**🡺What Can SQL do?**

SQL can execute queries against a database

SQL can retrieve data from a database

SQL can insert records in a database

SQL can update records in a database

SQL can delete records from a database

SQL can create new databases

SQL can create new tables in a database

SQL can create stored procedures in a database

SQL can create views in a database

SQL can set permissions on tables, procedures, and views

**🡺Some of The Most Important SQL Commands**

**SELECT** - extracts data from a database

**UPDATE** - updates data in a database

**DELETE** - deletes data from a database

INSERT INTO - inserts new data into a database

CREATE DATABASE - creates a new database

ALTER DATABASE - modifies a database

CREATE TABLE - creates a new table

ALTER TABLE - modifies a table

DROP TABLE - deletes a table

CREATE INDEX - creates an index (search key)

DROP INDEX - deletes an index

**🡺**The **ORDER BY** keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

**🡺IS NULL**

**🡺IS NOT NULL**

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

SELECT *column\_names*FROM *table\_name*  
WHERE *column\_name* **IS NOT NULL**;

**🡺**The **SELECT TOP** clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

**🡺MIN(),MAX(),COUNT(),AVG(),SUM()**

**🡺LIKE**

WHERE CustomerName **LIKE** 'a%' Finds any values that start with "a"

WHERE CustomerName **LIKE** '%a' Finds any values that end with "a"

WHERE CustomerName **LIKE** '%or%' Finds any values that have "or" in any position

WHERE CustomerName **LIKE** '\_r%' Finds any values that have "r" in the second position

WHERE CustomerName **LIKE** 'a\_%' Finds any values that start with "a" and are at least 2 characters in length

WHERE CustomerName **LIKE** 'a\_\_%' Finds any values that start with "a" and are at least 3 characters in length

WHERE ContactName **LIKE** 'a%o' Finds any values that start with "a" and ends with "o"

**🡺Wildcard Characters in SQl Server**

**%** Represents zero or more characters bl% finds bl, black, blue, and blob

**\_** Represents a single character **h\_t** finds hot, hat, and hit

**[]** Represents any single character within the brackets **h[oa]t** finds hot and hat, but not hit

**^** Represents any character not in the brackets **h[^oa]t** finds hit, but not hot and hat

**-** Represents any single character within the specified range

**c[a-b]t** finds cat and cbt

**🡺**The **IN** operator allows you to specify multiple values in a WHERE clause.

WHERE column\_name **IN** (value1, value2, ...);

**🡺**The **BETWEEN** operator selects values within a given range. The values can be numbers, text, or dates.

WHERE column\_name **BETWEEN** value1 AND value2;

**🡺**The **GROUP BY** statement is often used with aggregate functions (**COUNT(), MAX(), MIN(), SUM(), AVG())** to group the result-set by one or more columns.

**🡺**The **HAVING** clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

SELECT **COUNT**(CustomerID), Country

FROM Customers

**GROUP BY** Country

**HAVING** COUNT(CustomerID) > 5;

**🡺**The **EXISTS** operator is used to test for the existence of any record in a subquery.

The **EXISTS** operator returns TRUE if the subquery returns one or more records.

SELECT column\_name(s)

FROM table\_name

WHERE **EXISTS**

(SELECT column\_name FROM table\_name WHERE condition);

**🡺The ANY operator:**

returns a boolean value as a result

returns TRUE if **ANY** of the subquery values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

SELECT column\_name(s)

FROM table\_name

WHERE column\_name operator **ANY**

(SELECT column\_name

FROM table\_name

WHERE condition);

Note: The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

**🡺The ALL operator:**

returns a boolean value as a result

returns TRUE if **ALL** of the subquery values meet the condition

is used with SELECT, WHERE and HAVING statements

ALL means that the condition will be true only if the operation is true for all values in the range.

SELECT column\_name(s)

FROM table\_name

WHERE column\_name operator **ALL**

(SELECT column\_name

FROM table\_name

WHERE condition);

**🡺**The **SELECT INTO** statement copies data from one table into a new table.

**SELECT \***

**INTO** newtable [IN externaldb]

FROM oldtable

WHERE condition;

**SELECT** column1, column2, column3, ...

**INTO** newtable [IN externaldb]

FROM oldtable

WHERE condition;

**🡺**The **INSERT INTO SELECT** statement copies data from one table and inserts it into another table.

The **INSERT INTO SELECT** statement requires that the data types in source and target tables match.

Note: The existing records in the target table are unaffected.

INSERT INTO table2

SELECT \* FROM table1

WHERE condition;

**INSERT INTO** Customers (CustomerName, City, Country)

**SELECT** SupplierName, City, Country FROM Suppliers

**WHERE** Country='Germany';

**🡺**The **CASE** statement goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

SELECT OrderID, Quantity,

**CASE**

WHEN Quantity > 30 THEN 'The quantity is greater than 30'

WHEN Quantity = 30 THEN 'The quantity is 30'

ELSE 'The quantity is under 30'

END AS QuantityText

FROM OrderDetails;

**🡺SQL IFNULL(), ISNULL(), COALESCE(), and NVL()** Functions

SELECT ProductName, UnitPrice \* (UnitsInStock + **ISNULL**(UnitsOnOrder, 0))  
FROM Products;

**🡺What is a Stored Procedure?**

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

**CREATE PROCEDURE** procedure\_name

AS

sql\_statement

GO;

**EXEC** procedure\_name;

**🡺SQL Logical Operators**

Operator Description Example

**ALL** TRUE if all of the subquery values meet the condition

**AND** TRUE if all the conditions separated by AND is TRUE

**ANY** TRUE if any of the subquery values meet the condition

**BETWEEN** TRUE if the operand is within the range of comparisons

**EXISTS** TRUE if the subquery returns one or more records

**IN** TRUE if the operand is equal to one of a list of expressions

**LIKE** TRUE if the operand matches a pattern

**NOT** Displays a record if the condition(s) is NOT TRUE

**OR** TRUE if any of the conditions separated by OR is TRUE

**SOME** TRUE if any of the subquery values meet the condition