

Module 3: Writing Effective Queries

Designing for Performance
Kimberly L. Tripp

Overview

- **The Anatomy of a SELECT statement**
- **The Consistency List**
- **Data Retrieval**
- **Search Arguments**
- **Function “Top 10” List**
- **Inner/Outer Joins with Search Arguments**
- **Extra Info:**
 - **Comparison Operators**
 - **Related Table Info**
 - **Database Diagramming**
 - **Inner/Outer Joins**

The Anatomy of a SELECT

```
SELECT expression AS [column header],  
       expression AS [column header]  
FROM database.owner.tablename AS alias  
WHERE expression operator expression  
       logical_operator  
       expression operator expression
```

- **Expression is often a single column name but can include more complex computations, functions, concatenation, subqueries, etc.**
- **Not all three (SELECT, FROM, WHERE) components are required – but recommended!**
- **Eg. SELECT getdate()**

The Consistency List

- **Minimize data requested/returned**
 - Projection
 - Selection
 - Joins
- **Avoid Ambiguity**
- **Use table aliases**
- **Descriptive Column Headers**
- **Delimiters**
- **Single Quotes for character strings**
- **Formatting**

Accessing Data – Limit Data

- **Subset of columns = Projection**
 - Do not use *
 - Optimizer has more chances for optimizing query when NARROW
- **Subset of rows = Selection**
 - Use positive search arguments
 - Isolate the column to one side of the expression
 - USE: Birthday <= dateadd(yy, -21, getdate())
 - DO NOT USE: dateadd(yy, 21, Birthday) <= getdate()
 - Be cautious with LEADING wildcards
 - USE: LastName LIKE 'S%'
 - Try not to just append %value% to every application
- **Consider using Views, Stored Procedures and Functions to limit the columns/rows**

Avoid Ambiguity

- **When accessing objects always owner qualify objects**
 - USE: SELECT... FROM dbo.tablename
 - NOT: SELECT... FROM tablename
 - It works but it's ambiguous, causes overhead (cache lookup) and reduces overall scalability
 - USE: EXEC dbo.procname
 - NOT: EXEC procname
- **Functions REQUIRE two part naming to avoid function namespace collisions**
 - MUST USE:
 - SELECT dbo.fn_name(column)
 - FROM dbo.tablename

Aliasing Tables/Views

- Avoid ambiguity within your query

```
SELECT a.au_id, a.au_lname, ...  
FROM dbo.authors AS a  
WHERE a.city = 'Oakland'
```
- Alias every table, qualify every column – even for single table queries...why, best practice leads to:
 - Consistency in query format
 - Readability
 - Simplicity in analysis for performance tuning (can easily reference which column comes from which table)
- Adding Database Name not a bad idea!

Descriptive Column Headers

- Expression AS [Column Header] is recommended as is current standard
- [Column Header] = Expression is common but not current standard
- Expression [Column Header] is supported but often a mistake where you might be missing a column – always make sure to verify your output/data!

Delimiters

- Used to support non-standard identifiers
 - Identifiers that include spaces
 - Identifiers cannot begin with a number
 - Certain characters have special meaning
- Brackets - always on
 - USE: [owner].[identifier]
- Double Quotes – on when:
 - Session setting is turned on:
 - Use DBCC USEROPTIONS to see what are set
 - Turn on with SET QUOTED_IDENTIFIER ON
 - If using OSQL turn on with -I or /I
 - IF session setting is not on then you're not sure if it's on or not unless you look at database option
DATABASEPROPERTYEX('IsQuotedIdentifiersEnabled')

Character Strings

- All character strings should be quoted with SINGLE quotes
- Quotes within single quotes should be quote/quote - two single quotes NOT double quotes
 - USE: WHERE a.au_lname = 'O'Leary'
 - NOT: WHERE a.au_lname = "O'Leary"
- Double quotes should be avoided unless used for identifiers – should NOT turn quoted_identifier off in order to “simplify” your query – can cause performance problems...really!

Formatting

- Will save you time later
- Can make your query easier to read
- Makes aliases, columns and various parts of more complex queries – easier to find
- Recommendations – have a standard:
 - KEYWORDS in uppercase
 - Primary keywords lined up at left with a new line for each (SELECT, FROM, WHERE, GROUP BY, ORDER BY, COMPUTE BY...)
 - Line up commas (easier to confirm, easier to comment out without syntax error near comma)

Fully Qualified Names

- Fully Qualified Names
server.database.owner.object
server.catalog.schema.object
- Partially Specified Names
 - Server defaults to connected server
 - Database (or Catalog) defaults to current database (if in QA MUST be dropdown NOT object browser)
 - Owner (Schema) defaults to the current user name in the database – but if this check fails will look to see if object is “generically” owned by the “dbo”

Search Arguments

- Using the WHERE clause to limit rows
 - Using Comparison Operators
 - Using String Comparisons
 - Retrieving a Range of Values
 - Using a List of Values as Search Criteria
 - Retrieving Unknown Values
- Combining conditions using Logical Operators
- Understanding Operator Precedence

