[SQL: 3](#_Toc133183295)

[RDBMS (Relational Database Management System) 3](#_Toc133183296)

[MySQL 3](#_Toc133183297)

[Difference between SQL and MySQL 3](#_Toc133183298)

[SQL DATA TYPES 3](#_Toc133183299)

[Types of SQL commands: 4](#_Toc133183300)

[01. DDL (data definition language): 4](#_Toc133183301)

[02. DRL/DQL (data retrieval language / data query language): 4](#_Toc133183302)

[03. DML (data modification language): 4](#_Toc133183303)

[04. DCL (Data Control language): 4](#_Toc133183304)

[05. TCL (Transaction control language): 4](#_Toc133183305)

[DDL - MANAGING DB 5](#_Toc133183306)

[Creation of DB 5](#_Toc133183307)

[DATA RETRIEVAL LANGUAGE (DRL) 5](#_Toc133183308)

[WHERE 5](#_Toc133183309)

[BETWEEN 5](#_Toc133183310)

[IN 5](#_Toc133183311)

[AND/OR/NOT 5](#_Toc133183312)

[IS NULL 5](#_Toc133183313)

[Pattern Searching / Wildcard (‘%’, ‘\_’) 5](#_Toc133183314)

[ORDER BY 5](#_Toc133183315)

[GROUP BY 6](#_Toc133183316)

[DISTINCT 6](#_Toc133183317)

[GROUP BY HAVING 6](#_Toc133183318)

[CONSTRAINTS (DDL) 6](#_Toc133183319)

[Primary Key 6](#_Toc133183320)

[Foreign Key 7](#_Toc133183321)

[UNIQUE 7](#_Toc133183322)

[CHECK 7](#_Toc133183323)

[DEFAULT 7](#_Toc133183324)

[ALTER OPERATIONS 7](#_Toc133183325)

[ADD 7](#_Toc133183326)

[MODIFY 7](#_Toc133183327)

[CHANGE COLUMN 7](#_Toc133183328)

[DROP COLUMN 7](#_Toc133183329)

[RENAME 8](#_Toc133183330)

[DATA MANIPULATION LANGUAGE (DML) 8](#_Toc133183331)

[INSERT 8](#_Toc133183332)

[UPDATE 8](#_Toc133183333)

[ON UPDATE CASCADE 8](#_Toc133183334)

[DELETE 8](#_Toc133183335)

[ON DELETE CASCADE - (to overcome DELETE constraint of Referential constraints) 8](#_Toc133183336)

[ON DELETE NULL - (can FK have null values?) 8](#_Toc133183337)

[REPLACE 8](#_Toc133183338)

[JOINING TABLES 9](#_Toc133183339)

[INNER JOIN 9](#_Toc133183340)

[Alias in MySQL (AS) 9](#_Toc133183341)

[OUTER JOIN 9](#_Toc133183342)

[LEFT JOIN 9](#_Toc133183343)

[RIGHT JOIN 9](#_Toc133183344)

[FULL JOIN 9](#_Toc133183345)

[CROSS JOIN 9](#_Toc133183346)

[SELF JOIN 9](#_Toc133183347)

[Join without using join keywords. 9](#_Toc133183348)

[SET OPERATIONS 10](#_Toc133183349)

[UNION 10](#_Toc133183350)

[INTERSECT 10](#_Toc133183351)

[MINUS 10](#_Toc133183352)

[SUB QUERIES 10](#_Toc133183353)

[Subquery using FROM clause 11](#_Toc133183354)

[Subquery using SELECT 11](#_Toc133183355)

[Derived Subquery 11](#_Toc133183356)

[Co-related sub-queries 11](#_Toc133183357)

[JOIN VS SUB-QUERIES 11](#_Toc133183358)

[MySQL VIEWS 11](#_Toc133183359)

# 

# SQL:

* SQl stands for Structured Query Language, , and it is used to communicate with database .
* This is a standard language used to perform task such as CRUD operations(Create , Delete ,Update Delete) of data from a database
* SQL is not DB, is a query language.

There are lots of different database systems, or DBMS – Database Management Systems

* Microsoft SQL Server

1. Enterprise, Developer versions, etc.
2. Express version is free of charge

* Oracle
* MySQL (Oracle, previously Sun Microsystems) – MySQL can be used free of charge (open source license), Web sites that use MySQL: YouTube, Wikipedia, Facebook
* Microsoft Access
* IBM DB2
* Sybase

# Data

Data is a collection of information

# Database

A database is an organized collection of structured information, or data, typically stored electronically in a computer system.

This is also known as structured form of data which can be accessed in many ways

# DBMS(Database Management System)

DBMS is software used to identify, manage, and create a databse.

In dbms the data is stored as a file

Software - XML, Microsoft Access.

# RDBMS (Relational Database Management System)

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.

Example. MySQL, MS SQL, Oracle, IBM etc.

# MySQL

MySQL is a open source RDBMS software and database management system .

# Difference between SQL and MySQL

SQL is Structured Query language used to perform CRUD operations in R-DB, while MySQL is a RDBMS used to store, manage and administrate DB (provided by itself) using SQL.

# SQL DATA TYPES

In SQL DB, data is stored in the form of tables.

1. Data can be of different types, like INT, CHAR etc.

DATATYPE

|  |  |
| --- | --- |
| DATATYPE | Description |
| CHAR | string(0-255), string with size = (0, 255], e.g., CHAR(251) |
| VARCHAR | string(0-255) |
| TINYTEXT | String(0-255) |
| TEXT | string(0-65535) |
| BLOB | string(0-65535) |
| MEDIUMTEXT | string(0-16777215) |
| MEDIUMBLOB | string(0-16777215) |
| LONGTEXT | string(0-4294967295) |
| LONGBLOB | string(0-4294967295) |
| TINYINT | integer(-128 to 127) |
| SMALLINT | integer(-32768 to 32767) |
| MEDIUMINT | integer(-8388608 to 8388607) |
| INT | integer(-2147483648 to 2147483647) |
| BIGINT | integer (-9223372036854775808 to  9223372036854775807) |
| FLOAT | Decimal with precision to 23 digits |
| DOUBLE | Decimal with 24 to 53 digits |

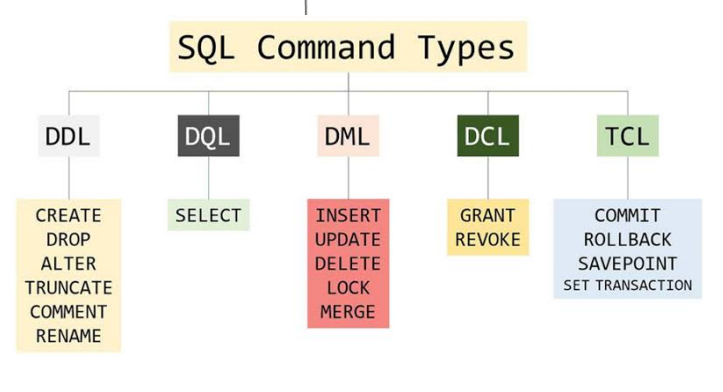
|  |  |
| --- | --- |
| DATATYPE | Description |
| DECIMAL | Double stored as string |
| DATE | YYYY-MM-DD |
| DATETIME | YYYY-MM-DD HH:MM:SS |
| TIMESTAMP | YYYYMMDDHHMMSS |
| TIME | HH:MM:SS |
| ENUM | One of the preset values |
| SET | One or many of the preset values |
| BOOLEAN | 0/1 |
| BIT | e.g., BIT(n), n upto 64, store values in bits. |

Size: TINY < SMALL < MEDIUM < INT < BIGINT.

**Variable length Data types** e.g., VARCHAR, are better to use as they occupy space equal to the actual data size.

Values can also be unsigned e.g., INT UNSIGNED.

# Types of SQL commands:



# SQL Table

Table is a collection of data, organized in terms of rows and columns. In DBMS term, table is known as relation and row as tuple.

# 01. DDL (data definition language):

DDL is used to create , modified , drop , rename an object on database.

