**Built In Functions(System Functions) IN SQL:**

SQL server

provide number of built in functions like mathematical functions, character

functions, date and time functions, aggregative functions,convertion functions

etc.these can be used to perform certain operations and return a value.

**Syntax: SELECT <Function Name> [Expressions]**

**Mathematical Functions:** These functions perform a calculation based on

input values provided as arguments, and return a numeric value.

**ABS ():** Returns the absolute, positive value of the given numeric expression.

**Ex:** select ABS(-15)---- 15

select ABS(45)----- 45

**CEILING ():**

Returns the smallest integer greater than, or equal to, the given

numeric expression.

**Ex:** select ceiling(15.000)----15

select ceiling(15.0001)----16

select ceiling(-12.34)-----(-12)

**FLOOR ():**

Returns the largest integer less than or equal to the given numeric

expression.

**Ex:** select floor(15.000)---15

select floor(15.0001)----15

select floor(-12.34)----(-13)

**SQUARE ():** Returns the square of the given expression.

**Ex:** select SQUARE(5)---25

**SQRT ():** Returns the square root of the given expression.

**Ex:** select SQUARE(25)---5

**POWER (n, m):** Returns the power value of given expression

Ex: select POWER (2, 3) ---------- 8

**SIGN ():** Returns the positive (+1), zero (0), or negative (-1) sign of the given

expression.

**Ex:** select SIGN(42)------------1

select SIGN(0)-------------0

select SIGN(-42)-----------(-1)

**PI ():** Returns the constant value of PI.

**Ex:** select PI()---------3.14159265358979

**LOG ():** Returns the natural logarithm of the given expression.

**Ex:** select LOG(2)------ 0.693147180559945

**LOG 10():** Returns the base-10 logarithm of the given expression.

**Ex:** select LOG10(10)----1

**SIN ():** Returns the trigonometric sine of the given angle (in radians) in an

approximate numeric expression.

**Ex:** select SIN (0) -------0

**COS ():** A mathematic function that returns the trigonometric cosine of the given

angle (in radians) in the given expression.

**Ex:** select COS (0) -------1

**TAN ():** Returns the tangent of the input expression.

**Ex:** select TAN (0) ---------0

**String Functions:** These functions perform an operation on a string input value

and return a string or numeric value.

**ASCII ():** Returns the ASCII code value of the leftmost character of a character

expression.

**Ex:** Select ASCII („Z‟) -----90

**CHAR ():** A string function that converts an **int** ASCII code to a character.

**Ex:** Select CHAR (90) -----Z

**CHARINDEX ():** Returns the starting position of the specified expression in a

character string.

**Ex:** Select CHARINDEX („S‟,‟SUDHAKAR‟) -------1

**LEFT ():** Returns the left part of a character string with the specified number of

characters.

**Ex:** Select LEFT („SUDHAKAR‟, 5) ----SUDHA

**RIGHT ():** Returns the right part of a character string with the specified number of

characters.

**Ex:** Select RIGHT („SUDHAKAR‟, 3) ------KAR

**LEN ():** Returns the number of characters, rather than the number of bytes, of the

given string expression.

**Ex:** Select LEN („WELCOME‟) ------------7

**LOWER ():** Returns a character expression after converting uppercase character

data to lowercase.

**Ex:** Select LOWER („SAI‟) --------sai

**UPPER ():** Returns a character expression with lowercase character data converted

to uppercase.

**Ex:** Select UPPER („sai‟) ------SAI

**LTRIM ():** Returns a character expression after removing leading blanks.

**Ex:** Select LTRIM („ HELLO‟) --------HELLO

**RTRIM ():** Returns a character string after truncating all trailing blanks.

**Ex:** Select RTRIM („HELLO „) -------HELLO

**REPLACE ():** Replaces all occurrences of the second given string expression in

the first string expression with a third expression.