Function “Top 10” List

- ISNULL(), NULLIF(), COALESCE()
- getdate(), getutcdate()
- datediff(), datepart(), datename()
- CONVERT, CAST, CONVERT with style (for formatting datetime data)
- LEFT(), RIGHT(), SUBSTRING()
- UPPER(), LOWER()
- Metadata – SERVERPROPERTY(), DATABASEPROPERTYEX(), OBJECTPROPERTY()
- OBJECT_NAME(ID), OBJECT_ID('name')
- @@version, @@error, @@trancount, @@rowcount
- AVG(), MIN(), MAX(), SUM(), COUNT(exp), COUNT(*)
- See “functions” in the BOL

Function Examples

● Datetime Functions

```
SELECT getdate() + 1
```

```
SELECT dateadd(yy, 3, getdate())
```

```
SELECT datediff(yy, '20000914', '20040101')
```

Be careful with DATEDIFF – Only takes out the boundaries requested (year boundary) 2004-2000

```
SELECT CONVERT(char(10), getdate(), 102) AS [yyyy.mm.dd]
```

Always use a column header with a style change on date!

● Aggregate Functions

```
SELECT AVG(t.price) FROM pubs.dbo.titles AS t
```

```
SELECT COUNT(t.price) FROM pubs.dbo.titles AS t
```

```
SELECT COUNT(*) FROM pubs.dbo.titles AS t
```

Count of a column is the count of not NULL values and count(*) is a row count. If the column does not allow null values then count(column) will be as efficient as count(*).

Function Examples

● Object Properties and Metadata

```
SELECT object_id('authors')
```

```
SELECT OBJECTPROPERTY(object_id('authors'),  
    'TableHasPrimaryKey')
```

```
SELECT OBJECTPROPERTY(object_id('byroyalty'),  
    'IsProcedure')
```

Avoid system table queries to determine an object's existence. Do NOT use system table queries – what if they (the system table names) were to change?

```
IF EXISTS (SELECT * FROM sysobjects WHERE  
    name = 'byroyalty' AND type = 'P')
```

Adding Search Arguments

- Adding a restriction to an INNER Join
 - Should it apply before or after the join?
 - Does it matter?
 - All conditions MUST be true
- Should be added to the WHERE clause

```
SELECT c.CustomerID AS [Customer Number]
      , c.FirstName + ' ' + c.LastName AS [Customer Name]
      , p.ProductName AS [Product Name]
      , o.Quantity
FROM   dbo.Customer AS c
      INNER JOIN dbo.[Order] AS o
            ON c.CustomerID = o.CustomerID
      INNER JOIN dbo.Product AS p
            ON p.ProductID = o.ProductID
      INNER JOIN dbo.Category AS Cat
            ON Cat.CategoryID = p.CategoryID
WHERE  cat.CategoryName = 'Water-related'
```

Adding Search Arguments

- Adding a restriction to an OUTER Join
 - Can be added to FROM clause OR WHERE clause
 - Does it matter? YES!
 - Completely different queries
- Adding restriction in FROM clause
 - Inner Join – intersection defined by FROM clause
 - Outer Join – adds rows that did not qualify
- Adding restriction in WHERE clause
 - All rows must adhere to this condition

ORDER BY Clause

- Only way to guarantee sort order
 - Expressions from SELECT list
 - Header from SELECT list (more readable)
 - Expressions, columns, etc. NOT included as output
 - If included as output special syntax allowed

```
SELECT p.CategoryID AS [Category ID]
      , p.ProductName AS [Product Name]
      , p.UnitPrice AS [Price]
FROM Northwind.dbo.Products AS p
ORDER BY [Category ID] ASC, [Price] DESC
ORDER BY p.CategoryID ASC, p.UnitPrice DESC
ORDER BY 1 ASC, 3 DESC
-- OR even something NOT included in SELECT list
ORDER BY p.CategoryID, UnitsInStock DESC
```

Review

- The Anatomy of a SELECT statement
- The Consistency List
- Data Retrieval
- Search Arguments
- Function “Top 10” List
- Inner/Outer Joins with Search Arguments
- Extra Info:
 - Comparison Operators
 - Related Table Info
 - Database Diagramming
 - Inner/Outer Joins

Using Comparison Operators

```
SELECT e.Lastname, e.City
FROM Northwind.dbo.Employees AS e
WHERE e.Country = 'USA'

SELECT e.Lastname, e.City
FROM Northwind.dbo.Employees AS e
WHERE e.EmployeeID > 'USA'
```

- **Standard comparison operators:**

- =, >, <, >=, <= and <>
- **USE:** WHERE a.city <> 'Oakland'
- **AVOID:** WHERE a.city != 'Oakland'
- **Or using the 'NOT' logical operator:**
NOT(WHERE a.city = 'Oakland')