**USE** db-name; //need to execute to choose on which DB CREATE TABLE etc commands will be executed.

**SHOW** DATABASES; list all the DBs in the server.

**SHOW** TABLES; list tables in the selected DB

## CREATE

Create is used to create table, DataBase, view.

**Create a database -** CREATE DATABASE databasename;

After creating databse you have to use it – USE DATABASE\_NAME

Creating table –

CREATE TABLE table\_name (  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
   .... );

CREATE VIEW Syntax

CREATE VIEW view\_name AS  
SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

## Drop

It is used to delete both the structure and record stored in the table.

Drop database\_name

Drop table\_name

Drop View\_name

## ALTER

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 – alter column**

To change existing tables column datatype and data size

Alter table table\_name alter column column\_name new\_datatype new\_datasize

Alter table student alter column name varchar(50)

### ADD

**Add new column (primary key , foreign key) in table**

|  |  |
| --- | --- |
| ALTER TABLE table\_name ADD column\_name datatype; | ALTER TABLE Customers ADD Email varchar(255); |

### DROP COLUMN

**To Drop a column completely.**

|  |  |
| --- | --- |
| ALTER TABLE table\_name DROP COLUMN column\_name; | ALTER TABLE Customers DROP COLUMN Email; |

### RENAME COLUMN

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

ALTER TABLE table\_name  
RENAME COLUMN old\_name to new\_name;

### MODIFY

To change the data type of a column in a table,

|  |  |
| --- | --- |
| ALTER TABLE table\_name MODIFY COLUMN column\_name datatype; |  |

### CHANGE COLUMN

**Change column , change the column name datatype , definition,datasize**

|  |  |
| --- | --- |
| ALTER TABLE table\_name  CHANGE COLUMN old-col-name new-col-name new-col-datatype; | ALTER TABLE customer  CHANGE COLUMN name customer-name VARCHAR(1024); |

### RENAME table name

1. **Rename table name itself.**
2. ALTER TABLE old\_table-name RENAME TO new-table-name;
3. e.g., ALTER TABLE old\_customer RENAME TO customer-details;

## TRUNCATE:

A truncate SQL statement is used to remove all rows (complete data) from a table. It is similar to the DELETE statement with no WHERE clause.

Drop table command can also be used to delete complete table but it deletes table structure too. TRUNCATE TABLE doesn't delete the structure of the table.

**TRUNCATE** **TABLE** table\_name;

## RENAME

rename DB name, table name, column name etc

RENAME **DATABASE** old\_database\_name **TO** new\_database\_name;

RENAME table  old\_table\_name **TO** new\_table\_name

# 02. DRL/DQL (data retrieval language / data query language):

## select

The SELECT statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

It uses only one command = SELECT

|  |  |
| --- | --- |
| SELECT column1, column2, ... FROM table\_name; | Select \* from table\_name  This is show all columns of table |

|  |  |
| --- | --- |
| SELECT column1, column2, ... FROM table\_name WHERE condition; | SELECT \* FROM Customers WHERE Country='Mexico'; |

# 03. DML (data modification language):

DML commands are used to modify the database.

It is responsible for all form of changes in the database.

DML command: INSERT , UPDATE , DELETE

## INSERT

It is used to insert data into the row of a table.

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

## UPDATE

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 Customers  
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'  
WHERE CustomerID = 1;

UPDATE Multiple Records

It is the WHERE clause that determines how many records will be updated.

The following SQL statement will update the ContactName to "Juan" for all records where country is "Mexico":

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

### ON UPDATE CASCADE

Can be added to the table while creating constraints. Suppose there is a situation where we have two tables such that primary key of one table is the foreign key for another table. if we update the primary key of the first table then using the ON UPDATE CASCADE foreign key of the second table automatically get updated.

## DELETE

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

|  |  |
| --- | --- |
| DELETE FROM table\_name WHERE condition; | DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste'; |

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;

### ON DELETE CASCADE

**ON DELETE CASCADE** constraint is used in MySQL to delete the rows from the child table automatically, when the rows from the parent table are deleted.

# Difference between DELETE and TRANCATE

|  |  |
| --- | --- |
| DELETE | TRANCATE |
| DML OPERATION | DLL OPERATION |
| It can delete a specific raw from table | It is not possible |
| It support where clause condition | It does not support where clause |
| It temporary data deletion | It is permanent data deletion |
| We can restore deleted data using rollback | We can not restore deleted data by using rollback |
| Execution speed is slow | Execution speed is fast |

# 04. DCL (Data Control language) :

DCL commands are used to grant and take back authority from any database user

## Grant:

It is used to give user access privileges to a database.

Example 1. GRANT SELECT, UPDATE ON MY\_TABLE TO SOME\_USER, ANOTHER\_USER;

## Revoke

It is used to take back permissions from the user.

Example 1. REVOKE SELECT, UPDATE ON MY\_TABLE FROM USER1, USER2;

# 05. TCL (Transaction control language):

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

## Commit:

Commit command is used to save all the transactions to the database.

Syntax:

Example: 1. DELETE FROM CUSTOMERS WHERE AGE = 25 COMMIT;

## Rollback:

Rollback command is used to undo transactions that have not already been saved to the database. Syntax:

Example: 1. DELETE FROM CUSTOMERS WHERE AGE = 25 ROLLBACK;

## SAVEPOINT:

It is used to roll the transaction back to a certain point without rolling back the entire transaction. Syntax:

SAVEPOINT SAVEPOINT\_NAME;

# WHERE

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

Syntax

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

|  |  |
| --- | --- |
| SELECT \* FROM Customers WHERE Country='Mexico'; | SELECT \* FROM Customers WHERE CustomerID=1; |

|  |  |
| --- | --- |
| SELECT \* FROM Products  WHERE Price BETWEEN 50 AND 60; | SELECT \* FROM Products  WHERE Price <> 18; (<> not equal to) |

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

# 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:

1. AND operator displays a record if all the conditions separated by AND are TRUE.
2. OR operator displays a record if any of the conditions separated by OR is TRUE.
3. 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;

# ORDER BY

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.

Syntax

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

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

SELECT \* FROM Customers  
ORDER BY Country;

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

SELECT \* FROM Customers  
ORDER BY Country DESC;

 Several Columns Example

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

SELECT \* FROM Customers  
ORDER BY Country, CustomerName;

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

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

# NULL

1. A field with a NULL value is a field with no value.(blank) empty values
2. 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.
3. We will have to use the IS NULL and IS NOT NULL operators instead.

|  |  |
| --- | --- |
| **IS NULL Syntax**  SELECT column\_namesFROM table\_name WHERE column\_name IS NULL; | **IS NOT NULL Syntax**  SELECT column\_namesFROM table\_name WHERE column\_name IS NOT NULL; |

|  |  |
| --- | --- |
| SELECT CustomerName, ContactName, Address FROM Customers WHERE Address IS NULL | SELECT CustomerName, ContactName, Address FROM Customers WHERE Address IS NOT NULL; |

# LIKE / wildcard

* The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
* A wildcard character is used to substitute one or more characters in a string.

The like operator is used in string compression.

There are two wildcards

% ( percent sign) represents zero, one, or multiple characters

 \_ (underscore sign) represents one, single character

In oracle \*==% , ?==\_

LIKE Syntax

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

|  |  |  |
| --- | --- | --- |
| **%**  A%  ABC..XYZ √ | %  %A  ZYX..CBA √ | %  %A%  …BCDABCD… √ |

|  |  |  |
| --- | --- | --- |
| **\_**  A\_  A X  AB √  AC √  ABC X | **\_**  A\_\_  ABC √  ABCD X  AB X  A X | **\_**  \_A  BA √  CA √  BCA X |

**Wildcard Characters in MS Access**

|  |  |  |
| --- | --- | --- |
| 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 |
| # | Represents any single numeric character | 2#5 finds 205, 215, 225, 235, 245, 255, 265, 275, 285, and 295 |

**Wildcard Characters in SQL Server**

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

# IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

|  |  |
| --- | --- |
| SELECT column\_name(s) FROM table\_name WHERE column\_name IN (value1, value2, ...); | SELECT column\_name(s) FROM table\_name WHERE column\_name IN (*SELECT* STATEMENT); |

The following SQL statement selects all customers that are located in "Germany", "France" or "UK":

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

The following SQL statement selects all customers that are NOT located in "Germany", "France" or "UK":

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

The following SQL statement selects all customers that are from the same countries as the suppliers:

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

# BETWEEN

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.

