SQL

SQL is a standard language for storing, manipulating and retrieving data in databases.

This tutorial will teach you how to use SQL in: MySQL, SQL Server, MS Access, Oracle, Sybase, Informix, Postgres, and other database systems.

Why SQL?

Nowadays, SQL is widely used in data science and analytics. Following are the reasons which explain why it is widely used:

- The basic use of SQL for data professionals and SQL users is to insert, update, and delete the data from the relational database.
- SQL allows the data professionals and users to retrieve the data from the relational database management systems.
- It also helps them to describe the structured data.
- It allows SQL users to create, drop, and manipulate the database and its tables.
- It also helps in creating the view, stored procedure, and functions in the relational database.
- It allows you to define the data and modify that stored data in the relational database.
- It also allows SQL users to set the permissions or constraints on table columns, views, and stored procedures.

History of SQL

"A Relational Model of Data for Large Shared Data Banks" was a paper which was published by the great computer scientist "E.F. Codd" in 1970.

The IBM researchers Raymond Boyce and Donald Chamberlin originally developed the SEQUEL (Structured English Query Language) after learning from the paper given by E.F. Codd. They both developed the SQL at the San Jose Research laboratory of IBM Corporation in 1970

Advantages of SQL

By
Amos

SQL provides various advantages which make it more popular in the field of data science. It is a perfect query language which allows data professionals and users to communicate with the database. Following are the best advantages or benefits of Structured Query Language:

1. No programming needed

SQL does not require a large number of coding lines for managing the database systems. We can easily access and maintain the database by using simple SQL syntactical rules. These simple rules make the SQL user-friendly.

2. High-Speed Query Processing

A large amount of data is accessed quickly and efficiently from the database by using SQL queries. Insertion, deletion, and updation operations on data are also performed in less time.

3. Standardized Language

SQL follows the long-established standards of ISO and ANSI, which offer a uniform platform across the globe to all its users.

4. Portability

The structured query language can be easily used in desktop computers, laptops, tablets, and even smartphones. It can also be used with other applications according to the user's requirements.

5. Interactive language

We can easily learn and understand the SQL language. We can also use this language for communicating with the database because it is a simple query language. This language is also used for receiving the answers to complex queries in a few seconds.

6. More than one Data View

The SQL language also helps in making the multiple views of the database structure for the different database users.

Disadvantages of SQL

With the advantages of SQL, it also has some disadvantages, which are as follows:

1. Cost

The operation cost of some SQL versions is high. That's why some programmers cannot use the Structured Query Language.

2. Interface is Complex

Another big disadvantage is that the interface of Structured query language is difficult, which makes it difficult for SQL users to use and manage it.

3. Partial Database control

The business rules are hidden. So, the data professionals and users who are using this query language cannot have full database control

Some of The Most Important SQL Commands

- `SELECT` - extracts data from a database
- `UPDATE` - updates data in a database
- `DELETE` - deletes data from a database
- `INSERT INTO` - inserts new data into a database
- `CREATE DATABASE` - creates a new database
- `ALTER DATABASE` - modifies a database
- `CREATE TABLE` - creates a new table
- `ALTER TABLE` - modifies a table
- `DROP TABLE` - deletes a table
- `CREATE INDEX` - creates an index (search key)
- `DROP INDEX` - deletes an index

The SQL SELECT Statement

The `SELECT` statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

SELECT Syntax

```
SELECT column1, column2, ...
FROM table_name;
```

By
Amos

Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

```
SELECT * FROM table_name;
```

The SQL SELECT DISTINCT Statement

The `SELECT DISTINCT` statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT Syntax

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

The SQL WHERE Clause

The `WHERE` clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

WHERE Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

The SQL AND, OR and NOT Operators

The `WHERE` clause can be combined with `AND`, `OR`, and `NOT` operators.

The `AND` and `OR` operators are used to filter records based on more than one condition:

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

OR Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

NOT Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

Combining AND, OR and NOT

You can also combine the AND, OR and NOT operators.

