SQL is a standard language for accessing and manipulating databases.

**SQL stands for Structured Query Language**

* SQL lets you access and manipulate databases

## What Can SQL do?

* Execute queries against a database
* Retrieve data from a database
* Insert records in a database
* Update records in a database
* Delete records from a database
* Create new databases
* Create new tables in a database
* Create stored procedures in a database
* Create views in a database
* Set permissions on tables, procedures, and views

## RDBMS

RDBMS stands for Relational Database Management System.

RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows

SQL Keep in Mind That...

SQL keywords are NOT case sensitive: select is the same as SELECT. The standard way is to use uppercase.

## Semicolon after SQL Statements?

Some database systems require a semicolon at the end of each SQL statement.

Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

we always use semicolon at the end of each SQL statement.

**Database Tables**

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

## SQL Statements

Most of the actions you need to perform on a database are done with SQL statements.

### **Example**

SELECT \* FROM Customers;

**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

**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.

### Syntax

SELECT \* FROM table\_name;

SELECT column1, column2, ...  
FROM table\_name;

### **DISTINCT Syntax**

SELECT DISTINCT column1, column2, ...  
FROM table\_name;

SELECT DISTINCT Country FROM Customers;

SELECT COUNT(DISTINCT Country) FROM Customers;

## 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;

Note: The WHERE clause is not only used in SELECT statements, it is also used in UPDATE, DELETE, etc.!

### Example

SELECT \* FROM Customers  
WHERE Country='Mexico';

SELECT \* FROM Customers  
WHERE CustomerID=1;

## Operators in The WHERE Clause

The following operators can be used in the WHERE clause:

|  |  |
| --- | --- |
| Operator | Description |
| = | Equal |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal |
| <= | Less than or equal |
| <> | Not equal. Note: In some versions of SQL this operator may be written as != |
| BETWEEN | Between a certain range |
| LIKE | Search for a pattern |
| IN | To specify multiple possible values for a column |

## 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 ...;

### Example

SELECT \* FROM Customers  
WHERE Country='Germany' AND City='Berlin';

### OR Syntax

SELECT column1, column2, ...  
FROM table\_name  
WHERE condition1 OR condition2 OR condition3 ...;

### Example

SELECT \* FROM Customers  
WHERE City='Berlin' OR City='München';

### NOT Syntax

SELECT column1, column2, ...  
FROM table\_name  
WHERE NOT condition;

### Example

SELECT \* FROM Customers  
WHERE NOT Country='Germany';

## Combining AND, OR and NOT

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

### Example

SELECT \* FROM Customers  
WHERE Country='Germany' AND (City='Berlin' OR City='München');

### Example

SELECT \* FROM Customers  
WHERE NOT Country='Germany' AND NOT Country='USA';

## ORDER BY Keyword

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;

### Example

SELECT \* FROM Customers  
ORDER BY Country;

### Example

SELECT \* FROM Customers  
ORDER BY Country DESC;

### Example

SELECT \* FROM Customers  
ORDER BY Country, CustomerName;

### Example

SELECT \* FROM Customers  
ORDER BY Country ASC, CustomerName DESC;

## INSERT INTO Statement

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

### 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, ...);

### Example

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)  
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

## 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;

### 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.

### Example

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":

### Example

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

## Update Warning!

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

## 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!

### Example

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

Delete All

DELETE FROM table\_name;

# SQL TOP, LIMIT, FETCH FIRST or ROWNUM Clause

Example

SELECT TOP 3 \* FROM Customers;

SELECT \* FROM Customers  
LIMIT 3;

SELECT TOP 50 PERCENT \* FROM Customers;

SELECT \* FROM Customers  
WHERE Country='Germany'  
LIMIT 3;

## 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.

### Example

SELECT MIN(Price) AS SmallestPrice  
FROM Products;

SELECT MAX(Price) AS LargestPrice  
FROM Products;

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

### Example

SELECT COUNT(ProductID)  
FROM Products;

Note: NULL values are not counted.

SELECT AVG(Price)  
FROM Products;

SELECT SUM(Quantity)  
FROM OrderDetails;

# 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

### LIKE Syntax

SELECT column1, column2, ...  
FROM table\_name  
WHERE columnN LIKE pattern;

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

|  |  |
| --- | --- |
| LIKE Operator | Description |
| LIKE 'a%' | Finds any values that start with "a" |
| LIKE '%a' | Finds any values that end with "a" |
| WLIKE '%or%' | Finds any values that have "or" in any position |
| LIKE '\_r%' | Finds any values that have "r" in the second position |
| LIKE 'a\_%' | Finds any values that start with "a" and are at least 2 characters in length |
| LIKE 'a\_\_%' | Finds any values that start with "a" and are at least 3  characters in length |
| LIKE 'a%o' | Finds any values that start with "a" and ends with "o" |

### Wildcard Characters in SQL Server

|  |  |  |
| --- | --- | --- |
| Symbol | Description | Example |
| % | Represents zero or more characters | bl% finds bl, black, blue, and blob |
| \_ | Represents a single character | h\_t finds hot, hat, and hit |
| [] | Represents any single character within the  brackets | h[oa]t finds hot and hat, but not hit |
| ^ | Represents any character not in the brackets | h[^oa]t finds hit, but not hot and  hat |
| - | Represents a range of characters | c[a-b]t finds cat and cbt |

