

SQL

What is SQL?

SQL is the abbreviation for structured query language, and you pronounce it either see-kwell or S-Q-L. Many different versions of SQL exist, but it's the current industry standard query language for requesting information from databases.

The SQL language is widely used today across web frameworks and database applications. Knowing SQL gives you the freedom to explore your data, and the power to make better decisions. By learning SQL, you will also learn concepts that apply to nearly every data storage system.

* **SELECT**

A **SELECT** query tells SQL Server what data you want from it. Technically speaking, SQL Server follows an ANSI-standard SQL syntax. A basic **SELECT** query takes the following form:

* SELECT { \* | column,column,column } FROM table [WHERE criteria] [ORDER BY column]

After the keyword **SELECT**, specify **\*** if every column from the table should be included. Otherwise you should provide a comma-delimited list of column names that you want.

Then use the **FROM** keyword to specify the table name the columns live in.

By default, SQL Server will return every row in the table. You can also specify a **WHERE** clause, which lists criteria for the rows you want returned.

* **Using the WHERE clause to specify certain records**

We’ve seen how the SELECT command lets you limit the number of columns that appear in your results—you simply list the columns you want to see. But by default, SQL returns all of the records in the table. Most of the time, when you’re querying a table, you want to display records that meet certain conditions or criteria.

For instance, suppose you want to display only the records for customers who live in Zip code 10010. Assuming that the Zip\_code field is a character field (not numeric), you'd issue a SELECT statement in the form:

* SELECT FName, Last\_Name, Phone FROM Customers WHERE Zip\_code="101010" ORDER BY Lname,Fname

In the WHERE clause, if the field you're checking is numeric, you don't need to include quotation marks around the data. For example, if you want to find customers whose last\_purchase field contains a value of 100 or greater, you'd issue a command like this:

* SELECT FName, LName, Phone FROM Customers WHERE Last\_purchase >=100
* **Looking to the records**

In SQL, SELECT means "show me." Specifically, it means "show me some records out of some table." Here’s the basic outline of a SELECT statement:

* SELECT [some or all of the columns] FROM [one or more tables] WHERE [the records meet some condition]

When you issue a SELECT command, SQL scans the records in the table and displays the results on-screen. To see all of the columns in the table, type an asterisk (\*) immediately following

* SELECT \* FROM mytable

To limit the number of columns in your results, simply list the columns you want in this form:

SELECT column1, column2, column3 FROM table

EXample:

* SELECT Name, Last\_Name, Phone FROM Customers
* **Controlling the order in which the records appear**

In addition to deciding which columns SQL displays, you can also control the order in which those records appear by adding the ORDER BY clause. Just add it to your SELECT statement in the form:

* SELECT Column1, Column2, Column3 FROM mytable ORDER BY Column3

By default, SQL assumes you want those records in ascending order. If you want to display the records in descending order, you add the keyword DESC to the SQL command in this form:

* SELECT Column1, Column2, Column3 FROM tablename ORDER BY Column3 DESC
* **BETWEEN, AND, LIKE (which uses % as wildcards), OR, and IN .**

The **BETWEEN** operator specifies ranges (note that dates are treated as strings and included in single quotes).

The **IN** operator provides a comma-separated list of acceptable values.

A useful alternative for the **IN** operator is a subquery:

…WHERE InvoiceNumber IN (SELECT InvoiceNumber FROM BadOrders)

The subquery executes first, and the**WHERE** clause will include all rows where the **InvoiceNumber** appears in the **BadOrders** table. Finally, the optional **ORDER BY** clause lets you specify the order that the rows are returned. Once again you can specify columns not included in the output. For example, you might sort on a date field you don't need to look at.

Here are some more examples:

* ORDER BY DateOrdered
* ORDER BY DateOrdered DESCENDING
* ORDER BY DateOrdered,InvoiceNumber

The default sort order is ascending, and you can specify the **ASCENDING** keyword if you wish. The **DESCENDING** keyword reverses that order.

The final option I want point out is you can select a subset of matching rows. For example:

* SELECT TOP 10 \* FROM Orders WHERE OrderAmount > 5000 ORDER BY DateOrdered DESCENDING

This query grabs all the rows that have an order amount more than 5,000 and orders them with the most recently-placed order first. It will then send you only the first 10 of those rows, and will include every column in the table.

* **INSERT**

The **INSERT** query allows you to add new rows to the database. To do this you need to specify a table to put the data in, the columns you are providing values for and then the values themselves. Rows are inserted one at a time.

query:

* INSERT INTO table (column,column,column) VALUES(value,value,value)

There is a way to omit the column list, but from a practical perspective it's easier to include it.

The values are listed in the same order as the columns – that's how SQL Server knows which value goes where. You need to specify columns and values for any column that doesn't either (a) have a default value or (b) allow an empty (null) value. Both the default and the "nullability" of a column is defined in the table's design.

Here's an **INSERT** example:

* INSERT INTO Orders (Invoice,Amount,ShipMethod) VALUES(10383,30.00,'Express')

The two biggest "gotchas" involve making sure you specify every column that requires a value (and not provided by default), and getting your punctuation – those single quotes and commas – correct.

* **UPDATE**

When you need to make a change to existing data, an **UPDATE** query is comes into play.

This type of query is designed to update multiple rows at once and takes the following basic form:

* UPDATE table SET column = value, column = value [WHERE criteria]

To change the completed column on *every* row in the **Orders** table, enter:

* UPDATE Orders SET Completed = 1

However, you probably won't always want to change every row. In this case, add a **WHERE** clause:

* UPDATE Orders SET Completed = 1 WHERE DateOrdered = DATE()
* **DELETE**

**DELETE** query, which removes rows from a table.

This query is almost always combined with a **WHERE** clause:

* DELETE FROM Customers WHERE Inactive = 1

Without a **WHERE** clause, every row in the table will be deleted!