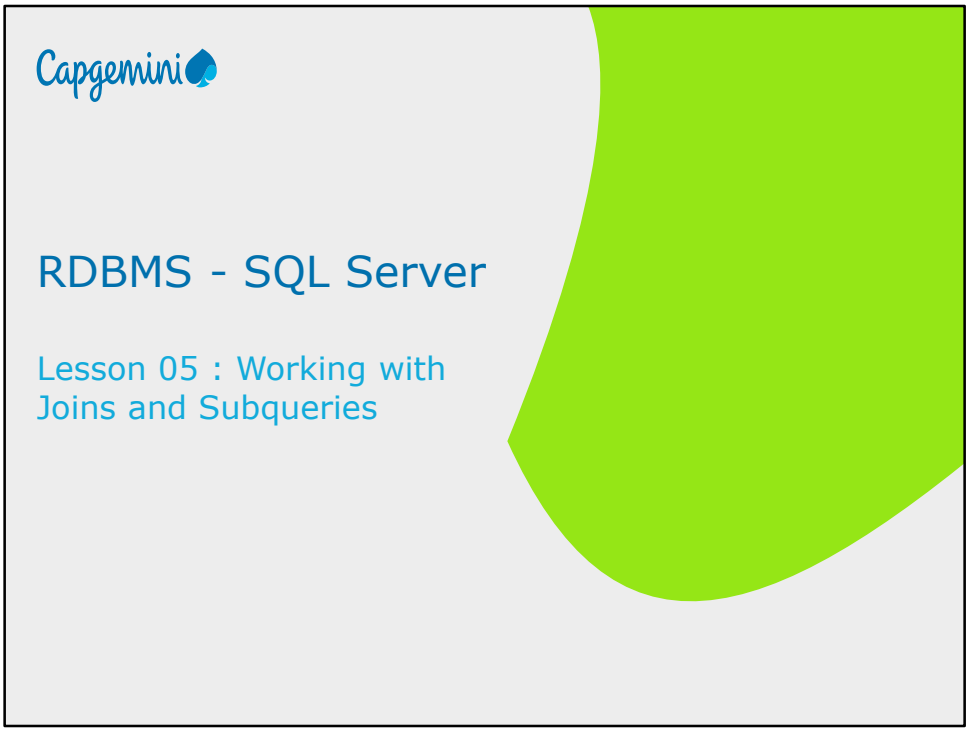
Instructor Notes:



Instructor Notes:

Lesson Objectives

- In this lesson, you will learn:
  - What are joins?
  - Types of Joins
  - Using Subqueries
  - Restrictions on Subqueries

Instructor Notes:

Joins

- Retrieving data from several tables ,based on a relationship between certain columns in these tables
- Most of the time it would be the Foreign key
- One can join maximum 256 tables in single query

By using joins, you can retrieve data from two or more tables based on logical relationships between the tables. Joins indicate how SQL Server should use data from one table to select the rows in another table.
A join condition defines the way two tables are related in a query by:
A join operation compares two or more tables (or views) by specifying a column from each, comparing the values in those columns row by row, and linking the rows that have matching values. It then displays the results in a new table. The tables specified in the join can be in the same database or in different databases.
When you join two or more tables, the columns being compared must have similar values--that is, values using the same or similar datatypes.
There are several types of joins, such as equijoins, natural joins, and outer joins. The most common join, the equijoin, is based on equality

If the joining table have the same column names the query engine would report an error because of ambiguity . Therefore one has to qualify the column names with Table names as given in the example.

Instructor Notes:

Types of Joins

- INNER join
  - Equijoin
  - Nonequijoin
- Outer join
  - LEFT outer
  - RIGHT outer
  - FULL outer
- Self join
- Cross join

Inner Join
Retrieves records from multiple tables based on the equality on the values of the common column. Inner joins are commutative in nature i.e same results come when A is joined with B or B is joined with A .

