



Lesson 12: Basic queries using SQL

12.1 Introduction: The difference between QBE and SQL

Query-By-Example (QBE) and Structured Query Language (SQL) are both well-known, industry-standard languages for extracting information from relational database systems. The advantage of QBE (as you saw in [Lesson 10](#) and [Lesson 11](#)) that it is graphical and relatively easy to use. The advantage of SQL is that it has achieved nearly universal adoption within the relational database world.

With only a few exceptions (which you probably will not encounter in this project) QBE and SQL are completely interchangeable. If you understand the underlying concepts (projection, selection, sorting, joining, and calculated fields) of one, you understand the underlying concepts of both. In fact, in ACCESS you can switch between QBE and SQL versions of your queries with the click of a mouse.

Although you typically use QBE to create queries in ACCESS, the ubiquity of SQL in the rest of the database world necessitates a brief overview.

Introduction: The difference between QBE and

12.2 Learning objectives

- understand the difference between QBE and SQL
- create an SQL query
- use SQL as a data definition language

12.3 Exercises

12.3.1 Select queries

Recall from [Lesson 10](#) that a select query is a query that allows you to select and organize data from one or more underlying tables. In these exercises, you are going to use SQL to achieve the same result.



The queries you build in these exercises are for practice only. They are not an integral part of your order entry project and it is not critical that they be saved as part of your database.

- ➔ Create a new query, but close the "Show Table" dialog box without adding any tables.
- ➔ Select **View** → **SQL** from the main menu to switch to the SQL editor. You get white text editor that contains nothing but an empty SELECT statement.



A typical SQL statement resembles the following:

```
NL SELECT ProductID, Description
NL FROM Products
NL WHERE Unit <> "ea";
```

There are four parts to the SQL statement:

1. **SELECT** *<field₁, field₂, ..., field_n>* ...
— specifies which fields to project;
2. ... **FROM** *<table>* ... — specifies the underlying table (or tables) for the query;
3. ... **WHERE** *<condition₁ AND/OR condition₂, ..., AND/OR condition_n>* — specifies one or more conditions that each record must satisfy in order to be included in the results set;
4. **;** (semicolon) — all SQL statements must end with a semicolon (but if you forget, ACCESS will add one for you).



Despite the use of the new line symbol (NL) in the examples in this lesson, the SQL interpreter ignores whitespace and allows lines to be split between any two words. In order to make the statements more readable, however, it is common practice is to use a new line for each major SQL keyword (**SELECT**, **FROM**, **WHERE**, etc.).

You will now use these basic constructs to create your own SQL query:

(lesson12-1.avi)

➔ Type the following into the SQL window:

```
NL SELECT ProductID, Description
NL FROM Products
NL WHERE Unit <> "ea";
```



If you look closely at your keyboard, you will notice that there is an equals sign (=) but no not-equals sign (≠). In some computer languages—including SQL—not-equals is represented using the less-than and greater-than signs together (<>).

➔ Select **View** → **Datasheet** to view the result, as shown in [Figure 12.1](#).

➔ Select **View** → **Query Design** to view the query in QBE mode, as shown in [Figure 12.2](#).



ACCESS automatically translates between QBE and SQL versions of the query.

➔ Save your query as **qrySQL**.

12.3.2 Complex WHERE clauses

As in QBE, you can use Boolean operators such as AND, OR, and NOT in your WHERE clauses to specify complex selection criteria.



FIGURE 12.2: The SQL and QBE views are interchangeable.

1 Use the **View** menu to switch between the QBE ("design"), SQL, and datasheet views of your query.

When you return to SQL view after modifying your query in QBE view, you will notice that ACCESS has added some additional text (square brackets, etc.). This optional text does not change the query in any way.

The screenshot shows the Microsoft Access interface. The 'View' menu is open, showing options: Design View, SQL View, Datasheet View, Totals, Table Names, Properties, Join Properties, and Toolbars. The 'SQL View' option is selected. The main window displays the SQL view of a query named 'Query1: Select Query'. The SQL code is:


```
SELECT Products.ProductID, Products.Description
FROM Products
WHERE (((Products.Unit)<>'ea');
```

 Below the SQL code, there is a table grid showing the fields: ProductID, Description, and UnitPrice. The 'Show' column has checkboxes for each field, and the 'Criteria' column is empty. A yellow callout box with a question mark icon explains that when returning to SQL view from QBE, Access adds optional text like square brackets, which does not change the query's meaning.

- ➔ Change your query to show all the products that normally sell for less than \$2 each AND all the products that are not sold in units of one, but which normally sell for less than \$5.

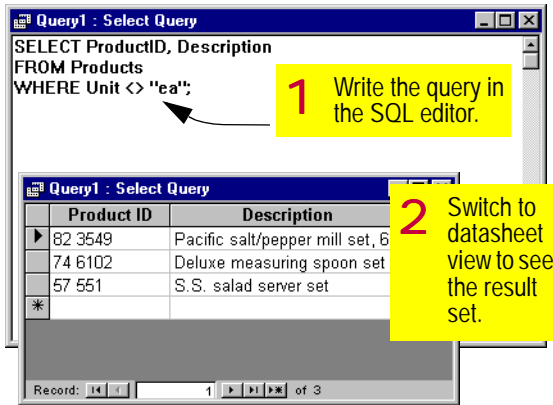
```
NL SELECT *
NL FROM Products
NL WHERE UnitPrice < 2
NL OR (UnitPrice < 5 AND NOT
NL Unit = "ea");
```

- ❓ As in QBE, selecting the asterisk (*) is shorthand for all fields in the underlying table. Thus, **SELECT *** is identical to **SELECT ProductID, Description, Unit, UnitPrice ...**

- ⚠ Note that since **unit** is a text field, its criterion must be a string (in this case, a **literal string** enclosed in quotation marks). **UnitPrice**, in contrast, is a numeric field and its criterion must be a number (no quotation marks).



