Introduction to Access SQL

https://support.office.com/en-za/article/Introduction-to-Access-SQL-d5f21d10-cd73-4507-925e-bb26e377fe7e

SQL (Structured Query Language) is a computer language that closely resembles English that database programs understand. Knowing SQL is important because every query in Microsoft Access uses SQL. Understanding how SQL works can help create better queries, and can make it easier for you to fix a query when it is not returning the results that you want.

NOTE TO SELF: this should be part of single table queries going forward

Here is an example of a simple SQL statement that retrieves a list of last names for contacts whose first name is Mary might resemble this:

```
SELECT Last_Name                    This is the field
FROM Contacts                       This is the table
WHERE First_Name = 'Mary';          This is the criteria
```

SELECT statements

To describe a set of data by using SQL, you write a SELECT statement. A SELECT statement contains a complete description of a set of data that you want to obtain from a database. This includes the following:

- What tables contain the data.
- How data from different sources is related.
- Which fields or calculations will produce the data.
- Criteria that data must match to be included.
- Whether and how to sort the results.

SQL clauses

Like a sentence, a SQL statement has clauses. Each clause performs a function for the SQL statement. Some clauses are required in a SELECT statement.

SQL clause	What it does	Required
SELECT	Lists the fields that contain data of interest.	Yes
FROM	Lists the tables that contain the fields listed in the SELECT clause.	Yes
WHERE	Specifies field criteria that must be met by each record to be included in the results.	No
ORDER BY	Specifies how to sort the results.	No
GROUP BY	In a SQL statement that contains aggregate functions, lists fields that are not summarized in the SELECT clause.	Only if there are such fields
HAVING	In a SQL statement that contains aggregate functions, specifies conditions that apply to fields that are summarized in the SELECT statement.	No

SQL terms

Each SQL clause is composed of terms — comparable to parts of speech.

SQL term	Comparable part of speech	Definition	Example
identifier	noun	A name that you use to identify a database object, such as the name of a field.	Customers. [Phone Number]
operator	verb or adverb	A keyword that represents an action or modifies an action.	AS
constant	noun	A value that does not change, such as a number or NULL.	42
expression	adjective	A combination of identifiers, operators, constants, and functions that evaluates to a single value.	>= Products. [Unit Price]

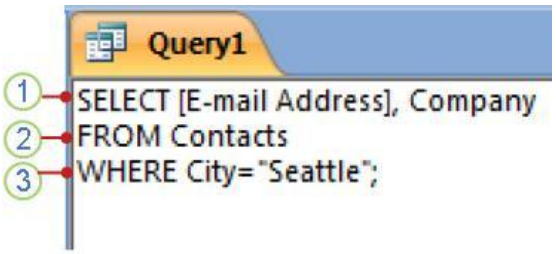
Basic SQL clauses: SELECT, FROM, and WHERE

An SQL statement takes the general form:

```
SELECT field_1
FROM table_1
WHERE criterion_1
;
```

- Access ignores line breaks in a SQL statement. However, consider using a line for each clause to help improve the readability of your SQL statements for yourself and others.
- Every SELECT statement ends with a semi-colon (;). The semi-colon can appear at the end of the last clause or on a line by itself at the end of the SQL statement.

The following illustrates what a SQL statement for a simple select query might look like in Access:



1. SELECT clause
2. FROM clause
3. WHERE clause

This example SQL statement reads "Select the data that is stored in the fields named E-mail Address and Company from the table named Contacts, specifically those records in which the value of the field City is Seattle."

Let's look at the example, one clause at a time, to see how SQL syntax works.

The SELECT clause

SELECT [E-mail Address], Company

This is the SELECT clause. It consists of an operator (SELECT) followed by two identifiers ([E-mail Address] and Company).

If an identifier contains spaces or special characters (such as "E-mail Address"), it must be enclosed in square brackets.

A SELECT clause does not have to say which tables contain the fields, and it cannot specify any conditions that must be met by the data to be included.

The SELECT clause always appears in front of the FROM clause in a SELECT statement.