**Ex:** Select REPLACE („JACK AND JUE‟, „J‟, „BL‟) ------BLACK AND BLUE

**REPLICATE ():** Repeats a character expression for a specified number of times.

**Ex:** Select REPLICATE („SAI‟, 3) -------SAISAISAI

**REVERSE ():** Returns the reverse of a character expression.

**Ex:** Select REVERSE („HELLO‟) --------OLLEH

**SPACE ():** Returns a string of repeated spaces.

**Ex:** Select („SAI‟+SPACE (50) +‟SUDHAKAR‟) -----SAI SUDHAKAR

**SUBSTRING (expression, start, length):** Returns a part of a string from

expression from starting position, where length is no. of chars to be picked.

**Ex**: Select SUBSTRING („HELLO‟, 1, 3) ----------- HEL

Select SUBSTRING („HELLO‟, 3, 3) ----------- LLO

**Date and Time Functions:** These functions perform an operation on a

date and time input value and return a string, numeric, or date and time value.

**GETDATE ():** Returns the current system date and time in the SQL Server

standard internal format for date time values.

**Ex:** Select GETDATE () ------- 2014-02-15 15:35:22.670

**DAY ():** Returns an integer representing the day date part of the specified date.

**Ex:** Select DAY (get date ())

**MONTH ():** Returns an integer that represents the month part of a specified date.

**Ex:** Select MONTH (get date ())

**YEAR ():** Returns an integer that represents the year part of a specified date.

**Ex:** Select YEAR (get Date ())

**GETUTCDATE ():** Returns the date time value representing the current UTC

time (Coordinated Universal Time).

**Ex:** Select GETUTCDATE ();

**DATE NAME ():** Returns a character string representing the specified date part of

the specified date.

**Ex:** Select DATE NAME (DW, get date ())

**DATE PART ():** Returns an integer representing the specified date part of the

specified date.

**Ex:** Select DATEPART (DD, get date ())

**DATE ADD ():** Returns a new date time value based on adding an interval to the

specified date.

**Ex:** Select DATEADD (DD, 5, get date ())

**DATE DIFF ():** Returns the difference between the start and end dates in the give

date part format.

**Ex:** Select DATEDIFF (MM, „2012-12-15‟, get date ())

**Conversion Functions:** These functions are used to convert one data type

to another. We have two conversion functions are CAST and CONVERT both

provide similar functionality.

**CAST ():** Convert to one data type to another type.

**Syntax:** CAST (Expression as data type [size])

**Ex:** Select CAST (10.2587 as Int) -------------10

**CONVERT ():** Convert function can be used to display date time data in different

format.

**Syntax:** Convert (Data type [size], Expression, Style value)

**Ex:** Select Convert (Varchar (24), get date (), 113)

The table below represents the style values for date time or small date time

conversion to character data:

**Sno**

**Value**

**Output Standard**

- 0 or 100 mon dd yyyy hh:mi AM (or PM) Default

1 101 mm/dd/yy USA

2 102 yy.mm.dd ANSI

3 103 dd/mm/yy British/French

4 104 dd.mm.yy German

5 105 dd-mm-yy Italian

6 106 dd mon yy

7 107 Mon dd, yy

8 108 hh:mm:ss

- 9 or 109

mon dd yyyy hh:mi:ss:mmmAM (or

PM)

Default+millisec

10 110 mm-dd-yy USA

11 111 yy/mm/dd Japan

12 112 Yymmdd ISO

- 13 or 113 dd mon yyyy hh:mi:ss:mmm (24h)

14 114 hh:mi:ss:mmm (24h)

- 20 or 120 yyyy-mm-dd hh:mi:ss (24h)

- 21 or 121 yyyy-mm-dd hh:mi:ss.mmm (24h)

- 126

yyyy-mm-ddThh:mi:ss.mmm (no

spaces)

ISO8601

- 130 dd mon yyyy hh:mi:ss:mmmAM Hijiri

- 131 dd/mm/yy hh:mi:ss:mmmAM Hijiri

**Aggregate functions/Group functions:** Aggregate functions perform

a calculation on a set of values and return a single value. Aggregate functions are

often used with the GROUP BY clause of the SELECT statement.

**SUM ():** Returns the sum of all the values .Sum can be used with numeric columns

only. Null values are ignored.