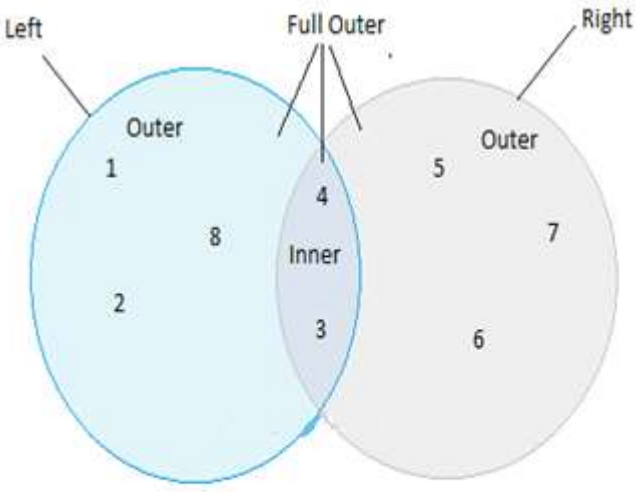
Self Join
A table joined with itself.

Outer Join
Displays the result set containing all the rows from one table and the matching rows from the another table. The table from where all the rows are retrieved is called OUTER table and other table is called as inner table . Based on the direction of the OUTER table , the outer joins can be LEFT , RIGHT
FULL outer joins have both matched and unmatched rows of all tables

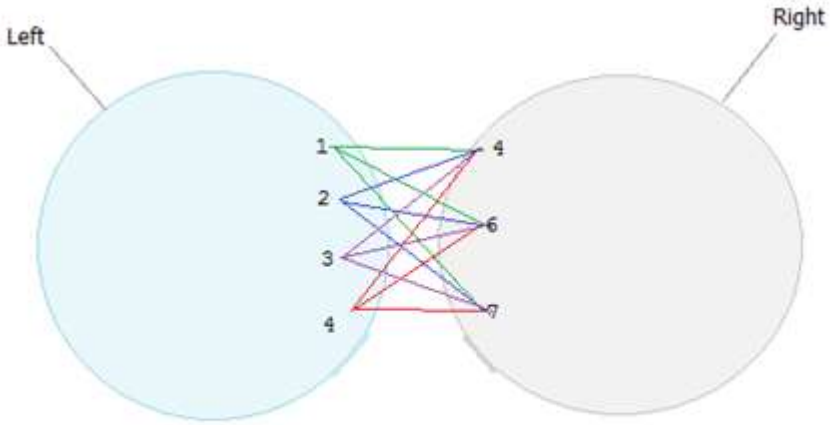
Cross Join (Cartesian Product)
Cross Join happens when each row from one table is joined with all the row of the other table. This typically can be done by removing the where clause

The syntax of all the joins are explained in the next section

Instructor Notes:



Inner Join Result : (4,3)
Left Join Result : (1,2,8,4,3)
Right Join Result : (5,6,7,4,3)
Full Outer Join Result : (1,2,8,4,3,5,6,7)




```
Cross Join Result :((1,4),(1,6),(1,7),(2,4),(2,6),(2,7),
                    (3,4),(3,6),(3,7),(4,4),(4,6),(4,7))
```

Instructor Notes:

Introducing T- SQL JOIN Syntax

```
SELECT   <<Column List>>
FROM   Table name1
[INNER]  JOIN Table name2
ON join criteria
WHERE condition criteria

OR

SELECT   <<Column List>>
FROM   Table name1 , Table name2
WHERE Join criteria
AND condition criteria
```

Instructor Notes:

Inner Join

```
USE Northwind
GO

SELECT ProductID, ProductName, CategoryName
FROM Products, Categories
WHERE Products.CategoryID=Categories.CategoryID
GO
```

Instructor Notes:

INNER JOIN – More than 2 tables

- Number of joins = number of tables - 1

```
USE northwind
GO

SELECT  categoryname, description,
productname, productid, companyname, suppliers.city
FROM products, categories, suppliers
WHERE Products.categoryid = Categories.categoryid
AND Products.supplierid = Suppliers.supplierid
AND  Suppliers.city = 'London'
ORDER by  productname
GO
```

When more than 2 tables need to be joined, the number of joins to be created will be one less than the total number of tables to be joined.

