Structured Query Language (SQL)

SQL is a standard language for accessing and manipulating databases.

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

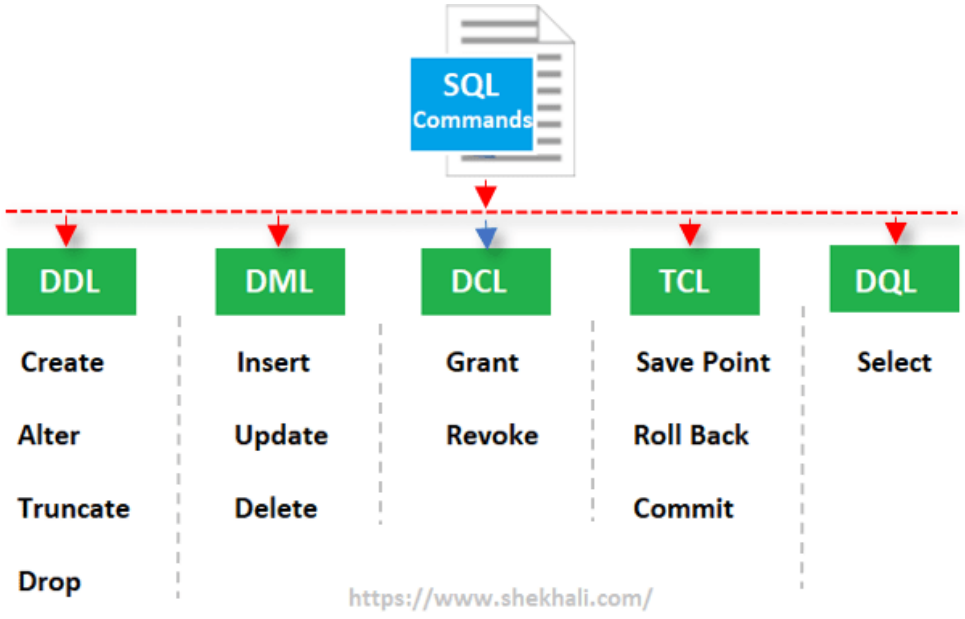
What Can SQL do?

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

SQL commands/statements are mainly categorized into four categories as:

1. DDL – Data Definition Language Commands
2. DQl – Data Query Language Commands
3. DML – Data Manipulation Language Commands
4. DCL – Data Control Language Commands
5. TCL- Transaction Control Language Commands



1. DDL (Data Definition Language):

DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.DDL is a set of SQL commands used to create, modify, and delete database structures but not data. These commands are normally not used by a general user, who should be accessing the database via an application.

List of DDL commands:

- **CREATE**: This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).
- **DROP**: This command is used to delete objects from the database.
- **ALTER:** This is used to alter the structure of the database.
- **TRUNCATE:** This is used to remove all records from a table, including all spaces allocated for the records are removed.
- **RENAME:** This is used to rename an object existing in the database.

2. DQL (Data Query Language):

DQL statements are used for performing queries on the data within schema objects. The purpose of the DQL Command is to get some schema relation based on the query passed to it. We can define DQL as follows it is a component of SQL statement that allows getting data from the database and imposing order upon it. It includes the SELECT statement. This command allows getting the data out of the database to perform operations with it. When a SELECT is fired against a table or tables the result is compiled into a further temporary table, which is displayed or perhaps received by the program i.e. a front-end.

List of DQL:

- **SELECT:** It is used to retrieve data from the database.

3. DML(Data Manipulation Language):

The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements. It is the component of the SQL statement that controls access to data and to the database. Basically, DCL statements are grouped with DML statements.

List of DML commands:

- **INSERT** : It is used to insert data into a table.
- **UPDATE:** It is used to update existing data within a table.
- **DELETE** : It is used to delete records from a database table.
- **LOCK:** Table control concurrency.

4. DCL (Data Control Language):

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

List of DCL commands:

- **GRANT:** This command gives users access privileges to the database.
- **REVOKE:** This command withdraws the user's access privileges given by using the GRANT command.

