custid | prodid | prodname | sumqtyprice | ProdSuggestedPrice |
|---|---|---|---|---|---|
| 1 | 2555 | 64 | Desk | 4346.9400000 | 330.00 |

## Join the table(s) and additional columns desired for the result table.

```
SELECT  vDesk.CustID,
        CustomerName,
        ProdName,
        SumQtyPrice TotalExtendedPrice
FROM vDesk
INNER JOIN xCust
ON vDesk.custID = xCust.custid
WHERE sumqtyprice =
      (SELECT MAX(sumqtyprice)
       FROM vDesk)
```



| | CustID | CustomerName | ProdName | TotalExtendedPrice |
|---|---|---|---|---|
| 1 | 2555 | Mountain Design | Desk | 4346.9400000 |

```
WITH cteSumExtPrice AS
(SELECT  custid,
         prodid,
        SUM(qtyOrdered*Price) SumQtyPrice
FROM xOrderline
INNER JOIN xOrd
ON xOrderline.orderid = xOrd.orderid
GROUP BY custid, prodid),

cteDesk as
(SELECT  custid,
         xprod.prodid,
         prodname,
         sumqtyprice,
         ProdSuggestedPrice
FROM     cteSumExtPrice
INNER JOIN xProd
ON ctesumextprice.prodid = xProd.prodid
WHERE prodname = 'desk')

SELECT   cteDesk.CustID,
         CustomerName,
         ProdName,
         SumQtyPrice TotalExtendedPrice
FROM cteDesk
INNER JOIN xCust
ON cteDesk.custID = xCust.custid
WHERE  sumqtyprice =
      (SELECT MAX(sumqtyprice)
        FROM cteDesk)
```

Can accomplish the same goal with CTE

```sql
WITH cteSumExtPrice AS
(SELECT  custid,
        xOrderline.prodid,
        prodname,
     SUM(qtyOrdered*price) AS SumQtyPrice
FROM xOrderline
INNER JOIN xOrd
ON xOrderline.orderid = xOrd.orderid
INNER JOIN xProd
ON xOrderline.prodid = xProd.prodid
WHERE prodname = 'desk'
GROUP BY custid, xOrderline.prodid, prodname)

SELECT  cteSumExtPrice.CustID,
        CustomerName,
        ProdName,
        SumQtyPrice TotalExtendedPrice
FROM cteSumExtPrice
INNER JOIN xCust
ON cteSumExtPrice.custID = xCust.custid
WHERE  sumqtyprice =
      (SELECT MAX(sumqtyprice)
        FROM cteSumExtPrice)
```

Can accomplish the same goal with a single CTE

Or a single VIEW…

# Why bother using Views or CTEs?

- Group functions and joins are complex.  Helps to split up a problem into smaller pieces of code.
- Difficult to do a group function of a group function. Examples:
  - The maximum of the sum of hours.
  - The minimum of a count of products.
- Joining multiple tables can yield full or partial cross joins making it difficult to trouble-shoot your SQL code.
- Good way to secure the data – users only see a view and don’t know the actual structure of the tables.

# Inserting Data vs. Bulk Load of Data

- A DBMS keeps an audit trail of each individual INSERT, UPDATE, and DELETE statement executed against the database.  The log of the transactions can be extensive.
- A “bulk load” of data allows the database programmer to insert much data without also creating a huge listing of all the data that is being INSERTed.

# Two general methods of bulk loading data

- One method requires that a table be created prior to loading the data.
- Another method creates a new table and bulk loads the data in a single statement.

# Two general methods of “bulk loading” data

## Examples of SQL Code used to “bulk load” data

```
INSERT INTO xCust (CustID, CustomerName)
SELECT      CustomerID,
            Name
FROM        xAnotherTable
WHERE       CustomerID NOT IN
            (SELECT CustID
             from xCust)
```

This syntax requires that a table be created prior to executing the INSERT statement. The SELECT can include a WHERE clause to determine which rows to insert.