**Ex:** SELECT SUM (SALARY) FROM EMP

**AVG** (): Returns the average of the values in a group. Null values are ignored.

**Ex:** SELECT AVG (SALARY) FROM EMP

**MAX** (): Returns the maximum value in the expression.

**Ex:** SELECT MAX (SALARY) FROM EMP

**MIN ():** Returns the minimum value in the expression.

**Ex:** SELECT MIN (SALARY) FROM EMP

**COUNT ():** Returns the number of records in a table. This function again use in

three ways.

1. **COUNT (\*):** It Returns total number of records in a table

**Ex:** SELECT COUNT (\*) FROM EMP

2. **COUNT (Expression/Column name):** It returns number of records

including duplicate values but not null vales.

**Ex:** SELECT COUNT (ENAME) FROM EMP

3. **COUNT (Distinct Column name):** It returns number of records without

null and duplicate values.

**Ex:** SELECT COUNT (Distinct ENAME) FROM EMP

**Distinct Key:** If we use this key word on a column with in a query then it will

retrieve the values of the column without duplicates.

**OPERATORS IN SQL:** Operator is a symbol which performs some

specific operation on operands or expressions. These operators are classified into 6

types in SQL.

1. Assignment operator

2. Arithmetic operator

3. Comparison operator

4. Logical operator

5. Set operator

**Assignment operator:** Assignment operator contain only one operator is

knows as equal „=‟ operator.

**Ex1:** Write a Query to display the employee details whose salary is equal to10000

SELECT \* FROM EMP WHERE SAL=10000

**Ex2:** Write a query to change the deptno as „10‟whose employee id is 101

UPDATE EMP SET DEPTNO=10 WHERE EID=101

**Ex3:** Write a query to delete a record whose employee id is 107

DELETE FROM EMP WHERE EID=107

**Arithmetic operator:** Arithmetic operators perform mathematical

operations on two expressions. The lists of arithmetic operators are + (Add), -

Subtraction,\* Multiplication. / (Divide) Division and % (Modulo) Returns the

integer remainder of a division. For example, 12 % 5 = 2 because the remainder of

12 divided by 5 is 2.

**Ex1:** Select 100+250

Select 245-400

Select 20\*20

Select 25/5

Select 37%6

Select 20/5+20/5

Select 35.50+20

**Ex2:** WAQ to find student TOTAL, AVERAGE AND CLASS OF a table

Step1: Create table student (Sid int, sname varchar (50), math‟s int, phy int, che

int, total int, average int, class varchar (max))

Step2: Update student set total=maths+phy+che

Step3: Update student set average=total/3

Step4: Update student set class=

Case

When average>=60 then 'First class'

When average>=50 then 'second class'

When average>=40 then 'third class'

Else

'Fail'

End

**CASE ():** This function is used to execute list of conditions and returns a value.

**Syntax:** Case

<Condition 1>-----------<Condition N>

Else

<Statement>

End

**Comparison operators:** Comparison operators test whether two

expressions are the same. Comparison operators can be used on all expressions

except expressions of the text, ntext, or image data types. The following table lists

the Transact-SQL comparison operators are > (Greater Than),< (Less Than) ,>=

(Greater Than or Equal To) ,<= (Less Than or Equal To) ,!= (Not Equal To),!< (Not

Less Than),!> (Not Greater Than)

**Examples:**

Select ename from EMP where salary<50000

Update EMP set salary=1000 where salary>90000

Update EMP set ename='joshitha' where salary<=25000

Update EMP set salary=98000 where salary>=1000

Select ename from Emp where salary !>98000

Select ename from Emp where salary !<98000

Select ename from Emp where salary !=98000

**Logical operator**: Logical operators test for the truth of some condition.

Logical operators, like comparison operators, return a Boolean data type with a

value of TRUE or FALSE. Logical operators are AND , OR , NOT**,** BETWEEN,

NOT BETWEEN, LIKE, NOT LIKE, IN, NOT IN, EXISTS,NOT EXISTS, ANY,

ALL, SOME.