Using Wildcards

- **Must use LIKE for pattern matching**
- **Wildcards supported:**
 - % = 0 to many characters
 - _ = Exactly 1 character
 - [] are used for a character list
 - ^ specifies NOT this character
- **Escape character supported**
 - Must be a single character

```
SELECT a.au_lname AS [Last Name]
      , a.au_fname AS [First Name]
FROM pubs.dbo.authors AS a
WHERE a.au_lname LIKE 'M%'
WHERE a.au_lname LIKE '[S-Z]%'
WHERE a.au_lname LIKE '[CK]ars_n'
WHERE a.au_lname LIKE 'M[^c]%'

SELECT so.[Name]
FROM master.dbo.sysobjects AS so
WHERE so.[Name] LIKE 'sp_%'
WHERE so.[Name] LIKE 'sp[_]%'
WHERE so.[Name]
      LIKE 'sp#_%' ESCAPE '#'
```

Ranges – Between

```
SELECT p.ProductName, p.UnitPrice
FROM Northwind.dbo.Products AS p
WHERE p.UnitPrice BETWEEN 10 AND 20
ORDER BY p.UnitPrice
```

```
SELECT p.ProductName, p.UnitPrice
FROM Northwind.dbo.Products AS p
WHERE p.UnitPrice NOT BETWEEN 10 AND 20
ORDER BY p.UnitPrice
```

- **Between is INCLUSIVE**
- **Between is re-written to be $\geq$ lower value and $\leq$ higher value**
- **If you don't want it to be inclusive and really want values only "between" then you need $>$ lower value and $<$ higher value**

Ranges – IN

```
SELECT s.CompanyName, s.Country
FROM Northwind.dbo.Suppliers AS s
WHERE s.Country IN ('Japan', 'Italy', 'Germany')

SELECT s.CompanyName, s.Country
FROM Northwind.dbo.Suppliers AS s
WHERE s.Country NOT IN ('Japan', 'Italy', 'Germany')

SELECT s.CompanyName, s.Country
FROM Northwind.dbo.Suppliers AS s
WHERE s.Country IN ('Japan', 'Italy', 'Germany')
OR s.Country LIKE 'S%'
```

- **The list used with IN is for EXACT matches only**
- **Wildcards are NOT supported**
- **IN is re-written to be column = *firstvalue* OR column = *secondvalue* OR...**
- **If you need wildcards you must combine with OR**

NULL Values

```
SELECT t.title, t.price
FROM pubs.dbo.titles AS t
WHERE t.price IS NULL

SELECT t.title, t.price
FROM pubs.dbo.titles AS t
WHERE t.price IS NOT NULL

SELECT t.title, ISNULL(t.price, 0)
FROM pubs.dbo.titles AS t
```

- **NULL Values must be evaluated using IS or IS NOT**
- **Do NOT use column = NULL when ANSI_NULLS is ON**
- **Do NOT turn ANSI_NULLS off as it can have a negative impact on performance**
- **Replacing a NULL Value with a “real” value can be easy with ISNULL function**

Operator Precedence

```
SELECT t.type, t.title, t.pub_id, t.price
FROM pubs.dbo.titles AS t
WHERE t.type = 'business' OR t.title LIKE '%comp%'
      AND t.price > 19.99

SELECT t.type, t.title, t.pub_id, t.price
FROM pubs.dbo.titles AS t
WHERE (t.type = 'business' OR t.title LIKE '%comp%')
      AND t.price > 19.99

SELECT t.type, t.title, t.pub_id, t.price
FROM pubs.dbo.titles AS t
WHERE t.type = 'business' OR (t.title LIKE '%comp%'
      AND price > 19.99)
```

- **Which two queries produce the same results?**
- **Hint: 1 and 2 OR 1 and 3**
- **Which one are you struggling with?**

Joins

- Introduction to Joins
- Finding metadata for table relationships
- Using Inner Joins
- Using Outer Joins
- Joining a table to itself
- Using search arguments in a Join

Related Data

- Entities in a relational database can be joined to create detailed output of related data from multiple tables
- FROM clause defines all tables to be joined and their join types
- ON clause (within FROM clause) defines the criteria on which the join is performed
- Usually based on Primary/Foreign Key relationship
- Joins v. subqueries
 - Might be able to re-write a query in multiple ways
 - For important/high priority queries you might consider re-writing if performance is not acceptable.