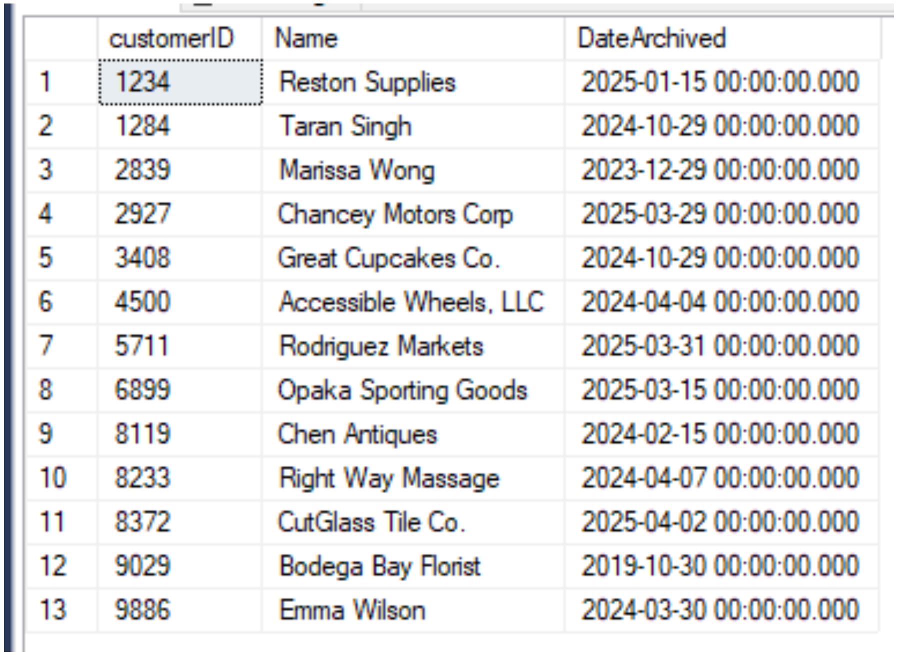
```
SELECT      CustomerID,
            Name
INTO        xCustNew
FROM        xAnotherTable
```

This syntax creates a table and inserts the data in a single statement. The SELECT can also include a WHERE clause to determine which rows to insert.

# Bulk load example

Create a new table called “xCustPast” by executing this script file: k:\CoB\IS475\LabFiles\SQLCustPast. This is the data that we want to add to the xCust table. But we only want to add those customers who don’t already exist in the xCust table.

SELECT * FROM xCustPast



| | customerID | Name | DateArchived |
|---|---|---|---|
| 1 | 1234 | Reston Supplies | 2025-01-15 00:00:00.000 |
| 2 | 1284 | Taran Singh | 2024-10-29 00:00:00.000 |
| 3 | 2839 | Marissa Wong | 2023-12-29 00:00:00.000 |
| 4 | 2927 | Chancey Motors Corp | 2025-03-29 00:00:00.000 |
| 5 | 3408 | Great Cupcakes Co. | 2024-10-29 00:00:00.000 |
| 6 | 4500 | Accessible Wheels, LLC | 2024-04-04 00:00:00.000 |
| 7 | 5711 | Rodriguez Markets | 2025-03-31 00:00:00.000 |
| 8 | 6899 | Opaka Sporting Goods | 2025-03-15 00:00:00.000 |
| 9 | 8119 | Chen Antiques | 2024-02-15 00:00:00.000 |
| 10 | 8233 | Right Way Massage | 2024-04-07 00:00:00.000 |
| 11 | 8372 | CutGlass Tile Co. | 2025-04-02 00:00:00.000 |
| 12 | 9029 | Bodega Bay Florist | 2019-10-30 00:00:00.000 |
| 13 | 9886 | Emma Wilson | 2024-03-30 00:00:00.000 |

# Use bulk load to create a new table called xCustHold

```
SELECT        CustID,
              CustomerName
INTO          xCustHold
FROM          xCust
```

We are going to modify xCust by adding new data, so it is a good idea to make a backup copy of the xCust table.  xCustHold is our backup copy of xCust.

# Use bulk load to add data to the existing customer table

```
INSERT INTO xCust (CustID, CustomerName)
SELECT CustomerID,
       Name
FROM    xCustPast
WHERE  CustomerID NOT IN
       (SELECT CustID
        FROM xCust)
```

We are adding rows to xCust that do not already exist in the table.

The two tables have different field names, but that is okay – the data is entered on the position of the field name rather than the actual name

# Bulk load will be used for HW#10

SQL Lab exercise 9 also shows how to do a bulk load of data.

Creating a backup copy of a table is optional, but a good idea when you intend to modify the contents of a table.