# Tools:

1. **Software Tools :**
   1. MySQL (LAMP) 2) MongoDB(Terminal) 3) SQL and PL/SQL(Oracle)

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

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

**Some of The Most Important SQL / MYSQL / MongoDB 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

### SQL CREATE DATABASE Statement

Syntax

CREATE DATABASE *databasename*;

Example

CREATE DATABASE demo;

### SQL DROP DATABASE Statement

Syntax

DROP DATABASE *databasename*;

Example

DROP DATABASE demo;

### SQL CREATE TABLE Statement

Syntax

CREATE TABLE *table\_name* ( *column1 datatype*, *column2 datatype*, *column3 datatype*,

....

);

Example

CREATE TABLE students ( rollno int,

stdname varchar(255), DOB date,

Address text, City varchar(255)

);

### SQL DROP TABLE Statement

Syntax

DROP TABLE *table\_name*;

Example

DROP TABLE students;

## 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*;

Example

TRUNCATE TABLE students;

### SQL ALTER TABLE Statement

**ALTER TABLE - ADD Column**

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

ALTER TABLE *table\_name*

ADD *column\_name datatype*;

**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*;

#### My SQL / Oracle (prior version 10G):

ALTER TABLE *table\_name*

MODIFY COLUMN *column\_name datatype*;

#### Oracle 10G and later:

ALTER TABLE *table\_name*

MODIFY *column\_name datatype*;

### SQL Constraints

Syntax

CREATE TABLE *table\_name* ( *column1 datatype constraint*, *column2 datatype constraint*, *column3 datatype constraint*,

....

);

The following constraints are commonly used in SQL:

* [**NOT NULL**](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value
* [**UNIQUE**](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different
* [**PRIMARY KEY**](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* [**FOREIGN KEY**](https://www.w3schools.com/sql/sql_foreignkey.asp) - Uniquely identifies a row/record in another table
* [**CHECK**](https://www.w3schools.com/sql/sql_check.asp) - Ensures that all values in a column satisfies a specific condition
* [**DEFAULT**](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column when no value is specified
* [**INDEX**](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly

### 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.

Example

CREATE TABLE stduent ( rollno int NOT NULL,

StdName varchar(255) NOT NULL, Address varchar(255) NOT NULL, mobileno int

);

### SQL UNIQUE Constraint

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

#### Oracle :

CREATE TABLE students (

rollno int NOT NULL UNIQUE, stdName varchar(255) NOT NULL, address varchar(255),

mobileno int

);

#### MySQL:

CREATE TABLE students ( rollno int NOT NULL,

stdName varchar(255) NOT NULL, address varchar(255),

mobileno int, UNIQUE (rollno)

);

.

#### MySQL / Oracle :

CREATE TABLE students ( rollno int NOT NULL,

stdName varchar(255) NOT NULL, address varchar(255),

mobileno int,

CONSTRAINT UC\_stduents UNIQUE (rollno, stdName)

);

# SQL UNIQUE Constraint on ALTER TABLE

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

#### MySQL / Oracle :

ALTER TABLE students ADD UNIQUE (rollno);

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

#### MySQL / Oracle :

ALTER TABLE students

ADD CONSTRAINT UC\_ students UNIQUE (rollno,stdName);

# DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL:

#### MySQL:

ALTER TABLE students DROP INDEX UC\_ students;

**Oracle:**

ALTER TABLE students

DROP CONSTRAINT UC\_students;

SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a database table. Primary keys must contain UNIQUE values, and cannot contain NULL values.

#### MySQL:

CREATE TABLE students ( rollno int NOT NULL,

stdName varchar(255) NOT NULL, address varchar(255),

mobileno int,

PRIMARY KEY (rollno)

);

#### Oracle :

CREATE TABLE students (

rollno int NOT NULL PRIMARY KEY, stdName varchar(255) NOT NULL, address varchar(255),

mobileno int

);

#### MySQL / Oracle :

CREATE TABLE students ( rollno int NOT NULL,

stdName varchar(255) NOT NULL, address varchar(255),

mobileno int,

CONSTRAINT PK\_ students PRIMARY KEY (rollno, stdName)

);

# SQL PRIMARY KEY on ALTER TABLE

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

#### MySQL / Oracle :

ALTER TABLE students ADD PRIMARY KEY (rollno);

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

#### MySQL / Oracle :

ALTER TABLE students

ADD CONSTRAINT PK\_ students PRIMARY KEY (rollno,stdName);

# DROP a PRIMARY KEY Constraint

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

#### MySQL:

ALTER TABLE students DROP PRIMARY KEY;

#### SQL Server / Oracle / MS Access:

ALTER TABLE students

DROP CONSTRAINT PK\_ students;

SQL FOREIGN KEY Constraint

A FOREIGN KEY is a key used to link two tables together.

