

# SQL

A database is a collection of organised or arranged data that can be easily accessed, updated/ modified or controlled. Information within the database can be easily placed into rows and columns, or tables.

## → What is a Database Management System (DBMS)?

A database management system (or DBMS) is essentially nothing more than a computerized data-keeping system.

Database Management System (DBMS) is software used to identify, manage, and create a database that provides administered access to the data.

## → What is a Relational Database Management System (RDBMS)?

The software used to store, manage, query, and retrieve data stored in a relational database is called a relational database management system (RDBMS).

The RDBMS provides an interface between users and applications and the database, as well as administrative functions for managing data storage, access, and performance.

Relational Database Management System (RDBMS) is a more advanced version of a DBMS system that allows access to data in a more efficient way. It is used to store or manage only the data that are in the form of tables.

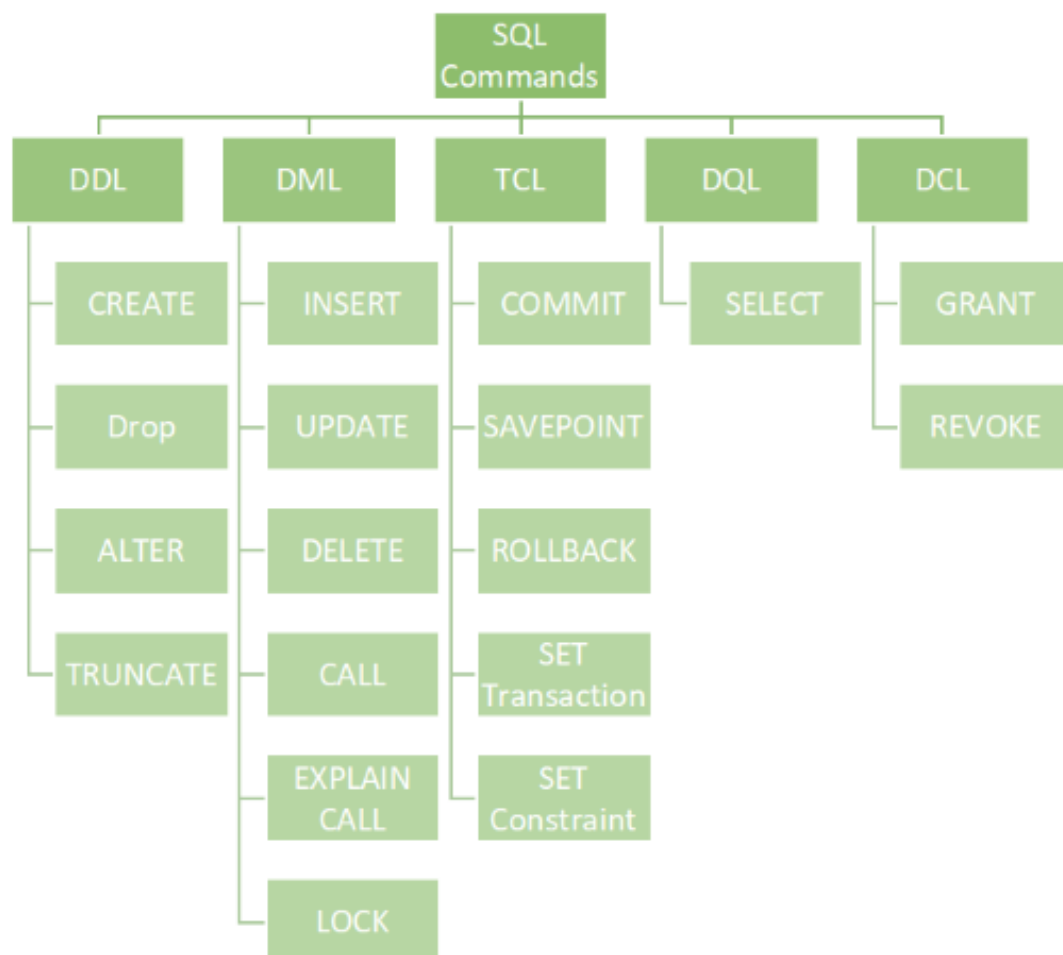
RDBMS	DBMS
Data stored is in table format	Data stored is in the file format
Multiple data elements are accessible together	Individual access of data elements
Data in the form of a table are linked together	No connection between data
Support distributed database	No support for distributed database
Data is stored in a large amount	Data stored is a small quantity
RDBMS supports multiple users	DBMS supports a single user
The software and hardware requirements are higher	The software and hardware requirements are low
Example: Oracle, SQL Server.	Example: XML, Microsoft Access.