Instructor Notes:

INNER JOIN – More than 2 tables

```
Example :-
SELECT  categoryname, description, productname,
productid, companyname, Suppliers.city
FROM products
INNER JOIN categories
ON Products.categoryid = Categories.categoryid
INNER JOIN Suppliers
ON  Products.supplierid = Suppliers.supplierid
WHERE Suppliers.city = 'london'
ORDER by Prodcts.productname
```

An inner join is a join in which the values in the columns being joined are compared using a comparison operator.
In the SQL-92 standard, inner joins can be specified in either the FROM or WHERE clause. This is the only type of join that SQL-92 supports in the WHERE clause. Inner joins specified in the WHERE clause are known as old-style inner joins.
This inner join is known as an equijoin. It returns all the columns in both tables, and returns only the rows for which there is an equal value in the join column.

Example:
```
select   colum1   FROM Table1
INNER JOIN Table2
ON Table1.col1=Table2.col1
```

Instructor Notes:

OUTER JOIN

- Inner joins eliminates rows which doesn't have a match in the joining tables
- Outer joins returns all rows from one of the joining tables and matched rows from the others
- Outer joins can be further classified as
  - LEFT OUTER JOIN or LEFT JOIN
  - RIGHT OUTER JOIN or RIGHT JOIN
  - FULL OUTER JOIN or FULL JOIN

Inner joins return rows only when there is at least one row from both tables that matches the join condition. Inner joins eliminate the rows that do not match with a row from the other table. Outer joins, however, return all rows from at least one of the tables (or views) mentioned in the FROM clause, as long as those rows meet any WHERE or HAVING search conditions. All rows are retrieved from the left table referenced with a left outer join, and all rows from the right table referenced in a right outer join. All rows from both tables are returned in a full outer join.

Microsoft SQL Server 2008 uses these SQL-92 keywords for outer joins specified in a FROM clause.

- LEFT OUTER JOIN or LEFT JOIN
- RIGHT OUTER JOIN or RIGHT JOIN
- FULL OUTER JOIN or FULL JOIN

Instructor Notes:

Using Left Outer Joins

- Left Outer - all records from left table and corresponding matching records from right

T-SQL Syntax

```
SELECT  categoryname, description,
productname, productid
FROM CATEGORIES
LEFT OUTER JOIN PRODUCTS
ON PRODUCTS.categoryid = CATEGORIES.categoryid
GO
```

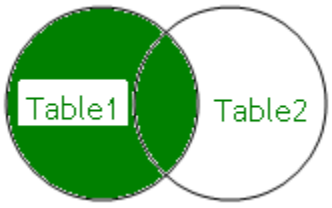
Using Left Outer Joins

Consider a join of the Products table and the Categories table on their CategoryID columns.

To include all category id, regardless of whether a product exists for that category or not, use an SQL-92 left outer join. The following is the query:

```
 SELECT  categoryname, description,
productname, productid
FROM CATEGORIES
LEFT OUTER JOIN PRODUCTS
ON PRODUCTS.categoryid = CATEGORIES.categoryid
GO
```

The LEFT OUTER JOIN includes all rows in the categories table in the results, whether or not there is a match on the CategoryID column in the Products table. Notice that in the results where there is no matching categoryID for a product, the row contains a null value in the productname and productID column.



Instructor Notes:

RIGHT OUTER Join

- T-SQL Syntax

```
SELECT  categoryname, description,
productname, productid
FROM CATEGORIES
RIGHT OUTER JOIN PRODUCTS
ON PRODUCTS.categoryid = CATEGORIES.categoryid
GO
```

A right outer join (or right join) closely resembles a left outer join, except with the treatment of the tables reversed Every row from the right table is retrieved with corresponding match from the left table.