Syntax

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name BETWEEN value1 AND value2;

The following SQL statement selects all products with a price between 10 and 20:

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

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

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

The following SQL statement selects all products with a price between 10 and 20. In addition; do not show products with a CategoryID of 1,2, or 3:

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

The following SQL statement selects all products with a ProductName between Carnarvon Tigers and Mozzarella di Giovanni:

SELECT \* FROM Products  
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'  
ORDER BY ProductName;

The following SQL statement selects all orders with an OrderDate between '01-July-1996' and '31-July-1996':

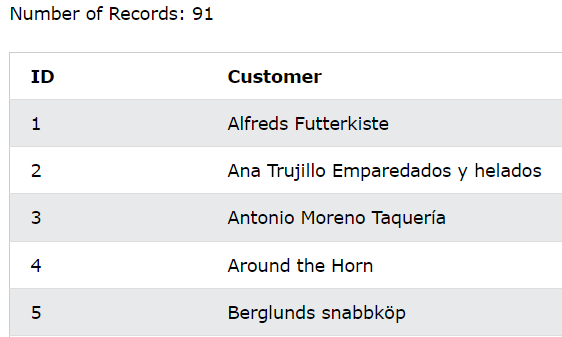
SELECT \* FROM Orders  
WHERE OrderDate BETWEEN #07/01/1996# AND #07/31/1996#;

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

# Aliases

* SQL aliases are used to give a table, or a column in a table, a temporary name.
* Aliases are often used to make column names more readable.
* An alias only exists for the duration of that query.
* An alias is created with the AS keyword.

|  |  |
| --- | --- |
| Alias Column Syntax  SELECT column\_name AS alias\_name FROM table\_name; | Alias Table Syntax  SELECT column\_name(s) FROM table\_name AS alias\_name; |



SELECT CustomerID AS ID, CustomerName AS Customer  
FROM Customers;

# SELECT DISTINCT

* 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.

Syntax

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

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ProductID | ProductName | SupplierID | CategoryID | Unit | Price |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 - 550 ml bottles | 10 |
| 4 | Chef Anton's Cajun Seasoning | 2 | 2 | 48 - 6 oz jars | 22 |
| 5 | Chef Anton's Gumbo Mix | 2 | 2 | 36 boxes | 21.35 |

MIN() Example

|  |
| --- |
| SmallestPrice |
| 2.5 |
|  |

finds the price of the cheapest product: Number of Records: 1

SELECT MIN(Price) AS SmallestPrice   
FROM Products;

MAX() Example

The following SQL statement finds the price of the most expensive product:

|  |
| --- |
| largestPrice |
| 21.35 |
|  |

SELECT MAX(Price) AS LargestPrice   
FROM Products;

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

The COUNT() function returns the number of rows that matches a specified criterion.

 is the function to count the number of rows in a table

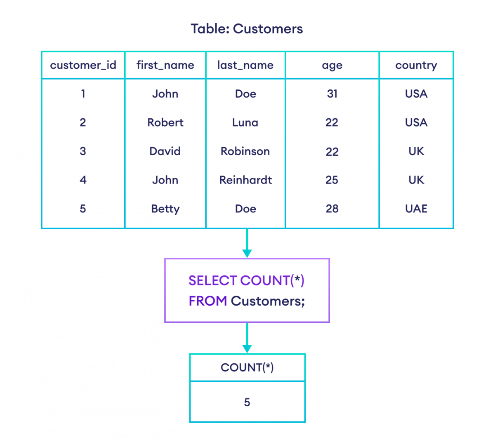
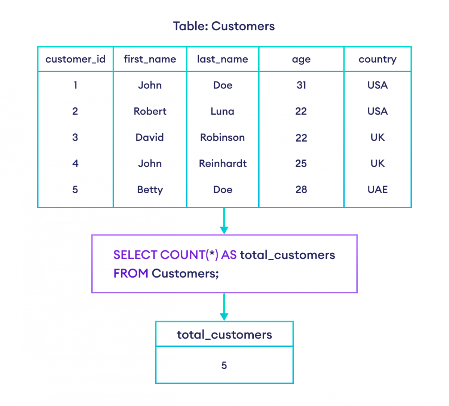
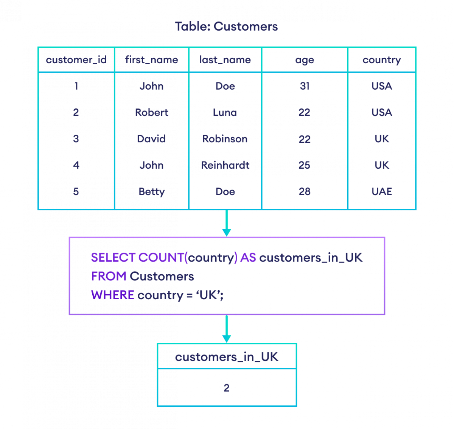
SELECT COUNT(column\_name)  
FROM table\_name  
WHERE condition;

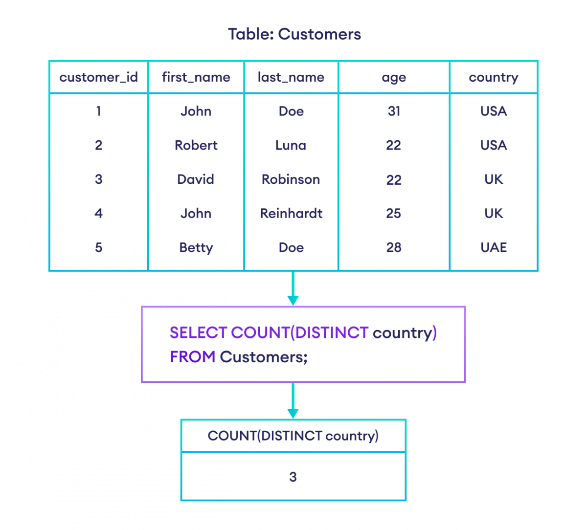
The AVG() function returns the average value of a numeric column.

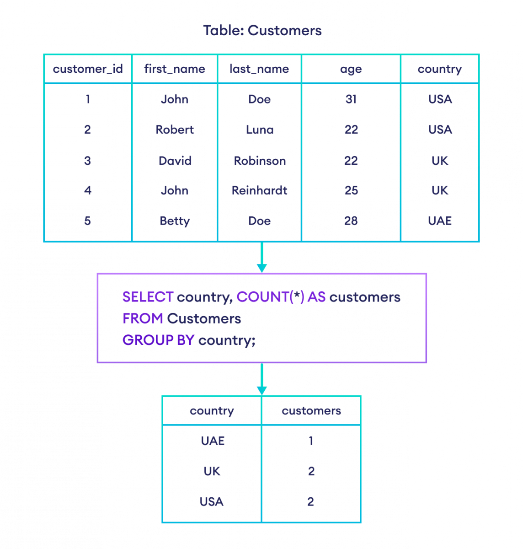
SELECT AVG(column\_name)  
FROM table\_name  
WHERE condition;

