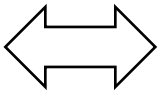
Cis 111 Chapter 4 Student Lecture Notes

Topic #1: Joining Tables in the Select Statement – Inner and Outer Joins

You focused on the SQL Select statement in the previous chapter where only one table was used as the data source. In this chapter you will again be working with the Select statement but will be using **two or more tables**. When more than one table is used in a Select statement a **Join** has to be defined between two tables that have a common field. Please review the following example:

Customers table

CustomerID	CustomerName
1	John Smith
2	Sue Doe
3	Dave Brown
4	Tina Jones
5	Tom Jackson



Orders table

OrderID	Date	CustomerID	Total
1	02/22/18	1	100
2	02/26/18	1	50
3	03/2/18	3	20
4	03/12/18	3	50
5	04/3/18	3	70
6	05/30/18	5	100
7	06/22/18	5	150
8	06/24/18	8	80

Customers primary key:_____ Orders primary key:_____

What field is common between these tables? _____

Questions about the relationship between the Customers and Orders table…

Which table is the primary key table in this relationship? _____

Which table is the foreign key table in this relationship? _____

What type of relationship exists between Customers and the Orders table? Please circle one:

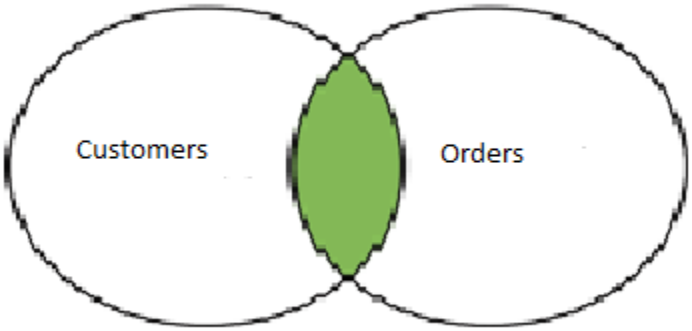
One to one (1:1) One to Many (1:M) Many to Many (M:N)

Referential Integrity Problem

The example above contains a foreign key value that does not exist in the primary table. Which foreign key value in the Orders table contains a problem? CustomerID_____. This is referred to as an **orphaned record** and should not happen. This can be prevented when the table relationships are created during design time by defining the **referential integrity** between these two tables. This is discussed in a future chapter. The reason the example above does contain an orphaned record is to see the effect it has on the different types of Joins.

Inner Join

The **INNER JOIN** keyword selects records that have **matching** values in **both tables**. Please refer to the following diagram using the Customers and the Orders table as an example:



Select Customers.CustomerID as ID, CustomerName as Name, OrderDate, Total

From Customers **Inner Join** Orders **On** Customers.CustomerID=Orders.CustomerID

Common field

Query Result Set: Review which customer IDs are missing from the Customers table. Also, the orphaned record in the Orders table is missing.

ID	Name	OrderDate	Total
1	John Smith	02/22/18	100
1	John Smith	02/26/18	50
3	Dave Brown	03/2/18	20
3	Dave Brown	03/12/18	50
3	Dave Brown	04/3/18	70
5	Tom Jackson	05/30/18	100
5	Tom Jackson	06/22/18	150

Note: Select Customers.CustomerID as ID, CustomerName as Name, OrderDate, Total

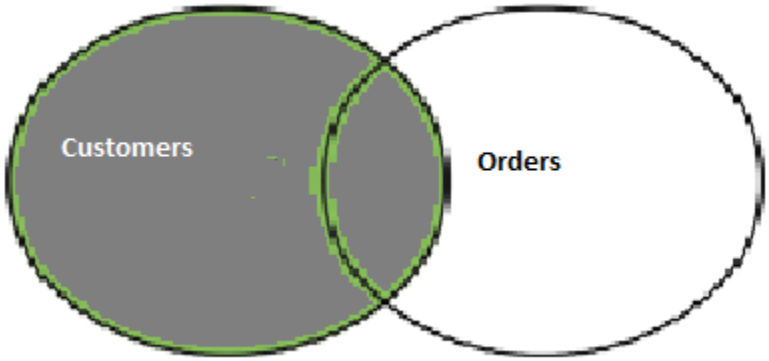
From Customers **Join** Orders **On** Customers.CustomerID=Orders.CustomerID

The keyword **Join** by itself is also a correct syntax to define an inner join.

Why is CustomerID **qualified** with a table name in front of it? If you code a Select statement that joins tables that have fields with the same name then you are required to qualify it if it is coded somewhere in the Select statement by coding the table name in front of it. The dot (.) is referred to as a **dot operator** which means the item on the right of the dot belongs to the item to the left of the dot.

Outer Join

Three outer joins exist: **left, right and full outer join**. You will first review a **left outer join**.



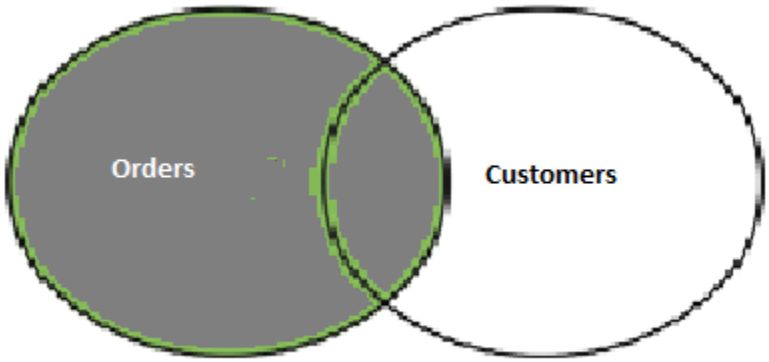
Select Customers.CustomerID as ID, CustomerName as Name, OrderDate, Total

From Customers **Left Join** Orders **On** Customers.CustomerID=Orders.CustomerID

Query Result Set: All customers will be in the result set regardless if they have an order or not. The customers that are not located in the Orders table will contain a NULL value for each order field that is coded in the Select clause. The orphaned record in the Orders table will not be in the result set.

