Data 604 – Data Management

Week 5 – SQL Programming 2

Learning Objectives

Write single- and multiple-table queries using SQL commands

Understand Boolean and Control and Flow operators

Understand difference between Loop and Batch operations and when to use each

Define three types of join commands and use SQL to write these commands

Write noncorrelated and correlated subqueries and know when to write each

Write queries to create views

Understand common uses of database operation functions

Homework 2 Review

- Review Homework 2 Solution

Conditional Expressions - CASE

```
SELECT CASE
      WHEN ProductLine = 1 THEN ProductDescription
      ELSE '####'
END AS ProductDescription
FROM Product_T;
```

A CASE expression acts like an if-then statement. It allows you to choose what will appear in a column of the result set, depending on a condition.

Result:

PRODUCTDESCRIPTION
End Table
####
####
####
Writers Desk
####
####
####

CASE

- Used to apply conditional logic within the query

```
select empid, birthdate,
  case when datediff(year,birthdate,getdate()) > 55
  then 'Yes, can retire'
  else 'No they cannot retire'
  end as RetireEligible
  from hr.Employees

 Select orderid, orderdate,
  case when month(orderdate) in (10,11) then
    year(dateadd(year,1,orderdate))
  else year(orderdate) end as FiscalYear
  from sales.orders
```

Multiple Tables

Assemble all information necessary to create an invoice for order number 1006.

Each pair of tables requires an equality-check condition in the WHERE clause, matching primary keys against foreign keys.

```
SELECT Customer_T.CustomerID, CustomerName, CustomerAddress,
   CustomerCity, CustomerState, CustomerPostalCode, Order_T.OrderID,
   OrderDate, OrderedQuantity, ProductDescription, StandardPrice,
   (OrderedQuantity * ProductStandardPrice)
FROM Customer_T, Order_T, OrderLine_T, Product_T
   WHERE Order_T.CustomerID = Customer_T.CustomerID
      AND Order_T.OrderID = OrderLine_T.OrderID
      AND OrderLine_T.ProductID = Product_T.ProductID
      AND Order_T.OrderID = 1006;
```

Multiple Tables - Results

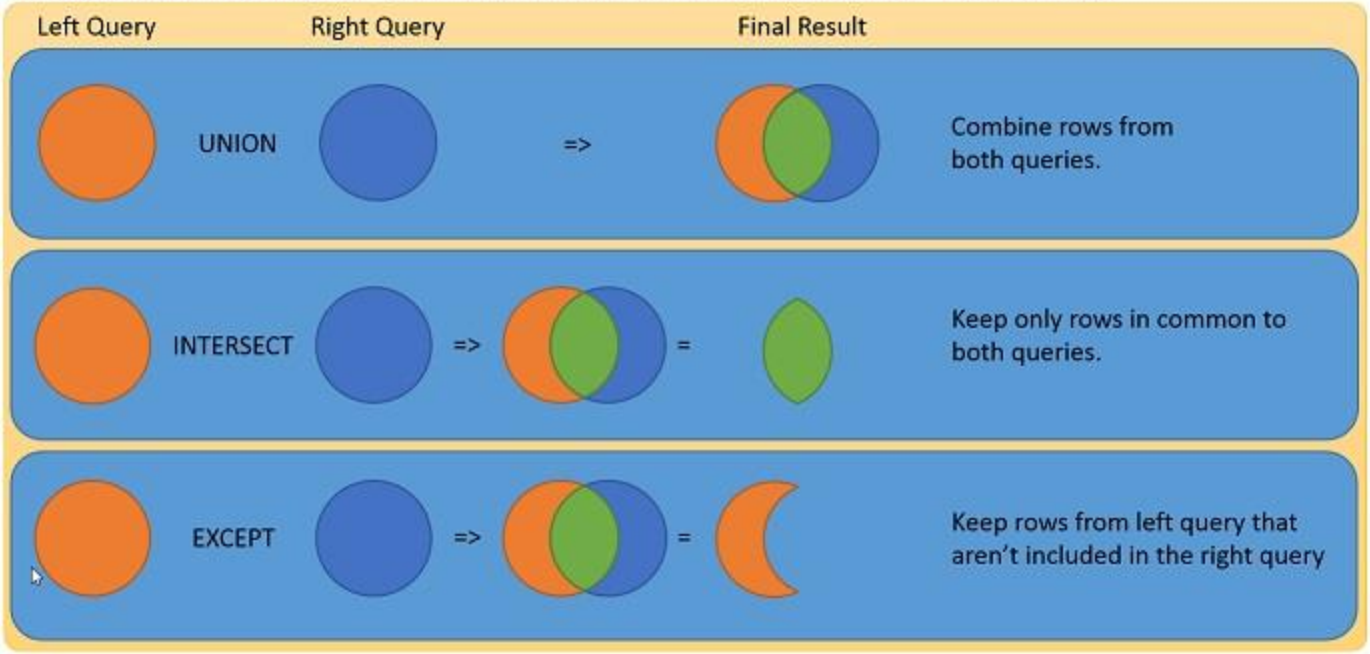
CUSTOMERID	CUSTOMERNAME	CUSTOMERADDRESS	CUSTOMER CITY	CUSTOMER STATE	CUSTOMER POSTALCODE
2	Value Furniture	15145 S. W. 17th St.	Plano	TX	75094 7743
2	Value Furniture	15145 S. W. 17th St.	Plano	TX	75094 7743
2	Value Furniture	15145 S. W. 17th St.	Plano	TX	75094 7743