**All the wildcards can also be used in combinations!**

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

|  |  |
| --- | --- |
| LIKE Operator | Description |
| LIKE 'a%' | Finds any values that starts with "a" |
| LIKE '%a' | Finds any values that ends with "a" |
| LIKE '%or%' | Finds any values that have "or" in any position |
| LIKE '\_r%' | Finds any values that have "r" in the second position |
| LIKE 'a\_\_%' | Finds any values that starts with "a" and are at least 3  characters in length |
| LIKE 'a%o' | Finds any values that starts with "a" and ends with "o" |

## Using the % Wildcard

starting with "ber":

SELECT \* FROM Customers  
WHERE City LIKE 'ber%';

containing the pattern "es":

SELECT \* FROM Customers  
WHERE City LIKE '%es%';

## Using the \_ Wildcard

The following SQL statement selects all customers with a City starting with any character, followed by "ondon":

SELECT \* FROM Customers  
WHERE City LIKE '\_ondon';

tarting with "L", followed by any character, followed by "n", followed by any character, followed by "on"

SELECT \* FROM Customers  
WHERE City LIKE 'L\_n\_on';

## Using the [charlist] Wildcard

starting with "b", "s", or "p":

SELECT \* FROM Customers  
WHERE City LIKE '[bsp]%';

starting with "a", "b", or "c":

SELECT \* FROM Customers  
WHERE City LIKE '[a-c]%';

NOT starting with "b", "s", or "p":

SELECT \* FROM Customers  
WHERE City LIKE '[!bsp]%';

SELECT \* FROM Customers  
WHERE City NOT 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, ...);

or:

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name IN (SELECT STATEMENT);

all customers that are located in "Germany", "France" or "UK":

### Example

SELECT \* FROM Customers  
WHERE Country IN ('Germany', 'France', 'UK');

SELECT \* FROM Customers  
WHERE Country NOT IN ('Germany', 'France', 'UK');

selects all customers that are from the same countries as the suppliers:

### Example

SELECT \* FROM Customers  
WHERE Country IN (SELECT Country FROM Suppliers);

## 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;

all products with a price between 10 and 20:

### Example

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

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

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

**Date example**

SELECT \* FROM Orders  
WHERE OrderDate BETWEEN #07/01/1996# AND #07/31/1996#;

### Example

SELECT \* FROM Orders  
WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';

## 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;

SELECT CustomerID AS ID, CustomerName AS Customer  
FROM Customers;

SELECT CustomerName AS Customer, ContactName AS [Contact Person]  
FROM Customers;

SELECT CustomerName, Address + ', ' + PostalCode + ' ' + City + ', '+ Country AS Address  
FROM Customers;

SELECT CustomerName, CONCAT(Address,', ',PostalCode,', ',City,', ',Country) AS Address  
FROM Customers;

SELECT o.OrderID, o.OrderDate, c.CustomerName  
FROM Customers AS c, Orders AS o  
WHERE c.CustomerName='Around the Horn' AND c.CustomerID=o.CustomerID;

ELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName  
FROM Customers, Orders  
WHERE Customers.CustomerName='Around the Horn' AND Customers.CustomerID=Orders.CustomerID;

# SQL Joins

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

**Order table**

|  |  |  |
| --- | --- | --- |
| **OrderID** | CustomerID | OrderDate |
| 1001 | 2 | 2020-09-18 |
| 1002 | 3 | 2020-10-18 |
| 1003 | 77 | 2020-11-18 |

**Customers table**

|  |  |  |
| --- | --- | --- |
| CustomerID | CustomerName | Country |
| 1 | Tohid Ahmed | Germany |
| 2 | Akram Hossain | Mexico |
| 3 | Mustakin | Italy |

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:

## **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

 

 

## 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;

**Single line comments**

--Select all  
SELECT \* FROM Customers;

**multi line comments**

/\*Select all the columns  
of all the records  
in the Customers table:\*/  
SELECT \* FROM Customers;

# SQL Operators

## SQL Arithmetic Operators

|  |  |  |
| --- | --- | --- |
| Operator | Description |  |
| + | Add |  |
| - | Subtract |  |
| \* | Multiply |  |
| / | Divide |  |
| % | Modulo |  |

## SQL Bitwise Operators

|  |  |
| --- | --- |
| Operator | Description |
| & | Bitwise AND |
| | | Bitwise OR |
| ^ | Bitwise exclusive OR |

## SQL Comparison Operators

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| = | Equal to |  |
| > | Greater than |  |
| < | Less than |  |
| >= | Greater than or equal to |  |
| <= | Less than or equal to |  |
| <> | Not equal to |  |

## SQL Compound Operators

|  |  |
| --- | --- |
| Operator | Description |
| += | Add equals |
| -= | Subtract equals |
| \*= | Multiply equals |
| /= | Divide equals |
| %= | Modulo equals |
| &= | Bitwise AND equals |
| ^-= | Bitwise exclusive equals |
| |\*= | Bitwise OR equals |

## SQL Logical Operators

|  |  |  |
| --- | --- | --- |
| Operator | Description |  |
| ALL | TRUE if all of the subquery values meet the condition |  |
| AND | TRUE if all the conditions separated by AND is TRUE |  |
| ANY | TRUE if any of the subquery values meet the condition |  |
| BETWEEN | TRUE if the operand is within the range of comparisons |  |
| EXISTS | TRUE if the subquery returns one or more records |  |
| IN | TRUE if the operand is equal to one of a list of expressions |  |
| LIKE | TRUE if the operand matches a pattern |  |
| NOT | Displays a record if the condition(s) is NOT TRUE |  |
| OR | TRUE if any of the conditions separated by OR is TRUE |  |
| SOME | TRUE if any of the subquery values meet the condition |  |