ID	Name	OrderDate	Total
1	John Smith	02/22/18	100
1	John Smith	02/26/18	50
2	Sue Doe	*NULL*	*NULL*
3	Dave Brown	03/2/18	20
3	Dave Brown	03/12/18	50
3	Dave Brown	04/3/18	70
4	Tina Jones	*NULL*	*NULL*
5	Tom Jackson	05/30/18	100
5	Tom Jackson	06/22/18	150

Another Left Join example:



The Orders table can be coded so that it is considered the table on the left. Please review this example:

Select Customers.CustomerID as ID, CustomerName as Name, OrderDate, Total

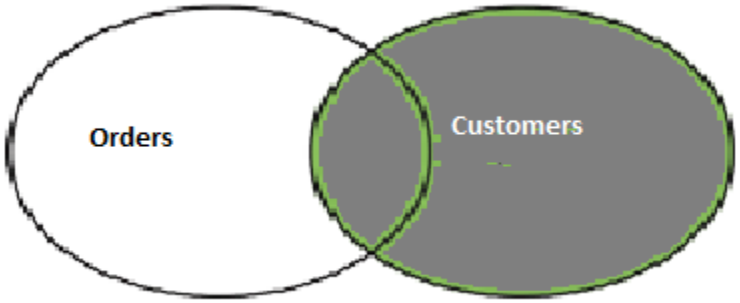
From Orders **Left Join** Customers **On** Orders.CustomerID=Customers.CustomerID

Query Result Set: All Orders will be in the result set including the orphaned record. Customers that do not have any orders will not be in the result set.

ID	Name	OrderDate	Total
1	John Smith	02/22/18	100
1	John Smith	02/26/18	50
3	Dave Brown	03/2/18	20
3	Dave Brown	03/12/18	50
3	Dave Brown	04/3/18	70
5	Tom Jackson	05/30/18	100
5	Tom Jackson	06/22/18	150
Null	*Null*	06/24/18	80

Right Outer Join

The idea for a right outer join is the same as the left except that all records are taken from the right table regardless if there is a matching field to the table on the left.



Select Customers.CustomerID as ID, CustomerName as Name, OrderDate, Total

From Orders **Right Join** Customers **On** Orders.CustomerID=Customers.CustomerID

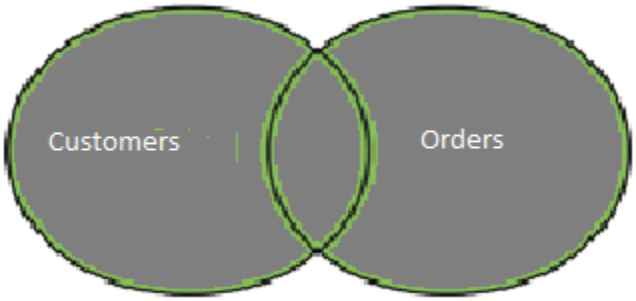
Query Result Set: All customers will be in the result set regardless if they have an order or not. The customers that are not located in the Orders table will contain a NULL value for each order field that is coded in the Select clause. The orphaned record in the Orders table will not be in the result set.

ID	Name	OrderDate	Total
1	John Smith	02/22/18	100
1	John Smith	02/26/18	50
2	Sue Doe	*NULL*	*NULL*
3	Dave Brown	03/2/18	20
3	Dave Brown	03/12/18	50
3	Dave Brown	04/3/18	70
4	Tina Jones	*NULL*	*NULL*
5	Tom Jackson	05/30/18	100
5	Tom Jackson	06/22/18	150

Full Outer Join

The FULL OUTER JOIN keyword return all records when there is a match in either left or right table records.

Note: FULL OUTER JOIN can potentially return very large result-sets!



Select Customers.CustomerID as ID, CustomerName as Name, OrderDate, Total

From Customers **Full Join** Orders **On** Customers.CustomerID=Orders.CustomerID

Query Result Set: All customers will be in the result set regardless if they have an order or not. The customers that are not located in the Orders table will contain a NULL value for each order field that is coded in the Select clause. The orphaned record in the Orders table will also be included in the result set and will contain null values for all fields from the Customers table.

ID	Name	OrderDate	Total
1	John Smith	02/22/18	100
1	John Smith	02/26/18	50
2	Sue Doe	*NULL*	*NULL*
3	Dave Brown	03/2/18	20
3	Dave Brown	03/12/18	50
3	Dave Brown	04/3/18	70
4	Tina Jones	*NULL*	*NULL*
5	Tom Jackson	05/30/18	100
5	Tom Jackson	06/22/18	150
Null	*Null*	06/24/18	80

Topic #2: Self Join

There are times when a table has to be joined to itself in a query Select statement. You will review an example using the following Employees table:

EmpID	Name	Gender	SupervisorID
1	John Smith	M	4
2	Sue Brown	F	4
3	Tom Doe	M	*2*
4	Dave Green	M	*Null*
5	Tina Sims	F	1
6	Bob Jackson	M	2

The values for the **SupervisorID** field represents an employee id, therefore, John Smith's supervisor is an employee with an ID of 4, which is Dave Green.