ORDERID	ORDERDATE	ORDERED QUANTITY	PRODUCTNAME	PRODUCT STANDARDPRICE	(QUANTITY* STANDARDPRICE)
1006	24-OCT-18	1	Entertainment Center	650	650
1006	24-OCT-18	2	Writer's Desk	325	650
1006	24-OCT-18	2	Dining Table	800	1600

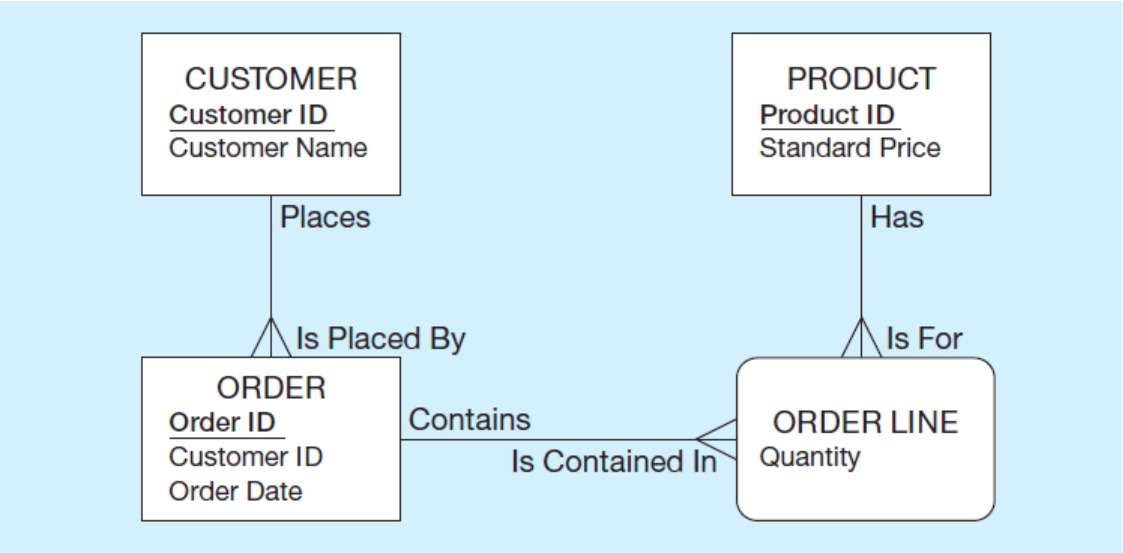
All rows returned from this query will pertain to Order ID 1006.

Note that the full query results include columns from four different tables.

Visual Explanation of UNION, INTERSECT, and EXCEPT operators



Sample Table



Inner join

```
SELECT Customer_T.CustomerID, Order_T.CustomerID,
   CustomerName, OrderID
FROM Customer_T  INNER JOIN  Order_T  ON
   Customer_T.CustomerID = Order_T.CustomerID
ORDER BY OrderID;
```

INNER JOIN clause is an alternative to WHERE clause, and is used to match primary and foreign keys.