Accessing Metadata

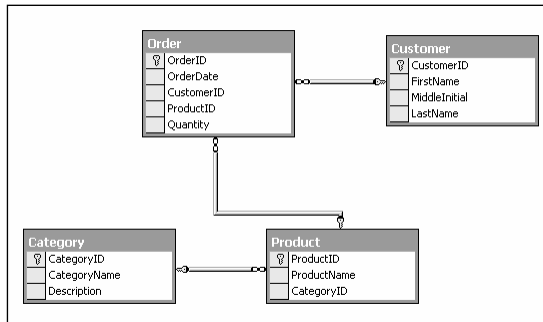
- In SQL Enterprise Manager
 - Database Diagramming Wizard
 - Graphical Query Designer
- In Query Analyzer
 - Object Browser
 - sp_help (highlight an object, press ALT+F1)
 - sp_helpconstraint

Note: Other key stroke information... Can program in other key strokes like Ctrl+3 for sp_helpconstraint. Use Tools, Customize
For list of doc'ed keystrokes see BOL (click the keyboard at the top of the right pane)

Database Diagramming Wizard

- Only accessible if:
 - You have SELECT on all of the tables
 - Cannot “open table” without SELECT rights
 - Cannot create a new diagram and add tables you don't have select rights on
 - Might be able to see other diagrams
 - Really works best with db_datareader, db_owner or higher (i.e. system administrator)
- Can automatically grab related tables – choose this ☒ checkbox before selecting tables
- Can select a subset of tables just to get that set displayed and those relationships described
- Not great for printing unless you manually tweak and print to plotter
- Other tools better – Visio Enterprise Architect

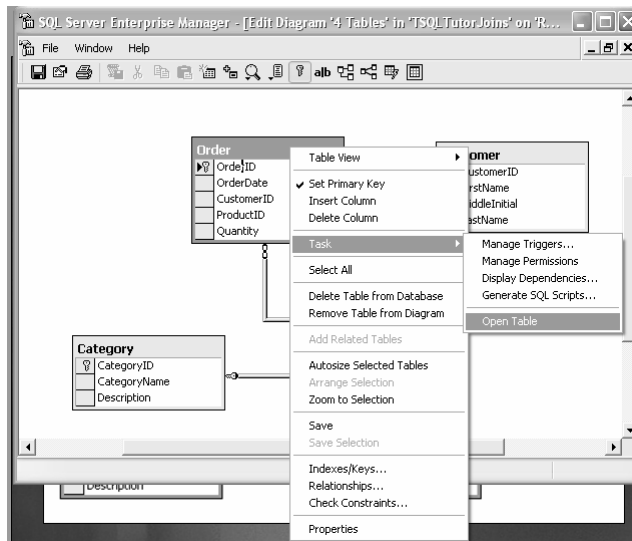
SQLEM – DB Diagram



- Each Order has **ONE** Customer
- A Customer can have many Orders
- Each Order has **ONE** Product

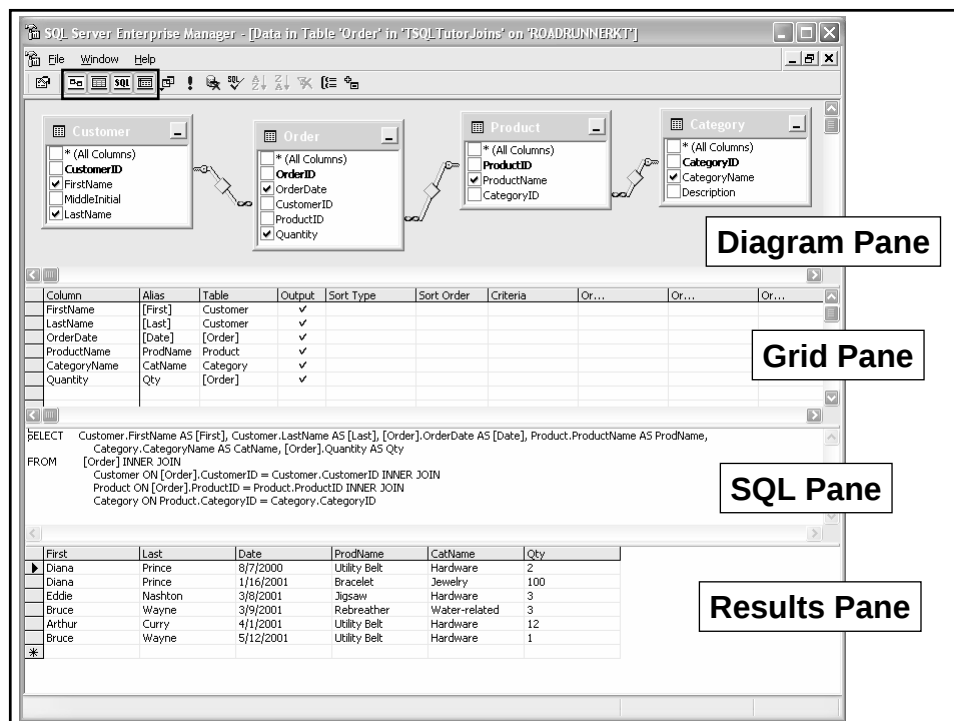
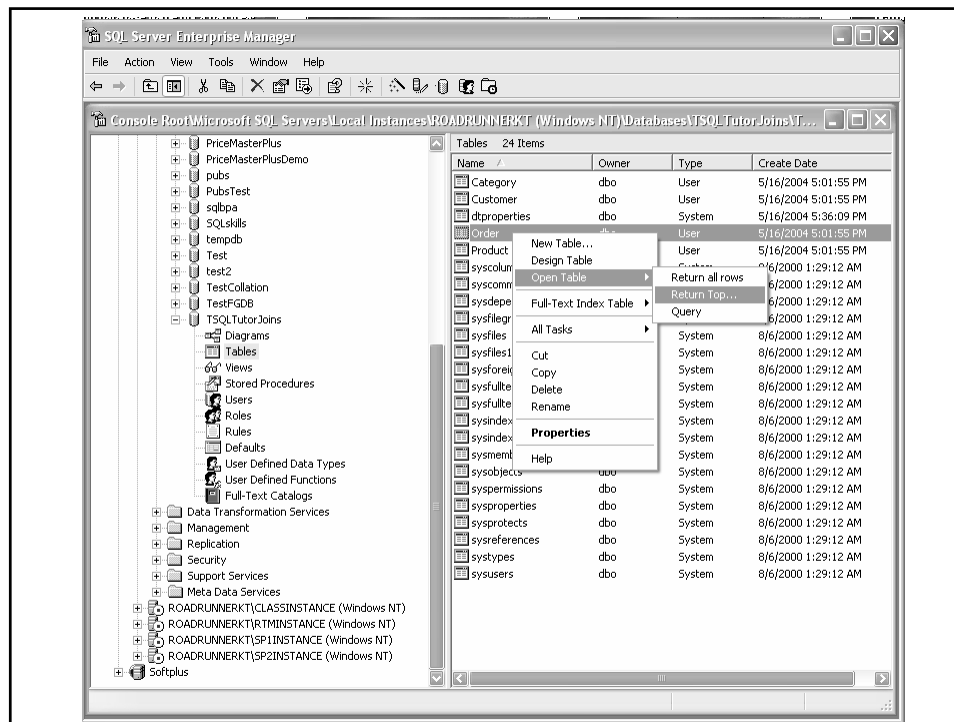
- Key to Infinity – 1 to many relationship
- Key to key – 1 to 1 relationship
- A middle table which has two infinity symbols meeting at it – represents a many to many relationship