## → What is a SQL?

- SQL stands for Structured Query Language
- SQL is a standard language for storing, manipulating and retrieving data in databases. SQL allows you to access and manipulate the databases. To use SQL in: MySQL, SQL Server, MS Access, Oracle, Sybase, Informix, Postgres, and other database systems.

These SQL commands are mainly categorized into five categories:

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



## Data types

<b>CHAR(size)</b>	A FIXED length string (can contain letters, numbers, and special characters). can be from 0 to 255. Default is 1
<b>VARCHAR(size)</b>	A VARIABLE length string (can contain letters, numbers, and special characters). can be from 0 to 65535
<b>INT(size)</b>	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295.
<b>INTEGER(size)</b>	Equal to INT(size)
<b>FLOAT(p)</b>	A floating point number. MySQL uses the <i>p</i> value to determine whether to use FLOAT or DOUBLE for the resulting data type. If <i>p</i> is from 0 to 24, the data type becomes FLOAT(). If <i>p</i> is from 25 to 53, the data type becomes DOUBLE()
<b>DATE</b>	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
<b>DATETIME(fsp)</b>	A date and time combination. Format: YYYY-MM-DD hh:mm:ss.
<b>TIME(fsp)</b>	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
<b>YEAR</b>	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

## SQL CREATE TABLE Statement

```
CREATE TABLE Employee (Emp_No int, Emp_Name varchar(50), Salary int);
```

→ Create Table Using another Table

```
CREATE TABLE TestEmployee AS SELECT Emp_No, Emp_Name FROM Employee;
```

### → The SQL DROP TABLE Statement

```
DROP TABLE TestEmployee;
```

### → SQL TRUNCATE TABLE

```
TRUNCATE TABLE TestEmployee;
```

To delete the data inside a table, but not the table itself.

### → The SQL INSERT INTO Statement

1. **INSERT INTO** Employee **VALUES** (101,'Chinmayee', 50000);

2. **INSERT INTO** Employee (Emp\_No, Emp\_Name) **VALUES** (101,'Chinmayee');

The rest field will contain NULL value automatically. E.g here about Salary.

### → The SQL SELECT Statement

```
SELECT * FROM Employee;
```

```
SELECT Emp_No, Emp_Name from Employee;
```

### → The SQL SELECT DISTINCT Statement

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

```
SELECT DISTINCT * FROM Employee;
```

```
SELECT DISTINCT Emp_No, Emp_Name FROM Employee;
```

## → The SQL WHERE Clause

1. **SELECT \* FROM Employee WHERE Emp\_No=101;**
2. **SELECT \* FROM Employee WHERE Emp\_Name='Chinmayee';**

### 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. <b>Note:</b> 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.

1. **SELECT \* FROM Employee WHERE Salary>25000 AND Emp\_No>5;**
2. **SELECT \* FROM Employee WHERE Salary>25000 OR Emp\_No>5;**
3. **SELECT \* FROM Employee WHERE NOT Salary=20000;**

### → Combining AND, OR and NOT

1. **SELECT \* FROM Employee WHERE Emp\_No>=5 AND (Salary<20000 OR Salary>50000);**
2. **SELECT \* FROM Employee WHERE NOT Salary<15000 AND NOT Salary>55000;**

### → 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 * FROM Employee ORDER BY Emp_Name;  
  
SELECT * FROM Employee ORDER BY Emp_Name DESC;
```

### → 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.
- 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!
- We can use the IS NULL and IS NOT NULL operators.

