Introduction to T-SQL Queries

Kathi Kellenberger

Redgate Software

Kathi Kellenberger

- Editor and Advocate Redgate Software

- /Kathikellenberger
- @auntkathi
- http://auntkathisql.com
- Kathi.Kellenberger@red-gate.com

- Lifelong learner
- Teacher

- Co-leader of PASS WIT
- Data Platform MVP

Agenda

- Class 1
  - Module 1: Introduction
  - Module 2: Simple select statements
  - Module 3: Filtering
- Class 2
  - Module 4: Expressions
  - Module 5: Joining
- Class 3
  - Module 5: Joining (Continued)
  - Module 6: Grouping
- Class 4
  - Module 7: Subqueries
  - Module 8: UNION

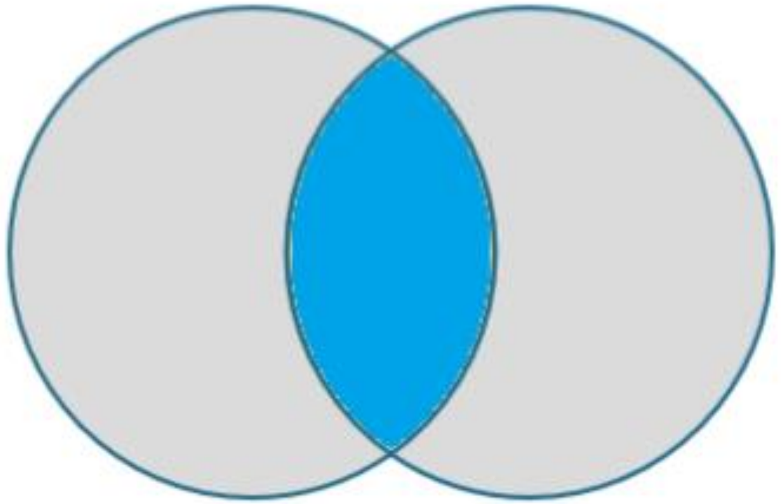
Module 5:
Joining Tables

INNER JOIN

- The columns from two tables where there is a match on a key
- Syntax

SELECT <table1>.<col1>,<table2>.<col2>

FROM <table1> [INNER] JOIN <table2>
ON <table1>.<col1> = <table2>.<col1>



Old join syntax: Comma join (Don’t use!)

```
SELECT Col1, Col1
FROM table1, table2
Where table1.col1 = table2.col1
```

- Used more often by Oracle developers than SQL Server devs

INNER JOIN

Customer

CustomerID (Primary Key)	Name
1	John
2	Sharon
3	Dana
4	Fox

Sale

SaleID (Primary Key)	CustomerID (Foreign Key)	Amt
1	3	100
2	1	200
3	3	75
4	3	90
5	1	100

Query results

SaleID	CustomerID	Name	Amt
1	3	Dana	100
2	1	John	200
3	3	Dana	75
4	3	Dana	90
5	1	John	100

Demo: INNER JOIN

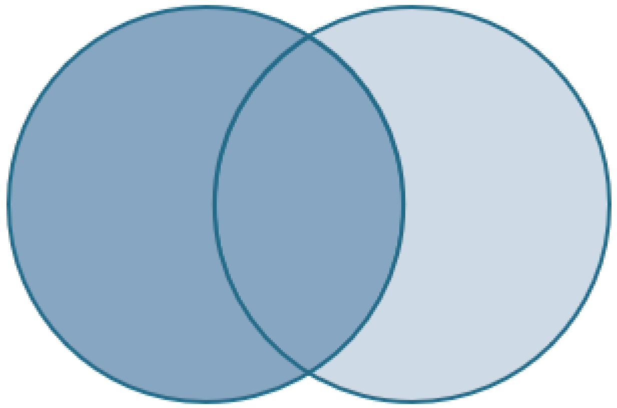
Lab

- Complete Module 5 Lab 1

LEFT OUTER JOIN

- All the rows from first table even if they don’t match
- Once you start down the left path, continue left
- Syntax

```
SELECT <table1>.<col1>, <table2><col2>
FROM <table1>
LEFT [OUTER] JOIN <table2>
ON <table1>.<col1> = <table2>.<col2>
```