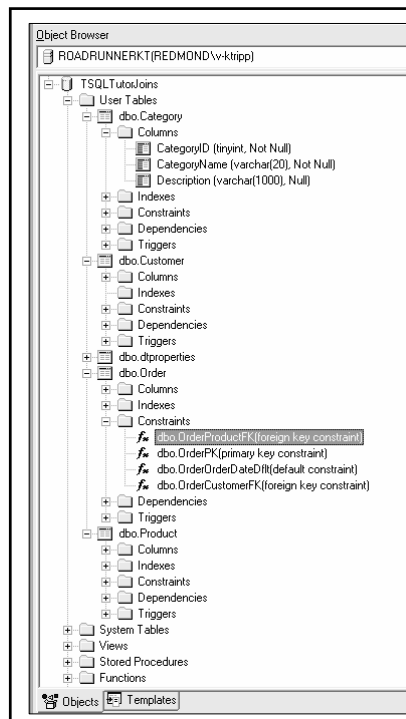
SQLEM – Query Designer



- Right Click
- Choose Task
- Open Table
- This will open the query designer...
- You cannot choose to limit the number of rows!

Is there another way?





Object Browser

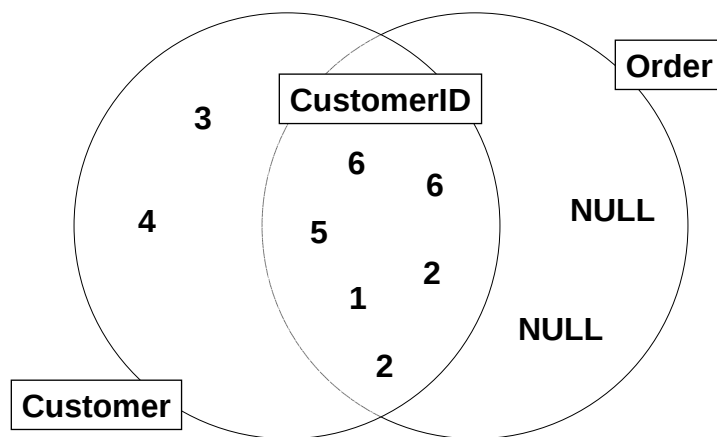
- Shows User Tables
- For each table:
 - Columns (type/nullability)
 - Indexes
 - Names of constraints
 - Object Dependencies
 - Triggers
- Can show/hide object browser with F8
- Can interact
 - Right click to “script” – good for SELECT
 - Drag/drop for column list

Join – Terms & Conditions

- Inner Join – determines intersection between two or more tables
- Outer Join – adds rows which did not qualify (from the outer table)
- Self Join – Joining a table to itself, used when self-referencing keys exist
- Cross Joins – Usually a mistake ☺ but often for generating sample data or projecting row sets
- Search Arguments – can apply to the entire set or the join condition...