Instructor Notes:

FULL OUTER JOIN

- FULL OUTER JOIN - includes all rows from both tables

```
USE Northwind;
GO
SELECT  categoryname, description,
productname, productid
FROM CATEGORIES
FULL OUTER JOIN PRODUCTS
ON PRODUCTS.categoryid = CATEGORIES.categoryid
GO
```

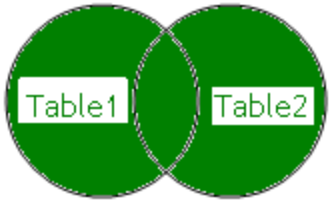
Using Full Outer Joins

To retain the non-matching information by including non-matching rows in the results of a join, use a full outer join. SQL Server provides the full outer join operator, FULL OUTER JOIN, which includes all rows from both tables, regardless of whether or not the other table has a matching value.

Consider a join of the Product table and the Categories table on their categoryID columns. The results show only the Products that have sales orders on them. The SQL-92 FULL OUTER JOIN operator indicates that all rows from both tables are to be included in the results, regardless of whether there is matching data in the tables.

You can include a WHERE clause with a full outer join to return only the rows where there is no matching data between the tables.

There is no T-SQL equivalent of FULL JOIN .



Instructor Notes:

SELF JOIN

- A self join is when a table joins with itself
- Self join is possible when the table has references to itself
  For example to query employees along with managers who are also employees

```
SELECT  emp.EmployeeID , emp.EmployeeName, mgr.EmpoyeeName
FROM  Employee emp
INNER JOIN Employee mgr
ON emp.ManagerID = mgr.EmployeeID
```

A table can be joined to itself in a self-join. This s typically happens , when the records in a table has a reference in the same table itself . For example Given an employee table with the following structure

EmployeeID
EmployeeName
SuperviosrID
Basic
Here supervisor is also an employee , therefore to list all employees along with manager names , one has to do a self join. Appropirate aliases are used to distinguish the columns .

A self join is a join in all possible ways except that it joins to another instance of itself

Instructor Notes:

CROSS JOIN

- Cross join does not have a WHERE clause
- Result set is the number of rows in the first table multiplied by the number of rows in the second table- a Cartesian product

```
USE Northwind;
GO
SELECT  categoryname, description, productname, productid
FROM CATEGORIES ,PRODUCTS
GO
```

```
USE Northwind;
GO
SELECT  categoryname, description, productname, productid
FROM CATEGORIES CROSS JOIN PRODUCTS
ORDER BY CategoryName
```

A cross join that does not have a WHERE clause produces the Cartesian product of the tables involved in the join. The size of a Cartesian product result set is the number of rows in the first table multiplied by the number of rows in the second table.

Instructor Notes:

Subquery

- A sub query is an SQL statement that is used within another SQL statement
- Subqueries are used to handle query requests that are expressed as the results of other queries
- Subquery can be embedded in WHERE /HAVING statement

```
USE Northwind;
GO
SELECT EmployeeID,EmployeeName
FROM Employees
WHERE Region=
 (SELECT Region from Employees
  WHERE EmployeeID=12345)
```