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

The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

#### MySQL:

CREATE TABLE semester ( sem\_id int NOT NULL,

sem\_name varchar(255) NOT NULL, course\_id int,

PRIMARY KEY (sem\_id),

FOREIGN KEY (course\_id) REFERENCES course(course\_id)

);

#### Oracle :

CREATE TABLE semester (

sem\_id int NOT NULL PRIMARY KEY, sem\_name varchar(255) NOT NULL,

course\_id int FOREIGN KEY REFERENCES course (course\_id)

);

#### MySQL / Oracle :

CREATE TABLE semester ( sem\_id int NOT NULL,

sem\_name varchar(255) NOT NULL, course\_id int,

PRIMARY KEY (sem\_id),

CONSTRAINT FK\_Semcourse FOREIGN KEY (course\_id) REFERENCES course (course\_id)

);

# SQL FOREIGN KEY on ALTER TABLE

To create a FOREIGN KEY constraint on the " course\_id " column when the " semester " table is already created, use the following SQL:

#### MySQL / Oracle:

ALTER TABLE semester

ADD FOREIGN KEY (course\_id) REFERENCES course (course\_id);

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

#### MySQL / Oracle:

ALTER TABLE semester

ADD CONSTRAINT FK\_PersonOrder

FOREIGN KEY (course\_id) REFERENCES course(course\_id);

# DROP a FOREIGN KEY Constraint

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

#### MySQL:

ALTER TABLE semester

DROP FOREIGN KEY FK\_semcourse;

**Oracle :**

ALTER TABLE semester

DROP CONSTRAINT FK\_ semcourse;

SQL CHECK Constraint

#### SQL CHECK on CREATE TABLE

The following SQL creates a CHECK constraint on the "Age" column when the "Employees" table is created. The CHECK constraint ensures that you can not have any person below 18 years:

#### MySQL:

CREATE TABLE Employees ( ID int NOT NULL,

LastName varchar(255) NOT NULL, FirstName varchar(255),

Age int,

CHECK (Age>=18)

);

#### Oracle :

CREATE TABLE Employees( ID int NOT NULL,

LastName varchar(255) NOT NULL, FirstName varchar(255),

Age int CHECK (Age>=18)

);

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

#### MySQL / Oracle :

CREATE TABLE Employees( ID int NOT NULL,

LastName varchar(255) NOT NULL, FirstName varchar(255),

Age int,

City varchar(255),

CONSTRAINT CHK\_Person CHECK (Age>=18 AND City='Surat')

);

# SQL CHECK on ALTER TABLE

To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:

#### MySQL / Oracle :

ALTER TABLE Employees ADD CHECK (Age>=18);

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

#### MySQL / Oracle :

ALTER TABLE Employees

ADD CONSTRAINT CHK\_EmployeeAge CHECK (Age>=18 AND City='Surat’);

# DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL:

#### Oracle :

ALTER TABLE Employees

DROP CONSTRAINT CHK\_EmployeesAge;

#### MySQL:

ALTER TABLE Employees

DROP CHECK CHK\_ EmployeesAge;

SQL DEFAULT Constraint

**SQL DEFAULT on CREATE TABLE**

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

#### My SQL / Oracle :

CREATE TABLE Employees ( ID int NOT NULL,

LastName varchar(255) NOT NULL, FirstName varchar(255),

Age int,

City varchar(255) DEFAULT 'Surat'

);

The DEFAULT constraint can also be used to insert system values, by using functions like GETDATE():

CREATE TABLE Projects( proj\_id int NOT NULL,

type\_of\_project varchar(255) NOT NULL, start\_date date DEFAULT GETDATE()

);

# SQL DEFAULT on ALTER TABLE

To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

#### MySQL:

ALTER TABLE Employees

ALTER City SET DEFAULT 'Surat’;

#### Oracle:

ALTER TABLE Employees MODIFY City DEFAULT 'Surat’;