The following SQL statement selects all fields from "Customers" where country is "Germany" AND city must be "Berlin" OR "München" (use parenthesis to form complex expressions):

```
SELECT * FROM Customers
WHERE condition AND (condition OR condition);
```

The SQL ORDER BY Keyword

By
Amos

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

ORDER BY Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

ORDER BY Several Columns Example

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName

```
SELECT * FROM Customers
ORDER BY column1, column2;
```

ORDER BY Several Columns Example 2

The following SQL statement selects all customers from the "Customers" table, sorted ascending by the "Country" and descending by the "CustomerName" column

```
SELECT * FROM Customers
ORDER BY column1 ASC, column1 DESC;
```

The SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

By
Amos

INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

What is a NULL Value?

A field with a NULL value is a field with no value.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

Note: A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation

How to Test for NULL Values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

IS NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

By
Amos

IS NOT NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

The SQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated

UPDATE Table

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person *and* a new city.

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

UPDATE Multiple Records

It is the WHERE clause that determines how many records will be updated.

The following SQL statement will update the ContactName to "Juan" for all records where country is "Mexico"

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

By
Amos

Update Warning!

Be careful when updating records. If you omit the `WHERE` clause, ALL records will be updated!

```
UPDATE Customers
SET ContactName='Juan';
```

The SQL DELETE Statement

The `DELETE` statement is used to delete existing records in a table.

DELETE Syntax

```
DELETE FROM table_name WHERE condition;
```

Note: Be careful when deleting records in a table! Notice the `WHERE` clause in the `DELETE` statement. The `WHERE` clause specifies which record(s) should be deleted. If you omit the `WHERE` clause, all records in the table will be deleted!

Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name;
```

The following SQL statement deletes all rows in the "Customers" table, without deleting the table:

```
DELETE FROM Customers;
```

The SQL SELECT TOP Clause

The `SELECT TOP` clause is used to specify the number of records to return.

The `SELECT TOP` clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

By
Amos

Note: Not all database systems support the `SELECT TOP` clause. MySQL supports the `LIMIT` clause to select a limited number of records, while Oracle uses `FETCH FIRST n ROWS ONLY` and `ROWNUM`.

SQL TOP, LIMIT and FETCH FIRST Examples

The following SQL statement selects the first three records from the "Customers" table (for SQL Server/MS Access):

The SQL MIN() and MAX() Functions

The `MIN()` function returns the smallest value of the selected column.

The `MAX()` function returns the largest value of the selected column.

MIN() Syntax

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

MAX() Syntax

```
SELECT MAX(column_name)
FROM table_name
--WHERE condition;
```

The SQL COUNT(), AVG() and SUM() Functions

The `COUNT()` function returns the number of rows that matches a specified criterion.

COUNT() Syntax

```
SELECT COUNT(column_name)
FROM table_name
--WHERE condition;
```

The `AVG()` function returns the average value of a numeric column.

By
Amos

AVG() Syntax

```
SELECT AVG(column_name)
FROM table_name
--WHERE condition;
```

The `SUM()` function returns the total sum of a numeric column.

SUM() Syntax

```
SELECT SUM(column_name)
FROM table_name
--WHERE condition;
```

The SQL LIKE Operator

The `LIKE` operator is used in a `WHERE` clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the `LIKE` operator:

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

Note: MS Access uses an asterisk (*) instead of the percent sign (%), and a question mark (?) instead of the underscore (_).

The percent sign and the underscore can also be used in combinations!

LIKE Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

LIKE Operator	Description

By
Amos

WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o

All the wildcards can also be used in combinations!

Here are some examples showing different `LIKE` operators with '%' and '_' wildcards:

LIKE Operator	**Description**

By
Amos

WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a__%'	Finds any values that starts with "a" and are at least length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o

Using the [charlist] Wildcard

The following SQL statement selects all customers with a City starting with "b", "s", or "p":

```
SELECT * FROM table name
WHERE columnName LIKE '[bsp]%';
```

The following SQL statement selects all customers with a City starting with "a", "b", or "c":

```
SELECT * FROM table name
WHERE Column Name LIKE '[a-c]%';
```

Using the [!charlist] Wildcard

By
Amos

The two following SQL statements select all customers with a City NOT starting with "b", "s", or "p":

