SQL is a standard language for storing, manipulating and retrieving data in databases. We will learn how to use SQL in: MySQL, SQL Server, MS Access, Oracle, Sybase, Informix, Postgres, and other database systems.

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can 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.
- Look at the "Customers" table:

SQL Statements

- Most of the actions you need to perform on a database are done with SQL statements.
- The following SQL statement selects all the records in the "Customers" table:

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;

Example:

SELECT * FROM table_name;

SELECT CustomerName, City FROM Customers;

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.

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

SELECT DISTINCT Syntax

SELECT DISTINCT column1, column2, ...

FROM table_name;

SELECT COUNT(DISTINCT Country) FROM Customers;

SELECT Count(*) AS DistinctCountries

FROM (SELECT DISTINCT Country FROM Customers);

The SQL WHERE Clause

The WHERE clause is used to filter records.

The WHERE clause is used to extract only those records that fulfill a specified condition.;

SELECT column1, column2, ...

FROM table_name

WHERE condition;

Example: SELECT * FROM Customers

WHERE Country='Mexico';

Operators in The WHERE Clause

The following operators can be used in the WHERE clause:

Operator	Description
=	Equal
<>	Not equal. Note: In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive 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 is 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.

SELECT column1, column2, ...

FROM table_name

WHERE condition1 AND condition2 AND condition3 ...;

SELECT column1, column2, ...

FROM table_name

WHERE condition1 OR condition2 OR condition3 ...;

SELECT column1, column2, ...

FROM table_name

WHERE NOT condition;

The SQL 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.

SELECT column1, column2, ...

FROM table_name

ORDER BY column1, column2, ... ASC|DESC;

Example: SELECT * FROM Customers

ORDER BY Country;

ORDER BY DESC Example

The following SQL statement selects all customers from the "Customers" table, sorted DESCENDING by the "Country" column:

ORDER BY Syntax

SELECT column1, column2, ...

FROM table_name

ORDER BY column1, column2, ... ASC|DESC;

SELECT * FROM Customers

ORDER BY Country;

SELECT * FROM Customers

ORDER BY Country DESC;

SELECT * FROM Customers

ORDER BY Country, CustomerName;

SELECT * FROM Customers

ORDER BY Country ASC, CustomerName DESC;

INSERT INTO Syntax

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

The first way specifies both the column names and the values to be inserted:

INSERT INTO table_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);

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.

SELECT column_names

FROM table_name

WHERE column_**name IS NULL;**

……

SELECT column_names

FROM table_name

WHERE column_name IS NOT NULL;

The IS NULL Operator

The following SQL statement uses the IS NULL operator to list all persons that have no address:

SELECT LastName, FirstName, Address FROM Persons

WHERE Address IS NULL;

The IS NOT NULL Operator

The following SQL statement uses the IS NOT NULL operator to list all persons that do have an address:

SELECT LastName, FirstName, Address FROM Persons

WHERE Address IS NOT NULL;

The SQL UPDATE Statement

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

UPDATE table_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

UPDATE Customers

SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'

WHERE CustomerID = 1;

UPDATE Multiple Records

It is the WHERE clause that determines how many records that 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';

The SQL DELETE Statement

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

DELETE Syntax

DELETE FROM table_name

WHERE condition;

DELETE FROM Customers

WHERE CustomerName='Alfreds Futterkiste';

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 on performance.

SELECT column_name(s)

FROM table_name

WHERE condition

LIMIT number;

SQL TOP, LIMIT and ROWNUM Examples

The following SQL statement selects the first three records from the "Customers" table:

SELECT TOP 3 * FROM Customers;

SELECT * FROM Customers

LIMIT 3;

SELECT * FROM Customers

WHERE ROWNUM <= 3;

SELECT TOP 50 PERCENT * FROM Customers;

SELECT TOP 3 * FROM Customers

WHERE Country='Germany';

SELECT * FROM Customers

WHERE Country='Germany'

LIMIT 3;

SELECT * FROM Customers

WHERE Country='Germany' AND ROWNUM <= 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.

SELECT MIN(column_name)

FROM table_name

WHERE condition;

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 criteria.

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

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

SELECT COUNT(column_name)

FROM table_name

WHERE condition;

SELECT AVG(column_name)

FROM table_name

WHERE condition;

SELECT SUM(column_name)

FROM table_name

WHERE condition;

COUNT() Example

The following SQL statement finds the number of products:

SELECT COUNT(ProductID) FROM Products;

AVG() Example

The following SQL statement finds the average price of all products:

SELECT AVG(Price) FROM Products;

SUM() Example

The following SQL statement finds the sum of the "Quantity" fields in the "OrderDetails" table:

SELECT SUM(Quantity)

FROM OrderDetails;

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 used in conjunction with the LIKE operator:

% - The percent sign represents zero, one, or multiple characters

_ - The underscore represents a single character

SELECT column1, column2, ...

FROM table_name

WHERE columnN LIKE pattern;

SELECT column1, column2, ...