LEFT OUTER JOIN

Customer

CustomerID	Name
1	John
2	Sharon
3	Dana
4	Fox

Sale

SaleID	CustomerID	Amt
1	3	100
2	1	200
3	3	75
4	3	90
5	1	100

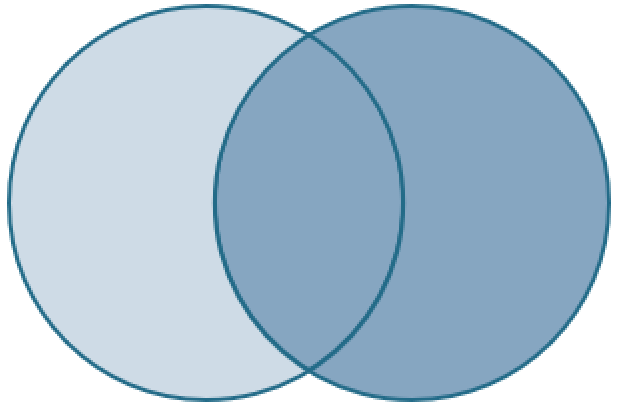
Query results

SaleID	CustomerID	Name	Amt
1	3	Dana	100
2	1	John	200
3	3	Dana	75
4	3	Dana	90
5	1	John	100
NULL	2	Sharon	NULL
NULL	4	Fox	NULL

RIGHT OUTER JOIN

- All the rows from second table even if they don’t match
- Not used as much
- Syntax

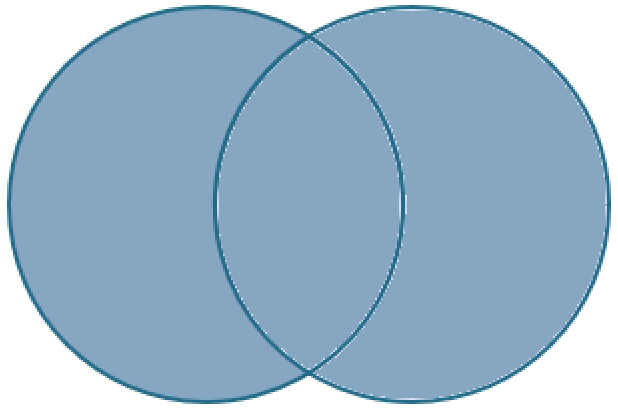
```
SELECT <table1>.<col1>, <table2><col2>
FROM <table1>
RIGHT [OUTER] JOIN <table2>
ON <table1>.<col1> = <table2>.<col2>
```



FULL OUTER JOIN

- All the rows from both tables even if they don’t match
- Rarely used
- Syntax

```
SELECT <table1>.<col1>, <table2><col2>
FROM <table1>
FULL [OUTER] JOIN <table2>
ON <table1>.<col1> = <table2>.<col2>
```



Demo: OUTER JOIN

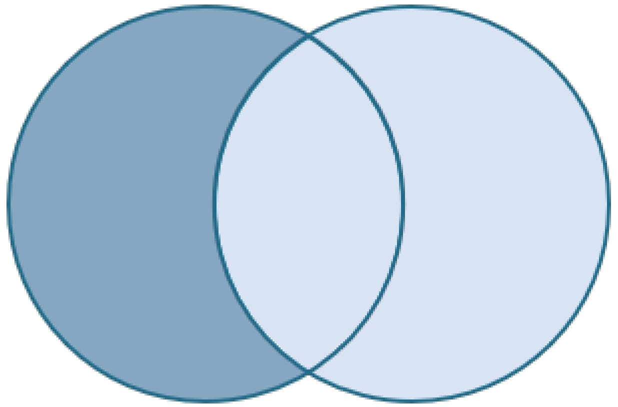
Lab

- Complete Module 5 Lab 2

LEFT OUTER JOIN with NULL RIGHT Filter

- Use to find rows that don’t match
- Filter on a key from the table on the right
- Syntax

```
SELECT <table1>.<col1>,<table2>.<col2>
FROM <table1>
LEFT [OUTER] JOIN <table2>
ON <table1>.<col1 > = <table2>.<col1>
WHERE <table2>.<col1> IS NULL
```



LEFT OUTER JOIN with NULL right table filter

Customer	
CustomerID	**Name**
1	John
2	Sharon
3	Dana
4	Fox

Sale		
SaleID	**CustomerID**	**Amt**
1	3	100
2	1	200
3	3	75
4	3	90
5	1	100

Query results			
SaleID	**CustomerID**	**Name**	**Amt**
NULL	2	Sharon	NULL
NULL	4	Fox	NULL