You need to code a Select statement that will display the **Employee name** in column one and the **Supervisor Name** in column two.

SELECT Employees.Name as [Employee Name], Employees_1.Name as [Supervisor Name]

FROM **Employees** INNER JOIN **Employees AS Employees_1**

ON Employees.**SupervisorID** = Employees_1.**EmployeeID**

Employees

EmpID	Name	Gender	SupervisorID
1	John Smith	M	4
2	Sue Brown	F	4
3	Tom Doe	M	*2*
4	Dave Green	M	*Null*
5	Tina Sims	F	1
6	Bob Jackson	M	2

Employees_1

EmpID	Name	Gender	SupervisorID
1	John Smith	M	4
2	Sue Brown	F	4
3	Tom Doe	M	*2*
4	Dave Green	M	*Null*
5	Tina Sims	F	1
6	Bob Jackson	M	2

Notice when the SQL Database engine processes the first employee record from the Employees table that it will be joined to Dave Green's record located in the Employees_1 table.

The result set of the self-join query

Employee Name	Supervisor Name
John Smith	Dave Green
Sue Brown	Dave Green
Tom Doe	Sue Brown
Tina Sims	John Smith
Bob Jackson	Sue Brown

Another Self-Join example

Assume you want to schedule an onsite meeting with your customers but would like to group customers at one location for those who are from the same city and state. Please review the following example:

Customers table

CustID	Name	City	State
1	Customer One	Canton	MI
2	Customer Two	Canton	MI
3	Customer Three	Canton	OH
4	Customer Four	Novi	MI
5	Customer Five	Novi	MI
6	Customer Six	Novi	MI
7	Customer Seven	Dearborn	MI

This Select statement should display all of the onsite locations that will need to have a meeting scheduled for multiple customers from the same city and state:

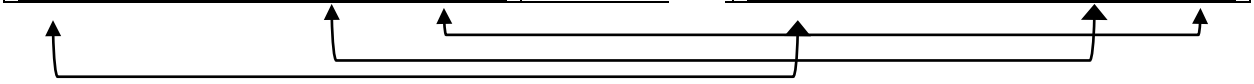
```
SELECT DISTINCT Customers1.City,
    Customers1.State
FROM Customers AS Customers1 JOIN Customers AS Customers2
    ON (Customers1.City = Customers2.City) AND
       (Customers1.State = Customers2.State) AND
       (Customers1.CustID <> Customers2.CustID)
ORDER BY Customers1.State, Customers1.City;
```

CUSTOMERS1

CustID	Name	City	State
1	Customer One	Canton	MI
2	Customer Two	Canton	MI
3	Customer Three	Canton	OH
4	Customer Four	Novi	MI
5	Customer Five	Novi	MI
6	Customer Six	Novi	MI
7	Customer Seven	Dearborn	MI

CUSTOMERS2

CustID	Name	City	State
1	Customer One	Canton	MI
2	Customer Two	Canton	MI
3	Customer Three	Canton	OH
4	Customer Four	Novi	MI
5	Customer Five	Novi	MI
6	Customer Six	Novi	MI
7	Customer Seven	Dearborn	MI



Prior to the SQL Server database engine **distinctly** returning the result set, fill-in the following table

Write down the customer id that created the row	**City**	**State**
	Canton	MI
	Canton	MI
	Novi	MI
	Novi	MI
	Novi	MI
	Novi	MI
	Novi	MI
	Novi	MI

The final result set would be

City	**State**
Canton	MI
Novi	MI

Topic #3: Interim Table

Interim table is a table that is generated by joining two tables by the SQL Server engine but is not a final result that is returned to the client. In other words, when two tables are joined they create an interim table that require further processing by the SQL Server engine. The self-join examples described in the previous topics used interim tables. Now you will review a Select statement that includes four tables:

```
SELECT VendorName, InvoiceNumber, InvoiceDate,
    InvoiceLineItemAmount AS LineItemAmount,
    AccountDescription
FROM Vendors
    JOIN Invoices ON Vendors.VendorID = Invoices.VendorID
    JOIN InvoiceLineItems
      ON Invoices.InvoiceID = InvoiceLineItems.InvoiceID
    JOIN GLAccounts
      ON InvoiceLineItems.AccountNo = GLAccounts.AccountNo
WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0
ORDER BY VendorName, LineItemAmount DESC;
```

First interim table

```
FROM Vendors
    JOIN Invoices ON Vendors.VendorID = Invoices.VendorID
```

The first interim table

	VendorName	InvoiceNumber	InvoiceDate
1	Blue Cross	547480102	2016-04-01 00:00:00
2	Cardinal Business Media, Inc.	134116	2016-03-28 00:00:00
3	Data Reproductions Corp	39104	2016-03-10 00:00:00
4	Federal Express Corporation	963253264	2016-03-18 00:00:00
5	Federal Express Corporation	263253268	2016-03-21 00:00:00
6	Federal Express Corporation	263253270	2016-03-22 00:00:00
7	Federal Express Corporation	263253273	2016-03-22 00:00:00

```
(11 rows)
```

Notice the columns returned will only be columns listed in the Select clause that are from the Vendors or Invoices tables. The common field between these two tables is VendorID.