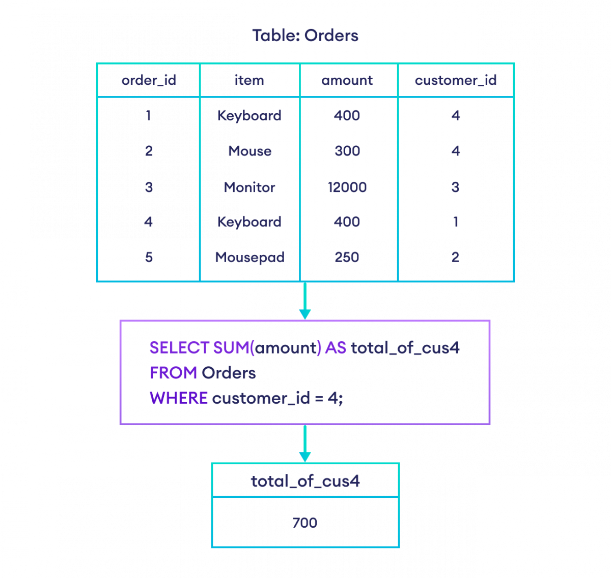
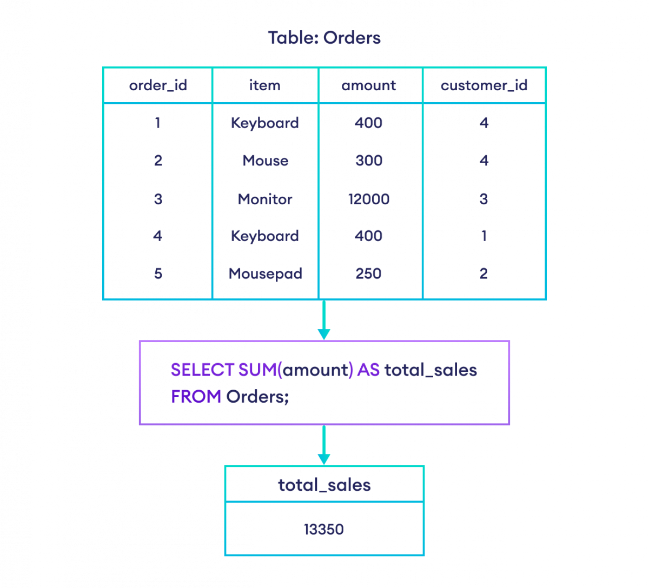
The SUM() function returns the total sum of a numeric column.

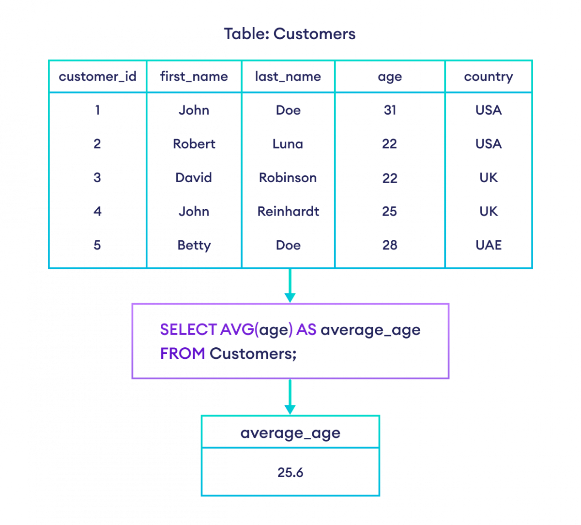
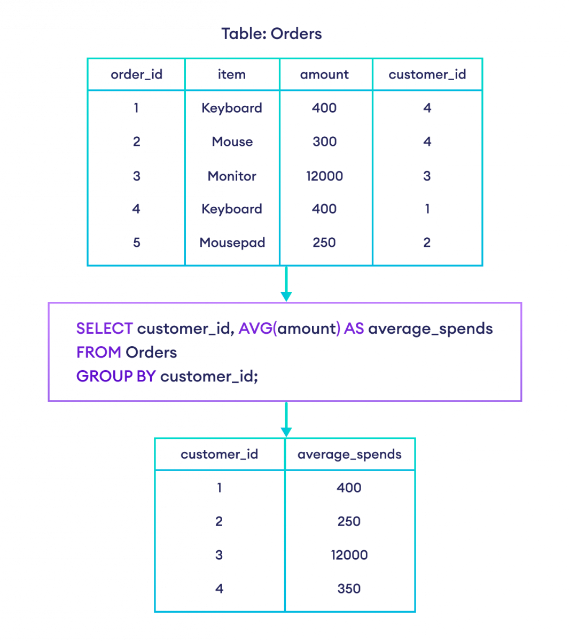
SELECT SUM(column\_name)  
FROM table\_name  
WHERE condition;







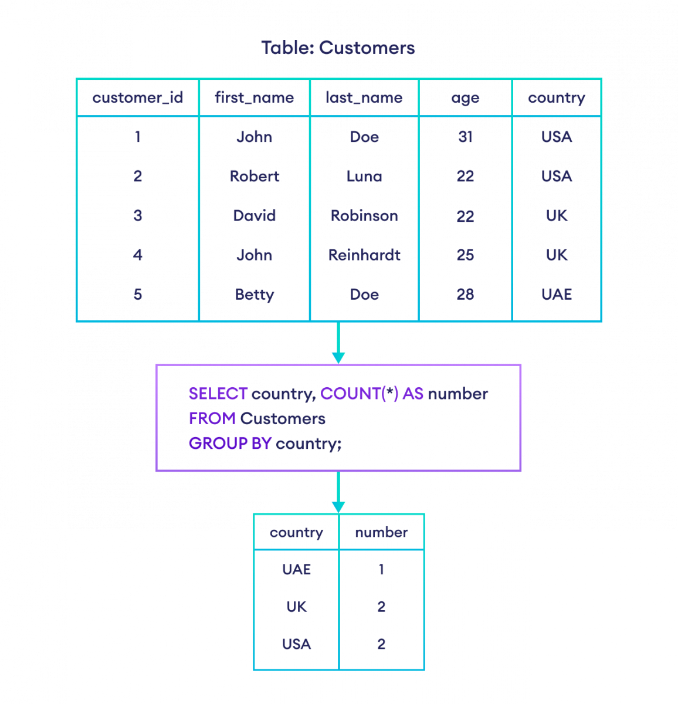
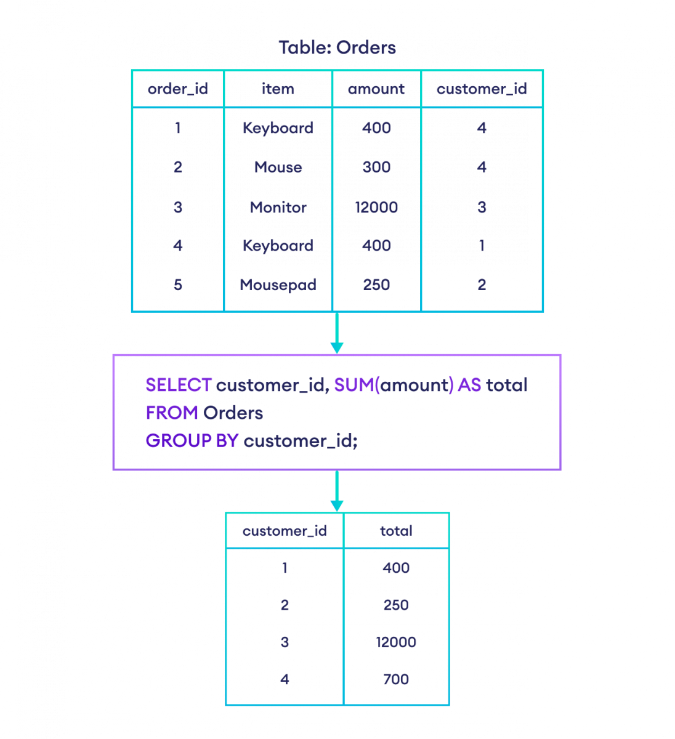




# GROUP BY

* GROUP BY clause is used to group rows by one or more columns.
* The GROUP BY clause is used in conjunction with aggregate functions such as [MIN() and MAX()](https://www.programiz.com/sql/min-and-max), [SUM() and AVG()](https://www.programiz.com/sql/sum-avg), [COUNT()](https://www.programiz.com/sql/count), etc.

|  |  |
| --- | --- |
| SELECT column ,column2,…….  FROM TABLE  GROUP BY COULMNA , COLUMNB… | SELECT column\_name(s) FROM table\_name WHERE condition GROUP BY column\_name(s) ORDER BY column\_name(s); |