```
SELECT * FROM Employee WHERE Salary IS NULL;  
  
SELECT * FROM Employee WHERE Salary IS NOT NULL;
```

### → The SQL UPDATE Statement

```
UPDATE Employee SET Salary=25000 WHERE Emp_Name='Dhara';
```

```
UPDATE Employee SET Salary=25000;
```

### → The SQL DELETE Statement

```
DELETE FROM Employee WHERE Emp_No=104;
```

```
DELETE FROM Employee;
```

(Delete all records from the table)

### → 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(Salary) AS Minimum_Salary FROM Employee;
```

**Output :**

Minimum_Salary
15000

```
SELECT MAX(Salary) AS Maximum_Salary FROM Employee;
```

**Output :**

Maximum_Salary
55000

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

- The COUNT() function returns the number of rows that matches a specified criterion.
  - NULL values are not counted.

```
SELECT COUNT(Emp_No) Total_Employees FROM Employee;
```

**Output :**

Total_Employees
9

**SELECT COUNT(Emp\_No) Total\_Employees FROM Employee WHERE Emp\_No>105;**

**Output :**

Total_Employees
6

- The AVG() function returns the average value of a numeric column.
- NULL values are ignored.

**SELECT AVG(Salary) Average\_Salary\_IS FROM Employee;**

**Output :**

Average_Salary_IS
314444.4444

**SELECT AVG(Salary) Average\_Salary\_IS FROM Employee WHERE Salary>25000;**

**Output :**

Average_Salary_IS
38600.0000

- The SUM() function returns the total sum of a numeric column.
- NULL values are ignored.

**SELECT SUM(Emp\_No) Sum\_OF\_ID FROM Employee;**

**Output :**

Sum_OF_ID
959

**SELECT SUM(Emp\_No) Sum\_OF\_ID FROM Employee WHERE Emp\_No>105;**

**Output :**

Sum_OF_ID
646



## → 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
- ✓ The percent sign and the underscore can also be used in combinations!
- ✓ You can also combine any number of conditions using AND or OR operators.

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"

Name start with 'A' or 'a'. Case doesn't matter.

**SELECT \* FROM Employee WHERE Emp\_Name LIKE 'A%';**

**Output:**

Emp_No	Emp_Name	Salary
109	Ameet	55000

Name start with 'A' or 'a'. Case doesn't matter.

**SELECT \* FROM Employee WHERE Emp\_Name LIKE '%a';**

**Output :**

Emp_No	Emp_Name	Salary
105	Bhavika	25000
106	Dhara	28000
107	Neha	15000

Name the Employees that have 'et' in any position.

**SELECT \* FROM Employee WHERE Emp\_Name LIKE '%et%';**

**Output :**

Emp_No	Emp_Name	Salary
108	Saket	35000
109	Ameet	55000

Name the Employees that have 'h' in the second position.

**SELECT \* FROM Employee WHERE Emp\_Name LIKE '\_h%';**

**Output :**

Emp_No	Emp_Name	Salary
105	Bhavika	25000
107	Dhara	15000
109	Chinmayee	45000

Name the Employees that starts with "a" and are at least 3 characters in length.

**SELECT \* FROM Employee WHERE Emp\_Name LIKE 'a\_\_%';**

**Output :**

Emp_No	Emp_Name	Salary
109	Ameet	55000

All employees with a Name that starts with "c" and ends with "e".

**SELECT \* FROM Employee WHERE Emp\_Name LIKE 'c%e';**

**Output :**

Emp_No	Emp_Name	Salary
109	Chinmayee	45000

Selects all employees with a Emp\_Name that does NOT start with "a":

**SELECT \* FROM Employee WHERE Emp\_Name NOT LIKE 'a%';**

## → 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 * FROM Employee WHERE Emp_Name IN('Chinmayee', 'Ameet','Rahul');
```

Output :

Emp_No	Emp_Name	Salary
107	Rahul	25000
109	Ameet	55000
109	Chinmayee	45000

```
SELECT * FROM Employee WHERE Emp_Name NOT IN('Chinmayee', 'Ameet','Rahul');
```

Output :

Emp_No	Emp_Name	Salary
103	Monali	30000
105	Bhavika	25000
105	Bhavika	25000
106	Neha	28000
107	Dhara	15000
108	Saket	35000
(NULL)	(NULL)	(NULL)

## → 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.