# DROP a DEFAULT Constraint

To drop a DEFAULT constraint, use the following SQL:

#### MySQL:

ALTER TABLE Employees ALTER City DROP DEFAULT;

#### Oracle :

ALTER TABLE Employees

ALTER COLUMN City DROP DEFAULT;

SQL SELECT Statement

SELECT Syntax

SELECT \* FROM *table\_name*;

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

Example

SELECT stdName, mobileno FROM students;

### 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*;

Example

SELECT DISTINCT rollno FROM students;

Example

SELECT COUNT(DISTINCT rollno) FROM students;

### 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.

WHERE Syntax

SELECT *column1*, *column2, ...*FROM *table\_name* WHERE *condition*; SELECT \* FROM Students WHERE stdname='XYZ';

SELECT \* FROM Students WHERE rollno=1;

# 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 |

### SQL AND, OR and NOT Operators

AND Syntax

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

Example

SELECT \* FROM Students WHERE stdname='patel' AND rollno=5;

OR Syntax

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

Example

SELECT \* FROM Students WHERE stdname ='patel' OR stdname ='shah';

NOT Syntax

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

Example

SELECT \* FROM Students WHERE NOT stdname ='patel';

# Combining AND, OR and NOT

Example

SELECT \* FROM Students WHERE stdname ='patel' AND (rollno=3 OR rollno =5);

Example

SELECT \* FROM Students WHERE NOT stdname ='patel' AND NOT rollno=3;

### 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.

ORDER BY Syntax

SELECT *column1*, *column2, ...* FROM *table\_name* ORDER BY *column1, column2, ...* ASC|DESC;

Example

SELECT \* FROM Students ORDER BY stdname;

Example

SELECT \* FROM Students ORDER BY stdname DESC;

Example

SELECT \* FROM Students ORDER BY rollno, stdname;

Example

SELECT \* FROM Students ORDER BY rollno ASC, stdname DESC;

### SQL INSERT INTO Statement

INSERT INTO Syntax

INSERT INTO *table\_name* (*column1*, *column2*, *column3*, ...) VALUES (*value1*, *value2*, *value3*, ...);

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. The INSERT INTO syntax would be as follows:

INSERT INTO *table\_name* VALUES (*value1*, *value2*, *value3*, ...);

Example

INSERT INTO department (dept\_no , dept\_name, location) VALUES (10, 'Account', 'NY');

### SQL NULL Values

#### 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.

# 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 IS NULL Operator

The following SQL statement uses the IS NULL operator to list all department that have no

dept\_name:

SELECT dept\_no , dept\_name, location FROM department WHERE dept\_name IS NULL;

# The IS NOT NULL Operator

The following SQL statement uses the IS NOT NULL operator to list all department that do have an dept\_name:

SELECT dept\_no , dept\_name, location FROM department WHERE dept\_name IS NOT NULL;

### SQL UPDATE Statement

#### 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*;

Example

UPDATE department SET , dept\_name = 'Clerk', location = 'USA' WHERE dept\_no = 10;

Example

UPDATE department SET dept\_name ='Account' WHERE location ='USA';

Example

UPDATE department SET location ='IND';

### SQL DELETE Statement

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

DELETE Syntax

DELETE FROM *table\_name* WHERE *condition*;

Example

DELETE FROM department WHERE location ='USA';

# 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*;

or:

DELETE \* FROM *table\_name*;

### SQL TOP, LIMIT or ROWNUM Clause

#### 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.

#### 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 ROWNUM.

**SQL Server / MS Access Syntax:**

SELECT TOP *number*|*percent column\_name(s)* FROM *table\_name* WHERE *condition*;

#### MySQL Syntax:

SELECT *column\_name(s)* FROM *table\_name* WHERE *condition* LIMIT *number*;

#### Oracle Syntax:

SELECT *column\_name(s)* FROM *table\_name* WHERE ROWNUM <= *number*;

Example

SELECT TOP 3 \* FROM department; SELECT \* FROM department LIMIT 3;

SELECT \* FROM department WHERE ROWNUM <= 3; SELECT TOP 50 PERCENT \* FROM department;

SELECT TOP 3 \* FROM department WHERE location ='USA'; SELECT \* FROM department WHERE location ='USA' LIMIT 3;

SELECT \* FROM department WHERE location ='USA' AND ROWNUM <= 3;

### 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*;

Example

SELECT MIN(salary) AS Smallestsal FROM EMPLOYEE;