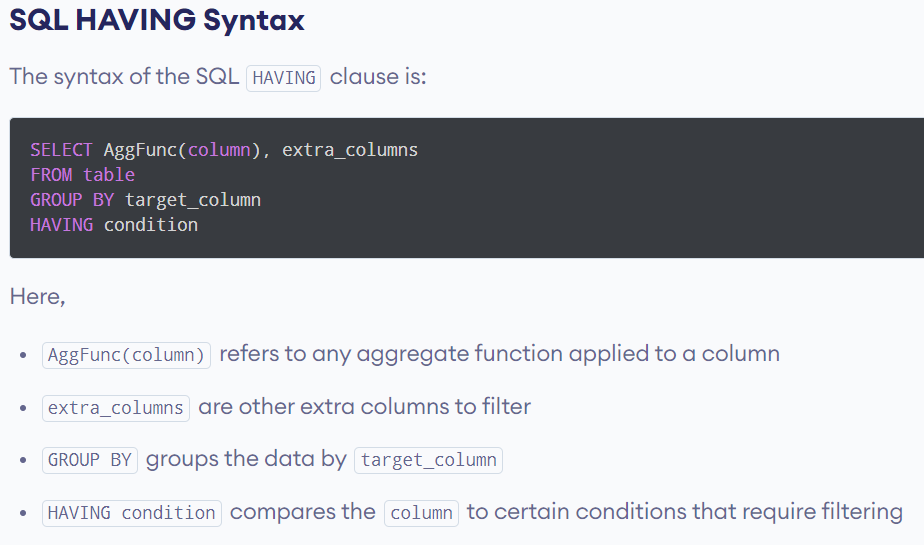


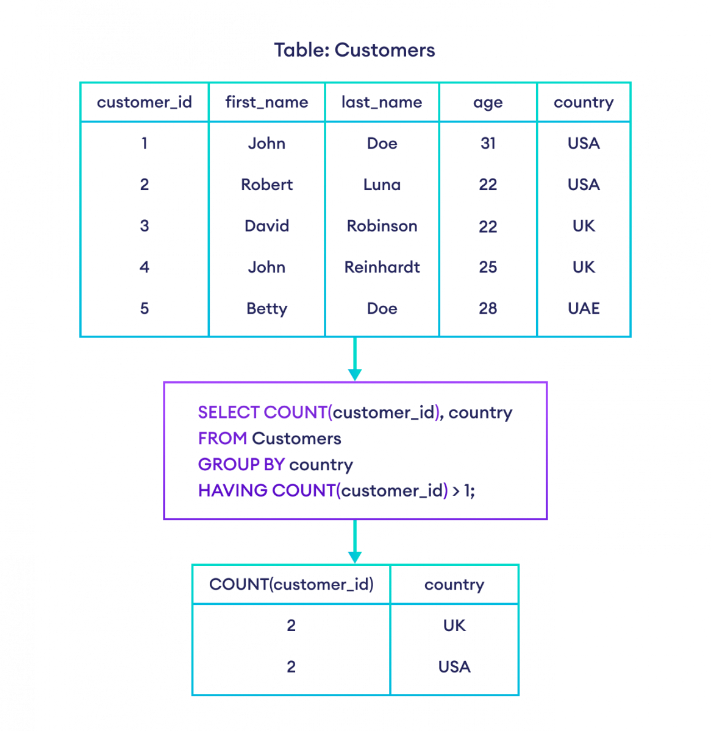
The following SQL statement lists the number of customers in each country, sorted high to low:

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

# GROUP BY HAVING

* HAVING clause is used if we need to filter the result set based on aggregate functions such as [MIN() and MAX()](https://www.programiz.com/sql/min-and-max), [SUM() and AVG()](https://www.programiz.com/sql/sum-avg), and [COUNT()](https://www.programiz.com/sql/count).
* The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.





|  |  |  |  |
| --- | --- | --- | --- |
| order\_id | item | amount | customer\_id |
| 1 | Keyboard | 400 | 4 |
| 2 | Mouse | 300 | 4 |
| 3 | Monitor | 12000 | 3 |
| 4 | Keyboard | 400 | 1 |
| 5 | Mousepad | 250 | 2 |

Orders

|  |  |
| --- | --- |
| customer\_id | total |
| 1 | 400 |
| 2 | 250 |

SELECT customer\_id, SUM(amount) AS total

FROM Orders

GROUP BY customer\_id

HAVING SUM(amount) < 500;

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5  
ORDER BY COUNT(CustomerID) DESC;

# Difference having and where

|  |  |
| --- | --- |
| HAVING Clause | WHERE Clause |
| The HAVING clause checks the condition on **a group of rows**. | The WHERE clause checks the condition **on each individual row**. |
| HAVING is used with aggregate functions. | The WHERE clause cannot be used with aggregate functions. |
| The HAVING clause is executed after the GROUP BY clause. | The WHERE clause is executed before the GROUP BY clause. |

# CONSTRAINTS (DDL)

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

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*,  
    .... );

**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) - Prevents actions that would destroy links between tables

[CHECK](https://www.w3schools.com/sql/sql_check.asp) - Ensures that the values in a column satisfies a specific condition

[DEFAULT](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column if no value is specified

[CREATE INDEX](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly

## [NOT NULL](https://www.w3schools.com/sql/sql_notnull.asp)

Ensures that a column cannot have a NULL value , MUST ASSIGN A VALUE

## 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.
* A PRIMARY KEY constraint automatically has a UNIQUE constraint.
* However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.
* **To create more than one primary key use UNIQUE + NOT NULL**

## Primary Key

* 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).
* An attribute can be **PK and FK both** in a table.

## Foreign Key

When a primary key of table referred to another table this key is foreign key.

Foreign key with reference is used to link two table in real manners

Foreign key with reference is known as referential integrity

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.

Look at the following two tables:

**Persons Table**

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

**MySQL:**

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

## CHECK

* 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.

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));

## DEFAULT

1. Set default value of the column.
2. CREATE TABLE account (

…

saving-rate DOUBLE NOT NULL DEFAULT 4.25,

…);

## INDEX

* The CREATE INDEX statement is used to create indexes in tables.
* Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.
* **Note:** Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

Creates an index on a table. Duplicate values are allowed:

CREATE INDEX index\_name  
ON table\_name (column1, column2, ...);

Creates a unique index on a table. Duplicate values are not allowed:

CREATE UNIQUE INDEX index\_name  
ON table\_name (column1, column2, ...);

## AUTO INCREMENT

* Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.
* Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

The following SQL statement defines the "Personid" column to be an auto-increment primary key field in the "Persons" table:

CREATE TABLE Persons (  
    Personid int NOT NULL AUTO\_INCR EMENT,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    PRIMARY KEY (Personid));

MySQL uses the AUTO\_INCREMENT keyword to perform an auto-increment feature.

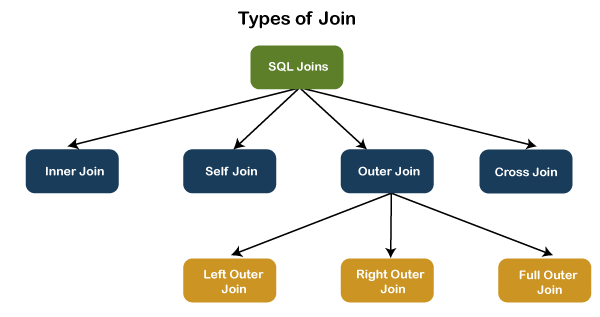
By default, the starting value for AUTO\_INCREMENT is 1, and it will increment by 1 for each new record.