Second interim table

```
JOIN InvoiceLineItems
      ON Invoices.InvoiceID = InvoiceLineItems.InvoiceID
```

The second interim table

	VendorName	InvoiceNumber	InvoiceDate	LineItemAmount
1	Blue Cross	547480102	2016-04-01 00:00:00	224.00
2	Cardinal Business Media, Inc.	134116	2016-03-28 00:00:00	90.36
3	Data Reproductions Corp	39104	2016-03-10 00:00:00	85.31
4	Federal Express Corporation	263253270	2016-03-22 00:00:00	67.92
5	Federal Express Corporation	263253268	2016-03-21 00:00:00	59.97
6	Federal Express Corporation	963253264	2016-03-18 00:00:00	52.25
7	Federal Express Corporation	263253273	2016-03-22 00:00:00	30.75

(11 rows)

Results of first interim table will be joined with the interim table described above which involves the relationship between the Invoices and InvoiceLineItems table (common field is InvoiceID). Notice the columns resulting in the second interim table will be from the first interim table and columns listed in the Select clause from the InvoiceLineItems table.

Final result set

```
JOIN GLAccounts
      ON InvoiceLineItems.AccountNo = GLAccounts.AccountNo
```

The final result set

	VendorName	InvoiceNumber	InvoiceDate	LineItemAmount	AccountDescription
1	Blue Cross	547480102	2016-04-01 00:00:00	224.00	Group Insurance
2	Cardinal Business Media, Inc.	134116	2016-03-28 00:00:00	90.36	Direct Mail Advertising
3	Data Reproductions Corp	39104	2016-03-10 00:00:00	85.31	Book Printing Costs
4	Federal Express Corporation	263253270	2016-03-22 00:00:00	67.92	Freight
5	Federal Express Corporation	263253268	2016-03-21 00:00:00	59.97	Freight
6	Federal Express Corporation	963253264	2016-03-18 00:00:00	52.25	Freight
7	Federal Express Corporation	263253273	2016-03-22 00:00:00	30.75	Freight

(11 rows)

Results of the second interim table will be joined with the interim table described above which involves the relationship between the GLAccounts and InvoiceLineItems table (common field is AccountNo). Notice the columns resulting in the final result set will be from the second interim table and columns listed in the Select clause from the GLAccounts table.

When was the Where clause conditions used? (Please circle your answer)

First interim table second interim table final result set

Topic #4: Join Syntax – Implicit and Explicit

You now know that the Join clause is used to combine rows from two or more tables, based on a related column between them. You can code a Join clause using the **explicit** syntax or the **implicit** syntax.

Join Clause – Implicit Syntax

```
SELECT InvoiceNumber, VendorName

FROM Vendors, Invoices

WHERE Vendors.VendorID = Invoices.VendorID;
```

Implicit Join simply has you code all tables used in the Select statement listed in the From clause separated by a comma.

The common field between tables is defined in the Where clause.

Join Clause – Explicit Syntax

The relationship between tables is completely defined in the From clause using the correct Join clause along with defining the related field(s) between tables using the On clause.

```
SELECT InvoiceNumber, VendorName

FROM Vendors Join Invoices
 On Vendors.VendorID=Invoices.VendorID

WHERE Vendors.VendorID = Invoices.VendorID;
```

Which is better to use? The implicit syntax is "older" syntax and may "go away" in the future. Most developers think the explicit syntax more clearly defines the relationship between tables then the implicit syntax.

Topic #5: Cross Join – Cartesian Product

Although **a Cross Join** is seldom used it needs to be understood since some developers might code it by mistake because they did not include all of the necessary tables needed for a Select statement. Please review the following three tables:

The Departments table

	DeptName	DeptNo
1	Accounting	1
2	Payroll	2
3	Operations	3
4	Personnel	4
5	Maintenance	5

The Employees table

	EmployeeID	LastName	FirstName	DeptNo
1	1	Smith	Cindy	2
2	2	Jones	Elmer	4
3	3	Simonian	Ralph	2
4	4	Hernandez	Olivia	1
5	5	Aaronsen	Robert	2
6	6	Watson	Denise	6
7	7	Hardy	Thomas	5
8	8	O'Leary	Rhea	4
9	9	Locario	Paulo	6

The Projects table

	ProjectNo	EmployeeID
1	P1011	8
2	P1011	4
3	P1012	3
4	P1012	1
5	P1012	5
6	P1013	6
7	P1013	9
8	P1014	10

Assume you have been asked to list all of the department names along with the project numbers that have been assigned to it. Please review the following Select statement using implicit syntax:

```
Select DeptName, ProjectNo From Departments,
Projects Order by DeptName
```

Notice a Where clause was left out since there isn't a common field between the **Departments** and **Projects** table. What would happen?

SQL Server Engine: It will start with the first department in the Departments table and join all records contained in the Projects table. It will then take the next department in the Departments table and join all records contained in the Projects table. It will do this for each department record in the departments table. How many records will be contained in the result set? _____

What would be the **explicit** syntax of the Select statement that was coded on the previous page?

```
Select DeptName, ProjectNo
     From Departments Cross Join Projects
     Order by DeptName
```

What might be the correct way to code this select statemen?

```
SELECT DeptName, ProjectNo
FROM Departments
 LEFT JOIN Employees
   ON Departments.DeptNo = Employees.DeptNo
 LEFT JOIN Projects ON
    Employees.EmployeeID = Projects.EmployeeID
ORDER BY DeptName;
```

The reason Left Joins are used instead of Inner Joins is because of the orphaned records that exist in the Employees table and the Projects table. Again, orphaned records should not happen if referential integrity is implemented correctly.