```
SELECT * FROM Employee WHERE Salary BETWEEN 30000 AND 50000;
```

<input type="checkbox"/>	Emp_No	Emp_Name	Salary
<input checked="" type="checkbox"/>	103	Monali	30000
<input type="checkbox"/>	108	Saket	35000
<input type="checkbox"/>	109	Chinmayee	45000
*	(NULL)	(NULL)	(NULL)

```
SELECT * FROM Employee WHERE Salary NOT BETWEEN 30000 AND 50000;
```

<input type="checkbox"/>	Emp_No	Emp_Name	Salary
<input checked="" type="checkbox"/>	105	Bhavika	25000
<input type="checkbox"/>	105	Bhavika	25000
<input type="checkbox"/>	106	Neha	28000
<input type="checkbox"/>	107	Dhara	15000
<input type="checkbox"/>	107	Rahul	25000
<input type="checkbox"/>	109	Ameet	55000
*	(NULL)	(NULL)	(NULL)

**SELECT \* FROM Employee WHERE Salary NOT BETWEEN 30000 AND 50000 AND Emp\_No NOT IN (107);**

<input type="checkbox"/>	Emp_No	Emp_Name	Salary
<input checked="" type="checkbox"/>	105	Bhavika	25000
<input type="checkbox"/>	105	Bhavika	25000
<input type="checkbox"/>	106	Neha	28000
<input type="checkbox"/>	109	Ameet	55000
*	(NULL)	(NULL)	(NULL)

**SELECT \* FROM Employee WHERE Emp\_Name IN('Rahul','Ameet','Chinmayee') ORDER BY Emp\_Name;**

<input type="checkbox"/>	Emp_No	Emp_Name	Salary
<input checked="" type="checkbox"/>	109	Ameet	55000
<input type="checkbox"/>	109	Chinmayee	45000
<input type="checkbox"/>	107	Rahul	25000
*	(NULL)	(NULL)	(NULL)

### → BETWEEN Dates Example

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

OR

**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.

```
SELECT C.Cust_ID, C.Customer_Name, E.Emp_No, E.Emp_Name FROM Customer C, Employee E
WHERE C.Emp_No=E.Emp_No;
```

<input type="checkbox"/>	Cust_ID	Customer_Name	Emp_No	Emp_Name
<input type="checkbox"/>	1	Mr. Dhaval Patel	105	Bhavika
<input type="checkbox"/>	1	Mr. Dhaval Patel	105	Bhavika
<input type="checkbox"/>	2	Mr. Raghav Patel	106	Neha
<input type="checkbox"/>	3	Mr. Vinit Pandya	107	Dhara
<input type="checkbox"/>	3	Mr. Vinit Pandya	107	Rahul
*	(NULL)	(NULL)	(NULL)	(NULL)

## → 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

## → 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.

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

### → SQL UNIQUE Constraint

- The UNIQUE constraint ensures that all values in a column are different.
- Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

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

### → SQL PRIMARY KEY Constraint

- The PRIMARY KEY constraint uniquely identifies each record in a table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

```
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 PRIMARY KEY in another table.
- The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

**Keys**

**table1 - Student**

id	name	cityId	city
101	karan	1	Pune
102	arjun	2	Mumbai
103	ram	1	Pune
104	shyam	3	Delhi

**table2 - City**

id	city_name
1	Pune
2	Mumbai
3	Delhi

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Orders Table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

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

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

### → SQL CHECK Constraint

- The **CHECK** constraint is used to limit the value range that can be placed in a column.
- If you define a **CHECK** constraint on a column it will allow only certain values for this column.
- If you define a **CHECK** constraint on a table it can limit the values in certain columns based on values in other columns in the row.

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

### → SQL DEFAULT Constraint

- The **DEFAULT** constraint is used to set a default value for a column.
- The default value will be added to all new records, if no other value is specified.