**Examples:**

Select \* from EMP where ename='siddhu' and salary=45000

Select \* from EMP where ename='joshitha' or salary=98000

Select \* from EMP where not ename='joshitha'

Select \* from EMP where salary between 10000 and 50000

Update EMP set ename='SAI' where eid=101 and salary=25000

**Queries Using ‘Select’ with ‘where’ clause:**

Write a Query to display the employee details whose salary is less than

10000

SELECT \* FROM EMP WHERE SAL<10000

Write a Query to display the employee details whose salary is greater than or

equal to 9000 and less than 15000

SELECT \* FROM EMP WHERE SAL>=9000 AND SAL<=15000

(OR)

SELECT \* FROM EMP WHERE SAL BETWEEN 9000 AND 15000

Write a Query to display the employee details whose salary is not between

9000 and 15000

SELECT \* FROM EMP WHERE SAL NOT BETWEEN 9000 AND15000

Write a Query to display the employee details whose name starts with „r‟

SELECT \* FROM EMP WHERE ENAME LIKE „r%‟

Write a Query to display the employee details whose name ends with „y‟

SELECT \* FROM EMP WHERE ENAME LIKE „%Y‟

Write a Query to display the employee details whose name contains the

letter „a‟

SELECT \* FROM EMP WHERE ENAME LIKE „%A%‟

Write a Query to display the employee details whose names contains only

three letters

SELECT \* FROM EMP WHERE ENAME LIKE „---„

Write a Query to display the employee details whose names contain „r‟ and

salary greater than 9000

SELECT \* FROM EMP WHERE ENAME LIKE „%R%‟ AND SAL>9000

Write a Query to display the employee details whose greater than ram

SELECT \* FROM EMP WHERE ENAME>‟RAM‟

Write a Query to display the employee details whose employee id starts with

1 and ends with 1

SELECT \* FROM EMP WHERE EID LIKE „1%1‟

(SQL commands are not case sensitive and also data available in SQL also not case

sensitive, in oracle Data available is case sensitive)

**Queries using ‘Update’ with ‘where’ clause:**

Write a query to change the deptno as „10‟whose employee id is 101, 103,

107

UPDATE EMPSET DEPTNO=10 WHERE EID=101 OR EID=103 OR

EID=107

Write a query to change the deptno as 20 who does not have deptno

UPDATE EMPSET DEPTNO=20 WHERE DEPTNO IS NULL

Write a query to change the employee salaries as 12000 who are working

under 10 dept and their names starts with „r‟

UPDATE EMPSET SAL=12000 WHERE DEPTNO=10 AND ENAME

LIKE „R%‟

Write a query to change the deptno as 30 whose second letter is „a‟

UPDATE EMPSET DEPTNO=30 WHERE ENAME=‟-A%‟

Write a query to change the employee salaries as 8500 who are working

under 10 and 20 deptno

UPDATE EMPSET SAL=8500 WHERE DEPTNO=10 OR DEPTNO=20

(OR)

UPDATE EMPSET SAL=8500 WHERE DEPTNO IN(10,20)

Write a query to change the employee salaries as 8500 who are not working

under 10 and 20 deptno

UPDATE EMPSET SAL=8500 WHERE DEPTNO NOT IN (10,20)

Write a query to change the employee salaries as 15000 and names ends

with „m‟ & working under 10 deptno

UPDATE EMPSET SAL=15000 WHERE ENAME=‟%M‟ AND

DEPTNO=10

Write a query to change the employee salaries as 5500 whose employee id

ends with 4 and deptno starts with 2

UPDATE EMPSET SAL=5500 WHERE EID LIKE „%4‟ AND DEPTNO

LIKE „2%‟

Write a query to change the employee salaries as 25000 whose salary less

than 10000 and the name contains letter „a‟ and working under dept 20

UPDATE EMPSET SAL=25000 WHERE SAL<10000 AND ENAME

LIKE „%A%‟ AND DEPTNO IN (20)

Write a query to change the employee salaries as 10000 whose salary is

greater than or equal to 8500 and less than or equal to 9000

UPDATE EMPSET SAL=10000 WHERE SAL BETWEEN 8500 AND

9000

**Set Operators:** Set operators combine results from two or more queries into a

single result set. SQL Server provides the following set operators.

UNION

UNION ALL

INTERSECT

EXCEPT

To combine the results of two queries we need to follow the below basic rules.

The number and the order of the columns must be the same in all queries.

