100 XP

Explore SQL

10 minutes

SQL stands for *Structured Query Language*, and 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.

You can use SQL statements such as **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **CREATE**, and **DROP** to accomplish almost everything that you need 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. For example, Microsoft SQL Server, and Azure database services that are based on the SQL Server database engine, use 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 services.
- *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.

> **Note**
>
> The SQL code examples in this module are based on the Transact-SQL dialect, unless otherwise indicated. The syntax for other dialects is generally similar, but may vary in some details.

SQL statement types

SQL statements are grouped into three main logical groups:

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

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

```
CREATE TABLE Product
(
    ID INT PRIMARY KEY,
    Name VARCHAR(20) NOT NULL,
    Price DECIMAL 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), DECIMAL (a decimal 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.

DCL statements

Database administrators generally use DCL statements to manage access to objects in a database by granting, denying, or revoking permissions to specific users or groups.

The three main DCL statements are:

Statement	Description
GRANT	Grant permission to perform specific actions
DENY	Deny permission to perform specific actions
REVOKE	Remove a previously granted permission

For example, the following **GRANT** statement permits a user named *user1* to read, insert, and modify data in the **Product** table.

SQL

```
GRANT SELECT, INSERT, UPDATE
ON Product
TO user1;
```

DML statements

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

The four main DML statements are:

Statement	Description
SELECT	Read rows from a table
INSERT	Insert new rows into a table
UPDATE	Modify data in existing rows
DELETE	Delete existing rows

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 columns (indicated by *) from the **Customer** table where the **City** column value is "Seattle":

```SQL
SELECT *
FROM Customer
WHERE City = 'Seattle';
```

To retrieve only a specific subset of columns from the table, you list them in the **SELECT** clause, like this:

```SQL
SELECT FirstName, LastName, Address, City
FROM Customer
WHERE City = 'Seattle';
```

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
SELECT FirstName, LastName, Address, City
FROM Customer
WHERE City = 'Seattle'
ORDER BY LastName;
```

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 typical join condition matches a foreign key from one table and its associated primary key in the other table.

The following query shows an example that joins **Customer** and **Order** tables. The query makes use of table *aliases* to abbreviate the table names when specifying which columns to retrieve in the **SELECT** clause and which columns to match in the **JOIN** clause.

```
SQL

SELECT o.OrderNo, o.OrderDate, c.Address, c.City
FROM Order AS o
JOIN Customer AS c
ON o.Customer = c.ID
```

The next example shows how to modify an existing row using SQL. It changes the value of the **Address** column in the **Customer** table for rows that have the value 1 in the **ID** column. All other rows are left unchanged:

```
SQL

UPDATE Customer
SET Address = '123 High St.'
WHERE ID = 1;
```

> ⚠ **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

DELETE FROM Product
WHERE ID = 162;
```

> ⚠ **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

```
INSERT INTO Product(ID, Name, Price)
VALUES (99, 'Drill', 4.99);
```

Note

This topic describes some basic SQL statements and syntax in order to help you understand how SQL is used to work with objects in a database. If you want to learn more about querying data with SQL, review the **Get Started Querying with Transact-SQL** learning path on Microsoft Learn.

Next unit: Describe database objects

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