The FROM clause

FROM Contacts

This is the FROM clause. It consists of an operator (FROM) followed by an identifier (Contacts).

A FROM clause does not list the fields to be selected.

The WHERE clause

WHERE City="Seattle"

This is the WHERE clause. It consists of an operator (WHERE) followed by an expression (City="Seattle").

NOTE: Unlike the SELECT and FROM clauses, the WHERE clause is not a required element of a SELECT statement.

Sorting the results: ORDER BY

Like Microsoft Office Excel, Access lets you sort query results in a datasheet. You can also specify in the query how you want to sort the results when the query is run, by using an ORDER BY clause. If you use an ORDER BY clause, it is the last clause in the SQL statement.

An ORDER BY clause contains a list of the fields that you want to use for sorting, in the same order that you want to apply the sort operations.

For example, suppose that you want your results sorted first by the value of the field Company in descending order, and — if there are records with the same value for Company — sorted next by the values in the field E-mail Address in ascending order. Your ORDER BY clause would resemble the following:

ORDER BY Company DESC, [E-mail Address]

NOTE By default, Access sorts values in ascending order (A-Z, smallest to largest). Use the DESC keyword to sort values in descending order instead.

Working with summarized data: GROUP BY and HAVING

Sometimes you want to work with summarized data, such as the total sales in a month, or the most expensive items in an inventory. To do this, you apply an aggregate function to a field in your SELECT clause. For example, if you want your query to show the count of e-mail addresses listed for each company, your SELECT clause might resemble the following:

SELECT COUNT([E-mail Address]), Company

The aggregate functions that you can use depend on the type of data that is in the field or expression that you want to use.

Specifying fields that are not used in an aggregate function: The GROUP BY clause

When you use aggregate functions, you usually must also create a GROUP BY clause. A GROUP BY clause lists all the fields to which you do not apply an aggregate function. If you apply aggregate functions to all the fields in a query, you do not have to create the GROUP BY clause.

A GROUP BY clause immediately follows the WHERE clause, or the FROM clause if there is no WHERE clause. A GROUP BY clause lists the fields as they appear in the SELECT clause.

For example, continuing the previous example, if your SELECT clause applies an aggregate function to [E-mail Address] but not to Company, your GROUP BY clause would resemble the following:

GROUP BY Company

Limiting aggregate values by using group criteria: the HAVING clause

If you want to use criteria to limit your results, but the field that you want to apply criteria to is used in an aggregate function, you cannot use a WHERE clause. Instead, you use a HAVING clause. A HAVING clause works like a WHERE clause, but is used for aggregated data.

For example, suppose that you use the AVG function (which calculates an average value) with the first field in your SELECT clause:

SELECT COUNT([E-mail Address]), Company

If you want the query to restrict the results based on the value of that COUNT function, you cannot use a criteria for that field in the WHERE clause. Instead, you put the criteria in a HAVING clause. For example, if you only want the query to return rows if there are more than one e-mail addresses associated with the company, the HAVING clause might resemble the following:

HAVING COUNT([E-mail Address])>1

NOTE A query can have a WHERE clause and a HAVING clause — criteria for fields that are not used in an aggregate function go in the WHERE clause, and criteria for fields that are used with aggregate functions go in the HAVING clause.

TO BE CONTINUED (have not yet covered)

http://www.sqlcourse.com/create.html
http://www.w3schools.com/sql/sql_foreignkey.asp cool “class” on sql

```
CREATE TABLE employees
( employee_number number(10) not null,
  employee_name varchar2(50) not null,
  salary number(6),
  CONSTRAINT employees_pk PRIMARY KEY (employee_number)
);

INSERT INTO employees (employee_number, employee_name, salary)
VALUES (1001, 'John Smith', 62000);

INSERT INTO employees (employee_number, employee_name, salary)
VALUES (1002, 'Jane Anderson', 57500);

INSERT INTO employees (employee_number, employee_name, salary)
VALUES (1003, 'Brad Everest', 71000);

INSERT INTO employees (employee_number, employee_name, salary)
VALUES (1004, 'Jack Horvath', 42000);
```