An INNER join will only return rows from each table that have matching rows in the other

Inner Join

What are the customer IDs and names of all customers, along with the order IDs for all the orders they have placed?

```
SELECT Customer_T.CustomerID, Order_T.CustomerID,
   CustomerName, OrderID
      FROM Customer_T, Order_T
         WHERE  Customer_T.CustomerID = Order_T. CustomerID
         ORDER BY OrderID
```

Result:

CUSTOMERID	CUSTOMERID	CUSTOMERNAME	ORDERID
1	1	Contemporary Casuals	1001
8	8	California Classics	1002
15	15	Mountain Scenes	1003
5	5	Impressions	1004
3	3	Home Furnishings	1005
2	2	Value Furniture	1006
11	11	American Euro Lifestyles	1007
12	12	Battle Creek Furniture	1008
4	4	Eastern Furniture	1009
1	1	Contemporary Casuals	1010

10 rows selected.

Inner Join – TSQLV4

```
select pc.[categoryid], [categoryname], [description],
p.productid, p.productname
from [Production].[Categories] pc
inner join [Production].[Products] p
on pc.categoryid = p.categoryid

select pc.[categoryid], [categoryname], [description],
p.productid, p.productname
from [Production].[Categories] pc, [Production].[Products] p
where pc.categoryid = p.categoryid
```

What happens if we leave off the ‘Where’?

```
select pc.[categoryid], [categoryname], [description],
p.productid, p.productname
from [Production].[Categories] pc, [Production].[Products] p

--Using ‘Where’ there returns 77 rows.
--Without ‘Where’ there are 616 rows. Why?
```

Outer join

List the customer name, ID number, and order number for all customers. Include customer information even for customers that do not have an order.

```
SELECT Customer_T.CustomerID, CustomerName, OrderID
   FROM Customer_T  LEFT OUTER JOIN  Order_T
   WHERE Customer_T.CustomerID = Order_T. CustomerID;
```

LEFT OUTER JOIN clause causes rows from the first mentioned table (customer) to appear even if there is no corresponding order data.

Unlike an INNER join, this will include customer rows with no matching order rows.

This will return 16 rows. That's because there are 15 customers, and one of these customers has 2 orders.

Result of Outer Join

Note two rows for customer #1 Contemporary Casuals.

Also note that several customers don't have orders.

This is because of the left outer join.

CUSTOMERID	CUSTOMERNAME	ORDERID
1	Contemporary Casuals	1001
1	Contemporary Casuals	1010
2	Value Furniture	1006
3	Home Furnishings	1005
4	Eastern Furniture	1009
5	Impressions	1004
6	Furniture Gallery	
7	Period Furniture	
8	California Classics	1002
9	M & H Casual Furniture	
10	Seminole Interiors	
11	American Euro Lifestyles	1007
12	Battle Creek Furniture	1008
13	Heritage Furnishings	
14	Kaneohe Homes	
15	Mountain Scenes	1003

16 rows selected.

Self Join

What are the employee ID and name of each employee and the name of his or her supervisor (label the supervisor's name Manager)?

```
SELECT E.EmployeeID, E.EmployeeName, M.EmployeeName AS Manager
   FROM Employee_T E, Employee_T M
   WHERE E.EmployeeSupervisor = M.EmployeeID;
```

Result:

EMPLOYEEID	EMPLOYEENAME	MANAGER
123-44-347	Jim Jason	Robert Lewis

The same table is used on both sides of the join; distinguished using table aliases. See the next slide for details.

Union – Combining Queries

Combine the output (union of multiple queries) together into a single result table

With UNION queries, the quantity and data types of the attributes in the SELECT clauses of both queries must be identical.