5. **TCL – Transaction Control Language**.

TCL commands deal with the transaction within the database.

List of TCL commands:

- **COMMIT:** Commits a Transaction.
- **ROLLBACK:** Rollbacks a transaction in case of any error occurs.
- **SAVEPOINT:**Sets a save point within a transaction.
- **SET TRANSACTION:** Specify characteristics for the transaction.

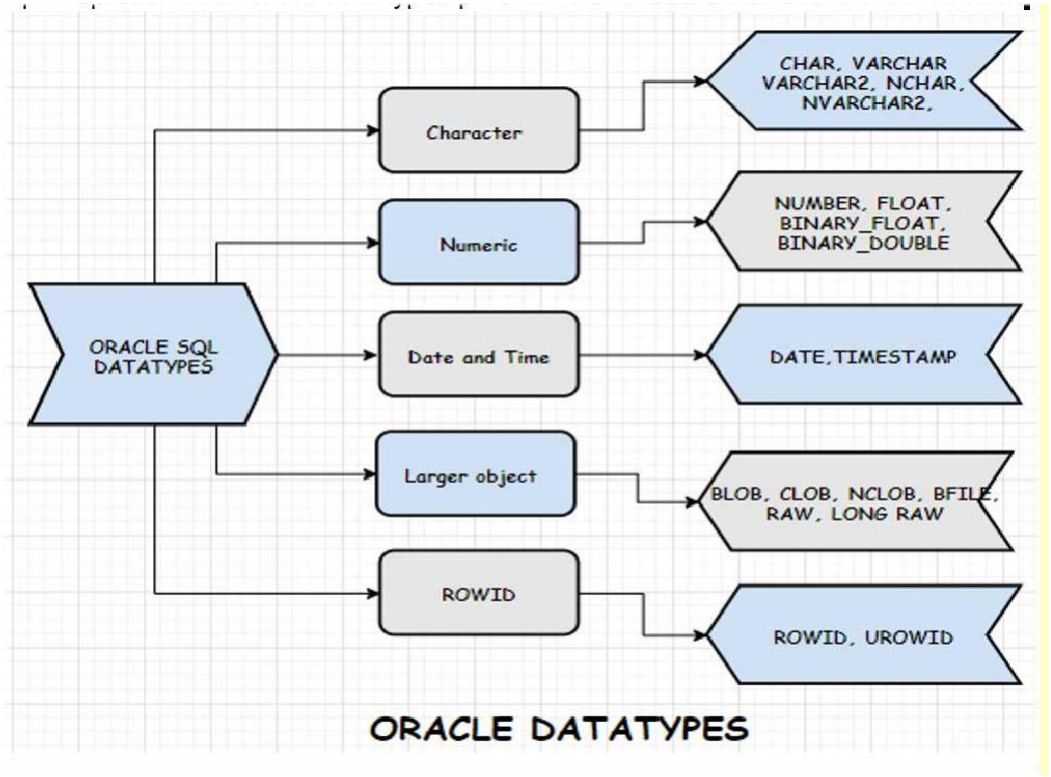
SQL Data Types

The data type of a column defines what value the column can hold: integer, character, money, date and time, binary, and so on.

Each column in a database table is required to have a name and a data type.

An SQL developer must decide what type of data that will be stored inside each column when creating a table.

Data types might have different names in different database. And even if the name is the same, the size and other details may be different.



Character Data types:

DATATYPE	DESCRIPTION
CHAR	It can be used to store fixed-length character Strings. The default value is 1 byte.
VARCHAR2(SIZE)	It is used to store variable-length character strings. You must specify **VARCHAR2(size)** maximum length between 1 to 4000 bytes.
VARCHAR(SIZE)	It is similar to VARCHAR2(size)

Numeric Data types:

DATATYPE	DESCRIPTION
NUMBER(p, s)	You can simply specify as NUMBER to store numeric values. Also besides, you can specify NUMBER(p, s) where p is precision(total number of digits) and s is scale (number of digits after the decimal point).
BINARY_FLOAT	It is 32 bit single precision floating point datatype
BINARY_DOUBLE	It is 64-bit double precision floating point datatype

DATE and TIME Data types:

DATATYPE	DESCRIPTION
DATE	It can be used to store the year, month, day, hours, minutes, and seconds. The standard format is 'DD-MM-YY'
TIMESTAMP	It is used to store the valid date along with time in 24-hour format 'HH:MM:SS'.