Demo: OUTER JOIN with FILTER

Lab

- Complete Module 5 Lab 3

Module 6:
Grouping

Grouping

- Summarize the data
- Special functions called aggregate
  - COUNT
  - SUM
  - AVG
  - MIN
  - MAX
- Return one row per group

Summary (aggregate) query

CustID	OrderID	Amt
1	101	5
2	102	10
3	103	7

```
SELECT
    COUNT(CustID) AS OrderCount,
    MIN(OrderID) AS FirstOrderID,
    SUM(Amt) SumOfOrders,
    AVG(Amt) AvgAmt
FROM Orders;
```

OrderCount	FirstOrderID	SumOfOrders	AvgAmt
3	101	22	7.33

GROUP BY clause

- Columns in the SELECT list or ORDER BY must be in an aggregate function or in the GROUP BY
- Syntax

```
SELECT <col1>, <ag function>(<col2>)
FROM <table1>
GROUP BY <col1>
```

Aggregate Queries

CustID	OrderID	TotalDue
1	1	50
2	2	76
1	3	150
1	4	25
3	5	100
3	6	30
2	7	60
1	8	90

Group by CustID

CustID	OrderID	TotalDue
1	1	50
2	2	76
1	3	150
1	4	25
3	5	100
3	6	30
2	7	60
1	8	90

CustID	OrderID	TotalDue
1	1	50
1	3	150
1	4	25
1	8	90

CustID	OrderID	TotalDue
2	2	76
2	7	60

CustID	OrderID	TotalDue
3	5	100
3	6	30

Apply any aggregate functions: SUM, AVG, COUNT…

CustID	OrderID	TotalDue
1	1	50
2	2	76
1	3	150
1	4	25
3	5	100
3	6	30
2	7	60
1	8	90

CustID	OrderID	TotalDue
1	1	50
1	3	150
1	4	25
1	8	90

COUNT(OrderID) = 4
SUM(TotalDue) =315
MAX(OrderID) = 8

CustID	OrderID	TotalDue
2	2	76
2	7	60

COUNT(OrderID) = 2
SUM(TotalDue) =136
MAX(OrderID) = 7

CustID	OrderID	TotalDue
3	5	100
3	6	30

COUNT(OrderID) = 2
SUM(TotalDue) =130
MAX(OrderID) = 6

Return 1 row per group

CustID	OrderID	TotalDue
1	1	50
2	2	76
1	3	150
1	4	25
3	5	100
3	6	30
2	7	60
1	8	90

CustID	OrderID	TotalDue
1	1	50
1	3	150
1	4	25
1	8	90

CustID	OrderID	TotalDue
2	2	76
2	7	60

CustID	OrderID	TotalDue
3	5	100
3	6	30

CustID	Order Count	Last Order	Total
1	4	8	315
2	2	7	136
3	2	6	130

```
SELECT CustID, COUNT(*) AS [Order Count], MAX(OrderID) AS [Last Order], SUM(TotalDue) AS Total
FROM Sales
GROUP BY CustID
```

DEMO: GROUP BY

Lab

- Complete Module 6 Lab 1

HAVING

- Use to filter the groups using an aggregate function
- Use WHERE to filter rows
- Use HAVING to filter groups

Return 1 row per group

CustID	OrderID	TotalDue
1	1	50
2	2	76
1	3	150
1	4	25
3	5	100
3	6	30
2	7	60
1	8	90

CustID	OrderID	TotalDue
1	1	50
1	3	150
1	4	25
1	8	90

CustID	OrderID	TotalDue
2	2	76
2	7	60

CustID	OrderID	TotalDue
3	5	100
3	6	30

CustID	Order Count	Last Order	Total
1	4	8	315
2	2	7	136
3	2	6	130

HAVING COUNT(OrderID) > 2

CustID	OrderID	TotalDue
1	1	50
2	2	76
1	3	150
1	4	25
3	5	100
3	6	30
2	7	60
1	8	90

CustID	OrderID	TotalDue
1	1	50
1	3	150
1	4	25
1	8	90

CustID	OrderID	TotalDue
2	2	76
2	7	60

CustID	OrderID	TotalDue
3	5	100
3	6	30

CustID	Order Count	Last Order	Total
1	4	8	315
2	2	7	136
3	2	6	130

CustID	Order Count	Last Order	Total
1	4	8	315

DEMO: HAVING

Lab

- Complete Module 6 Lab 2

Module 7:
Subqueries and Common Table Expressions

IN Subquery

- Use a query to generate a list for the WHERE clause

```
SELECT <column list>
FROM <schema>.<table1>
WHERE <col> IN (SELECT <col> FROM <schema>.<table2>)

SELECT <column list>
FROM <schema>.<table1>
WHERE <col> NOT IN (SELECT <col> FROM <schema>.<table2>)
```