Topic #5: Union Operator

Like a join, a **Union** combines data from two or more Select statements. Here are some important facts about the Union operator:

- ✓ Each SELECT statement within a UNION must have the **same number of columns**
- ✓ The **columns** must also have **similar data types**
- ✓ The **columns** in each SELECT statement must also be in the **same order**

Assume you have a Customers table and a Suppliers table and need to list all customer and supplier contacts from Italy. You could code the following

```
SELECT 'Customer'As Type, ContactName As Contact, City, Country
      FROM Customers
      Where Country='Italy'
UNION
SELECT 'Supplier', SupplierContactName, SupplierCity,
               SupplierCountry
      FROM Suppliers
      Where SupplierCountry='Italy'
Order by Country
```

How many columns are in each Select statement? _____

Based upon a column's position in each Select statement, do they all match in data type? _____

What column headings are used in the final result set? _____

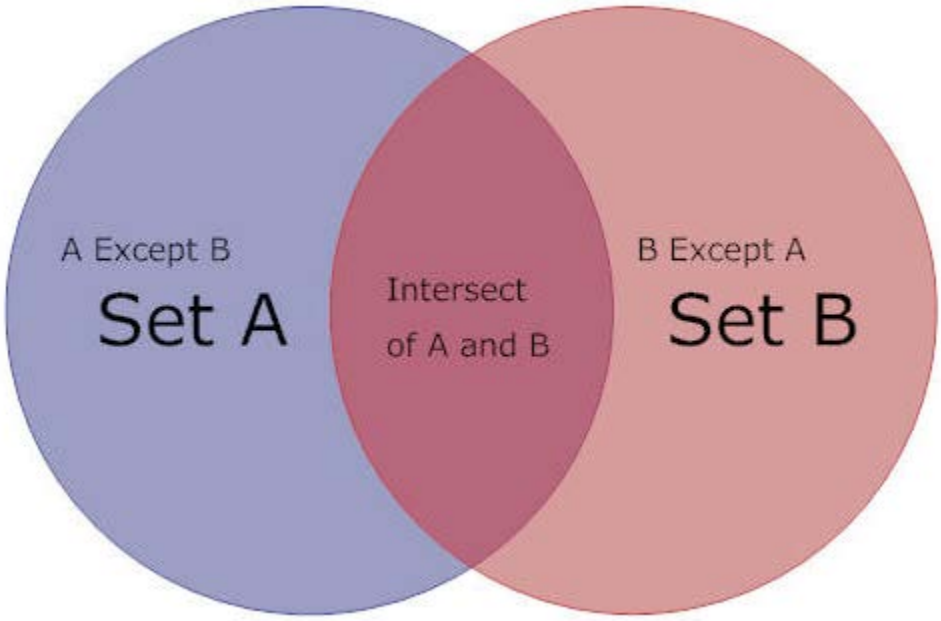
If an Order by clause is coded in a Union what columns are referenced? _____

This next example would only return one column in the final result set. What would happen with duplicate values? The Union operator automatically returns unique records. If duplicates should not be removed then you would to code the **Union All** operator.

```
SELECT City FROM Customers
UNION
SELECT SupplierCity FROM Suppliers
ORDER BY City;
```

Topic #6: Except and Intersect Operators

The UNION, **EXCEPT** and **INTERSECT** operators of SQL enable you to combine more than one SELECT statement to form a single result set. The UNION operator returns all rows. The INTERSECT operator returns all rows that are in both result sets. The EXCEPT operator returns the rows that are only in the first result set but not in the second.



To return the data in Set A that doesn't overlap with B, use **A EXCEPT B**.

To return only the data that overlaps in the two sets, use **A INTERSECT B**.

To return the data in Set B that doesn't overlap with A, use **B EXCEPT A**.

To return the data in all three areas without duplicates, use **A UNION B**.

To return the data in all three areas, including duplicates, use **A UNION ALL B**.

Except and **Intersect** Examples

```
CREATE VIEW Lunch
AS
 SELECT ‘Pop’ AS item
 UNION SELECT 'Olives'
 UNION SELECT 'Bread'
 UNION SELECT 'Salami'
 UNION SELECT 'Calamari'
 UNION SELECT 'Coffee';
```

Lunch View when it is used:

Item
Pop
Olives
Bread
Salami
Calamari
Coffee

```
CREATE VIEW Dinner
AS
 SELECT 'Juice' AS item
 UNION SELECT 'Olives'
 UNION SELECT 'Bread'
 UNION SELECT 'Steak'
 UNION SELECT 'eggplant'
 UNION SELECT 'Salad'
 UNION SELECT 'Coffee'
 UNION SELECT 'Apple';
```

Dinner View when it is used:

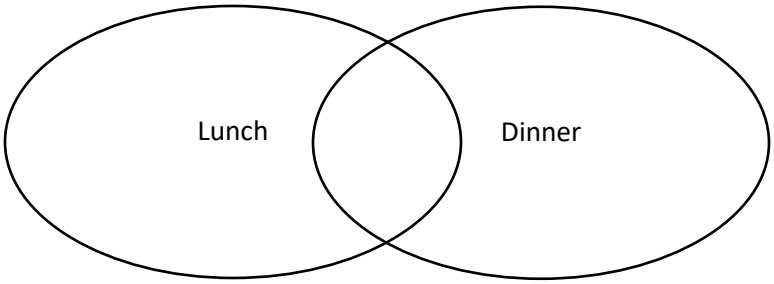
item
Juice
Olives
Bread
Steak
eggplant
Salad
Coffee
Apple