Larger Object (LOB) Data types:

Large unstructured data such as video clips, sound waves, texts, graphic images can be stored and manipulated in binary or character format.

DATATYPE	DESCRIPTION
BLOB	It can be used to specify unstructured binary data in the database which can store up to 128 terabytes of binary data.
CLOB	It can be used to store up to 128 terabytes of character data in the database.

ROWID Datatype:

DATATYPE	DESCRIPTION
ROWID	This helps us to store the row's ID that is the address of every row in the database.
UROWID	This is just a universal rowid datatype that supports all kinds of rowids.

What is SQL Operator?

An operator is a symbol specifying an action that is performed on one or more expressions.
An operator is a reserved character or word which is used in a SQL statement to query our database. We use a WHERE clause to query a database using operators.
Operators are needed to specify conditions in a SQL statement. The available operators act as a connector for various conditional statements.

Types of SQL Operators

We have various SQL operators, and they are as follows:

- SQL Arithmetic operators
- SQL Comparison operators
- SQL Logical operators
- SQL Compound Operators
- SQL Bitwise Operators
- SQL Unary Operator

SQL Arithmetic Operators

Sr.No	Arithmetic Operators	Description
1	+	Add
2	–	Subtract
3	*	Multiply
4	/	Division
5	%	Modulo

SQL Comparison Operators

Sr.No	Comparison Operators	Description
1	=	Equal
2	>	Greater than
3	<	Smaller than
4	>=	Greater than Equal to
5	<=	Smaller than Equal to
6	!<	Not less than
7	!=	Not Equal to
8	! >	Not greater than
9	< >	Value equal to

SQL Compound Operators

SQL compound operators are as shown below in the following table:

Sr.No	Compound Operators	Description
1	+=	Add equals
2	-=	Subtract equals
3	*=	Multiply equals
4	/=	Divide equals
5	%=	Modulo equals
6	&=	Bitwise AND equals
7	^-=	Bitwise Exclusive equals
8	|*=	Bitwise exclusive OR equals

SQL Logical Operators

SQL provides us with many logical operators to use while querying a database. All the logical operators are listed below in the table:

Sr.No	SQL Logical Operators	Description
1	ALL	Returns TRUE when all subqueries are satisfied
2	AND	Returns TRUE if all conditions in AND are satisfied
3	ANY	Returns TRUE if any subquery is satisfied
4	BETWEEN	Returns TRUE if the operand is in the given range
5	EXISTS	Returns true if one or more data record exists
6	IN	Returns TRUE if data matches the list of conditions
7	LIKE	Returns TRUE if our operand is similar to a pattern
8	NOT	Returns records if the condition is not satisfied
9	OR	Returns TRUE if any one of all subqueries is satisfied
10	SOME	Returns TRUE if any of our subqueries is satisfied.
11	IS NULL	Used to compare the NULL values.

SQL Unary Operators

Sr.No	Unary Operators in SQL	Description
1	(-)	Returns the negative value of the passed expression.
2	(+)	Returns the positive value of the passed expression.

SQL Expressions

An expression is a combination of one or more values, operators, and SQL functions that evaluate to a value.

SQL expression can be classified into following categories.

1. Boolean
2. Numeric
3. Date

SQL Boolean Expression

SQL Boolean Expression fetches data based on the condition that is mentioned as part of the SQL query. It should fetch just single value when the query is executed. Its syntax is given below.

```
SELECT column FROM table_name WHERE SINGLE_VALUE_MATCHING_EXPRESSION
```

SQL Numeric Expression

SQL Numeric Expression is used for performing mathematical operation in SQL query. Its syntax is as follows:

```
SELECT NUMERICAL_EXPRESSION as OPERATION_NAME FROM table_name
```

SQL Date Expression

SQL Date Expression results in datetime value.