MAX() Syntax

SELECT MAX(*column\_name*) FROM *table\_name* WHERE *condition*;

Example

SELECT MAX(salary) AS Largestsal FROM EMPLOYEE;

### 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.

COUNT() Syntax

SELECT COUNT(*column\_name*) FROM *table\_name* WHERE *condition*;

Example

SELECT COUNT(emp\_id) FROM EMPLOYEE;

AVG() Syntax

SELECT AVG(*column\_name*) FROM *table\_name* WHERE *condition*;

Example

SELECT AVG(salary) FROM EMPLOYEE;

SUM() Syntax

SELECT SUM(*column\_name*) FROM *table\_name* WHERE *condition*;

Example

SELECT SUM (salary) FROM EMPLOYEE;

### 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

**Note:** MS Access uses 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** |
| WHERE emp\_name LIKE 'a%' | Finds any values that start with "a" |
| WHERE emp\_name LIKE '%a' | Finds any values that end with "a" |
| WHERE emp\_name LIKE '%or%' | Finds any values that have "or" in any position |
| WHERE emp\_name LIKE '\_r%' | Finds any values that have "r" in the second position |
| WHERE emp\_name LIKE 'a\_%\_%' | Finds any values that start with "a" and are at least 3 characters in length |
| WHERE emp\_name LIKE 'a%o' | Finds any values that start with "a" and ends with "o" |

Example

SELECT \* FROM Employee WHERE emp\_name LIKE 'a%'; SELECT \* FROM Employee WHERE emp\_name LIKE '%a'; SELECT \* FROM Employee WHERE emp\_name LIKE '%or%'; SELECT \* FROM Employee WHERE emp\_name LIKE '\_r%'; SELECT \* FROM Employee WHERE emp\_name LIKE 'a\_%\_%'; SELECT \* FROM Employee WHERE emp\_name LIKE 'a%o'; SELECT \* FROM Employee WHERE emp\_name NOT LIKE 'a%';

### SQL Wildcards

A wildcard character is used to substitute any other character(s) in a string.

Wildcard characters are used with the [SQL LIKE](https://www.w3schools.com/sql/sql_like.asp) 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

**Note:** MS Access uses a question mark (?) instead of the underscore (\_). In MS Access and SQL Server you can also use:

* [*charlist*] - Defines sets and ranges of characters to match
* [^*charlist*] or [!*charlist*] - Defines sets and ranges of characters NOT to match The wildcards can also be used in combinations!

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

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

Example

SELECT \* FROM Employee WHERE emp\_name LIKE 'ber%'; SELECT \* FROM Employee WHERE emp\_name LIKE '%es%'; SELECT \* FROM Employee WHERE emp\_name LIKE '\_erlin'; SELECT \* FROM Employee WHERE emp\_name LIKE 'L\_n\_on';

#### Using the [charlist] Wildcard

SELECT \* FROM Employee WHERE emp\_name LIKE '[bsp]%'; SELECT \* FROM Employee WHERE emp\_name LIKE '[a-c]%';

#### Using the [!charlist] Wildcard

SELECT \* FROM Employee WHERE emp\_name LIKE '[!bsp]%'; SELECT \* FROM Employee WHERE emp\_name NOT LIKE '[bsp]%';

### 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*);

Example

SELECT \* FROM Department WHERE location NOT IN ('Germany', 'France', 'UK'); SELECT \* FROM Department WHERE location IN (SELECT location FROM Employee );

### 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;*

Example

SELECT \* FROM Employee WHERE salary BETWEEN 10000 AND 20000;

#### NOT BETWEEN Example

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

SELECT \* FROM Employee WHERE salary NOT BETWEEN 10000 AND 20000;

SELECT \* FROM Employee WHERE (salary BETWEEN 10000 AND 20000) AND NOT DepartNO IN (1,2,3);

SELECT \* FROM Employee WHERE emp\_name BETWEEN 'Dhaval Shah' AND 'Shashi Patel' ORDER BY

emp\_name;

#### NOT BETWEEN Text Values Example

The following SQL statement selects all products with a emp\_name NOT BETWEEN 'Dhaval Shah' and 'Shashi Patel':

Example

SELECT \* FROM Employee WHERE emp\_name NOT BETWEEN ' Dhaval Shah ' AND ' Shashi Patel ' ORDER BY emp\_name;

