|  |  |
| --- | --- |
| **SQL Statement** | **Syntax** |
| AND / OR | SELECT column\_name(s) FROM table\_name WHERE condition AND|OR condition |
| ALTER TABLE | ALTER TABLE table\_name  ADD column\_name datatype  or  ALTER TABLE table\_name  DROP COLUMN column\_name |
| AS (alias) | SELECT column\_name AS column\_alias FROM table\_name  or  SELECT column\_name FROM table\_name  AS table\_alias  e.g.  SELECT CustomerName, Address+', '+City+', '+PostalCode+', '+Country AS Address FROM Customers;  SELECT CustomerName, CONCAT(Address,', ',City,', ',PostalCode,', ',Country) AS Address FROM Customers; |
| BETWEEN /  NOT BETWEEN | SELECT column\_name(s) FROM table\_name WHERE column\_name BETWEEN value1 AND value2  Eg.  SELECT \* FROM Orders WHERE OrderDate BETWEEN #07/04/1996# AND #07/09/1996#; |
| CREATE DATABASE | CREATE DATABASE database\_name |
| CREATE TABLE | CREATE TABLE table\_name ( column\_name1 data\_type, column\_name2 data\_type, column\_name3 data\_type, ... ) |
| CREATE INDEX | CREATE INDEX index\_name ON table\_name (column\_name)  or  CREATE UNIQUE INDEX index\_name ON table\_name (column\_name) |
| CREATE VIEW | CREATE VIEW view\_name AS SELECT column\_name(s) FROM table\_name WHERE condition |
| constraint | CREATE TABLE table\_name ( column\_name1 data\_type(size) constraint\_name, column\_name2 data\_type(size) constraint\_name, column\_name3 data\_type(size) constraint\_name, .... );   * **NOT NULL** - Indicates that a column cannot store NULL value * **UNIQUE** - Ensures that each row for a column must have a unique value * **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have a unique identity which helps to find a particular record in a table more easily and quickly * **FOREIGN KEY** - Ensure the referential integrity of the data in one table to match values in another table * **CHECK** - Ensures that the value in a column meets a specific condition * **DEFAULT** - Specifies a default value for a column |
| DELETE | DELETE FROM table\_name WHERE some\_column=some\_value  or  DELETE FROM table\_name  (**Note:** Deletes the entire table!!)  DELETE \* FROM table\_name  (**Note:** Deletes the entire table!!) |
| DROP DATABASE | DROP DATABASE database\_name |
| DROP INDEX | DROP INDEX table\_name.index\_name (SQL Server) DROP INDEX index\_name ON table\_name (MS Access) DROP INDEX index\_name (DB2/Oracle) ALTER TABLE table\_name DROP INDEX index\_name (MySQL) |
| DROP TABLE | DROP TABLE table\_name |
| EXISTS | IF EXISTS (SELECT \* FROM table\_name WHERE id = ?) BEGIN --do what needs to be done if exists END ELSE BEGIN --do what needs to be done if not END |
| GROUP BY | SELECT column\_name, aggregate\_function(column\_name) FROM table\_name WHERE column\_name operator value GROUP BY column\_name |
| HAVING | SELECT column\_name, aggregate\_function(column\_name) FROM table\_name WHERE column\_name operator value GROUP BY column\_name HAVING aggregate\_function(column\_name) operator value |
| IN / NOT IN | SELECT column\_name(s) FROM table\_name WHERE column\_name IN (value1,value2,..) |
| INSERT INTO | INSERT INTO table\_name VALUES (value1, value2, value3,....)  *or*  INSERT INTO table\_name (column1, column2, column3,...) VALUES (value1, value2, value3,....) |
| INNER JOIN /  JOIN | This is simple join, An SQL INNER JOIN returns all rows from multiple tables where the join condition is met.  SELECT column\_name(s) FROM table\_name1 INNER JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name  Or  Select \*  from table1 join table2 using(column\_name)  where… |
| LEFT JOIN /  LEFT OUTER JOIN | The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.  SELECT column\_name(s) FROM table\_name1 LEFT JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| RIGHT JOIN/  RIGHT OUTER JOIN | The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.  SELECT column\_name(s) FROM table\_name1 RIGHT JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| FULL JOIN/  FULL OUTER JOIN | SELECT column\_name(s) FROM table\_name1 FULL JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| LIKE / NOT LIKE | SELECT column\_name(s) FROM table\_name WHERE column\_name LIKE pattern  Eg.  SELECT \* FROM Customers WHERE Country NOT LIKE '%land%'; |
| ORDER BY | SELECT column\_name(s) FROM table\_name ORDER BY column\_name [ASC|DESC], another\_column\_name |
| SELECT | SELECT column\_name(s) FROM table\_name |
| SELECT \* | SELECT \* FROM table\_name |
| SELECT DISTINCT | SELECT DISTINCT column\_name(s) FROM table\_name |
| SELECT INTO | SELECT \* INTO new\_table\_name [IN externaldatabase] FROM old\_table\_name  *or*  SELECT column\_name(s) INTO new\_table\_name [IN externaldatabase] FROM old\_table\_name |
| SELECT TOP | SELECT TOP number|percent column\_name(s) FROM table\_name  My SQL:  SELECT \* FROM table\_name  LIMIT 5 |
| TRUNCATE TABLE | TRUNCATE TABLE table\_name |
| UNION | Distinct by default  SELECT column\_name(s) FROM table\_name1 UNION SELECT column\_name(s) FROM table\_name2 |
| UNION ALL | Not distinct  SELECT column\_name(s) FROM table\_name1 UNION ALL SELECT column\_name(s) FROM table\_name2 |
| UPDATE | UPDATE table\_name SET column1=value, column2=value,... WHERE some\_column=some\_value |
| WHERE | SELECT column\_name(s) FROM table\_name WHERE column\_name operator value |

