**Subquery Fundamentals**

**SQL Server 2008 R2**

[Other Versions](javascript:;)

Description: http://i.technet.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

* [SQL Server 2008](http://technet.microsoft.com/en-us/library/ms189575(d=printer,v=sql.100).aspx)
* [SQL Server 2005](http://technet.microsoft.com/en-us/library/ms189575(d=printer,v=sql.90).aspx)

A subquery is a query that is nested inside a SELECT, INSERT, UPDATE, or DELETE statement, or inside another subquery. A subquery can be used anywhere an expression is allowed. In this example a subquery is used as a column expression named MaxUnitPrice in a SELECT statement.

USE AdventureWorks2008R2;

GO

SELECT Ord.SalesOrderID, Ord.OrderDate,

(SELECT MAX(OrdDet.UnitPrice)

FROM AdventureWorks.Sales.SalesOrderDetail AS OrdDet

WHERE Ord.SalesOrderID = OrdDet.SalesOrderID) AS MaxUnitPrice

FROM AdventureWorks2008R2.Sales.SalesOrderHeader AS Ord

A subquery is also called an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select.

Many Transact-SQL statements that include subqueries can be alternatively formulated as joins. Other questions can be posed only with subqueries. In Transact-SQL, there is usually no performance difference between a statement that includes a subquery and a semantically equivalent version that does not. However, in some cases where existence must be checked, a join yields better performance. Otherwise, the nested query must be processed for each result of the outer query to ensure elimination of duplicates. In such cases, a join approach would yield better results. The following is an example showing both a subquery SELECT and a join SELECT that return the same result set:

/\* SELECT statement built using a subquery. \*/

SELECT Name

FROM AdventureWorks2008R2.Production.Product

WHERE ListPrice =

(SELECT ListPrice

FROM AdventureWorks2008R2.Production.Product

WHERE Name = 'Chainring Bolts' );

/\* SELECT statement built using a join that returns

the same result set. \*/

SELECT Prd1. Name

FROM AdventureWorks2008R2.Production.Product AS Prd1

JOIN AdventureWorks2008R2.Production.Product AS Prd2

ON (Prd1.ListPrice = Prd2.ListPrice)

WHERE Prd2. Name = 'Chainring Bolts';

A subquery nested in the outer SELECT statement has the following components:

* A regular SELECT query including the regular select list components.
* A regular FROM clause including one or more table or view names.
* An optional WHERE clause.
* An optional GROUP BY clause.
* An optional HAVING clause.

The SELECT query of a subquery is always enclosed in parentheses. It cannot include a COMPUTE or FOR BROWSE clause, and may only include an ORDER BY clause when a TOP clause is also specified.

A subquery can be nested inside the WHERE or HAVING clause of an outer SELECT, INSERT, UPDATE, or DELETE statement, or inside another subquery. Up to 32 levels of nesting is possible, although the limit varies based on available memory and the complexity of other expressions in the query. Individual queries may not support nesting up to 32 levels. A subquery can appear anywhere an expression can be used, if it returns a single value.

If a table appears only in a subquery and not in the outer query, then columns from that table cannot be included in the output (the select list of the outer query).

Statements that include a subquery usually take one of these formats:

* WHERE expression [NOT] IN **(**subquery**)**
* WHERE expression comparison\_operator [ANY | ALL] **(**subquery**)**
* WHERE [NOT] EXISTS **(**subquery**)**

In some Transact-SQL statements, the subquery can be evaluated as if it were an independent query. Conceptually, the subquery results are substituted into the outer query (although this is not necessarily how Microsoft SQL Server actually processes Transact-SQL statements with subqueries).

There are three basic types of subqueries. Those that:

* Operate on lists introduced with IN, or those that a comparison operator modified by ANY or ALL.
* Are introduced with an unmodified comparison operator and must return a single value.
* Are existence tests introduced with EXISTS.