Now let’s look at how you would return only the food served (or drank) for **lunch**, **but was not served** for **dinner**:

```
SELECT item FROM Lunch
EXCEPT
SELECT item FROM Dinner;
```

item
Calamari
Pop
Salami

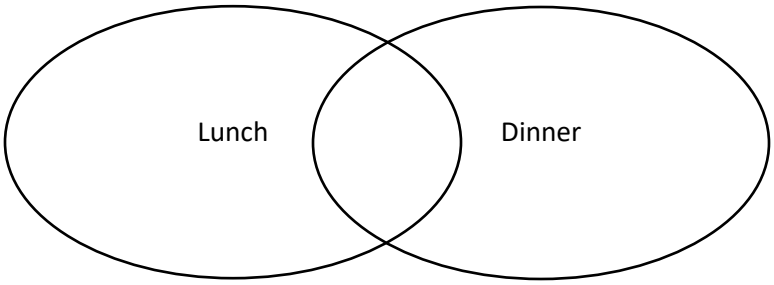
Shade in the area representing the results of this query:



In the next example, the INTERSECT operator is used to return only the **food that was served at both meals**:

SELECT item FROM Dinner **INTERSECT** **SELECT item FROM Lunch;**	**item** Bread Coffee Olives

Shade in the area representing the results of this query:

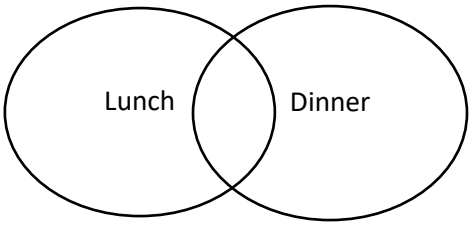


One last example…

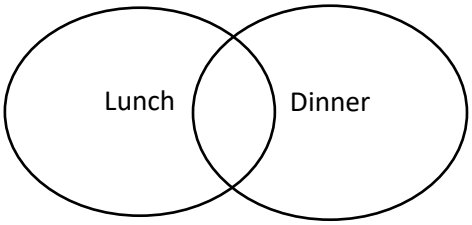
Now let's look at how you would return a list of food that you served at one of the meals, but not both meals, in other words, the food you served other than bread, olives, and coffee. In the following statement, a UNION operator is used to join two SELECT statements:

```
SELECT item FROM
 (
     SELECT item FROM Lunch
     EXCEPT
     SELECT item FROM Dinner
 ) As Only_Lunch
UNION
SELECT item FROM
 (
     SELECT item FROM Dinner
     EXCEPT
     SELECT item FROM Lunch
 ) As Only_Dinner;
```

Shade in results of first subquery:



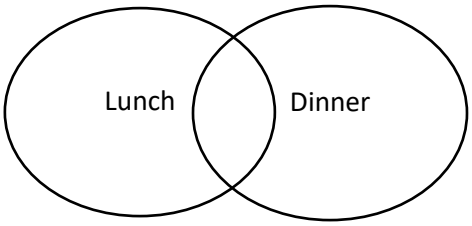
Shade in results of second subquery:



Notice that first statement retrieves only the food you served for lunch, and the second statement retrieves only the food served for dinner. The Union operator joins both result sets into one:

	item
1	Apple
2	Calamari
3	eggplant
4	Juice
5	Pop
6	Salad
7	Salami
8	Steak

Shade in results of final result set:

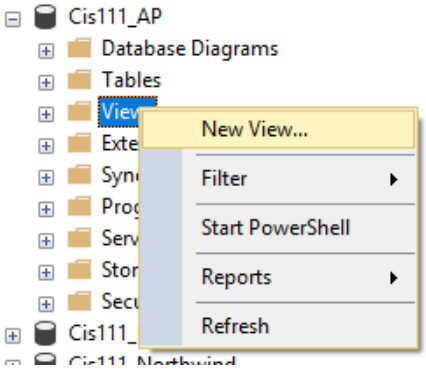


Lab Activity

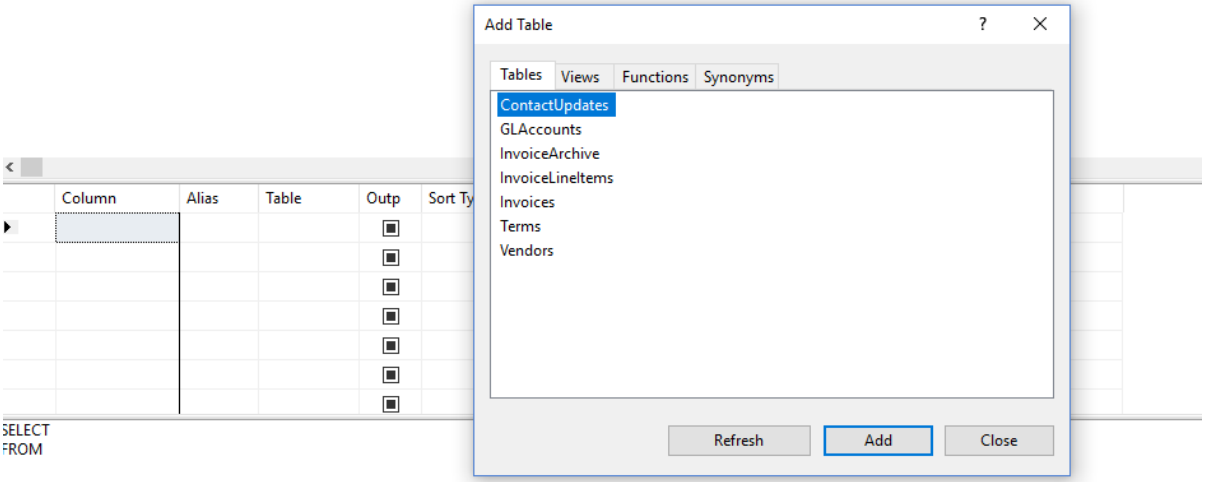
You will complete the exercises located at the end of chapter 4. The instructor will demonstrate how you can user the **Query Designer** to complete these problems.