Some other Syntax:

Intersect

Except

Exists/Not exists

All

Any

Case when then else end

If when else end

If(condition, A, B )

|  |  |
| --- | --- |
| **Operator** | **Description** |
| = | Equal |
| <> | Not equal. **Note:** In some versions of SQL this operator may be written as != |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal |
| <= | Less than or equal |
| BETWEEN | Between an inclusive range |
| LIKE | Search for a pattern |
| IN / NOT IN | To specify multiple possible values for a column |

**SQL Wildcard Characters**

In SQL, wildcard characters are used with the SQL LIKE operator.

SQL wildcards are used to search for data within a table.

With SQL, the wildcards are:

|  |  |
| --- | --- |
| **Wildcard** | **Description** |
| % | A substitute for zero or more characters |
| \_ | A substitute for a single character |
| [*charlist*] | Sets and ranges of characters to match |
| [^*charlist*] or [!*charlist*] | Matches only a character NOT specified within the brackets |

Injection: Server Code:

txtUserId = getRequestString("UserId");  
txtSQL = "SELECT \* FROM Users WHERE UserId = " + txtUserId;

Parameters for Protectin

txtUserId = getRequestString("UserId");  
txtSQL = "SELECT \* FROM Users WHERE UserId = @0";  
db.Execute(txtSQL,txtUserId);

**Tip:** The SELECT INTO statement can also be used to create a new, empty table using the schema of another. Just add a WHERE clause that causes the query to return no data:

SELECT \*  
INTO newtable  
FROM table1  
WHERE 1=0;

## Function:

## SQL General Data Types

|  |  |
| --- | --- |
| **Data type** | **Description** |
| CHARACTER(n) | Character string. Fixed-length n |
| VARCHAR(n) or CHARACTER VARYING(n) | Character string. Variable length. Maximum length n |
| BINARY(n) | Binary string. Fixed-length n |
| BOOLEAN | Stores TRUE or FALSE values |
| VARBINARY(n) or BINARY VARYING(n) | Binary string. Variable length. Maximum length n |
| INTEGER(p) | Integer numerical (no decimal). Precision p |
| SMALLINT | Integer numerical (no decimal). Precision 5 |
| INTEGER | Integer numerical (no decimal). Precision 10 |
| BIGINT | Integer numerical (no decimal). Precision 19 |
| DECIMAL(p,s) | Exact numerical, precision p, scale s. Example: decimal(5,2) is a number that has 3 digits before the decimal and 2 digits after the decimal |
| NUMERIC(p,s) | Exact numerical, precision p, scale s. (Same as DECIMAL) |
| FLOAT(p) | Approximate numerical, mantissa precision p. A floating number in base 10 exponential notation. The size argument for this type consists of a single number specifying the minimum precision |
| REAL | Approximate numerical, mantissa precision 7 |
| FLOAT | Approximate numerical, mantissa precision 16 |
| DOUBLE PRECISION | Approximate numerical, mantissa precision 16 |
| DATE | Stores year, month, and day values |
| TIME | Stores hour, minute, and second values |
| TIMESTAMP | Stores year, month, day, hour, minute, and second values |
| INTERVAL | Composed of a number of integer fields, representing a period of time, depending on the type of interval |
| ARRAY | A set-length and ordered collection of elements |
| MULTISET | A variable-length and unordered collection of elements |
| XML | Stores XML data |

## SQL Data Type Quick Reference

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data type** | **Access** | **SQLServer** | **Oracle** | **MySQL** | **PostgreSQL** |
| *boolean* | Yes/No | Bit | Byte | N/A | Boolean |
| *integer* | Number (integer) | Int | Number | Int Integer | Int Integer |
| *float* | Number (single) | Float Real | Number | Float | Numeric |
| *currency* | Currency | Money | N/A | N/A | Money |
| *string (fixed)* | N/A | Char | Char | Char | Char |
| *string (variable)* | Text (<256) Memo (65k+) | Varchar | Varchar Varchar2 | Varchar | Varchar |
| *binary object* | OLE Object Memo | Binary (fixed up to 8K) Varbinary (<8K) Image (<2GB) | Long Raw | Blob Text | Binary Varbinary |

## SQL Aggregate Functions

SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

* AVG() - Returns the average value
* COUNT() - Returns the number of rows eg. SELECT COUNT(DISTINCT column\_name) FROM table\_name;
* FIRST() - Returns the first value.

Same as:

SELECT column\_name FROM table\_name  
ORDER BY column\_name ASC  
LIMIT 1;

* LAST() - Returns the last value
* MAX() - Returns the largest value
* MIN() - Returns the smallest value
* SUM() - Returns the sum

## SQL Scalar functions

SQL scalar functions return a single value, based on the input value.

Useful scalar functions:

* UCASE() - Converts a field to upper case
* LCASE() - Converts a field to lower case
* MID() - Extract characters from a text field
* LEN() - Returns the length of a text field
* ROUND() - Rounds a numeric field to the number of decimals specified
* NOW() - Returns the current system date and time
* FORMAT() - Formats how a field is to be displayed