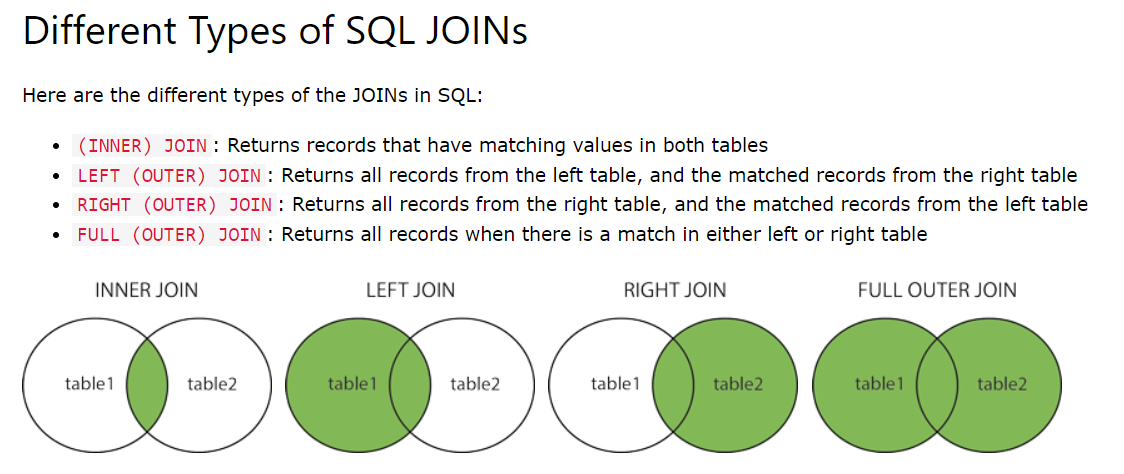
To let the AUTO\_INCREMENT sequence start with another value, use the following SQL statement:

ALTER TABLE Persons AUTO\_INCREMENT=100

# JOIN

* A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
* FK are used to do reference to other table.

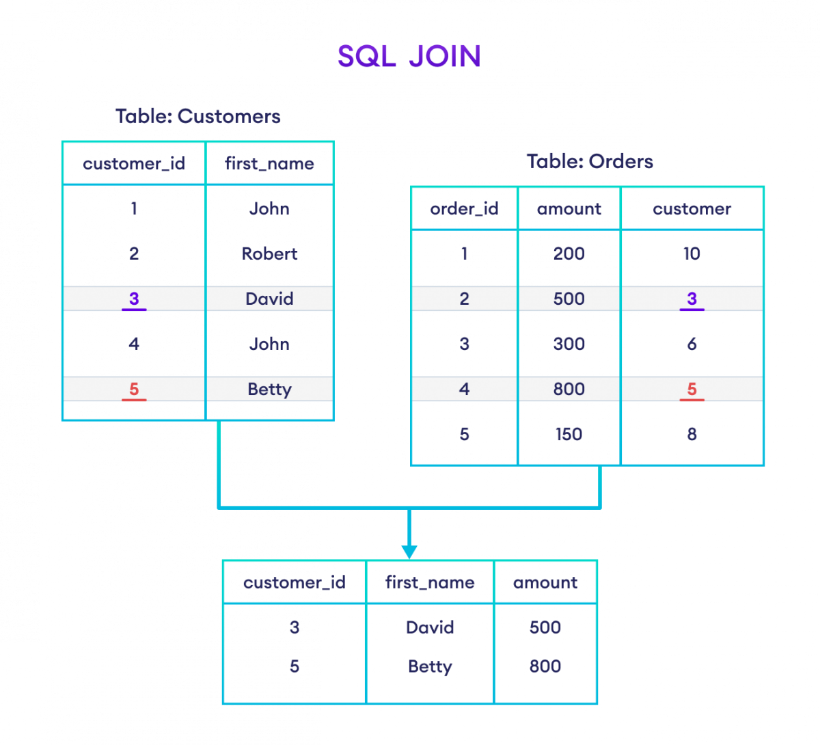


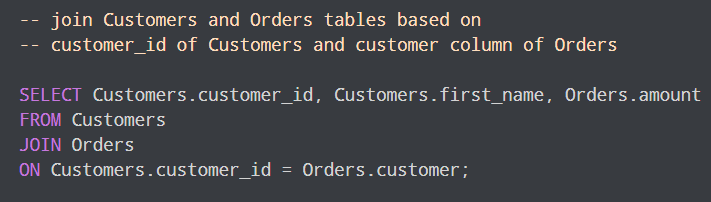


## INNER JOIN

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

SELECT column\_name(s)  
FROM table1  
INNER JOIN table2ON table1.column\_name = table2.column\_name;





|  |  |  |
| --- | --- | --- |
| order\_id | amount | customer |
| 1 | 200 | 10 |
| 2 | 500 | 3 |
| 3 | 300 | 6 |
| 4 | 800 | 5 |
| 5 | 150 | 8 |

### Alias in MySQL (AS)

Customers Orders

|  |  |
| --- | --- |
| customer\_id | first\_name |
| 1 | John |
| 2 | Robert |
| 3 | David |
| 4 | John |
| 5 | Betty |

-- use alias C for Customers table

-- use alias O for Orders table

SELECT C.customer\_id, C.first\_name, O.amount

FROM Customers AS C

JOIN Orders AS O

ON C.customer\_id = O.customer;

Result:

|  |  |  |
| --- | --- | --- |
| customer\_id | first\_name | amount |
| 3 | David | 500 |
| 5 | Betty | 800 |

 we can change the column names temporarily using AS aliases. For example,

-- use alias C for Customers table

-- use alias O for Orders table

SELECT C.customer\_id AS cid, C.first\_name AS name, O.amount

FROM Customers AS C

JOIN Orders AS O

ON C.customer\_id = O.customer;

## OUTER JOIN

### LEFT JOIN

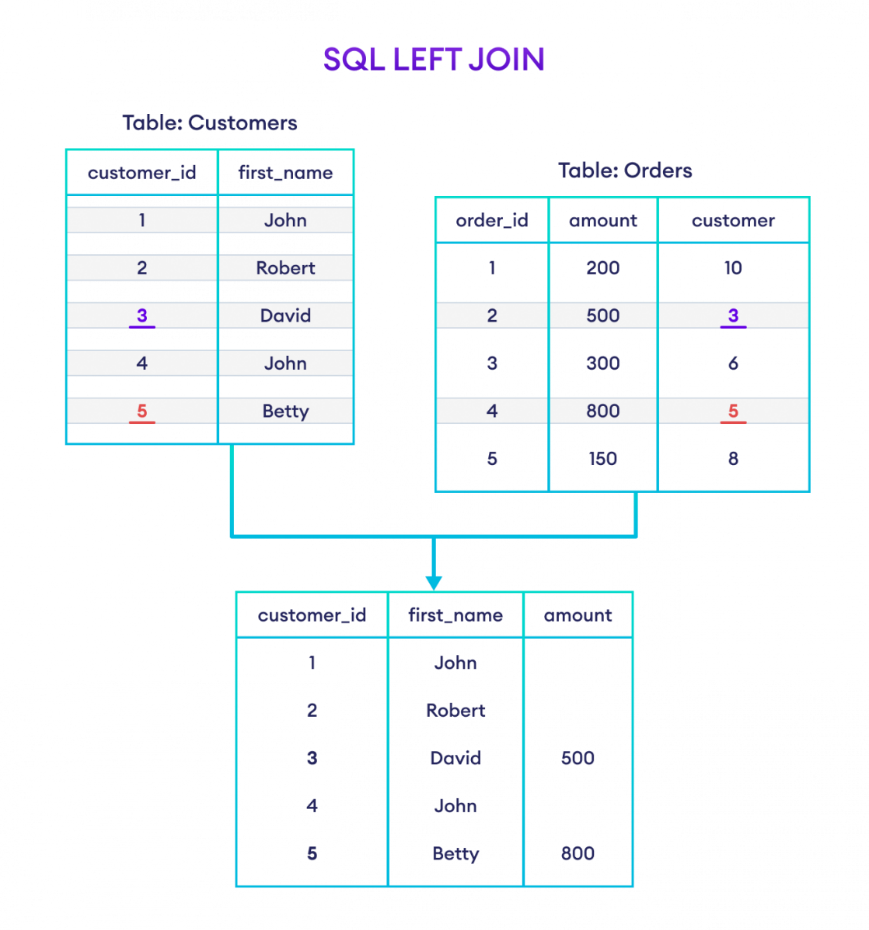
The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.