Inner Joins = Intersection

```
SELECT c.CustomerID AS [Customer Number]
      , c.FirstName + ' ' + c.LastName AS [Customer Name]
FROM   dbo.Customer AS c
       INNER JOIN dbo.[Order] AS o
           ON c.CustomerID = o.CustomerID
```



Writing Joins in QA

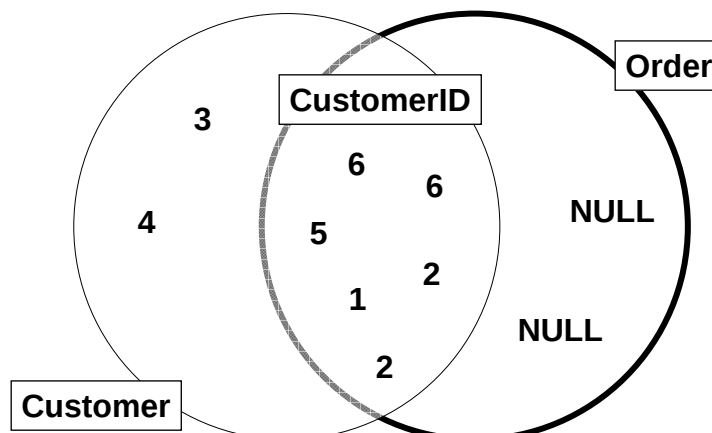
- **FROM clause first**
 - Establish alias
 - Qualify table name (always use two-part, may need three-part for cross database joins)
- **ON clause second**
 - Establish type of join
 - Define ALL criteria for join
- **Other clauses next**
 - SELECT List
 - WHERE clause
 - ORDER by

How does aliasing work?

- How can you put aliases in the FROM clause and then use them in the SELECT list?
- Query goes through phases
 - Parse (SELECT...FROM...WHERE...)
 - Resolve, Standardize, Normalize
 - Optimize
 - Compile
 - Execute
- Resolution happens after parsing and resolves table names, then column names, etc.

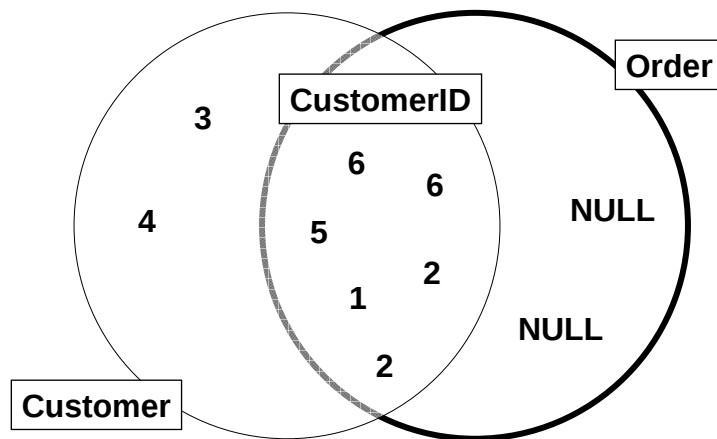
Inner Joins – Not Enough?

What if you want to see ALL orders regardless of whether or not they were purchased by a valid customer?



Outer Joins

```
SELECT c.CustomerID AS [Customer Number]
      , c.FirstName + ' ' + c.LastName AS [Customer Name]
FROM   dbo.Customer AS c
      RIGHT OUTER JOIN dbo.[Order] AS o
      ON c.CustomerID = o.CustomerID
```



Left v. Right – Any difference?

```
SELECT c.CustomerID AS [Customer Number]
      , c.FirstName + ' ' + c.LastName AS [Customer Name]
FROM   dbo.Customer AS c
      RIGHT OUTER JOIN dbo.[Order] AS o
      ON c.CustomerID = o.CustomerID
```

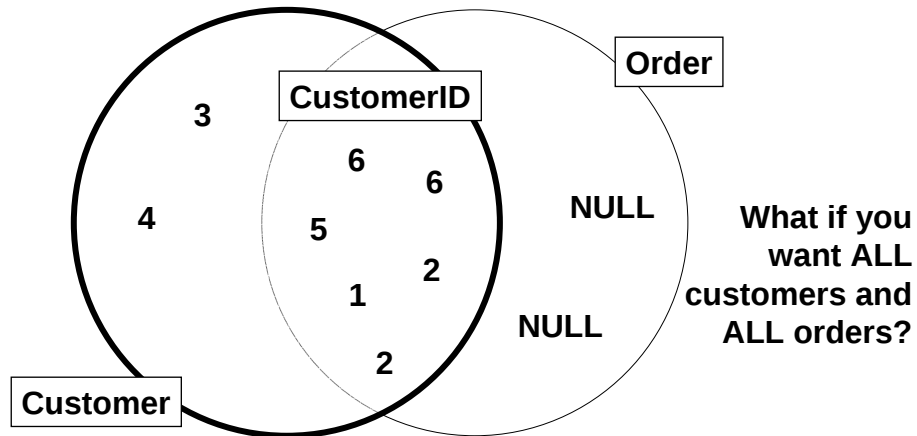
```
SELECT c.CustomerID AS [Customer Number]
      , c.FirstName + ' ' + c.LastName AS [Customer Name]
FROM   dbo.[Order] AS o
      LEFT OUTER JOIN dbo.Customer AS c
      ON c.CustomerID = o.CustomerID
```