The SQL CREATE DATABASE Statement

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

Syntax

```
CREATE DATABASE databasename;
```

CREATE DATABASE Example

The following SQL statement creates a database called "testDB":

Example:

```
CREATE DATABASE testDB;
```

Tip: Make sure you have admin privilege before creating any database. Once a database is created, you can check it in the list of databases with the following SQL command: SHOW DATABASES;

The SQL DROP DATABASE Statement

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

Syntax

Note: Be careful before dropping a database. Deleting a database will result in loss of complete information stored in the database!

```
DROP DATABASE databasename;
```

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

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

EXAMPLE:

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

The SQL 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, ...);
```

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, ...);
```

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;
```

The SQL DROP TABLE Statement

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

Syntax

```
DROP TABLE table_name;
```

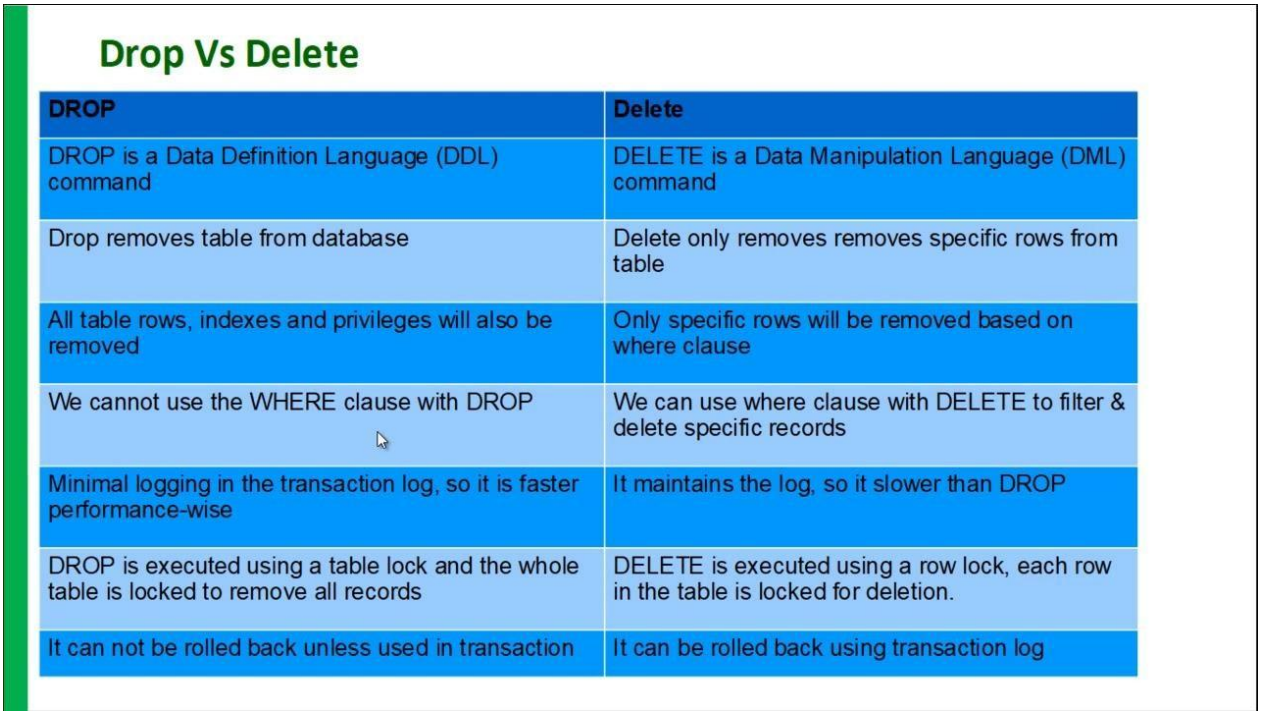
The SQL DELETE Statement

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!



Drop Vs Delete

DROP	Delete
DROP is a Data Definition Language (DDL) command	DELETE is a Data Manipulation Language (DML) command
Drop removes table from database	Delete only removes removes specific rows from table
All table rows, indexes and privileges will also be removed	Only specific rows will be removed based on where clause
We cannot use the WHERE clause with DROP	We can use where clause with DELETE to filter & delete specific records
Minimal logging in the transaction log, so it is faster performance-wise	It maintains the log, so it slower than DROP
DROP is executed using a table lock and the whole table is locked to remove all records	DELETE is executed using a row lock, each row in the table is locked for deletion.
It can not be rolled back unless used in transaction	It can be rolled back using transaction log

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.

ALTER TABLE - ADD Column

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

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

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

Example