```
SELECT * FROM table Name
WHERE coulomnName LIKE '[!bsp]%'
```

The SQL IN Operator

The `IN` operator allows you to specify multiple values in a `WHERE` clause.

The `IN` operator is a shorthand for multiple `OR` conditions.

IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

the following SQL statement selects all customers that are NOT located in "Germany", "France" or "UK":

```
SELECT * FROM table_name
WHERE column_name  NOT IN ('value1, value2, ...);
```

The following SQL statement selects all customers that are from the same countries as the suppliers:

```
SELECT * FROM table_name
WHERE column_name   IN (SELECT column_name  FROM value);
```

The SQL BETWEEN Operator

The `BETWEEN` operator selects values within a given range. The values can be numbers, text, or dates.

The `BETWEEN` operator is inclusive: begin and end values are included.

BETWEEN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

By
Amos

BETWEEN Example

The following SQL statement selects all products with a price between 10 and 20:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

NOT BETWEEN Example

To display the products outside the range of the previous example, use NOT BETWEEN:

```
SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

BETWEEN with IN Example

The following SQL statement selects all products with a price between 10 and 20. In addition; do not show products with a CategoryID of 1,2, or 3:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3);
```

BETWEEN Text Values Example

The following SQL statement selects all products with a ProductName between Carnarvon Tigers and Mozzarella di Giovanni:

```
SELECT * FROM Products
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

The following SQL statement selects all products with a ProductName between Carnarvon Tigers and Chef Anton's Cajun Seasoning:

```
SELECT * FROM Products
WHERE ProductName BETWEEN "Carnarvon Tigers" AND "Chef Anton's Cajun
```

By
Amos

```
Seasoning"
ORDER BY ProductName;
```

SQL Aliases

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of that query.

An alias is created with the AS keyword.

Alias Column Syntax

```
SELECT column_name AS alias_name
FROM table_name;
```

Alias Table Syntax

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

SQL JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Let's look at a selection from the "Orders" table:

OrderID	CustomerID	OrderDate
10308	2	1996-09-18

10309	37	1996-09-19
10310	77	1996-09-20

Then, look at a selection from the "Customers" table:

CustomerID	CustomerName	ContactName
1	Alfreds Futterkiste	Maria Anders
2	Ana Trujillo Emparedados y helados	Ana Trujillo
3	Antonio Moreno Taquería	Antonio Moreno

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

Then, we can create the following SQL statement (that contains an `INNER JOIN`), that selects records that have matching values in both tables:

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

and it will produce something like this:

OrderID	CustomerName

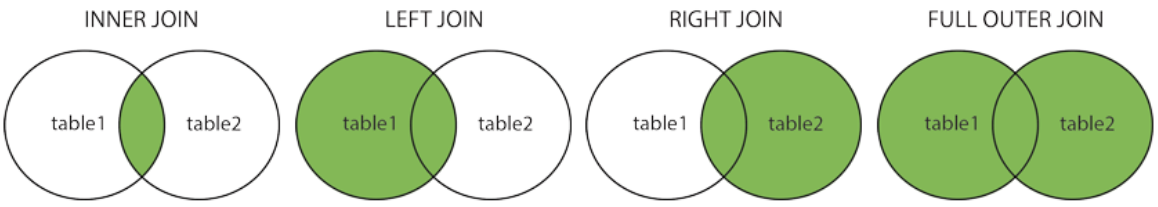
By
Amos

10308	Ana Trujillo Emparedados y helados
10365	Antonio Moreno Taquería
10383	Around the Horn
10355	Around the Horn
10278	Berglunds snabbköp

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- `(INNER) JOIN`: Returns records that have matching values in both tables
- `LEFT (OUTER) JOIN`: Returns all records from the left table, and the matched records from the right table
- `RIGHT (OUTER) JOIN`: Returns all records from the right table, and the matched records from the left table
- `FULL (OUTER) JOIN`: Returns all records when there is a match in either left or right table



SQL INNER JOIN Keyword

By
Amos

The INNER JOIN keyword selects records that have matching values in both tables.