FROM table_name

WHERE columnN LIKE pattern;

SELECT * FROM Customers

WHERE CustomerName LIKE '%or%';

Using the % Wildcard

The following SQL statement selects all customers with a City starting with "ber":

SELECT * FROM Customers

WHERE City LIKE 'ber%';

SELECT * FROM Customers

WHERE City LIKE '%es%';

SELECT * FROM Customers

WHERE City LIKE '_erlin';

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.

SELECT column_name(s)

FROM table_name

WHERE column_name IN (value1, value2, ...);

SELECT column_name(s)

FROM table_name

WHERE column_name IN (SELECT STATEMENT);

SELECT * FROM Customers

WHERE Country IN ('Germany', 'France', 'UK'

SELECT * FROM Customers

WHERE Country NOT IN ('Germany', 'France', 'UK');

SELECT * FROM Customers

WHERE Country IN (SELECT Country FROM Suppliers);

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;

SELECT * FROM Products

WHERE Price BETWEEN 10 AND 20;

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 the query.

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:

SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate

FROM Orders

INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

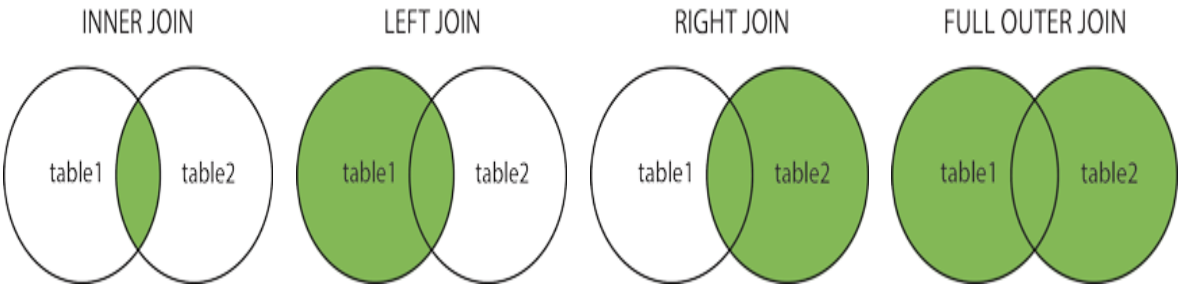
(INNER) JOIN: Returns records that have matching values in both tables

LEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the right table

RIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the left table

FULL (OUTER) JOIN: Return all records when there is a match in either left or right table

SQL INNER JOIN SQL LEFT JOIN SQL RIGHT JOIN SQL FULL OUTER JOIN



SQL INNER JOIN Keyword

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;

SELECT Orders.OrderID, Customers.CustomerName

FROM Orders

INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

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 RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

RIGHT JOIN Syntax

SELECT column_name(s)

FROM table1

RIGHT JOIN table2 ON table1.column_name = table2.column_name;

SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

SELECT column_name(s)

FROM table1

RIGHT JOIN table2 ON table1.column_name = table2.column_name;

SELECT Orders.OrderID, Employees.LastName, Employees.FirstName

FROM Orders

RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID

ORDER BY Orders.OrderID;

SQL FULL OUTER JOIN Keyword

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

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

FULL OUTER JOIN Syntax

SELECT column_name(s)

FROM table1

FULL OUTER JOIN table2 ON table1.column_name = table2.column_name;

SQL Self JOIN

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

Self JOIN Syntax

SELECT column_name(s)

FROM table1 T1, table1 T2

WHERE condition;

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.

Each SELECT statement within UNION must have the same number of columns

The columns must also have similar data types

The columns in each SELECT statement must also be in the same order

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;

SELECT City, Country FROM Customers

WHERE Country='Germany'

UNION

SELECT City, Country FROM Suppliers

WHERE Country='Germany'

ORDER BY City;

SELECT City, Country FROM Customers

WHERE Country='Germany'

UNION ALL

SELECT City, Country FROM Suppliers

WHERE Country='Germany'

ORDER BY City;

SELECT 'Customer' As Type, ContactName, City, Country

FROM Customers

UNION

SELECT 'Supplier', ContactName, City, Country

FROM Suppliers;

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;

SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country

ORDER BY COUNT(CustomerID) DESC;

The SQL HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword could not 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

GROUP BY Country

HAVING COUNT(CustomerID) > 5;

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);

SQL EXISTS Examples

The following SQL statement returns TRUE and lists the suppliers with a product price less than 20:

SELECT SupplierName

FROM Suppliers

WHERE EXISTS (SELECT ProductName FROM Products WHERE SupplierId = Suppliers.supplierId AND Price < 20);

The SQL SELECT INTO Statement

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

SELECT *

INTO newtable [IN externaldb]

FROM oldtable

WHERE condition;

SELECT column1, column2, column3, ...

INTO newtable [IN externaldb]

FROM oldtable

WHERE condition;

The SQL INSERT INTO SELECT Statement

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

INSERT INTO SELECT requires that data types in source and target tables match

The existing records in the target table are unaffected

INSERT INTO table2 (column1, column2, column3, ...)

SELECT column1, column2, column3, ...

FROM table1

WHERE condition;

The SQL CREATE DATABASE Statement

The CREATE DATABASE statement is used to create a new SQL database

CREATE DATABASE databasename;

The SQL DROP DATABASE Statement

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

DROP DATABASE databasename;