**Subquery Rules**

**SQL Server 2008 R2**

[Other Versions](javascript:;)



* [SQL Server 2008](http://technet.microsoft.com/en-us/library/ms189543(d=printer,v=sql.100).aspx)
* [SQL Server 2005](http://technet.microsoft.com/en-us/library/ms189543(d=printer,v=sql.90).aspx)

A subquery is subject to the following restrictions:

* The select list of a subquery introduced with a comparison operator can include only one expression or column name (except that EXISTS and IN operate on SELECT \* or a list, respectively).
* If the WHERE clause of an outer query includes a column name, it must be join-compatible with the column in the subquery select list.
* The ntext, text, and image data types cannot be used in the select list of subqueries.
* Because they must return a single value, subqueries introduced by an unmodified comparison operator (one not followed by the keyword ANY or ALL) cannot include GROUP BY and HAVING clauses.
* The DISTINCT keyword cannot be used with subqueries that include GROUP BY.
* The COMPUTE and INTO clauses cannot be specified.
* ORDER BY can only be specified when TOP is also specified.
* A view created by using a subquery cannot be updated.
* The select list of a subquery introduced with EXISTS, by convention, has an asterisk (\*) instead of a single column name. The rules for a subquery introduced with EXISTS are the same as those for a standard select list, because a subquery introduced with EXISTS creates an existence test and returns TRUE or FALSE, instead of data.

**Subquery Types**

**SQL Server 2008 R2**

[Other Versions](javascript:;)



* [SQL Server 2008](http://technet.microsoft.com/en-us/library/ms175838(d=printer,v=sql.100).aspx)
* [SQL Server 2005](http://technet.microsoft.com/en-us/library/ms175838(d=printer,v=sql.90).aspx)

Subqueries can be specified in many places:

* With aliases. For more information, see [Subqueries with Aliases](http://technet.microsoft.com/en-us/library/ms190410(v=sql.105).aspx).
* With IN or NOT IN. For more information, see [Subqueries with IN](http://technet.microsoft.com/en-us/library/ms190963(v=sql.105).aspx) and [Subqueries with NOT IN](http://technet.microsoft.com/en-us/library/ms189062(v=sql.105).aspx).
* In UPDATE, DELETE, and INSERT statements. For more information, see [Subqueries in UPDATE, DELETE, and INSERT Statements](http://technet.microsoft.com/en-us/library/ms191223(v=sql.105).aspx).
* With comparison operators. For more information, see [Subqueries with Comparison Operators](http://technet.microsoft.com/en-us/library/ms190609(v=sql.105).aspx).
* With ANY, SOME, or ALL. For more information, see [Comparison Operators Modified by ANY, SOME, or ALL](http://technet.microsoft.com/en-us/library/ms187074(v=sql.105).aspx).
* With EXISTS or NOT EXISTS. For more information, see [Subqueries with EXISTS](http://technet.microsoft.com/en-us/library/ms189259(v=sql.105).aspx) and [Subqueries with NOT EXISTS](http://technet.microsoft.com/en-us/library/ms184297(v=sql.105).aspx).
* In place of an expression. For more information, see [Subqueries Used in Place of an Expression](http://technet.microsoft.com/en-us/library/ms189623(v=sql.105).aspx).

**Multiple Levels of Nesting**

**SQL Server 2008 R2**

[Other Versions](javascript:;)



* [SQL Server 2008](http://technet.microsoft.com/en-us/library/ms177501(d=printer,v=sql.100).aspx)
* [SQL Server 2005](http://technet.microsoft.com/en-us/library/ms177501(d=printer,v=sql.90).aspx)

A subquery can itself include one or more subqueries. Any number of subqueries can be nested in a statement.

The following query finds the names of employees who are also sales persons.

Use AdventureWorks2008R2;

GO

SELECT LastName, FirstName

FROM Person.Person

WHERE BusinessEntityID IN

(SELECT BusinessEntityID

FROM HumanResources.Employee

WHERE BusinessEntityID IN

(SELECT BusinessEntityID

FROM Sales.SalesPerson)

)

Here is the result set.

LastName FirstName

-------------------------------------------------- -----------------------

Jiang Stephen

Abbas Syed

Alberts Amy

Ansman-Wolfe Pamela

Campbell David

Carson Jillian

Ito Shu

Mitchell Linda

Reiter Tsvi

Saraiva Jos

Vargas Garrett

Varkey Chudukatil Ranjit

Valdez Rachel

Tsoflias Lynn

Pak Jae

Blythe Michael

Mensa-Annan Tete

(17 row(s) affected)

The innermost query returns the sales person IDs. The query at the next higher level is evaluated with these sales person IDs and returns the contact ID numbers of the employees. Finally, the outer query uses the contact IDs to find the names of the employees.