FIGURE 12.1: A simple SQL select query.



SQL is a computer language. And like any computer language, it is picky about what you type. If your query contains spelling mistakes or other errors, it will not execute.

12.3.3 Join queries

When you create a join query in QBE and switch to SQL, you see an “INNER JOIN” statement that is unique to MICROSOFT’S SQL for the JET database engine. The ANSI¹ standard version of SQL uses a different syntax for joining tables (which

ACCESS also supports). In this section, you will use the ANSI standard approach to join the **Customers** and **Regions** tables.

- ➔ Create a new SQL query called `qrySQLJoin` and use the WHERE clause to specify the join relationships between the tables:

```
NL SELECT CustName, City, RegionName
NL FROM Customers, Regions
NL WHERE Customers.RegionCode =
      Regions.RegionCode
NL AND RepID = 9
NL ORDER BY CustName;
```

- ➔ View the result set and confirm that all the customers in regions serviced by **RepID = 9** (Ben Sidhu) are included, as shown in [Figure 12.3](#).



If you neglected to populate the **Regions.RepID** column in [Section 8.5](#), your query will return an empty recordset.

When we created join queries in QBE (recall [Section 10.3.2.5](#)), we did not worry about linking primary keys to foreign keys—all the relationships were inherited from those created using the relationships window in [Lesson 7](#).

However, the relationships window is a feature of ACCESS, not relational databases generally. Thus, SQL provides its own mechanism for

¹ American National Standards Institute



FIGURE 12.3: Using SQL to join two tables.

```

qrySQLJoin : Select Query
SELECT CustName, City, RegionName
FROM Customers, Regions
WHERE Customers.RegionCode=Regions.RegionCode
AND ReplID=9
ORDER BY CustName;

```

The **ORDER BY** clause sorts the results in ascending order by customer name.

A join in SQL is achieved by specifying the matching condition in the WHERE clause.

Customer name	City	RegionName
Gadgets "R" Us	North Vancouver	Central
Loonie Mart #107	Vancouver	Key Accounts
Rosch Dry Goods Inc.	Calgary	Key Accounts
Sam's Stock Pot	Vancouver	Central

Record: 1 of 4

specifying joins using the WHERE clause. To illustrate, consider the first part of the WHERE clause you just created:

```

WHERE Customers.RegionCode =
Regions.RegionCode

```

The only records that are joined from the **Customers** and **Regions** table are those for which the value of **RegionCode** is the same in both tables. Thus, you use the WHERE clause in SQL to specify equality between the primary

key on the “one” side and the foreign key on the “many” side. This equality condition is equivalent to the little connecting lines between tables used by Access in the QBE interface and the relationships window.



The second part of the WHERE clause (**AND RepID = 9**) is simply a standard selection criterion.

12.3.4 SQL as a data definition language

A distinction is often made between two types of database language constructs: **data definition language** (DDL) and **data manipulation language** (DML). So far, we have used QBE and SQL for data manipulation (joining, sorting, etc.). For data definition tasks (e.g., creating tables), we have used ACCESS’s table design form. However, like most databases, ACCESS support the use of SQL as a DDL.

➔ Open a new SQL query and type the following:

```

NL CREATE TABLE Suppliers
NL (SuppID INTEGER NOT NULL,
NL SuppName VARCHAR(20),
NL Phone VARCHAR(14),
NL CONSTRAINT PK_Suppliers PRIMARY KEY
NL (SuppID));

```



FIGURE 12.4: Using SQL as a data definition language.

1 Write a data definition SQL statement to create a list of suppliers.

```

CREATE TABLE Suppliers
(SuppID INTEGER NOT NULL,
 SuppName VARCHAR(20),
 Phone VARCHAR(14),
 CONSTRAINT PK_Suppliers PRIMARY KEY (SuppID));
  
```

2 Execute the SQL statement by selecting **Query** → **Run** from the main menu.

Suppliers : Table		
Field Name	Data Type	Description
SuppID	Number	
SuppName	Text	
Phone	Text	

Field Properties	
General	Lookup
Field Size	Long Integer
Format	
Decimal Places	Auto
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	Yes
Indexed	Yes (No Duplicates)

3 Examine the resulting table definition.

? Note that SQL does not support a number of the ACCESS-specific field properties (such as input masks). These must be created using the table design form.

Rather than using the ACCESS-specific data types Long and Text, the query uses standard SQL data types “Integer” and “Varchar” respectively. Fortunately, ACCESS is smart enough to map the standard SQL data types to its own internal data types.

? SQL is (nominally) a vendor-independent standard. As a consequence, the DDL SQL statement in Figure 12.4 will execute correctly on ORACLE, MICROSOFT SQL SERVER, or any other SQL-compliant DBMS. This would not be the case if ACCESS-specific data types were used.

- Select **Query** → **Run** from the main menu or press the exclamation mark (!) on the toolbar.
- Switch to the **Tables** tab of the database window and open the new **suppliers** table in design view.

As Figure 12.4, shows, the SQL statement created the table, specified the field names and data types, and set the primary key.



12.4 Discussion

Although the syntax of SQL is not particularly difficult, writing long SQL queries is tedious and error-prone. For this reason, you are advised to use QBE for your project.



When you say you know something about databases, it usually implies you know the DDL and DML aspects of SQL in your sleep. If you plan to pursue a career in information systems, a comprehensive SQL reference book and lots of practice can be worthwhile investments.