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

```
CREATE TABLE Orders (  
    ID int NOT NULL,  
    OrderNumber int NOT NULL,  
    OrderDate date DEFAULT GETDATE()  
);
```

→ Alter keyword

To add column :

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

To delete column :

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

To alter column:

The ALTER COLUMN command is used to change the data type of a column in a table.

The following SQL changes the data type of the column named "BirthDate" in the "Employees" table to type year:

```
ALTER TABLE Employees ALTER COLUMN BirthDate year;
```

### To alter constraint:

The ADD CONSTRAINT command is used to create a constraint after a table is already created.

The following SQL adds a constraint named "PK\_Person" that is a PRIMARY KEY constraint on multiple columns (ID and LastName):

```
ALTER TABLE Persons  
ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);
```

### To drop the constraint :

The DROP CONSTRAINT command is used to delete a UNIQUE, PRIMARY KEY, FOREIGN KEY, or CHECK constraint.

```
ALTER TABLE Persons DROP CONSTRAINT UC_Person;
```

### → GROUP BY Clause

- Groups those rows that have the same values into summary rows.
- It collects data from multiple records and groups the results by one or more columns.
- Generally we use the group by with some aggregate functions.

Count number of students in each city

```
SELECT city, count(name)  
FROM student  
GROUP BY city;
```

Query to find average marks in each city in ascending order.

```
SELECT city, avg(marks)  
FROM student  
GROUP BY city  
ORDER BY city;
```

## → HAVING Clause

Similar to Where i.e. applies some condition on rows.  
Used when we want to apply any **condition after grouping**.

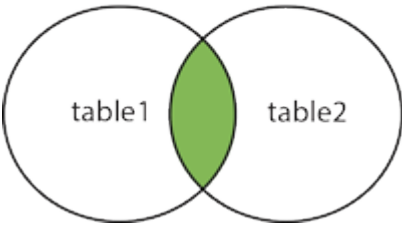
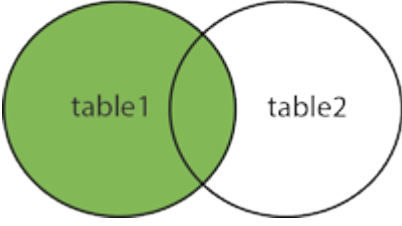
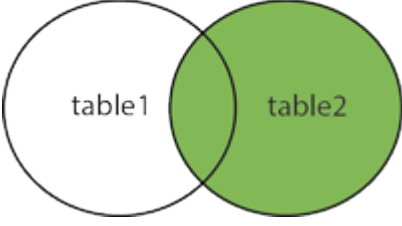
Count number of students in each city where max marks cross 90.

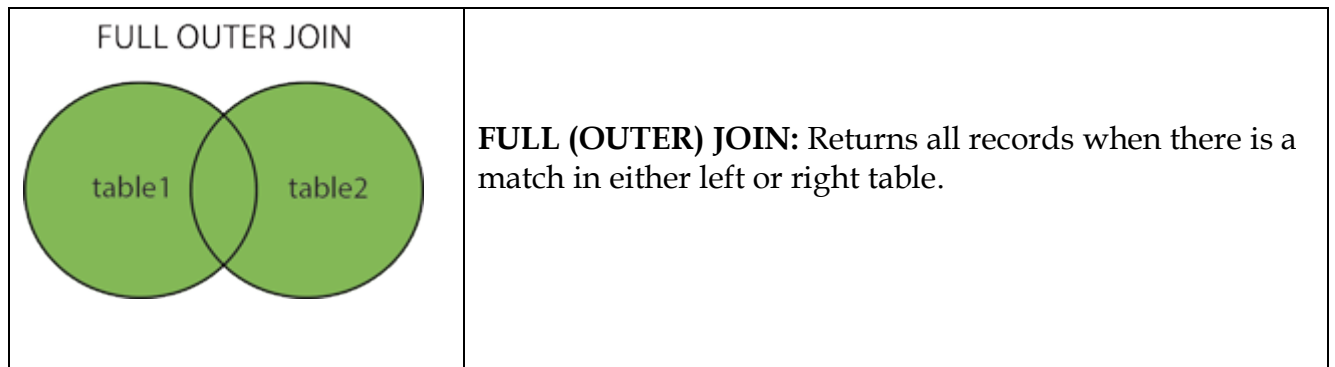
```
SELECT count(name), city  
FROM student  
GROUP BY city  
HAVING max(marks) > 90;
```