#### BETWEEN Dates Example

SELECT \* FROM Employee WHERE birth\_date BETWEEN #01/07/1996# AND #31/07/2006#; OR:

Example

SELECT \* FROM Employee WHERE birth\_date BETWEEN '1996-07-01' AND '2006-07-31';

### 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 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;*

Example

SELECT emp\_id AS ID, emp\_name AS Emp FROM Employee;

SELECT emp\_name AS emp, emp\_name AS [Employee Person] FROM Employee;

SELECT emp\_name, address + ', ' + designation + ' ' + salary + ', ' + experience AS Address FROM Employee;

SELECT emp, CONCAT(address', ', designation,', ', salary,', ', experience) AS Address FROM Employee;

#### Alias for Tables Example

SELECT d.dept\_no, d.dept\_name, e. emp\_name FROM Employee AS e, department AS d WHERE e.

emp\_name ="Dhaval Shah" AND d.dept\_no =e. dept\_no;

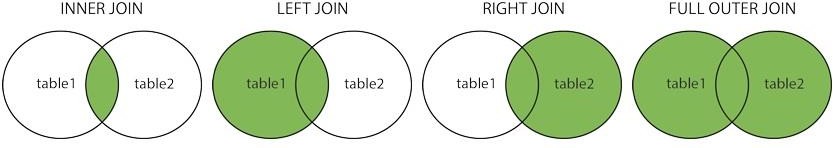
SELECT department. dept\_no, department. dept\_name, Employee.emp\_name FROM Employee, department WHERE Employee. emp\_name =" Dhaval Shah " AND Employee. dept\_no= department. dept\_no;

### SQL Joins

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

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



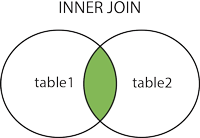
### SQL INNER JOIN

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*;



Example

SELECT department.dept\_no, employee.emp\_name FROM department INNER JOIN employee ON

department.dept\_no = employee.dept\_no;

#### JOIN Three Tables

Example

SELECT department.dept\_no, employee. emp\_name, project. type\_of\_project FROM ((department INNER JOIN employee ON department.dept\_no = dept\_no.dept\_no) INNER JOIN project ON employee. emp\_id = project. emp\_id);

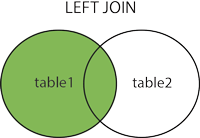
### SQL LEFT JOIN

The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL 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*;



LEFT JOIN Syntax

SELECT *column\_name(s)* FROM *table1* LEFT JOIN *table2* ON *table1.column\_name* =

*table2.column\_name*;

Example

SELECT employee.emp\_name, department.dept\_no FROM employee LEFT JOIN department ON employee.dept\_no = department. dept\_no ORDER BY employee. emp\_name;

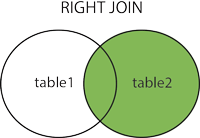
### SQL RIGHT JOIN

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*;



Example

SELECT department.dept\_no, Employee.emp\_name, Employee. gender FROM department RIGHT JOIN Employee ON department.dept\_no = Employee.dept\_no ORDER BY department. dept\_no;

### SQL FULL OUTER JOIN

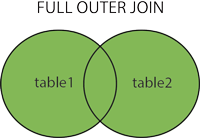
The FULL OUTER JOIN keyword return all records when there is a match in either left (table1) or right (table2) table records.

**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 FULL OUTER JOIN Example

SELECT Employee.emp\_name, department. dept\_no FROM Employee FULL OUTER JOIN department

ON Employee.dept\_no = emp\_name department.dept\_no ORDER BY Employee.emp\_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. emp\_name AS empnm1, B. emp\_name AS empnm 2, A.City FROM Employee A, Employee B WHERE A.emp\_id <> B. emp\_id AND A.City = B.City ORDER BY A.City;

### SQL GROUP BY Statement

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);*

Example

SELECT COUNT(emp\_id), emp\_name FROM Employee GROUP BY emp\_name;

Example

SELECT COUNT(emp\_id), emp\_name FROM Employee GROUP BY emp\_name ORDER BY COUNT(emp\_id) DESC;

### 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);*

Example

SELECT COUNT(emp\_id),emp\_name FROM Employee GROUP BY emp\_name HAVING COUNT(emp\_id) > 5;

SELECT COUNT(emp\_id), emp\_name FROM Employee GROUP BY emp\_name HAVING COUNT(emp\_id) > 5 ORDER BY COUNT(emp\_id) DESC;