How do you get into the Query Designer? You will go through an example of using the Query Designer by working through a query problem:
Query problem: Create a query to list the Vendor name, Invoice date and total for invoice totals over 500. Display the results by vendor name.

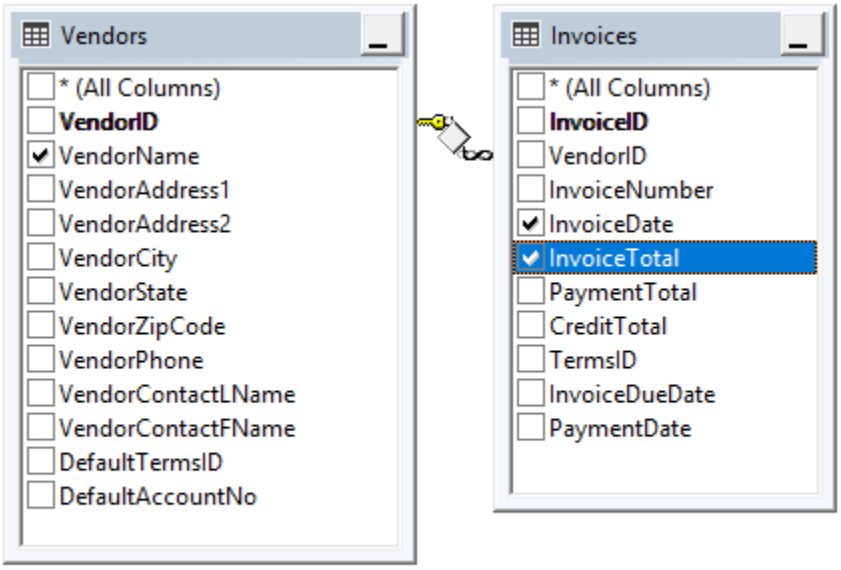
1. **Right-click** the **View folder** contained in the database you want to query, then click **New View…**:



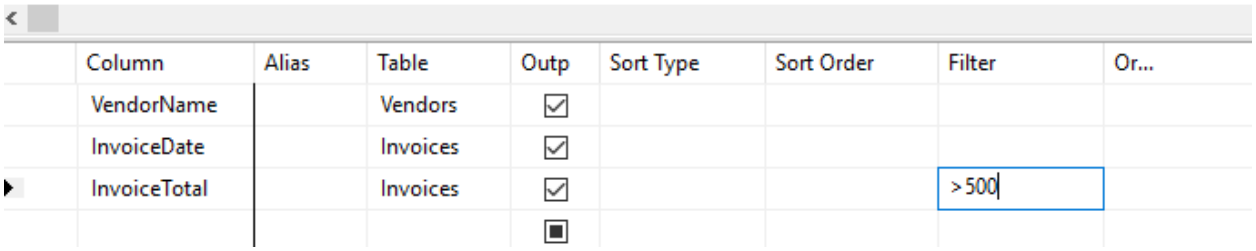
2. An Add Table dialog will appear. Click the Vendors table first and then the Invoices table.



3. Expand each table so you can see all fields for each table. Click the check box for each field needed in the Select statement:

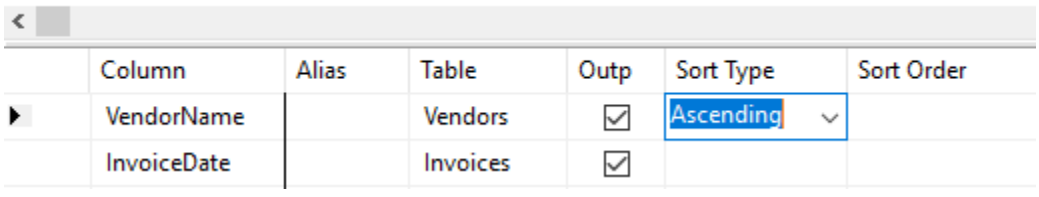


4. In the grid containing the field names enter **> 500** in the **filter** column for the **InvoiceTotal row**.



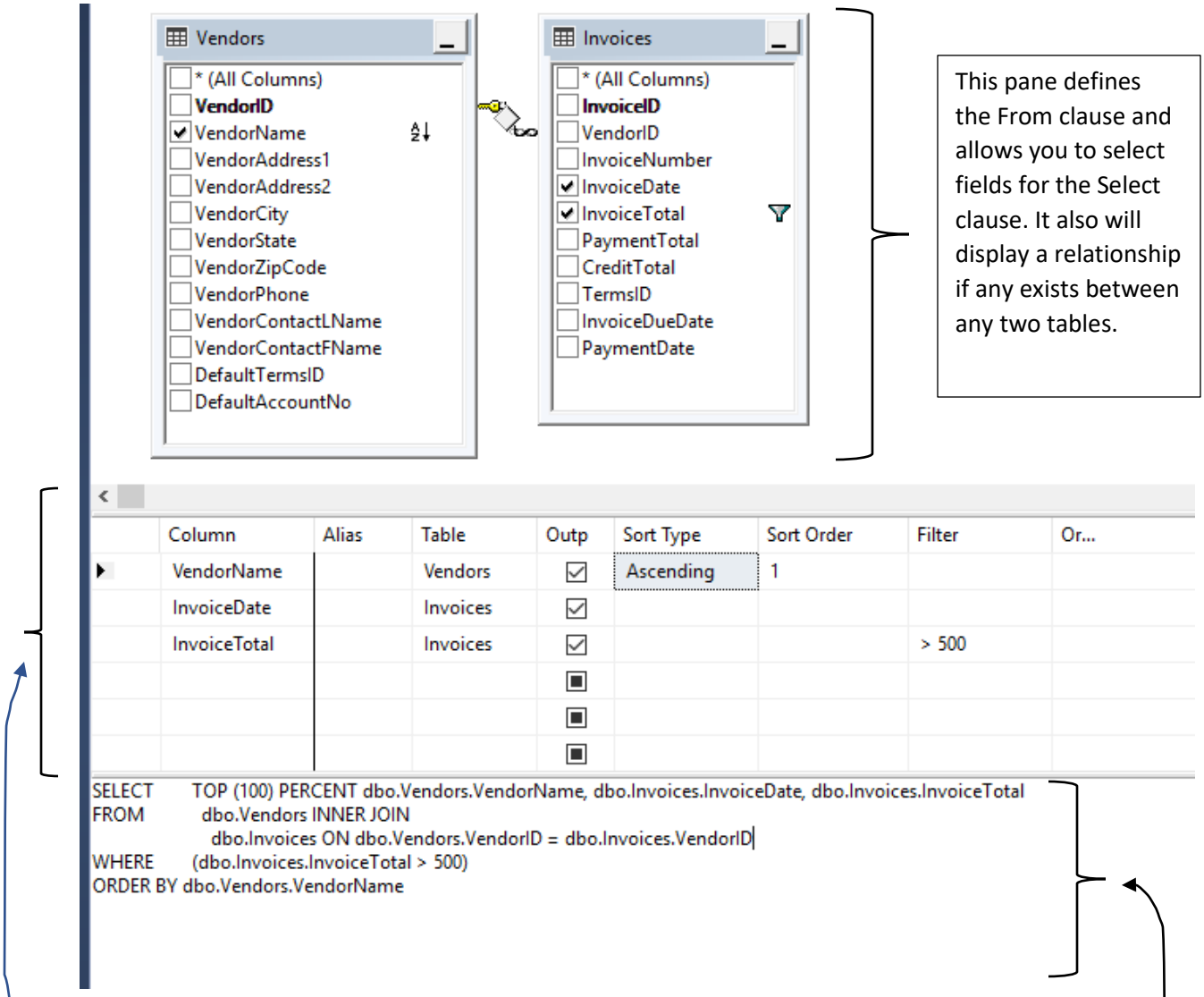
Column	Alias	Table	Outp	Sort Type	Sort Order	Filter	Or...
VendorName		Vendors	☑				
InvoiceDate		Invoices	☑				
InvoiceTotal		Invoices	☑			> 500	
			■				

5. Select the **Ascending** sort type for the **VendorName** column.

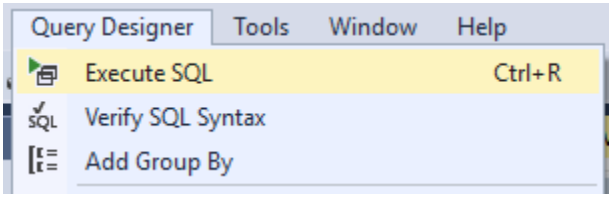


Column	Alias	Table	Outp	Sort Type	Sort Order
VendorName		Vendors	☑	Ascending	
InvoiceDate		Invoices	☑		

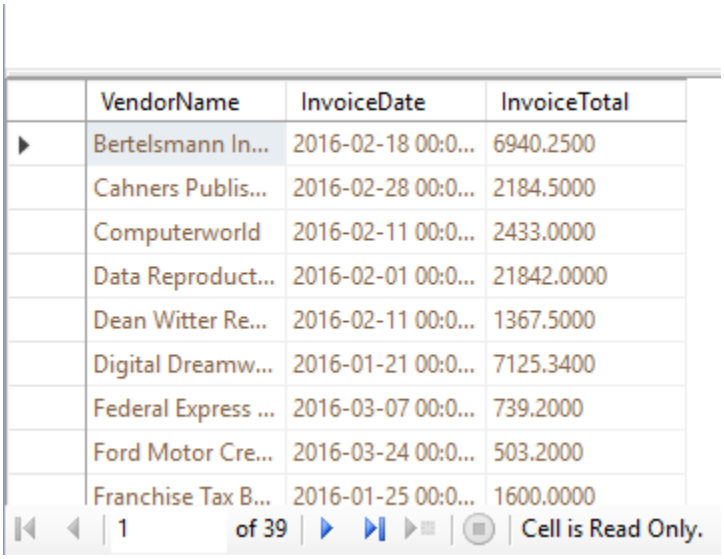
6. Note the three panes of the query designer below:



7. Since you are using the Query Designer there should be a **Query Designer** menu title. Click it and select **Execute SQL**.



A new pane should be visible showing the result set returned from the execution of the Select statement:



VendorName	InvoiceDate	InvoiceTotal
Bertelsmann In...	2016-02-18 00:0...	6940.2500
Cahners Publis...	2016-02-28 00:0...	2184.5000
Computerworld	2016-02-11 00:0...	2433.0000
Data Reproduct...	2016-02-01 00:0...	21842.0000
Dean Witter Re...	2016-02-11 00:0...	1367.5000
Digital Dreamw...	2016-01-21 00:0...	7125.3400
Federal Express ...	2016-03-07 00:0...	739.2000
Ford Motor Cre...	2016-03-24 00:0...	503.2000
Franchise Tax B...	2016-01-25 00:0...	1600.0000

1 of 39 Cell is Read Only.

You can adjust the SQL Select statement if you need to and then execute the statement again. Lastly, you can copy/paste the SQL Select statement to where you need to use it.