- In two table joins – there's no difference between LEFT and RIGHT except for where you place the table within the query (relative to the word JOIN)
- In n-table joins can make a difference in the result set returned
- Personal preference is LEFT

Outer Join

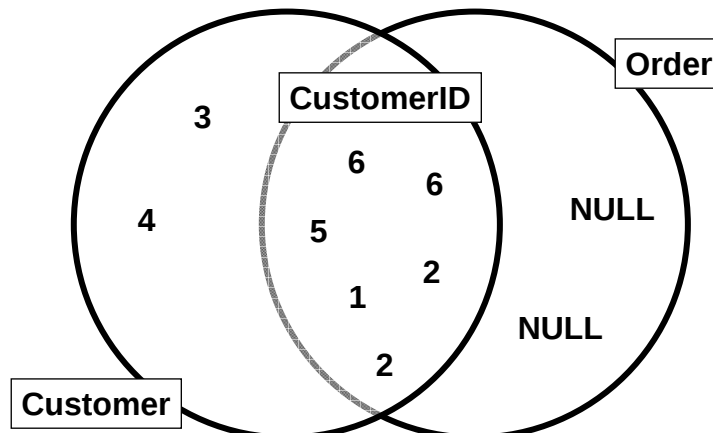
What if you want to see **ALL** customers
regardless of whether or not they
placed an order?

Make Customer the **OUTER** table



FULL Outer Join

```
SELECT c.CustomerID AS [Customer Number]
      , c.FirstName + ' ' + c.LastName AS [Customer Name]
FROM   dbo.Customer AS c
      FULL OUTER JOIN dbo.[Order] AS o
      ON c.CustomerID = o.CustomerID
```



Joining a Table to Itself

EmployeeSelfJoin

EmpID	LName	MgrID
1	Smith	NULL
2	Tripp	1

```

SELECT E.FName + ' ' + E.LName AS 'Employee Name',
       M.FName + ' ' + M.LName AS 'Manager Name'
FROM   dbo.EmployeeSelfJoin AS E
       INNER JOIN dbo.EmployeeSelfJoin AS M
       ON E.MgrID = M.EmpID
  
```

Employee Name	Manager Name
Kimberly Tripp	Bob Smith

*What about Bob?
As an "Employee" Bob should show up...*

```

--So how about a LEFT OUTER SELF Join?!
SELECT E.FName + ' ' + E.LName AS 'Employee Name',
       M.FName + ' ' + M.LName AS 'Manager Name'
FROM   dbo.EmployeeSelfJoin AS E
       LEFT OUTER JOIN dbo.EmployeeSelfJoin AS M
       ON E.MgrID = M.EmpID
  
```

Employee Name	Manager Name
Bob Smith	NULL
Kimberly Tripp	Bob Smith

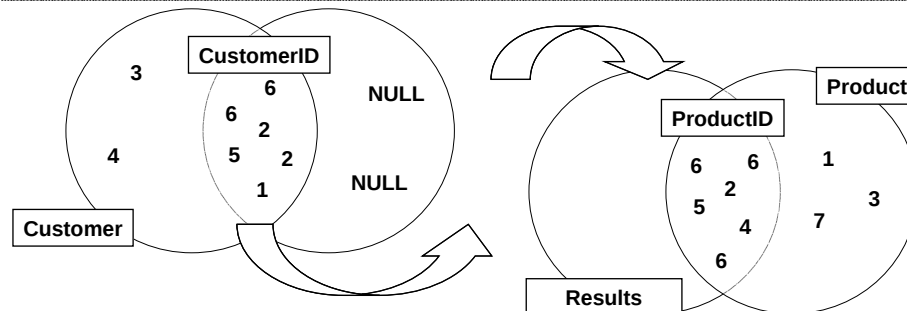
Adding More Tables

- Order is not relevant for performance
- Processing Order and internal join type are determined by SQL Server (unless hints used)
- Added table only needs to join to ANY previously listed table
- Watch logic if LEFT/RIGHT/FULL are used – may need to continue adding LEFT/RIGHT/FULL to preserve rows
- Might consider parenthesis to separate LEFT/RIGHT/FULL