SELECT Employee.emp\_name, COUNT(department.dept\_no) AS NumberOfdepart FROM (department INNER JOIN Employee ON department.dept\_no = Employee. dept\_no) GROUP BY Employee.emp\_name HAVING COUNT(department. dept\_no) > 10;

SELECT Employee.emp\_name, COUNT(department.dept\_no) AS NumberOfdepart FROM department INNER JOIN Employee ON department. dept\_no = Employee.dept\_no WHERE emp\_name = 'Patel' OR emp\_name = 'shah' GROUP BY Employee.emp\_name HAVING COUNT(department. dept\_no) > 25;

### 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.

SELECT *column\_name(s)* FROM *table\_name* WHERE EXISTS (SELECT *column\_name* FROM *table\_name*

WHERE *condition*);

Example

SELECT emp\_name FROM Employee WHERE EXISTS (SELECT dept\_name FROM department WHERE

emp\_id = Employee.emp\_id AND salary < 2000);

SELECT emp\_name FROM Employee WHERE EXISTS (SELECT dept\_name FROM department WHERE

emp\_id = Employee.emp\_id AND salary = 2000);

### SQL ANY and ALL Operators

The ANY and ALL operators are used with a WHERE or HAVING clause.

The ANY operator returns true if any of the subquery values meet the condition. The ALL operator returns true if all of the subquery values meet the condition.

ANY Syntax

SELECT *column\_name(s)* FROM *table\_name* WHERE *column\_name operator* ANY (SELECT *column\_name*

FROM *table\_name* WHERE *condition*);

SELECT ProductName FROM Products WHERE ProductID = ANY (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);

ALL Syntax

SELECT *column\_name(s)* FROM *table\_name* WHERE *column\_name operator* ALL (SELECT *column\_name*

FROM *table\_name* WHERE *condition*);

Example

SELECT ProductName FROM Products WHERE ProductID = ALL (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);

### 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

SELECT \* INTO CustomersBackup2017 FROM Customers;

SELECT CustomerName, ContactName INTO CustomersBackup2018 FROM Customers; SELECT \* INTO CustomersGermany FROM Customers WHERE Country = 'INDIA';

# 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;

Execute a Stored Procedure

EXEC *procedure\_name*;

Example

CREATE PROCEDURE SelectAllCustomers AS

SELECT \* FROM Customers GO;

Example

EXEC SelectAllCustomers;

### Oracle Functions

**Oracle String Functions**

|  |  |
| --- | --- |
| **Function** | **Description** |
| ASCII | Returns the number code that represents the specified character |
| ASCIISTR | Converts a string in any character set to an ASCII string using the database character set |
| CHR | Returns the character based on the number code |
| COMPOSE | Returns a Unicode string |
| CONCAT | Allows you to concatenate two strings together |
| Concat with || | Allows you to concatenate two or more strings together |
| CONVERT | Converts a string from one character set to another |
| DECOMPOSE | Accepts a string and returns a Unicode string |
| DUMP | Returns a varchar2 value that includes the datatype code, the length in bytes, and the internal representation of the expression |
| INITCAP | Sets the first character in each word to uppercase and the rest to lowercase |
| INSTR | Returns the location of a substring in a string |
| INSTR2 | Returns the location of a substring in a string, using UCS2 code points |
| INSTR4 | Returns the location of a substring in a string, using UCS4 code points |
| INSTRB | Returns the location of a substring in a string, using bytes instead of characters |

|  |  |
| --- | --- |
| INSTRC | Returns the location of a substring in a string, using Unicode complete characters |
| LENGTH | Returns the length of the specified string |
| LENGTH2 | Returns the length of the specified string, using UCS2 code points |
| LENGTH4 | Returns the length of the specified string, using UCS4 code points |
| LENGTHB | Returns the length of the specified string, using bytes instead of characters |
| LENGTHC | Returns the length of the specified string, using Unicode complete of characters |
| LOWER | Converts all letters in the specified string to lowercase |
| LPAD | Pads the left-side of a string with a specific set of characters |
| LTRIM | Removes all specified characters from the left-hand side of a string |
| NCHR | Returns the character based on the number code in the national character set |
| REGEXP\_INSTR | Returns the location of a regular expression pattern in a string |
| REGEXP\_REPLACE | Allows you to replace a sequence of characters in a string with another set of characters using regular expression pattern matching |
| REGEXP\_SUBSTR | Allows you to extract a substring from a string using regular expression pattern matching |
| REPLACE | Replaces a sequence of characters in a string with another set of characters |
| RPAD | Pads the right-side of a string with a specific set of characters |
| RTRIM | Removes all specified characters from the right-hand side of a string |
| SOUNDEX | Returns a phonetic representation (the way it sounds) of a string |