The data types must be compatible(Well-Matched)

**Example**:

CREATE TABLE EMP\_HYD (EID INT, ENAME VARCHAR (50), SALARY

MONEY)

CREATE TABLE EMP\_CHENNAI (EID INT, ENAME VARCHAR (50))

EMP\_HYD

EID ENAME SALARY

101 SAI 25000.00

102 SIDDHU 32000.00

103 KAMAL 42000.00

104 NEETHU 63000.00

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

EMP\_CHENNAI

EID ENAME

101 SAI

105 POOJA

106 JASMIN

**UNION:** it combines the result of two or more select statements into a single result

set that includes all the records belongs to all queries except duplicate values.

**Ex:** Select Ename from EMP\_HYD

Union

Select Ename from EMP\_CHENNAI

**OUTPUT**: ENAME

JASMIN

KAMAL

NEETHU

POOJA

SAI

SIDDHU

**UNION ALL:** it is same as union but returns duplicate values

**Ex:** Select Ename from EMP\_HYD

Union ALL

Select Ename from EMP\_CHENNAI

**OUTPUT**: ENAME

SAI

SIDDHU

KAMAL

NEETHU

SAI

POOJA

JASMIN

**INTERSECT:** INTERSECT returns any distinct values that are common in left

and right tables.

**Ex:** Select Ename from EMP\_HYD

Intersect

Select Ename from EMP\_CHENNAI

**OUTPUT**: ENAME

SAI

**EXCEPT:** EXCEPT returns any distinct values from the left query that are not

found on the right query.

**Ex:** Select Ename from EMP\_HYD

Except

Select Ename from EMP\_CHENNAI

**OUTPUT**: ENAME

KAMAL

NEETHU

SIDDHU

**Ex:** Select Ename from EMP\_CHENNAI

Except

Select Ename from EMP\_HYD

**OUTPUT**: ENAME

JASMIN

POOJA

**CLAUSES**

**QL Server: GROUP BY Clause**

This SQL Server tutorial explains how to use the **GROUP BY clause** in SQL Server (Transact-SQL) with syntax and examples.

**Description**

The SQL Server (Transact-SQL) GROUP BY clause is used in a SELECT statement to collect data across multiple records and group the results by one or more columns.

**Syntax**

The syntax for the GROUP BY clause in SQL Server (Transact-SQL) is:

SELECT expression1, expression2, ... expression\_n,

aggregate\_function (expression)

FROM tables

[WHERE conditions]

GROUP BY expression1, expression2, ... expression\_n;

Parameters or Arguments

**expression1, expression2, ... expression\_n**

The expressions that are not encapsulated within an aggregate function and must be included in the GROUP BY clause.

**aggregate\_function**

It can be a function such as[SUM](https://www.techonthenet.com/sql_server/functions/sum.php),[COUNT](https://www.techonthenet.com/sql_server/functions/count.php),[MIN](https://www.techonthenet.com/sql_server/functions/min.php),[MAX](https://www.techonthenet.com/sql_server/functions/max.php), or [AVG](https://www.techonthenet.com/sql_server/functions/avg.php) functions.

**tables**

The tables that you wish to retrieve records from. There must be at least one table listed in the FROM clause.

**WHERE conditions**

Optional. The conditions that must be met for the records to be selected.

**Example - Using SUM function**

Let's look at a SQL Server GROUP BY query example that uses the [SUM function.](https://www.techonthenet.com/sql_server/functions/sum.php)

For example:

SELECT product\_name, SUM(quantity) AS "Total quantity"

FROM products

GROUP BY product\_name;

This SQL Server GROUP BY example uses the SUM function to return the *product\_name* and the total *quantity* (for the *product\_name*).

Because you have listed one column (the *product\_name* field) in your SELECT statement that is not encapsulated in the SUM function, you must use the GROUP BY clause. The *product\_name* field must, therefore, be listed in the GROUP BY clause.

**Example - Using COUNT function**

Let's look at how we could use the GROUP BY clause with the [COUNT function](https://www.techonthenet.com/sql_server/functions/count.php).

For example:

SELECT manager\_id, COUNT(\*) AS "Number of employees"

FROM employees

WHERE last\_name = 'Anderson'

GROUP BY manager\_id;

This GROUP BY example uses the COUNT function to return the *manager\_id* and the number of employees whose last\_name is 'Anderson'.