## → SQL JOIN

- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
- The join keyword merges two or more tables and creates a temporary image of the merged table. Then according to the conditions provided, it extracts the required data from the image table, and once data is fetched, the temporary image of the merged tables is dumped.

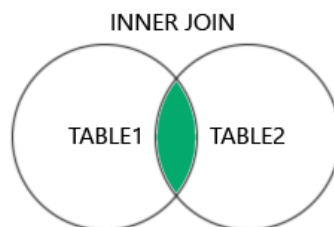
## → Different Types of SQL JOINS

<p>INNER JOIN</p>  <p>The diagram shows two overlapping circles labeled 'table1' and 'table2'. The intersection of the two circles is shaded green, representing the records that have matching values in both tables.</p>	<p><b>(INNER) JOIN:</b> Returns records that have matching values in both tables.</p>
<p>LEFT JOIN</p>  <p>The diagram shows two overlapping circles labeled 'table1' and 'table2'. The entire circle for 'table1' is shaded green, including the intersection with 'table2', representing all records from the left table and the matched records from the right table.</p>	<p><b>LEFT (OUTER) JOIN:</b> Returns all records from the left table, and the matched records from the right table.</p>
<p>RIGHT JOIN</p>  <p>The diagram shows two overlapping circles labeled 'table1' and 'table2'. The entire circle for 'table2' is shaded green, including the intersection with 'table1', representing all records from the right table and the matched records from the left table.</p>	<p><b>RIGHT (OUTER) JOIN:</b> Returns all records from the right table, and the matched records from the left table.</p>



## INNER JOIN

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



```
SELECT column-name
FROM table-1 INNER JOIN table-2
WHERE table-1.column-name = table-2.column-name;
```

ProductID	ProductName	CategoryID	Price
1	Chais	1	18
2	Chang	1	19
3	Aniseed Syrup	2	10

And a selection of the Categories table:

CategoryID	CategoryName	Description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads

```
SELECT ProductID, ProductName, CategoryName
FROM Products
INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID;
```

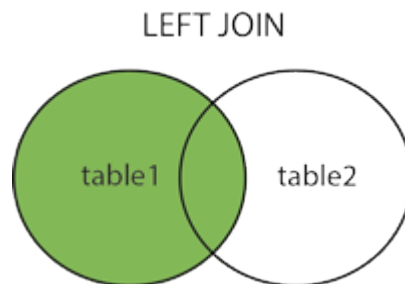
## OUTER JOIN

SQL Outer joins give both matched and unmatched rows of data depending on the type of outer joins. These types of outer joins are sub-divided into the following types:

- Left Outer Join
- Right Outer Join
- Full Outer Join

## LEFT OUTER JOIN

Rows belonging to the left-hand table as well as records available in both the tables, and not having values from the right-hand table are presented.



```
SELECT column-name(s)
FROM table1 LEFT OUTER JOIN table2
ON table1.column-name = table2.column-name;
```

```
SELECT Customers. Name, Shopping_Details.Item_Name
FROM Customers LEFT OUTER JOIN Shopping_Details;
ON Customers.ID = Shopping_Details.ID;
```

**Table\_Name : Customers Left**

CustID	Name	Phone_Number
1	Raj Mehta	98540XXXXX
2	Sanjay Mishra	88888XXXXX
3	Aditi Gupta	67809XXXXX
4	Manish Chopra	12345XXXXX