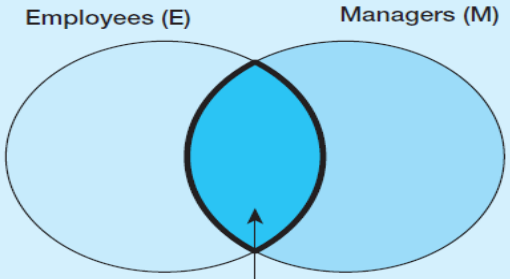
```
SELECT C1.CustomerID, CustomerName, OrderedQuantity,
'Largest Quantity' AS Quantity
FROM Customer_T C1,Order_T O1, OrderLine_T Q1
      WHERE C1.CustomerID = O1.CustomerID
      AND O1.OrderID = Q1.OrderID
      AND OrderedQuantity =
      (SELECT MAX(OrderedQuantity)
      FROM OrderLine_T)
UNION
SELECT C1.CustomerID, CustomerName, OrderedQuantity,
'Smallest Quantity'
FROM Customer_T C1, Order_T O1, OrderLine_T Q1
      WHERE C1.CustomerID = O1.CustomerID
      AND O1.OrderID = Q1.OrderID
      AND OrderedQuantity =
        (SELECT MIN(OrderedQuantity)
        FROM OrderLine_T)
ORDER BY 3;
```

Result:

CUSTOMERID	CUSTOMERNAME	ORDEREDQUANTITY	QUANTITY
1	Contemporary Casuals	1	Smallest Quantity
2	Value Furniture	1	Smallest Quantity
1	Contemporary Casuals	10	Largest Quantity

Example of a Self Join

Self join involve tables that implement 1-to-many unary relationships.



Employees who have supervisors; i.e.,
WHERE E.EmployeeSupervisor = M.EmployeeID

Employees (E)

EmployeeID	EmployeeName	EmployeeSupervisor
098-23-456	Sue Miller	
107-55-789	Stan Getz	
123-44-347	Jim Jason	678-44-546
547-33-243	Bill Blass	
678-44-546	Robert Lewis	

Managers (M)

EmployeeID	EmployeeName	EmployeeSupervisor
098-23-456	Sue Miller	
107-55-789	Stan Getz	
123-44-347	Jim Jason	678-44-546
547-33-243	Bill Blass	
678-44-546	Robert Lewis	

Table Expressions

Returns data that can be used as a table

- Derived Tables – subqueries defined in From clause of a query
- Common Table Expressions – uses WITH statement followed by outer query
- Views – reusable queries with definitions stored as permanent objects
- Inline table-valued functions –reusable queries that also accepts parameters (performance can be similar to a correlated subquery)

Sub Queries

- Subquery – placing an inner query (SELECT statement) inside an outer query
- Options:
  - In a condition of the WHERE clause
  - As a “table” of the FROM clause
  - Returning a field for the SELECT clause
  - Within the HAVING clause
- Subqueries can be:
  - Noncorrelated – executed once for the entire outer query
  - Correlated – executed once for each row returned by the outer query

https://www.vertica.com/docs/9.2.x/HTML/Content/Authoring/AnalyzingData/Queries/Subqueries/NoncorrelatedAndCorrelatedSubqueries.htm

Uncorrelated SubQuery

What are the name and address of the customer who placed order number 1008?

```
SELECT CustomerName, CustomerAddress, CustomerCity, CustomerState,
CustomerPostalCode
   FROM Customer_T
      WHERE Customer_T.CustomerID =
         (SELECT Order_T.CustomerID
            FROM Order_T
               WHERE OrderID = 1008);
```

Correlated Subquery

List the details about the product with the highest standard price.

```
SELECT ProductDescription, ProductFinish, ProductStandardPrice
FROM Product_T PA
   WHERE PA.ProductStandardPrice > ALL
      (SELECT ProductStandardPrice FROM Product_T PB
         WHERE PB.ProductID ! = PA.ProductID);
```

Result:

PRODUCTDESCRIPTION	PRODUCTFINISH	PRODUCTSTANDARDPRICE
Dining Table	Natural Ash	800

Correlated vs Noncorrelated Subqueries

- Noncorrelated subqueries:
  - Do not depend on data from the outer query
  - Execute once for the entire outer query
- Correlated subqueries:
  - Make use of data from the outer query
  - Execute once for each row of the outer query
  - Can use the EXISTS and ALL operators

Derived Query

What are the order IDs for all orders that have included furniture finished in natural ash?

```
SELECT ProductDescription, ProductStandardPrice, AvgPrice
  FROM
    (SELECT AVG(ProductStandardPrice) AvgPrice FROM Product_T),
      Product_T
  WHERE ProductStandardPrice > AvgPrice;
```

Here, the subquery forms the derived table used in the FROM clause of the outer query. The AvgPrice column from the subquery is used in the SELECT clause of the outer query.