**Example - Using MIN function**

Let's next look at how we could use the GROUP BY clause with the [MIN function](https://www.techonthenet.com/sql_server/functions/min.php).

For example:

SELECT product\_type, MIN(quantity) AS "Lowest quantity"

FROM products

GROUP BY product\_type;

This GROUP BY example uses the MIN function to return the *product\_type* and the minimum quantity for that *product\_type*.

**Example - Using MAX function**

Finally, let's look at how we could use the GROUP BY clause with the [MAX function](https://www.techonthenet.com/sql_server/functions/max.php).

For example:

SELECT department, MAX(salary) AS "Highest salary"

FROM employees

GROUP BY department;

This GROUP BY example uses the MAX function to return the name of each department and the maximum salary in the department.

**SQL Server: HAVING Clause**

This SQL Server tutorial explains how to use the **HAVING clause** in SQL Server (Transact-SQL) with syntax and examples.

**Description**

The SQL Server (Transact-SQL) HAVING clause is used in combination with the[GROUP BY clause](https://www.techonthenet.com/sql_server/group_by.php) to restrict the groups of returned rows to only those whose the condition is TRUE.

**Syntax**

The syntax for the HAVING clause in SQL Server (Transact-SQL) is:

SELECT expression1, expression2, ... expression\_n,

aggregate\_function (expression)

FROM tables

[WHERE conditions]

GROUP BY expression1, expression2, ... expression\_n

HAVING having\_condition;

Parameters or Arguments

**aggregate\_function**

It can be a function such as [SUM](https://www.techonthenet.com/sql_server/functions/sum.php), [COUNT](https://www.techonthenet.com/sql_server/functions/count.php), [MIN](https://www.techonthenet.com/sql_server/functions/min.php), [MAX](https://www.techonthenet.com/sql_server/functions/max.php), or [AVG](https://www.techonthenet.com/sql_server/functions/avg.php) functions.

**expression1, expression2, ... expression\_n**

The expressions that are not encapsulated within an aggregate function and must be included in the GROUP BY clause.

**WHERE conditions**

Optional. These are the conditions for the records to be selected.

**HAVING having\_condition**

This is a further condition applied only to the aggregated results to restrict the groups of returned rows. Only those groups whose condition evaluates to TRUE will be included in the result set.

**Example - Using SUM function**

Let's look at a SQL Server HAVING clause example that uses the [SUM function.](https://www.techonthenet.com/sql_server/functions/sum.php)

For example:

SELECT department, SUM(quantity) AS "Total Quantity"

FROM products

GROUP BY department

HAVING SUM(quantity) > 100;

This HAVING clause example uses the [SUM function](https://www.techonthenet.com/sql_server/functions/sum.php) to return the name of the department and the total quantity (in the associated department). The SQL Server HAVING clause will filter the results so that only departments with total quantity greater than 100 will be returned.

**Example - Using COUNT function**

Let's look at how we could use the HAVING clause with the [COUNT function](https://www.techonthenet.com/sql_server/functions/count.php).

For example:

SELECT city, COUNT(\*) AS "Number of employees"

FROM employees

WHERE state = 'California'

GROUP BY city

HAVING COUNT(\*) > 20;

This HAVING clause example uses the [COUNT function](https://www.techonthenet.com/sql_server/functions/count.php) to return the city and the number of employees (residing in that city) that are in the state of 'California'. The SQL Server HAVING clause will filter the results so that only cities in California with more than 20 employees will be returned.

**Example - Using MIN function**

Let's next look at how we could use the HAVING clause with the [MIN function](https://www.techonthenet.com/sql_server/functions/min.php).

For example:

SELECT department, MIN(salary) AS "Lowest salary"

FROM employees

GROUP BY department

HAVING MIN(salary) >= 50000;

This HAVING example uses the [MIN function](https://www.techonthenet.com/sql_server/functions/min.php) to return the name of each department and the minimum salary in the department. The SQL Server HAVING clause will return only those departments where the minimum salary is greater than or equal to $50,000.