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

-- left join the Customers and Orders tables

SELECT Customers.customer\_id, Customers.first\_name, Orders.amount

FROM Customers

LEFT JOIN Orders

ON Customers.customer\_id = Orders.customer;

### RIGHT JOIN

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.



SELECT column\_name(s)  
FROM table1  
RIGHT JOIN table2ON table1.column\_name = table2.column\_name;

-- join Customers and Orders tables

-- based on customer\_id of Customers and customer of Orders

-- Customers is the left table

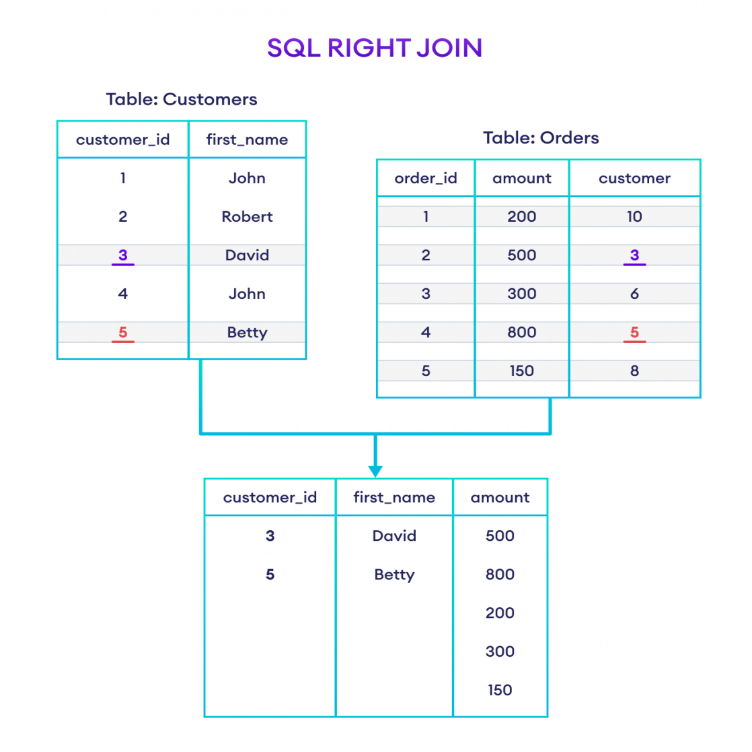
-- Orders is the right table

SELECT Customers.customer\_id, Customers.first\_name, Orders.amount

FROM Customers

RIGHT JOIN Orders

ON Customers.customer\_id = Orders.customer;



### FULL JOIN

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

MYSQL doesn’t support full join direct

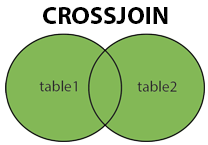
SELECT \* FROM TABLE1 LEFT JOIN TABLE2 ON TABLE1.COL= TABLE2.COL

UNION

SELECT \* FROM TABLE1 RIGHT JOIN TABLE2 ON TABLE1.COL= TABLE2.COL



## CROSS JOIN

CROSS JOIN keyword returns all records from both tables (table1 and table2).

SELECT column\_name(s)  
FROM table1  
CROSS JOIN table2;

1. Used rarely in practical purpose.
2. Table-1 has 10 rows and table-2 has 5, then resultant would have 50 rows.
3. SELECT column-lists FROM table1 CROSS JOIN table2;

## SELF JOIN

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

SELECT column\_name(s)  
FROM table1 T1, table1 T2  
WHERE condition;

T1 and T2 are different table aliases for the same table.

when the table has a FOREIGN KEY which references its own PRIMARY KEY. To join a table itself means that each row of the table is combined with itself and with every other row of the table.

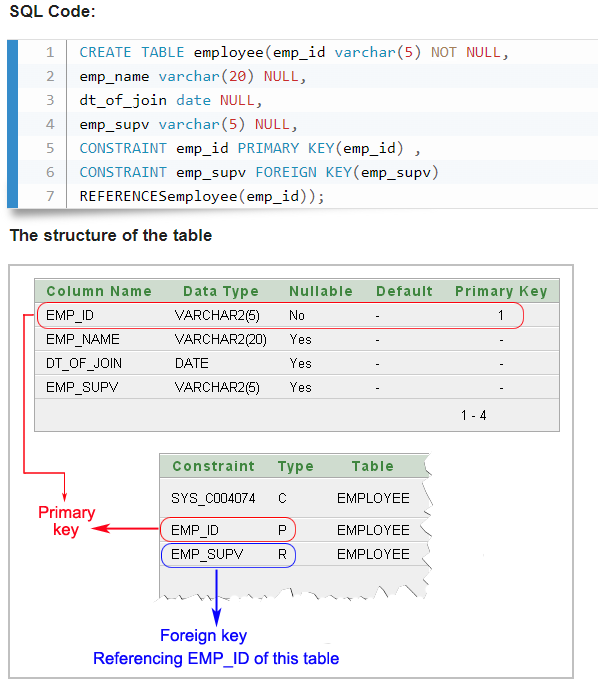
The self join can be viewed as a join of two copies of the same table. The table is not actually copied, but SQL performs the command as though it were.

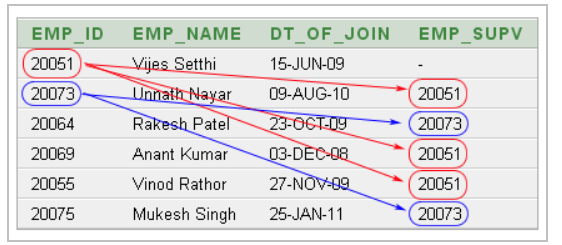
SYNTAX:

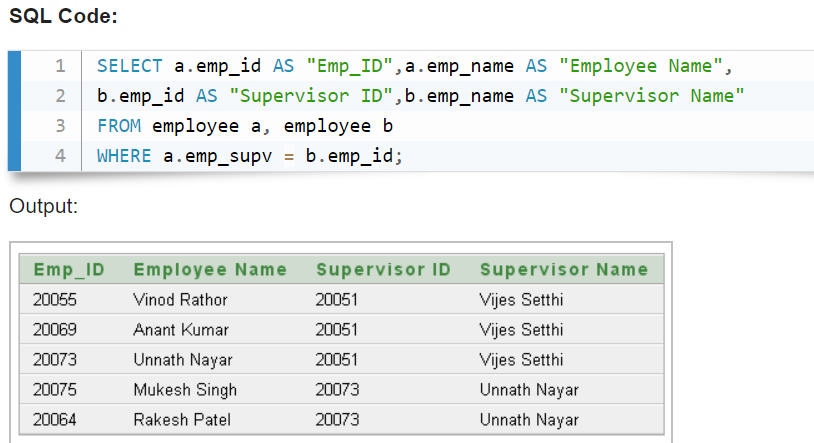
SELECT a.column\_name, b.column\_name...

FROM table1 a, table1 b

WHERE a.common\_filed = b.common\_field;







## Join without using join keywords.

SELECT \* FROM table1, table2 WHERE condition;

e.g., SELECT artist\_name, album\_name, year\_recordedFROM artist, albumWHERE artist.id = album.artist\_id;

# SET OPERATIONS

1. Used to combine multiple select statements.
2. Always gives distinct rows.

## UNION

The UNION operator is used to combine the result-set of two or more SELECT statements.