INNER JOIN Syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

SQL INNER JOIN Example

The following SQL statement selects all orders with customer information

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Note: The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns. If there are records in the "Orders" table that do not have matches in "Customers", these orders will not be shown!

JOIN Three Tables

The following SQL statement selects all orders with customer and shipper information:

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

By
Amos

LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

Note: In some databases LEFT JOIN is called LEFT OUTER JOIN

SQL LEFT JOIN Example

The following SQL statement will select all customers, and any orders they might have:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Note: The `LEFT JOIN` keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

SQL RIGHT JOIN Keyword

The `RIGHT JOIN` keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Note: In some databases `RIGHT JOIN` is called `RIGHT OUTER JOIN`

SQL RIGHT JOIN Example

The following SQL statement will return all employees, and any orders they might have placed:

By
Amos

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

Note: The `RIGHT JOIN` keyword returns all records from the right table (Employees), even if there are no matches in the left table (Orders).

SQL FULL OUTER JOIN Keyword

The `FULL OUTER JOIN` keyword returns all records when there is a match in left (table1) or right (table2) table records.

Tip: `FULL OUTER JOIN` and `FULL JOIN` are the same.

FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

Note: `FULL OUTER JOIN` can potentially return very large result-sets!

SQL FULL OUTER JOIN Example

The following SQL statement selects all customers, and all orders:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Note: The `FULL OUTER JOIN` keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

SQL Self Join

A self join is a regular join, but the table is joined with itself.

By
Amos

Self Join Syntax

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

SQL Self Join Example

The following SQL statement matches customers that are from the same city:

```
SELECT A.CustomerName AS CustomerName1,
B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

The SQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

UNION Syntax

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

UNION ALL Syntax

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

By
Amos

Note: The column names in the result-set are usually equal to the column names in the first `SELECT` statement.

SQL UNION Example

The following SQL statement returns the cities (only distinct values) from both the "Customers" and the "Suppliers" table:

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

Note: If some customers or suppliers have the same city, each city will only be listed once, because `UNION` selects only distinct values. Use `UNION ALL` to also select duplicate values!

SQL UNION ALL Example

The following SQL statement returns the cities (duplicate values also) from both the "Customers" and the "Suppliers" table:

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

SQL UNION With WHERE

The following SQL statement returns the German cities (only distinct values) from both the "Customers" and the "Suppliers" table:

```
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

SQL UNION ALL With WHERE

The following SQL statement returns the German cities (duplicate values also) from both the "Customers" and the "Suppliers" table

```
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION ALL
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

Another UNION Example

The following SQL statement lists all customers and suppliers:

```
SELECT 'Customer' AS Type, ContactName, City, Country
FROM Customers
UNION
SELECT 'Supplier', ContactName, City, Country
FROM Suppliers;
```

The SQL GROUP BY Statement

The `GROUP BY` statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The `GROUP BY` statement is often used with aggregate functions (`COUNT()`, `MAX()`, `MIN()`, `SUM()`, `AVG()`) to group the result-set by one or more columns.

GROUP BY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

SQL GROUP BY Examples

The following SQL statement lists the number of customers in each country:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

The following SQL statement lists the number of customers in each country, sorted high to low:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

GROUP BY With JOIN Example

The following SQL statement lists the number of orders sent by each shipper

```
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM
Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

The SQL HAVING Clause

The `HAVING` clause was added to SQL because the `WHERE` keyword cannot be used with aggregate functions.

HAVING Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

SQL HAVING Examples

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

```
SELECT COUNT(CustomerID), Country
FROM Customers
```

By
Amos

```
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

The following SQL statement lists the number of customers in each country, sorted high to low (Only include countries with more than 5 customers):

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

More HAVING Examples

The following SQL statement lists the employees that have registered more than 10 orders:

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;
```

The following SQL statement lists if the employees "Davolio" or "Fuller" have registered more than 25 orders:

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
WHERE LastName = 'Davolio' OR LastName = 'Fuller'
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25;
```

The SQL EXISTS Operator

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

EXISTS Syntax

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

The SQL ANY and ALL Operators

The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.

The SQL ANY Operator

The ANY operator:

- returns a boolean value as a result
- returns TRUE if ANY of the subquery values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

ANY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

The SQL ALL Operator

The ALL operator:

- returns a boolean value as a result
- returns TRUE if ALL of the subquery values meet the condition
- is used with SELECT, WHERE and HAVING statements

By
Amos

ALL means that the condition will be true only if the operation is true for all values in the range.

ALL Syntax With SELECT

```
SELECT ALL column_name(s)
FROM table_name
WHERE condition;
```

The SQL SELECT INTO Statement

The SELECT INTO statement copies data from one table into a new table.

SELECT INTO Syntax

Copy all columns into a new table:

```
SELECT *
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

Copy only some columns into a new table:

```
SELECT column1, column2, column3, ...
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

SQL SELECT INTO Examples

The following SQL statement creates a backup copy of Customers

```
SELECT * INTO CustomersBackup2017
FROM Customers;
```

The following SQL statement uses the IN clause to copy the table into a new table in another database:

```
SELECT * INTO CustomersBackup2017 IN 'Backup.mdb'
FROM Customers;
```

By
Amos

The SQL INSERT INTO SELECT Statement

The `INSERT INTO SELECT` statement copies data from one table and inserts it into another table.

The `INSERT INTO SELECT` statement requires that the data types in source and target tables match.

Note: The existing records in the target table are unaffected.

INSERT INTO SELECT Syntax

Copy all columns from one table to another table:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

Copy only some columns from one table into another table:

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```

The SQL CASE Expression

The `CASE` expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the `ELSE` clause.

If there is no `ELSE` part and no conditions are true, it returns NULL.

CASE Syntax

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

By
Amos

SQL CASE Examples

The following SQL goes through conditions and returns a value when the first condition is met:

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN 'The quantity is greater than 30'
    WHEN Quantity = 30 THEN 'The quantity is 30'
    ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```

The following SQL will order the customers by City. However, if City is NULL, then order by Country:

```
SELECT CustomerName, City, Country
FROM Customers
ORDER BY
(CASE
    WHEN City IS NULL THEN Country
    ELSE City
END)
```

What is a Stored Procedure?

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

Stored Procedure Syntax

```
CREATE PROCEDURE procedure_name
AS
sql_statement
GO;
```

By
Amos

Stored Procedure Example

The following SQL statement creates a stored procedure named "SelectAllCustomers" that selects all records from the "Customers" table:

```
CREATE PROCEDURE SelectAllCustomers
AS
SELECT * FROM Customers
GO;
```

Execute the stored procedure above as follows:

```
EXEC SelectAllCustomers;
```

SQL Comments

Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.

Note: The examples in this chapter will not work in Firefox and Microsoft Edge!

Comments are not supported in Microsoft Access databases. Firefox and Microsoft Edge are using Microsoft Access database in our examples.

Single Line Comments

Single line comments start with `--`.

Any text between -- and the end of the line will be ignored (will not be executed).

The following example uses a single-line comment as an explanation:

```
--Select all:
SELECT * FROM Customers;
```

By
Amos

The SQL DROP DATABASE Statement

The DROP DATABASE statement is used to drop an existing SQL database.

Syntax

```
DROP DATABASE databasename;
```

The SQL BACKUP DATABASE Statement

The BACKUP DATABASE statement is used in SQL Server to create a full back up of an existing SQL database.

Syntax

```
BACKUP DATABASE databasename
TO DISK = 'filepath';
```

The SQL BACKUP WITH DIFFERENTIAL Statement

A differential back up only backs up the parts of the database that have changed since the last full database backup.

Syntax

```
BACKUP DATABASE databasename
TO DISK = 'filepath'
WITH DIFFERENTIAL;
```

The SQL CREATE TABLE Statement

The CREATE TABLE statement is used to create a new table in a database.

Syntax

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
```

By
Amos

```
    column3 datatype,
   ....
);
```

SQL CREATE TABLE Example

The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

The SQL DROP TABLE Statement

The DROP TABLE statement is used to drop an existing table in a database.