SQL INSERT Statement

```
INSERT INTO PRODUCT(PRODNR, PRODNAME, PRODTYPE,
AVAILABLE_QUANTITY) VALUES
('980', 'Chateau Angelus, Grand Clu Classé, 1960', 'red', 6),
('1000', 'Domaine de la Vougeraie, Bâtard Montrachet', Grand
cru, 2010’, 'white', 2),
('1002', 'Leeuwin Estate Cabernet Sauvignon 2011', 'white',
20)
```

```
INSERT INTO INACTIVE-SUPPLIERS(SUPNR)
SELECT SUPNR
   FROM SUPPLIER
   EXCEPT
   SELECT SUPNR
   FROM SUPPLIES
```

SQL DELETE Statement

```
DELETE FROM PRODUCT
WHERE PRODNR = '1000'


DELETE FROM SUPPLIER
WHERE SUPSTATUS IS NULL


DELETE FROM SUPPLIES
WHERE PRODNR IN (SELECT PRODNR
                 FROM PRODUCT
                 WHERE PRODNAME LIKE '%CHARD%')
```

SQL DELETE Statement

```
DELETE FROM SUPPLIER R
WHERE NOT EXISTS
      (SELECT PRODNR
       FROM SUPPLIES S
       WHERE R.SUPNR = S.SUPNR)


DELETE FROM SUPPLIES S1
WHERE S1.PURCHASE_PRICE >
(SELECT 2 * AVG(S2.PURCHASE_PRICE)
FROM SUPPLIES S2
WHERE S1.PRODNR = S2.PRODNR)


DELETE FROM PRODUCT
```

SQL UPDATE Statement

```
UPDATE PRODUCT
SET AVAILABLE_QUANTITY = 26
WHERE PRODNR = '0185'


UPDATE SUPPLIER
SET SUPSTATUS = DEFAULT


UPDATE SUPPLIES
SET DELIV_PERIOD = DELIV_PERIOD+7
WHERE SUPNR IN (SELECT SUPNR
                FROM SUPPLIER
                WHERE SUPNAME = 'Deliwines')
```

SQL UPDATE Statement

```
UPDATE SUPPLIES S1
SET (PURCHASE_PRICE, DELIV_PERIOD) =
(SELECT MIN(PURCHASE_PRICE), MIN(DELIV_PERIOD)
FROM SUPPLIES S2
WHERE S1.PRODNR = S2.PRODNR)
WHERE SUPNR = '68'
```

```
ALTER TABLE SUPPLIER ADD SUPCATEGORY VARCHAR(10) DEFAULT
'SILVER'
UPDATE SUPPLIER
SET SUPCATEGORY =
CASE WHEN SUPSTATUS >= 70 AND SUPSTATUS <= 90 THEN 'GOLD'
WHEN SUPSTATUS >= 90 THEN 'PLATINUM'
ELSE 'SILVER'
END
```

SQL UPDATE Statement

SUPNR	SUPNAME	SUPADDRESS	SUPCITY	SUPSTATUS	SUPCATEGORY
21	Deliwines	20, Avenue of the Americas	New York	20	SILVER
32	Best Wines	660, Market Street	San Francisco	90	GOLD
37	Ad Fundum	82, Wacker Drive	Chicago	95	PLATINUM
52	Spirits & co.	928, Strip	Las Vegas	NULL	SILVER
68	The Wine Depot	132, Montgomery Street	San Francisco	10	SILVER
69	Vinos del Mundo	4, Collins Avenue	Miami	92	PLATINUM
84	Wine Trade Logistics	100, Rhode Island Avenue	Washington	92	PLATINUM
94	The Wine Crate	330, McKinney Avenue	Dallas	75	GOLD

Query Efficiency

- Instead of SELECT *, identify the specific attributes in the SELECT clause; this helps reduce network traffic of result set
- Limit the number of subqueries; try to make everything done in a single query if possible
- If data is to be used many times, make a separate query and store it as a view
- Loop vs Batch

Views

- Production databases contain hundreds or even thousands of tables, and tables could include hundreds of columns.
- So, sometimes query requirements can be very complex.
- Sometimes it's useful to combine queries, through the use of Views.
- If you use a view (which is a query), you could have another query that uses the view as if it were a table.