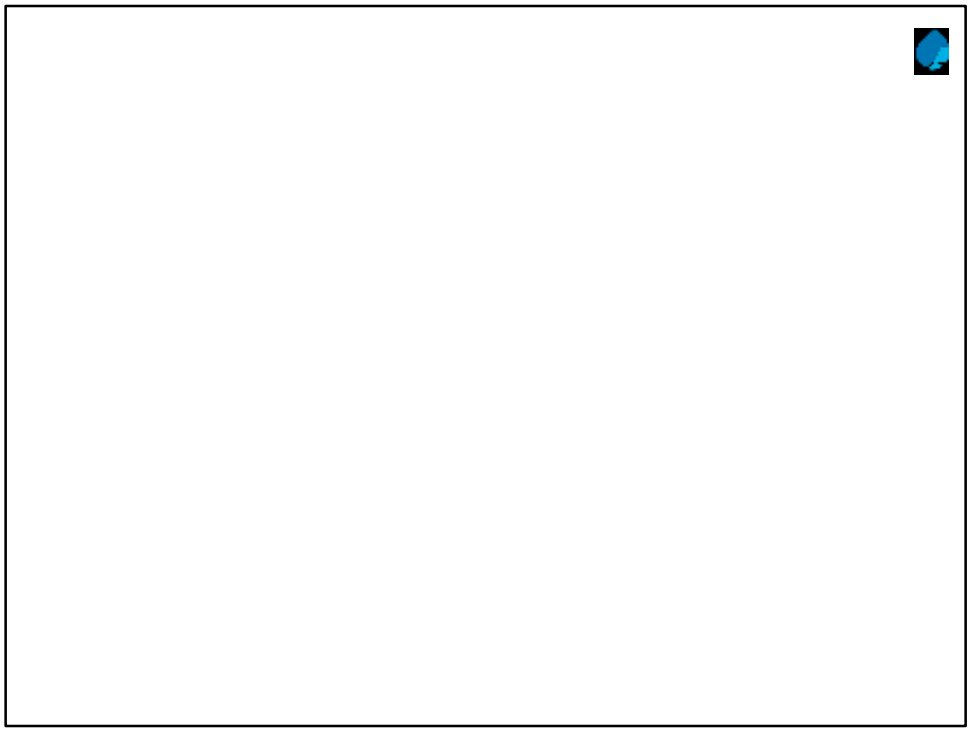
How subqueries work

Subqueries, also called inner queries, appear within a **where** or **having** clause of another SQL statement or in the select list of a statement. You can use subqueries to handle query requests that are expressed as the results of other queries. A statement that includes a subquery operates on rows from one table, based on its evaluation of the subquery's **select** list, which can refer either to the same table as the outer query, or to a different table. In Transact-SQL, a subquery can also be used almost anywhere an expression is allowed, if the subquery returns a single value. A **case** expression can also include a subquery.

For example, this subquery lists the employee details of all employees who belong to the same region as employee 12345

```
USE Northwind;
GO
SELECT EmployeeID, EmployeeName
FROM Employees
WHERE Region=
 (SELECT Region from Employees
  WHERE EmployeeID=12345)
```

Instructor Notes:



select statements that contain one or more subqueries are sometimes called nested queries or nested select statements.
You can formulate as joins many SQL statements that include a subquery. Other questions can be posed only with subqueries. Some people find subqueries easier to understand. Other SQL users avoid subqueries whenever possible. You can choose whichever formulation you prefer.
The result of a subquery that returns no values is NULL. If a subquery returns NULL, the query failed.

Instructor Notes:

Subquery restrictions

- Subquery_select_list can consist of only one column name
- Subqueries can be nested inside the where or having clause
- Subquery can appear almost anywhere an expression can be used, if it returns a single value
- Subqueries cannot manipulate their results internally i.e. ORDER BY Clause cannot be used inside the subquery
- If the sub query returns more than 1 row then outer query has to use appropirate operator like IN , ANY etc.

A subquery is subject to the following restrictions:

1. The subquery_select_list can consist of only one column name, except in the **exists** subquery, where an **(*)** is usually used in place of the single column name. Do not specify more than one column name. Qualify column names with table or view names if there is ambiguity about the table or view to which they belong.
2. Subqueries can be nested inside the **where** or **having** clause of an outer **select**, **insert**, **update**, or **delete** statement, inside another subquery, or in a select list. Alternatively, you can write many statements that contain subqueries as joins; Adaptive Server processes such statements as joins.
3. In Transact-SQL, a subquery can appear almost anywhere an expression can be used, if it returns a single value.
4. The select list of an inner subquery introduced with a comparison operator can include only one expression or column name, and the subquery must return a single value. The column you name in the **where** clause of the outer statement must be join-compatible with the column you name in the subquery select list.
5. Subqueries cannot manipulate their results internally, that is, a subquery cannot include the **order by** clause, the **compute** clause, or the **into** keyword.