n-Table Inner Joins

```

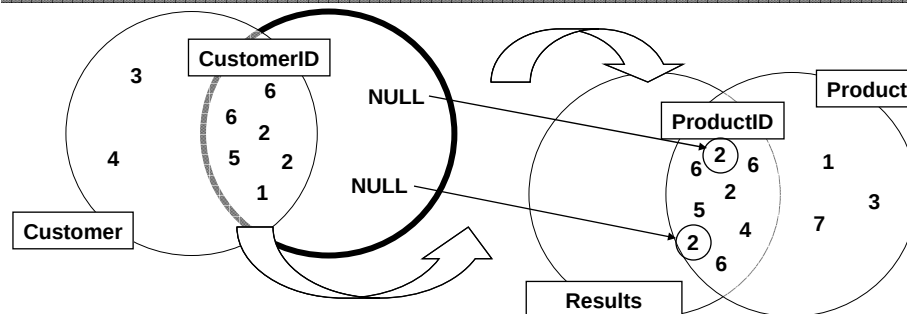
SELECT c.CustomerID AS [Customer Number]
      , c.FirstName + ' ' + c.LastName AS [Customer Name]
      , p.ProductName AS [Product Name]
      , o.Quantity
FROM dbo.Customer AS c
     INNER JOIN dbo.[Order] AS o
           ON c.CustomerID = o.CustomerID
     INNER JOIN dbo.Product AS p
           ON p.ProductID = o.ProductID
  
```



n-Table Outer Joins

```

SELECT c.CustomerID AS [Customer Number]
      , c.FirstName + ' ' + c.LastName AS [Customer Name]
      , p.ProductName AS [Product Name]
      , o.Quantity
FROM dbo.Customer AS c
     RIGHT OUTER JOIN dbo.[Order] AS o
           ON c.CustomerID = o.CustomerID
     INNER JOIN dbo.Product AS p
           ON p.ProductID = o.ProductID
  
```

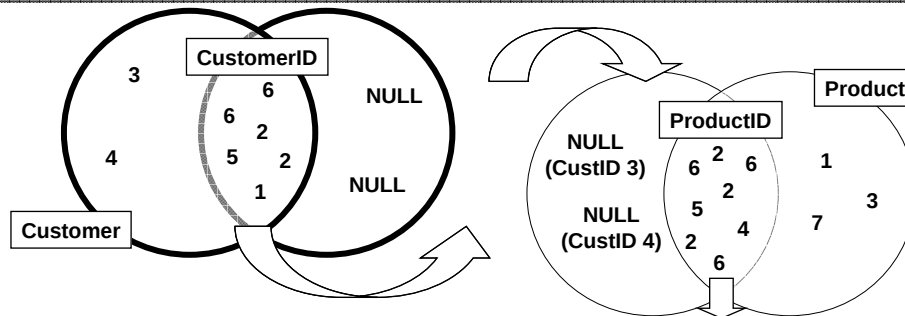


n-Table Outer Joins

```

SELECT c.CustomerID AS [Customer Number]
      , c.FirstName + ' ' + c.LastName AS [Customer Name]
      , p.ProductName AS [Product Name]
      , o.Quantity
FROM   dbo.Customer AS c
      FULL OUTER JOIN dbo.[Order] AS o
              ON c.CustomerID = o.CustomerID
      INNER JOIN dbo.Product AS p
              ON p.ProductID = o.ProductID
  
```

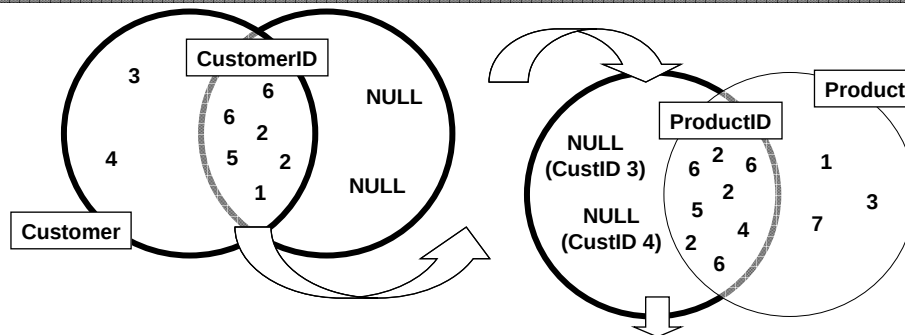
**What
happens
here?**



n-Table Outer Joins

```

SELECT c.CustomerID AS [Customer Number]
      , c.FirstName + ' ' + c.LastName AS [Customer Name]
      , p.ProductName AS [Product Name]
      , o.Quantity
FROM   dbo.Customer AS c
      FULL OUTER JOIN dbo.[Order] AS o
              ON c.CustomerID = o.CustomerID
      LEFT INNER JOIN dbo.Product AS p
              ON p.ProductID = o.ProductID
  
```



Joins Summary

- Write your FROM clause first
- Add tables slowly
- Try to determine if your result set is correct before you add next table, next condition, etc.
- Consider adding parenthesis to isolate groupings of joins