DEMO: IN Subquery

Lab

- Complete Module 7 Lab 1

Correlated subquery

- Typically in the SELECT list
- Pull a scalar or single value into a query
- Inner query can see outer query
- Only one column allowed in the subquery
- Often used to separate logic from outer query

Correlated subquery

CustID	OrderID	Amt
1	101	5
2	102	10
1	103	70
3	104	30
2	105	90
1	106	15

```
SELECT
   CustID, OrderID, Amt
FROM Orders AS Ord;
```

CustID	OrderID	Amt	AvgAmt
1	101	5	30
2	102	10	50
1	103	70	30
3	104	30	30
2	105	90	50
1	106	15	30

```
SELECT
   CustID, OrderID, Amt,
   (SELECT AVG(AMT)
    FROM Orders
    WHERE CustID = Ord.CustID ) AS AvgAmt
FROM Orders AS Ord;
```

DEMO: Correlated Subquery

Lab

- Complete Module 7 Lab 2

Derived table

- Subquery in the FROM clause
- Also used to separate the logic
- Join to the subquery
- Outer query can see the columns in the SELECT list
- Often nested

Common table expression (CTE)

- Similar to derived table
- Defined up front
- No nesting, but one CTE can use other CTEs
- Also used to separate logic
- Previous statement MUST end with ;

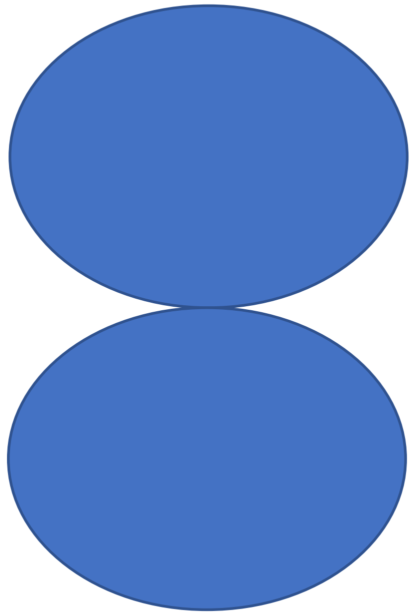
DEMO: Derived tables and CTEs

Lab

- Complete Module 7 Lab 3

UNION

- Combine the results of two queries
- Rules
  - Same number of columns
  - Compatible data types
  - Names from first query
  - ORDER BY at end
- Also UNION ALL, EXCEPT, and INTERCEPT



UNION – Eliminates Duplicates

Names

ID	Name
1	Kevin
2	Sam
3	Jane
4	Kathi

Customers

CustID	FirstName
100	Bill
120	Denise
130	Anna
4	Kathi

ID	Name
1	Kevin
2	Sam
3	Jane
4	Kathi
100	Bill
120	Denise
130	Anna

```
SELECT ID, NAME
FROM Names
UNION
SELECT CustID, FirstName
FROM Customers;
```

UNION ALL – Retains duplicates

Names

ID	Name
1	Kevin
2	Sam
3	Jane
4	Kathi

Customers

CustID	FName
100	Bill
120	Denise
130	Anna
4	Kathi

ID	Name
1	Kevin
2	Sam
3	Jane
4	Kathi
100	Bill
120	Denise
130	Anna
4	Kathi

```
SELECT ID, NAME
FROM Names
UNION  ALL
SELECT CustID, FirstName
FROM Customers;
```

Except – Find items that don't match

Names

ID	Name
1	Kevin
2	Sam
3	Jane
4	Kathi

Customers

CustID	FName
100	Bill
120	Denise
130	Anna
4	Kathi

ID	Name
1	Kevin
2	Sam
3	Jane

```
SELECT ID, NAME
FROM Names
EXCEPT
SELECT CustID, FirstName
FROM Customers;
```

INTERSECT – Find rows that match

Names

ID	Name
1	Kevin
2	Sam
3	Jane
4	Kathi

Customers

CustID	FName
100	Bill
120	Denise
130	Anna
4	Kathi

ID	Name
4	Kathi

```
SELECT ID, NAME
FROM Names
INTERSECT
SELECT CustID, FirstName
FROM Customers;
```

Demo: UNION

Lab

- Complete Module 8 Lab 1