Instructor Notes:

Types of Subqueries

- Single Row Sub query
- Multi Row Sub query
- Sub query for Existence
- Correlated Sub Query

6. Correlated (repeating) subqueries are not allowed in the **select** clause of an updatable cursor defined by **declare cursor**.
7. There is a limit of 16 nesting levels.
8. The maximum number of subqueries on each side of a union is 16.
9. The **where** clause of a subquery can contain an aggregate function only if the subquery is in a **having** clause of an outer query and the aggregate value is a column from a table in the **from** clause of the outer query.

There are three basic types of subqueries.
Single ROW subquery
Such subquery returns only 1 value . Therefore the operator which can be used
=, >, >=, <, <=, and <>.
The subquery can be used in WHERE and HAVING clause

Multiple ROW Subqueries
Subqueries which return more than 1 row . Some operators that can be used with multipe-row subqueries are:
IN, equal to any member in the list
ANY, compare values to each value returned by the subquery.
ALL , All the values
The examples of each type is explained in the next few sections

Instructor Notes:

Multi Row Subquery

- Subqueries returns a list of zero or more values and can include a GROUP BY or HAVING clause
- >ALL means greater than every value
- >ANY means greater than at least one value

```
SELECT ProductID, ProductName, UnitPrice
FROM PRODUCTS WHERE SupplierID  IN
(SELECT SupplierID
 FROM Suppliers
 WHERE CITY="NEW York")
```

Comparison operators that introduce a subquery can be modified by the keywords ALL or ANY. SOME is an SQL-92 standard equivalent for ANY.
Subqueries introduced with a modified comparison operator return a list of zero or more values and can include a GROUP BY or HAVING clause. These subqueries can be restated with EXISTS.
Using the > comparison operator as an example, >ALL means greater than every value. In other words, it means greater than the maximum value. For example, >ALL (1, 2, 3) means greater than 3. >ANY means greater than at least one value, that is, greater than the minimum. So >ANY (1, 2, 3) means greater than 1.
For a row in a subquery with >ALL to satisfy the condition specified in the outer query, the value in the column introducing the subquery must be greater than each value in the list of values returned by the subquery.
Similarly, >ANY means that for a row to satisfy the condition specified in the outer query, the value in the column that introduces the subquery must be greater than at least one of the values in the list of values returned by the subquery.

Instructor Notes:

Multi Row Subquery

```
SELECT ProductID, ProductName, SupplierID
FROM Products
WHERE UnitPrice > ALL
(Select UnitPrice
 FROM  Products
 WHERE SupplierID=10098)
```

In this example you are listing all those products whose price is above all the price of products supplied by supplier 10098

Instructor Notes:

EXISTS

- EXISTS checks for a existence of a condition
- The EXISTS condition is considered "to be met" if the subquery returns at least one row.

```
SELECT SupplierID
FROM suppliers
WHERE EXISTS
  (select 'A'
    from orders
    where suppliers.supplier_id = orders.supplier_id);
```

Subquery that is introduced with **exists** is different from other subqueries, in these ways:

- The keyword **exists** is not preceded by a column name, constant, or other expression.
- The subquery **exists** evaluates to TRUE or FALSE rather than returning any data.

Since EXISTS evaluates to TRUE /FALSE for a condition , there Is no need to return all the columns , a single column or a value can be returned back which will improve performance.

Instructor Notes:

Summary

- In this lesson, you have learnt:
  - Joins are used to fetch data from more than 2 tables.
  - Join types: equijoin, non-equijoin, outer join, self join
  - Subquery is query within a query or nested query
  - Subquery types

Instructor Notes:

Review Question

- Question 1: If we do not include a join condition then the result leads to _____
- Question 2: _____ return all rows from at least one of the tables in the FROM clause
- Question 3: When subquery depends on the outer query for its execution then it is _____ subquery.
- Question 4: Subquery _____ evaluates to TRUE or FALSE rather than returning any data