```
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:

Example

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

ALTER TABLE - ALTER/MODIFY COLUMN

To change the data type of a column in a table, use the following syntax:

SQL Server / MS Access:

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype;
```

Oracle 10G and later:

```
ALTER TABLE table_name
MODIFY column_name datatype;
```

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;
```

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;
```

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

WHERE Clause Example

The following SQL statement selects all the customers from the country "Mexico", in the "Customers" table:

Example

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

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.

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 Server / MS Access Syntax:

```
SELECT TOP number|percent column_name(s)
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;
```

AND Example

The following SQL statement selects all fields from "Customers" where country is "Germany" AND city is "Berlin":

Example

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

OR Example

The following SQL statement selects all fields from "Customers" where city is "Berlin" OR "München":

Example

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

NOT Example

The following SQL statement selects all fields from "Customers" where country is NOT "Germany":

Example

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

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;
```

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

LIKE Operator	Description
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 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

SQL Wildcard Characters

A wildcard character is used to substitute one or more characters in a string.

Wildcard characters are used with the `LIKE` operator. The `LIKE` operator is used in a `WHERE` clause to search for a specified pattern in a column.

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 any single character within the specified range	c[a-b]t finds cat and cbt

The SQL ORDER BY Clause

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 Example

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

Example

```
SELECT * FROM Customers
ORDER BY Country;
```

The SQL GROUP BY Clause

The GROUP BY Statement in SQL is used to arrange identical data into groups with the help of some functions. i.e if a particular column has same values in different rows then it will arrange these rows in a group.

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:

Example

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

The SQL HAVING Clause

The **HAVING Clause** enables you to specify conditions that filter which group results appear in the results.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY 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);
```

What is an Aggregate Function in SQL?

An aggregate function in SQL performs a calculation on multiple values and returns a single value. SQL provides many aggregate functions that include avg, count, sum, min, max, etc. An aggregate function ignores NULL values when it performs the calculation, except for the count function.
An aggregate function in SQL returns one value after calculating multiple values of a column. We often use aggregate functions with the GROUP BY and HAVING clauses of the SELECT statement.

Various types of SQL aggregate functions are:

Count()

Sum()

Avg()

Min()

Max()

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.

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;
```

Note: NULL values are ignored.

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

MIN() Syntax

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

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

MAX() Syntax

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

SQL Date Functions

- Introduces some build-in functions to deal with dates

MySQL Date Functions

Function	Description
NOW()	Returns the current date and time
CURDATE()	Returns the current date
CURTIME()	Returns the current time
DATE()	Extracts the date part of a date or date/time expression
EXTRACT()	Returns a single part of a date/time
DATE_ADD()	Adds a specified time interval to a date
DATE_SUB()	Subtracts a specified time interval from a date
DATEDIFF()	Returns the number of days between two dates
DATE_FORMAT()	Displays date/time data in different formats

