

Introduction to SQL

7 minutes

SQL stands for Structured Query Language. SQL is used to communicate with a relational database. It's the standard language for relational database management systems. SQL statements are used to perform tasks such as update data in a database, or retrieve data from a database. Some common relational database management systems that use SQL include Microsoft SQL Server, MySQL, PostgreSQL, MariaDB, and Oracle.

Note

SQL was originally standardized by the American National Standards Institute (ANSI) in 1986, and by the International Organization for Standardization (ISO) in 1987. Since then, the standard has been extended several times as relational database vendors have added new features to their systems. Additionally, most database vendors include their own proprietary extensions that are not part of the standard, which has resulted in a variety of dialects of SQL.

In this unit, you'll learn about SQL. You'll see how it's used to query and maintain data in a database, and the different dialects that are available.

Understand SQL dialects

You can use SQL statements such as **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **CREATE**, and **DROP** to accomplish almost everything that one needs to do with a database. Although these SQL statements are part of the SQL standard, many database management systems also have their own additional proprietary extensions to handle the specifics of that database management system. These extensions provide functionality not covered by the SQL standard, and include areas such as security management and programmability. Microsoft SQL Server, for example, uses Transact-SQL. This implementation includes proprietary extensions for writing stored procedures and triggers (application code that can be stored in the database), and managing user accounts. PostgreSQL and MySQL also have their own versions of these features.

Some popular dialects of SQL include:

- *Transact-SQL (T-SQL)*. This version of SQL is used by Microsoft SQL Server and Azure SQL Database.

- *pgSQL*. This is the dialect, with extensions implemented in PostgreSQL.
- *PL/SQL*. This is the dialect used by Oracle. PL/SQL stands for Procedural Language/SQL.

Users who plan to work specifically with a single database system should learn the intricacies of their preferred SQL dialect and platform.

Understand SQL statement types

SQL statements are grouped into two main logical groups, and they are:

- Data Manipulation Language (DML)
- Data Definition Language (DDL)

Use DML statements

You use DML statements to manipulate the rows in a relational table. These statements enable you to retrieve (query) data, insert new rows, or edit existing rows. You can also delete rows if you don't need them anymore.

The four main DML statements are:


Statement	Description
SELECT	Select/Read rows from a table
INSERT	Insert new rows into a table
UPDATE	Edit/Update existing rows
DELETE	Delete existing rows in a table

The basic form of an **INSERT** statement will insert one row at a time. By default, the **SELECT**, **UPDATE**, and **DELETE** statements are applied to every row in a table. You usually apply a **WHERE** clause with these statements to specify criteria; only rows that match these criteria will be selected, updated, or deleted.


Warning

SQL doesn't provide *are you sure?* prompts, so be careful when using **DELETE** or **UPDATE** without a **WHERE** clause because you can lose or modify a lot of data.

The following code is an example of a SQL statement that selects all rows that match a single filter from a table. The **FROM** clause specifies the table to use:

SQL	 Copy
<pre>SELECT * FROM MyTable WHERE MyColumn2 = 'contoso'</pre>	


If a query returns many rows, they don't necessarily appear in any specific sequence. If you want to sort the data, you can add an **ORDER BY** clause. The data will be sorted by the specified column:

SQL	 Copy
<pre>SELECT * FROM MyTable ORDER BY MyColumn1</pre>	

You can also run **SELECT** statements that retrieve data from multiple tables using a **JOIN** clause. Joins indicate how the rows in one table are connected with rows in the other to determine what data to return. A join condition defines the way two tables are related in a query by:

- Specifying the column from each table to be used for the join. A typical join condition specifies a foreign key from one table and its associated primary key in the other table.
- Specifying a logical operator (for example, = or <>,) to be used in comparing values from the columns.

The following query shows an example that joins two tables, named *Inventory* and *CustomerOrder*. It retrieves all rows where the value in the *ID* column in the *Inventory* table matches the value in the *InventoryID* column in the *CustomerOrder* table.

SQL	 Copy
<pre>SELECT * FROM Inventory JOIN CustomerOrder ON Inventory.ID = CustomerOrder.InventoryID</pre>	

SQL provides aggregate functions. An aggregate function calculates a single result across a set of rows or an entire table. The example below finds the minimum value in the *MyColumn1* column across all rows in the *MyTable* table:

SQL	 Copy
-----	--

```
SELECT MIN(MyColumn1)
FROM MyTable
```

A number of other aggregate functions are available, including MAX (which returns the largest value in a column), AVG (which returns the average value, but only if the column contains numeric data), and SUM (which returns the sum of all the values in the column, but only if the column is numeric).

The next example shows how to update an existing row using SQL. It modifies the value of the second column but only for rows that have the value 3 in *MyColumn1*. All other rows are left unchanged:

SQL

 Copy

```
UPDATE MyTable
SET MyColumn2 = 'contoso'
WHERE MyColumn1 = 3
```

Warning

If you omit the **WHERE** clause, an **UPDATE** statement will modify **every** row in the table.

Use the **DELETE** statement to remove rows. You specify the table to delete from, and a **WHERE** clause that identifies the rows to be deleted:

SQL

 Copy

```
DELETE FROM MyTable
WHERE MyColumn2 = 'contoso'
```

Warning

If you omit the **WHERE** clause, a **DELETE** statement will remove **every** row from the table.

The **INSERT** statement takes a slightly different form. You specify a table and columns in an **INTO** clause, and a list of values to be stored in these columns. Standard SQL only supports inserting one row at a time, as shown in the following example. Some dialects allow you to specify multiple **VALUES** clauses to add several rows at a time:

SQL

 Copy


```
INSERT INTO MyTable(MyColumn1, MyColumn2, MyColumn3)
VALUES (99, 'contoso', 'hello')
```

Use DDL statements

You use DDL statements to create, modify, and remove tables and other objects in a database (table, stored procedures, views, and so on).


The most common DDL statements are:

Statement	Description
CREATE	Create a new object in the database, such as a table or a view.
ALTER	Modify the structure of an object. For instance, altering a table to add a new column.
DROP	Remove an object from the database.
RENAME	Rename an existing object.

 **Warning**

The **DROP** statement is very powerful. When you drop a table, all the rows in that table are lost. Unless you have a backup, you won't be able to retrieve this data.

The following example creates a new database table. The items between the parentheses specify the details of each column, including the name, the data type, whether the column must always contain a value (NOT NULL), and whether the data in the column is used to uniquely identify a row (PRIMARY KEY). Each table should have a primary key, although SQL doesn't enforce this rule.

 **Note**

Columns marked as **NOT NULL** are referred to as *mandatory* columns. If you omit the *NOT NULL* clause, you can create rows that don't contain a value in the column. An empty column in a row is said to have a *NULL* value.

SQL

Copy

```
CREATE TABLE MyTable
(
```

```
MyColumn1 INT NOT NULL PRIMARY KEY,  
MyColumn2 VARCHAR(50) NOT NULL,  
MyColumn3 VARCHAR(10) NULL  
);
```

The datatypes available for columns in a table will vary between database management systems. However, most database management systems support numeric types such as INT (an integer, or whole number), and string types such as VARCHAR (*VARCHAR* stands for variable length character data). For more information, see the documentation for your selected database management system.

Next unit: Query relational data in Azure SQL Database

[Continue >](#)