**Table\_Name : Shopping\_Details Right**

ItemID	CustID	Item_Name	Quantity
1	2	Chips	2
2	3	Chocolate	5
3	5	Dress	8

Primary key  
 Foreign key

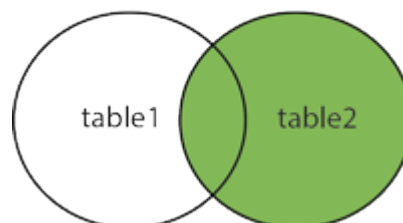
**Temporary Table**

Customers CustID	Name	Phone_Number	Item_ID	Shopping_d etails.CustID	Item_Name	Quantity
1	Raj Mehta	98540XXXXX	NULL	NULL	NULL	NULL
2	Sanjay Mishra	88888XXXXX	1	2	Chips	2
3	Aditi Gupta	67809XXXXX	2	3	Chocolate	5
4	Manish Chopra	12345XXXXX	NULL	NULL	NULL	NULL

## RIGHT OUTER JOIN

Rows belonging to the right-hand table as well as records available in both the tables, and not having values from the left-hand table are presented.

RIGHT JOIN



```
SELECT column-name(s)
FROM table1 RIGHT OUTER JOIN table2
ON table1.column-name = table2.column-name;
```

```
SELECT Customers.Name, Shopping_Details.Item_Name
FROM Customers RIGHT OUTER JOIN Shopping_Details;
ON Customers.ID = Shopping_Details.ID;
```

Table\_Name : Customers **Left**

CustID	Name	Phone_Number
1	Raj Mehta	98540XXXXX
2	Sanjay Mishra	88888XXXXX
3	Aditi Gupta	67809XXXXX
4	Manish Chopra	12345XXXXX

Table\_Name : Shopping\_Details **Right**

ItemID	CustID	Item_Name	Quantity
1	2	Chips	2
2	3	Chocolate	5
3	5	Dress	8



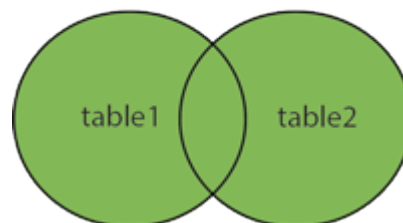
Temporary Table

Customers CustID	Name	Phone_Number	Item_ID	Shopping_d etails.CustID	Item_Name	Quantity
1	Raj Mehta	98540XXXXX	NULL	NULL	NULL	NULL
2	Sanjay Mishra	88888XXXXX	1	2	Chips	2
3	Aditi Gupta	67809XXXXX	2	3	Chocolate	5
4	Manish Chopra	12345XXXXX	NULL	NULL	NULL	NULL
NULL	NULL	NULL	3	5	Dress	8

## FULL OUTER JOIN

The full outer join (a.k.a. SQL Full Join) first adds all the rows matching the stated condition in the query and then adds the remaining unmatched rows from both tables. We need two or more tables for the join.

FULL OUTER JOIN



```
SELECT column-name(s)
FROM table1 FULL OUTER JOIN table2
ON table1.column-name = table2.column-name;
```

```
SELECT Customers.Name, Shopping_Details.Item_Name
FROM Customers FULL OUTER JOIN Shopping_Details
WHERE Customer.ID = Shopping_Details.ID;
```



Table Name: Customers

CustID	Name	Phone_Number
1	Raj Mehta	98540XXXXX
2	Sanjay Mishra	8888XXXXX
3	Aditi Gupta	67809XXXXX
4	Manish Chopra	12345XXXXX

Table Name: Shopping\_Details

ItemID	CustID	Item_Name	Quantity
1	2	Chips	2
2	3	Chocolate	5
3	3	Dress	8

Primary Key  
Foreign Key

Temporary table

Customers CustID	Name	Phone_Number	ItemID	Shopping_D etails.CustID	Item_Name	Quantity
1	Raj Mehta	98540XXXXX	NULL	NULL	NULL	NULL
2	Sanjay Mishra	8888XXXXX	1	2	Chips	2
3	Aditi Gupta	67809XXXXX	2	3	Chocolate	5
4	Manish Chopra	12345XXXXX	NULL	NULL	NULL	NULL
NULL	NULL	NULL	3	5	Dress	8