|  |  |
| --- | --- |
| SUBSTR | Allows you to extract a substring from a string |
| TRANSLATE | Replaces a sequence of characters in a string with another set of characters |
| TRIM | Removes all specified characters either from the beginning or the end of a string |
| UPPER | Converts all letters in the specified string to uppercase |
| VSIZE | Returns the number of bytes in the internal representation of an expression |

**Oracle Numeric Functions**

|  |  |
| --- | --- |
| **Function** | **Description** |
| ABS | Returns the absolute value of a number |
| ACOS | Returns the arc cosine of a number |
| ASIN | Returns the arc sine of a number |
| ATAN | Returns the arc tangent of a number |
| ATAN2 | Returns the arc tangent of n and m |
| AVG | Returns the average value of an expression |
| BITAND | Returns an integer representing an AND operation on the bits of expr1 and expr2 |
| CEIL | Returns the smallest integer value that is greater than or equal to a number |
| COS | Returns the cosine of a number |
| COSH | Returns the hyperbolic cosine of a number |

|  |  |
| --- | --- |
| COUNT | Returns the count of an expression |
| EXP | Returns e raised to the power of number |
| FLOOR | Returns the largest integer value that is equal to or less than a number |
| GREATEST | Returns the greatest value in a list of expressions |
| LEAST | Returns the smallest value in a list of expressions |
| LN | Returns the natural logarithm of a number |
| LOG | Returns the natural logarithm of a number to a specified base |
| MAX | Returns the maximum value of an expression |
| MEDIAN | Returns the median of an expression |
| MIN | Returns the minimum value of an expression |
| MOD | Returns the remainder of n divided by m |
| POWER | Returns m raised to the nth power |
| REGEXP\_COUNT | Counts the number of times that a pattern occurs in a string |
| REMAINDER | Returns the remainder of m divided by n |
| ROUND | Returns a number rounded to a certain number of decimal places |
| ROWNUM | Returns a number that represents the order that a row is |
| SIGN | Returns a value indicating the sign of a number |
| SIN | Returns the sine of a number |
| SQRT | Returns the square root of a number |

|  |  |
| --- | --- |
| SUM | Returns the summed value of an expression |
| TAN | Returns the tangent of a number |
| TANH | Returns the hyperbolic tangent of n |
| TRUNC | Returns a number truncated to a certain number of decimal places |

**Oracle Date Functions**

|  |  |
| --- | --- |
| **Function** | **Description** |
| ADD\_MONTHS | Returns a date with a specified number of months added |
| CURRENT\_DATE | Returns the current date in the time zone of the current SQL session as set by the ALTER SESSION command |
| CURRENT\_TIMESTAMP | Returns the current date and time in the time zone of the current SQL session as set by the ALTER SESSION command |
| DBTIMEZONE | returns the database time zone as a time zone offset or a time zone region name |
| EXTRACT | Extracts a value from a date or interval value |
| LAST\_DAY | Returns the last day of the month based on a date value |
| LOCALTIMESTAMP | Returns the current date and time in the time zone of the current SQL session as set by the ALTER SESSION command |
| MONTHS\_BETWEEN | Returns the number of months between date1 and date2 |
| NEW\_TIME | Converts a date from time zone1 to a date in time zone2 |
| NEXT\_DAY | Returns the first weekday that is greater than a date |

|  |  |
| --- | --- |
| ROUND | Returns a date rounded to a specific unit of measure |
| SESSIONTIMEZONE | Returns the current session's time zone as a time zone offset or a time zone region name |
| SYSDATE | Returns the current system date and time on your local database |
| SYSTIMESTAMP | Returns the current system date and time (including fractional seconds and time zone) on your local database |
| TRUNC | Returns a date truncated to a specific unit of measure |
| TZ\_OFFSET | Returns the time zone offset of a value |

19800

6000

10800

4200

18000

12600

6000

4300

22500

14400

6000

4700