**Example - Using MAX function**

Finally, let's look at how we could use the HAVING clause with the [MAX function](https://www.techonthenet.com/sql_server/functions/max.php).

For example:

SELECT last\_name, MAX(salary) AS "Highest salary"

FROM employees

GROUP BY last\_name

HAVING MAX(salary) > 34000;

This HAVING clause example uses the [MAX function](https://www.techonthenet.com/sql_server/functions/max.php) to return the *last\_name* of the employee and the maximum salary for that *last\_name* value. The SQL Server HAVING clause will return only those *last\_name* values whose maximum salary is greater than $34,000.

**SQL Server: SELECT TOP Statement**

This SQL Server tutorial explains how to use the **SELECT TOP statement** in SQL Server (Transact-SQL) with syntax and examples.

**Description**

The SQL Server (Transact-SQL) SELECT TOP statement is used to retrieve records from one or more tables in SQL Server and limit the number of records returned based on a fixed value or percentage.

**Syntax**

The syntax for the SELECT TOP statement in SQL Server (Transact-SQL) is:

SELECT TOP (top\_value) [ PERCENT ] [ WITH TIES ]

expressions

FROM tables

[WHERE conditions]

[ORDER BY expression [ ASC | DESC ]];

Parameters or Arguments

**TOP (top\_value)**

Returns the top number of rows in the result set based on *top\_value*. For example, TOP(10) would return the top 10 rows from the full result set.

**PERCENT**

Optional. If PERCENT is specified, then the top rows are based on a percentage of the total result set (as specfied by the *top\_value*). For example, TOP(10) PERCENT would return the top 10% of the full result set.

**WITH TIES**

Optional. If the *WITH TIES* clause is specified, then rows tied in last place within the limited result set are returned. This may result in more rows be returned than the *TOP* parameter permits.

**expressions**

The columns or calculations that you wish to retrieve.

**tables**

The tables that you wish to retrieve records from. There must be at least one table listed in the FROM clause.

**WHERE conditions**

Optional. The conditions that must be met for the records to be selected.

**ORDER BY expression**

Optional. It is used in the SELECT TOP statement so that you can order the results and target those records that you wish to return. ASC is ascending order and DESC is descending order.

**Example - Using TOP keyword**

Let's look at a SQL Server example, where we use the TOP keyword in the SELECT statement.

For example:

SELECT TOP(5)

employee\_id, last\_name, first\_name

FROM employees

WHERE last\_name = 'Anderson'

ORDER BY employee\_id;

This SQL Server SELECT TOP example would select the first 5 records from the *employees* table where the *last\_name* is 'Anderson'. If there are other records in the *employees* table that have a *last\_name* of 'Anderson', they will not be returned by the SELECT statement.

You could modify this example to include the WITH TIES clause as follows:

SELECT TOP(5) WITH TIES

employee\_id, last\_name, first\_name

FROM employees

WHERE last\_name = 'Anderson'

ORDER BY employee\_id;

The *WITH TIES* clause would include rows that may be tied in last place within the limited result set. So if the 5th top record is a tie, then all records in this tie position would be returned by the SELECT TOP statement. This will result in more than 5 records being returned.

**Example - Using TOP PERCENT keyword**

Let's look at a SQL Server example, where we use the TOP PERCENT keyword in the SELECT statement.

For example:

SELECT TOP(10) PERCENT

employee\_id, last\_name, first\_name

FROM employees

WHERE last\_name = 'Anderson'

ORDER BY employee\_id;

This SQL Server SELECT TOP example would select the first 10% of the records from the full result set. So in this example, the SELECT statement would return the top 10% of records from the *employees* table where the *last\_name* is 'Anderson'. The other 90% of the result set would not be returned by the SELECT statement.

You could modify this example to include the WITH TIES clause as follows:

SELECT TOP(10) PERCENT WITH TIES

employee\_id, last\_name, first\_name

FROM employees

WHERE last\_name = 'Anderson'

ORDER BY employee\_id;

The *WITH TIES* clause would include rows that may be tied in last place within the limited result set. So if the last position in the SELECT TOP(10) PERCENT record set is a tie, then these tied records would be returned by the SELECT TOP statement. This will result in more than 10% of the full record set being returne