PostgreSQL: http://www.postgresql.org/docs/8.2/static/functions-datetime.html

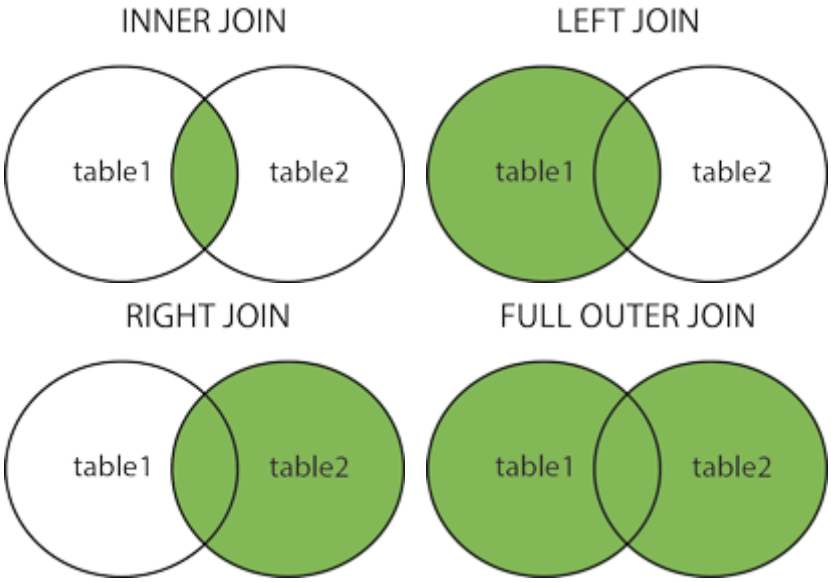
SQL JOIN

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

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

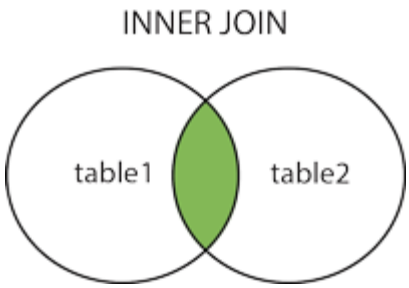


SQL INNER JOIN Keyword

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;
```



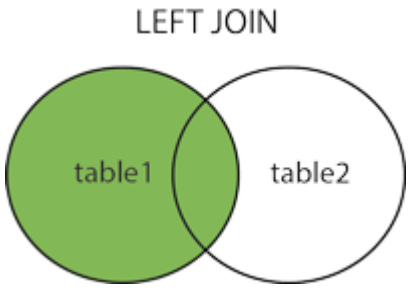
SQL LEFT JOIN Keyword

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.

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.



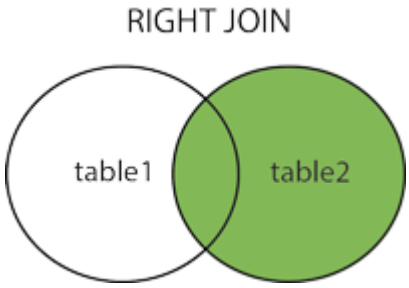
SQL RIGHT JOIN Keyword

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`.



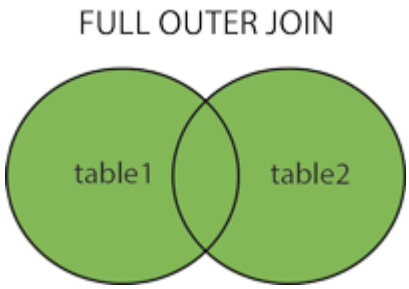
SQL FULL OUTER JOIN Keyword

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 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;
```

T1 and *T2* are different table aliases for the same tabl

SQL Views

SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the `CREATE VIEW` statement.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Note: A view always shows up-to-date data! The database engine recreates the view, every time a user queries it.

SQL CREATE VIEW Examples

The following SQL creates a view that shows all customers from Brazil:

Example

```
CREATE VIEW Brazil_Customers AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

SQL Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Prevents actions that would destroy links between tables
- CHECK - Ensures that the values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column if no value is specified
- CREATE INDEX - Used to create and retrieve data from the database very quickly

- 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,
    ....
);
```

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:

Example

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

SQL UNIQUE Constraint on CREATE TABLE

The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created:

SQL Server / Oracle / MS Access:

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

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:

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

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'
);
```

SQL PRIMARY KEY on CREATE TABLE

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 FOREIGN KEY Constraint

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

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

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

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

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