You can also express this query as a join:

USE AdventureWorks2008R2;

GO

SELECT LastName, FirstName

FROM Person.Person c

INNER JOIN HumanResources.Employee e

ON c.BusinessEntityID = e.BusinessEntityID

JOIN Sales.SalesPerson s

ON e.BusinessEntityID = s.BusinessEntityID

**Correlated Subqueries**

**SQL Server 2008 R2**

[Other Versions](javascript:;)



* [SQL Server 2008](http://technet.microsoft.com/en-us/library/ms187638(d=printer,v=sql.100).aspx)
* [SQL Server 2005](http://technet.microsoft.com/en-us/library/ms187638(d=printer,v=sql.90).aspx)

Many queries can be evaluated by executing the subquery once and substituting the resulting value or values into the WHERE clause of the outer query. In queries that include a correlated subquery (also known as a repeating subquery), the subquery depends on the outer query for its values. This means that the subquery is executed repeatedly, once for each row that might be selected by the outer query.

This query retrieves one instance of each employee's first and last name for which the bonus in the SalesPerson table is 5000 and for which the employee identification numbers match in the Employee and SalesPerson tables.

USE AdventureWorks2008R2;

GO

SELECT DISTINCT c.LastName, c.FirstName, e.BusinessEntityID

FROM Person.Person AS c JOIN HumanResources.Employee AS e

ON e.BusinessEntityID = c.BusinessEntityID

WHERE 5000.00 IN

(SELECT Bonus

FROM Sales.SalesPerson sp

WHERE e.BusinessEntityID = sp.BusinessEntityID) ;

GO

Here is the result set.

LastName FirstName BusinessEntityID

-------------------------- ---------- ------------

Ansman-Wolfe Pamela 280

Saraiva José 282

(2 row(s) affected)

The previous subquery in this statement cannot be evaluated independently of the outer query. It needs a value for Employee.BusinessEntityID, but this value changes as SQL Server examines different rows in Employee.

That is exactly how this query is evaluated: SQL Server considers each row of the Employee table for inclusion in the results by substituting the value in each row into the inner query. For example, if SQL Server first examines the row for Syed Abbas, the variable Employee.BusinessEntityID takes the value 285, which SQL Server substitutes into the inner query.

USE AdventureWorks2008R2;

GO

SELECT Bonus

FROM Sales.SalesPerson

WHERE BusinessEntityID = 285;

The result is 0 (Syed Abbas did not receive a bonus because he is not a sales person), so the outer query evaluates to:

USE AdventureWorks2008R2;

GO

SELECT LastName, FirstName

FROM Person.Person AS c JOIN HumanResources.Employee AS e

ON e.BusinessEntityID = c.BusinessEntityID

WHERE 5000 IN (0.00)

Because this is false, the row for Syed Abbas is not included in the results. Go through the same procedure with the row for Pamela Ansman-Wolfe. You will see that this row is included in the results.

Correlated subqueries can also include table-valued functions in the FROM clause by referencing columns from a table in the outer query as an argument of the table-valued function. In this case, for each row of the outer query, the table-valued function is evaluated according to the subquery.

**Qualifying Column Names in Subqueries**

**SQL Server 2008 R2**

[Other Versions](javascript:;)



* [SQL Server 2008](http://technet.microsoft.com/en-us/library/ms178050(d=printer,v=sql.100).aspx)
* [SQL Server 2005](http://technet.microsoft.com/en-us/library/ms178050(d=printer,v=sql.90).aspx)

In the following example, the CustomerID column in the WHERE clause of the outer query is implicitly qualified by the table name in the outer query's FROM clause, Sales.Store. The reference to CustomerID in the select list of the subquery is qualified by the subquery's FROM clause, that is, by the Sales.Customer table.

USE AdventureWorks2008R2;

GO

SELECT Name

FROM Sales.Store

WHERE BusinessEntityID NOT IN

(SELECT CustomerID

FROM Sales.Customer

WHERE TerritoryID = 5)

The general rule is that column names in a statement are implicitly qualified by the table referenced in the FROM clause at the same level. If a column does not exist in the table referenced in the FROM clause of a subquery, it is implicitly qualified by the table referenced in the FROM clause of the outer query.

Here is what the query looks like with these implicit assumptions specified:

USE AdventureWorks2008R2;

GO

SELECT Name

FROM Sales.Store

WHERE Sales.Store.BusinessEntityID NOT IN

(SELECT Sales.Customer.CustomerID

FROM Sales.Customer

WHERE TerritoryID = 5)

It is never wrong to state the table name explicitly, and it is always possible to override implicit assumptions about table names with explicit qualifications.

|  |
| --- |
| **Caution noteCaution** |
| If a column is referenced in a subquery that does not exist in the table referenced by the subquery's FROM clause, but exists in a table referenced by the outer query's FROM clause, the query executes without error. SQL Server implicitly qualifies the column in the subquery with the table name in the outer query. |