Syntax

```
DROP TABLE table_name;
```

SQL TRUNCATE TABLE

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

Syntax

```
TRUNCATE TABLE table_name;
```

SQL ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

By
Amos

ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD column_name datatype;
```

he following SQL adds an "Email" column to the "Customers" table:

```
ALTER TABLE Customers
ADD Email varchar(255);
```

ALTER TABLE - DROP COLUMN

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

The following SQL deletes the "Email" column from the "Customers" table:

```
ALTER TABLE Customers
DROP COLUMN Email;
```

SQL Create Constraints

Constraints can be specified when the table is created with the `CREATE TABLE` statement, or after the table is created with the `ALTER TABLE` statement.

Syntax

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

By
Amos

SQL NOT NULL Constraint

By default, a column can hold NULL values.

The `NOT NULL` constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

SQL NOT NULL on CREATE TABLE

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

SQL UNIQUE Constraint

The `UNIQUE` constraint ensures that all values in a column are different.

Both the `UNIQUE` and `PRIMARY KEY` constraints provide a guarantee for uniqueness for a column or set of columns.

A `PRIMARY KEY` constraint automatically has a `UNIQUE` constraint.

However, you can have many `UNIQUE` constraints per table, but only one `PRIMARY KEY` constraint per table

MySQL

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
```

By
Amos

```
    Age int,
    UNIQUE (ID)
);
```

SQL Server / Oracle / MS Access

```
CREATE TABLE Persons (
    ID int NOT NULL UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

SQL UNIQUE Constraint on ALTER TABLE

To create a `UNIQUE` constraint on the "ID" column when the table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access

```
ALTER TABLE Persons
ADD UNIQUE (ID);
```

SQL PRIMARY KEY Constraint

The `PRIMARY KEY` constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

SQL PRIMARY KEY on CREATE TABLE

The following SQL creates a `PRIMARY KEY` on the "ID" column when the "Persons" table is created:

MySQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
)
```

SQL PRIMARY KEY on ALTER TABLE

To create a `PRIMARY KEY` constraint on the "ID" column when the table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
```

DROP a PRIMARY KEY Constraint

To drop a `PRIMARY KEY` constraint, use the following SQL:

MySQL:

```
ALTER TABLE Persons
DROP PRIMARY KEY;
```

SQL FOREIGN KEY Constraint

The `FOREIGN KEY` constraint is used to prevent actions that would destroy links between tables.

By
Amos

A `FOREIGN KEY` is a field (or collection of fields) in one table, that refers to the `PRIMARY KEY` in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Look at the following two tables:

Persons Table

PersonID	LastName	FirstName
1	Hansen	Ola
2	Svendson	Tove
3	Pettersen	Kari

Orders Table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2

By
Amos

4	24562	1

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the `PRIMARY KEY` in the "Persons" table.

The "PersonID" column in the "Orders" table is a `FOREIGN KEY` in the "Orders" table.

The `FOREIGN KEY` constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table

SQL FOREIGN KEY on CREATE TABLE

The following SQL creates a `FOREIGN KEY` on the "PersonID" column when the "Orders" table is created:

MySQL:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

SQL FOREIGN KEY on ALTER TABLE

To create a `FOREIGN KEY` constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

By
Amos

DROP a FOREIGN KEY Constraint

To drop a `FOREIGN KEY` constraint, use the following SQL:

MySQL:

```
ALTER TABLE Orders
DROP FOREIGN KEY FK_PersonOrder;
```

SQL CHECK Constraint

The `CHECK` constraint is used to limit the value range that can be placed in a column.

If you define a `CHECK` constraint on a column it will allow only certain values for this column.

If you define a `CHECK` constraint on a table it can limit the values in certain columns based on values in other columns in the row.

SQL CHECK on CREATE TABLE

The following SQL creates a `CHECK` constraint on the "Age" column when the "Persons" table is created. The `CHECK` constraint ensures that the age of a person must be 18, or older:

MySQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

SQL DEFAULT Constraint

The `DEFAULT` constraint is used to set a default value for a column.

By
Amos

The default value will be added to all new records, if no other value is specified.

SQL DEFAULT on CREATE TABLE

The following SQL sets a `DEFAULT` value for the "City" column when the "Persons" table is created:

My SQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);
```

By
Amos