SQL Views

- SQL views are part of the external data model
- A view is defined by means of an SQL query and its content is generated upon invocation of the view by an application or other query
- A view is a virtual table without physical tuples
- Views allow for logical data independence which makes them a key component in the three-layer database architecture

Example of a View

For each salesperson, list his or her biggest-selling product.

The view:

```
CREATE VIEW TSales AS
SELECT SalespersonName,
      ProductDescription,
      SUM(OrderedQuantity) AS Totorders
FROM Salesperson_T, OrderLine_T, Product_T, Order_T
      WHERE Salesperson_T.SalespersonID=Order_T.SalespersonID
      AND Order_T.OrderID=OrderLine_T.OrderID
      AND OrderLine_T.ProductID=Product_T.ProductID
      GROUP BY SalespersonName, ProductDescription;
```

The query using the view:

```
SELECT SalespersonName, ProductDescription
      FROM TSales AS A
         WHERE Totorders = (SELECT MAX(Totorders) FROM TSales B
         WHERE B.SalesperssonName = A.SalespersonName);
```

SQL Views

```
CREATE VIEW TOPSUPPLIERS
AS SELECT SUPNR, SUPNAME FROM SUPPLIER
WHERE SUPSTATUS > 50
```

```
CREATE VIEW TOPSUPPLIERS_SF
AS SELECT * FROM TOPSUPPLIERS
WHERE SUPCITY = 'San Francisco'
```

SQL Views

```
CREATE VIEW ORDEROVERVIEW(PRODNR, PRODNAME,
TOTQUANTITY)
AS SELECT P.PRODNR, P.PRODNAME, SUM(POL.QUANTITY)
FROM PRODUCT AS P LEFT OUTER JOIN PO_LINE AS POL
ON (P.PRODNR = POL.PRODNR)
GROUP BY P.PRODNR
```

SQL Views

- Some views can be updated
  - In this case, the view serves as a window through which updates are propagated to the underlying base table(s)

SQL TO Query Metadata for Data Dictionary

```
SELECT TABLE_CATALOG,
CONCAT(TABLE_SCHEMA,'.',TABLE_NAME) AS TableName,
COLUMN_NAME , IS_NULLABLE, DATA_TYPE
FROM [INFORMATION_SCHEMA].[COLUMNS]
WHERE TABLE_CATALOG = 'TSQLV4'
ORDER BY Table_name, column_name
```

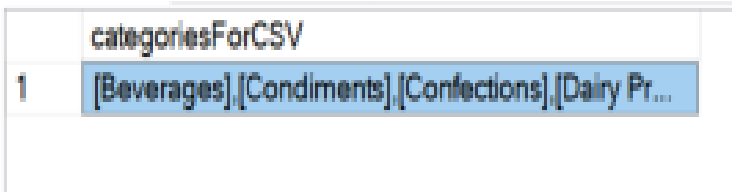
Advanced Topics

- Complex Queries
  - Solve complex queries one step at a time
  - Test incrementally at each step when you add in a new function, expression or table
  - Good idea to back up database or create a copy of the table when modifying data

String_agg Function

- String_agg: turn rows for a field into a one merged value

```
select  concat('[',(STRING_AGG(c.CATEGORYNAME, '],[')),']') as categoriesForCSV
  FROM
  [Production].[Categories] c
```



	categoriesForCSV
1	[Beverages],[Condiments],[Confections],[Dairy Pr...

String_split function

- String_split: turn string/column of data into separate rows

```
select VALUE
 FROM
 STRING_SPLIt('[Beverages],[Condiments],[Confections],[Dairy
Products],[Grains/Cereals],[Meat/Poultry],[Produce],[Seafood]',',')
```

	VALUE
1	[Beverages]
2	[Condiments]
3	[Confections]
4	[Dairy Products]
5	[Grains/Cereals]
6	[Meat/Poultry]
7	[Produce]
8	[Seafood]

Window Functions

- Computes a scalar result for each row based on a calculation against of subset of rows from the underlying query

```
SELECT orderid, custid, val,
SUM(val) OVER() as totalvalue,
SUM(val) OVER(PARTITION BY custid) as custtotalvalue
FROM Sales.OrderValues
```

Results | Messages

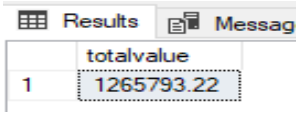
	orderid	custid	val	totalvalue	custtotalvalue
1	10643	1	814.50	1265793.22	4273.00
2	10692	1	878.00	1265793.22	4273.00
3	10702	1	330.00	1265793.22	4273.00
4	10835	1	845.80	1265793.22	4273.00

Window Functions Vs Grouping

Window functions allow you to rollup aggregates inline or you could traditionally run the groupings in separate grouping queries:

- Query to return one total value for entire dataset

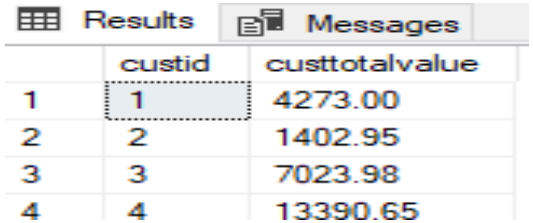
```
SELECT SUM(val) as totalvalue
from Sales.OrderValues
```



	totalvalue
1	1265793.22

- Query to return group total value for each customer

```
SELECT custid, sum(val) as custtotalvalue
FROM Sales.OrderValues
group by custid
order by custid
```



	custid	custtotalvalue
1	1	4273.00
2	2	1402.95
3	3	7023.98
4	4	13390.65

Pivot Vs Unpivot

Student	Subject	Marks
Jacob	Mathematics	100
Jacob	Science	95
Jacob	Geography	90
Amilee	Mathematics	90
Amilee	Science	95
Amilee	Geography	100

Original Records

Student	Mathematics	Science	Geography
Jacob	100	95	90
Amilee	90	95	100

PIVOT Data

Pivoting Data

- Pivoting swaps data from a row to a column
- In T-SQL, to pivot you must group, spread and aggregate your data

```
SELECT  productid, [Beverages],[Condiments],[Confections],[Dairy
Products],
[Grains/Cereals],[Meat/Poultry],[Produce],[Seafood]
FROM
(SELECT [productid],c.categoryname,c.categoryid
                FROM [TSQLV4].[Production].[Products] p
  inner join [Production].[Categories] c
  on p.categoryid = c.categoryid) AS c
  PIVOT(count(categoryid) for categoryname in ([Beverages],[Condiments],
  [Confections],[Dairy Products],[Grains/Cereals],
  [Meat/Poultry],[Produce],[Seafood])) as p
```

Unpivoting Data

- Unpivoting swaps data from a column to a row
- Create view in TSQLv4 datab

```
create view vPivot
as
SELECT  productid, [Beverages],[Condiments],[Confections],[Dairy
Products],
[Grains/Cereals],[Meat/Poultry],[Produce],[Seafood]
FROM
(SELECT [productid],c.categoryname,c.categoryid
                FROM [TSQLV4].[Production].[Products] p
  inner join [Production].[Categories] c
  on p.categoryid = c.categoryid) AS c
  PIVOT(count(categoryid) for categoryname in ([Beverages],[Condiments],
  [Confections],[Dairy Products],[Grains/Cereals],
  [Meat/Poultry],[Produce],[Seafood])) as pase  from using the pivot query
```

Unpivoting Data

- Now lets unpivot

```
SELECT productid, categoryname, cnt
 from vPivot
 UNPIVOT(cnt for categoryname in([Beverages],[Condiments],
 [Confections],[Dairy Products],[Grains/Cereals],
 [Meat/Poultry],[Produce],[Seafood])) as u
 where cnt = 1
```

Other Database Objects

- Routines
  - Program modules that execute on demand
- Functions
  - routines that return values and take input parameters
- Procedures
  - routines that do not return values and can take input or output parameters

  -- https://www.w3schools.com/sql/sql_stored_procedures.asp
- Triggers
  - routines that execute in response to a database event (INSERT, UPDATE, or DELETE)

  -- https://www.sqlservertutorial.net/sql-server-triggers/