1. Every SELECT statement within UNION must have the same number of columns
2. The columns must also have similar data types
3. The columns in every SELECT statement must also be in the same order

**UNION Syntax**

SELECT column\_name(s) FROM table1  
UNION  
SELECT column\_name(s) FROM table2;

**UNION ALL Syntax**

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

SELECT column\_name(s) FROM table1  
UNION ALL  
SELECT column\_name(s) FROM table2;

**Note:** The column names in the result-set are usually equal to the column names in the

The following SQL statement returns the cities (only distinct values) from both the "Customers" and the "Suppliers" table:

SELECT City FROM Customers  
UNION  
SELECT City FROM Suppliers  
ORDER BY City;

The following SQL statement returns the cities (duplicate values also) from both the "Customers" and the "Suppliers" table:

SELECT City FROM Customers  
UNION ALL  
SELECT City FROM Suppliers  
ORDER BY City;

The following SQL statement returns the German cities (only distinct values) from both the "Customers" and the "Suppliers" table:

SELECT City, Country FROM Customers  
WHERE Country='Germany'  
UNION  
SELECT City, Country FROM Suppliers  
WHERE Country='Germany'  
ORDER BY City;

## INTERSECT

* + 1. It Returns common values between tables.
    2. We have to Emulate using inner join.
    3. SELECT DISTINCT column-list FROM table-1 INNER JOIN table-2 USING(join\_cond);
    4. Ex. = SELECT DISTINCT \* FROM table1 INNER JOIN table2 ON USING(id);

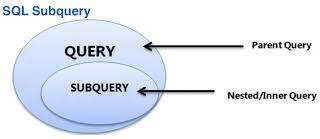
## MINUS

* + - 1. This operator returns the distinct row from the first table that does not occur in the second table.
      2. Emulated.
      3. SELECT column\_list FROM table1 LEFT JOIN table2 ON condition WHERE table2.column\_name IS NULL;
      4. e.g., SELECT id FROM table-1 LEFT JOIN table-2 USING(id) WHERE table-2.id IS NULL;

# Difference between join and set

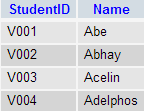
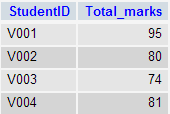
|  |  |
| --- | --- |
| JOIN | SET Operations |
| Combines multiple tables based on matching condition. | Combination is resulting set from two or more SELECT statements. |
| Column wise combination. | Row wise combination. |
| Data types of two tables can be different. | Datatypes of corresponding columns from each table should be the same. |
| Can generate both distinct or duplicate rows. | Generate distinct rows. |
| The number of column(s) selected may or may be the same from each table. | nTohte number of column(s) selected must be the same from each table. |
| Combines results horizontally. | Combines results vertically. |

# SUB QUERIES

* A subquery is a SQL query nested inside a larger query.
* The subquery (inner query) executes once before the main query (outer query) executes.
* The main query (outer query) use the subquery result.
* Alternative to joins.
* A subquery exist mainly in three clauses:
  + - A SELECT clause
  + - A FROM clause
  + - A WHERE clause

You can use a subquery in a SELECT, INSERT, DELETE, or UPDATE statement to perform the following tasks:.

In this section, you will learn the requirements of using subqueries. We have the following two tables 'student' and 'marks' with common field 'StudentID'.

  
           
            student                                         marks

1.Now we want to write a query to identify all students who get better marks than that of the student who's StudentID is 'V002', but we do not know the marks of 'V002'.

2.To solve the problem, we require two queries. One query returns the marks (stored in Total\_marks field) of 'V002' and a second query identifies the students who get better marks than the result of the

|  |  |
| --- | --- |
| **First query:**  SELECT \*  FROM `marks`  WHERE studentid = 'V002'; | **Query result:**  student query  The result of the query is 80. |

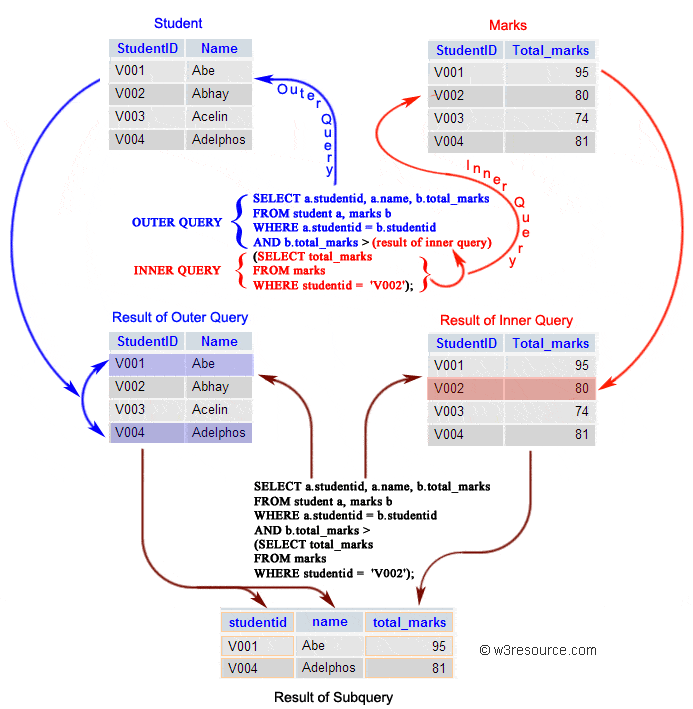
- Using the result of this query, here we have written another query to identify the students who get better marks than 80. Here is the query :

|  |  |
| --- | --- |
| **Second query:**  SELECT a.studentid, a.name, b.total\_marks  FROM student a, marks b  WHERE a.studentid = b.studentid  AND b.total\_marks >80; | **Query result:**  student marks query |

Above two queries identified students who get the better number than the student who's StudentID is 'V002' (Abhay).

You can combine the above two queries by placing one query inside the other. The subquery (also called the 'inner query') is the query inside the parentheses. See the following code and query result :

|  |  |
| --- | --- |
| **Sub queries**  SELECT a.studentid, a.name, b.total\_marks  FROM student a, marks b  WHERE a.studentid = b.studentid AND b.total\_marks >  (SELECT total\_marks  FROM marks  WHERE studentid = 'V002'); | **Query result:**  student marks query |

[](https://www.w3resource.com/sql/subqueries/sql-subqueries.gif)

+------------+----------------------+--------------------+------------+-----------------+---------+

| AGENT\_CODE | AGENT\_NAME | WORKING\_AREA | COMMISSION | PHONE\_NO | COUNTRY |

+------------+----------------------+--------------------+------------+-----------------+---------+

| A007 | Ramasundar | Bangalore | 0.15 | 077-25814763 | |

| A003 | Alex | London | 0.13 | 075-12458969 | |

| A008 | Alford | New York | 0.12 | 044-25874365 | |

| A011 | Ravi Kumar | Bangalore | 0.15 | 077-45625874 | |

| A010 | Santakumar | Chennai | 0.14 | 007-22388644 | |

| A012 | Lucida | San Jose | 0.12 | 044-52981425 | |

| A005 | Anderson | Brisban | 0.13 | 045-21447739 | |

| A001 | Subbarao | Bangalore | 0.14 | 077-12346674 | |

| A002 | Mukesh | Mumbai | 0.11 | 029-12358964 | |

| A006 | McDen | London | 0.15 | 078-22255588 | |

| A004 | Ivan | Torento | 0.15 | 008-22544166 | |

| A009 | Benjamin | Hampshair | 0.11 | 008-22536178 | |

+------------+----------------------+--------------------+------------+-----------------+---------+



ORD\_NUM ORD\_AMOUNT ADVANCE\_AMOUNT ORD\_DATE CUST\_CODE AGENT\_CODE

---------- ---------- -------------- --------- --------------- --------------- -----------------

200114 3500 2000 15-AUG-08 C00002 A008

200122 2500 400 16-SEP-08 C00003 A004

200118 500 100 20-JUL-08 C00023 A006

200119 4000 700 16-SEP-08 C00007 A010

200121 1500 600 23-SEP-08 C00008 A004

200130 2500 400 30-JUL-08 C00025 A011

200134 4200 1800 25-SEP-08 C00004 A005

200108 4000 600 15-FEB-08 C00008 A004

200103 1500 700 15-MAY-08 C00021 A005

200105 2500 500 18-JUL-08 C00025 A011

200109 3500 800 30-JUL-08 C00011 A010

200101 3000 1000 15-JUL-08 C00001 A008

200111 1000 300 10-JUL-08 C00020 A008

200104 1500 500 13-MAR-08 C00006 A004

200106 2500 700 20-APR-08 C00005 A002

200125 2000 600 10-OCT-08 C00018 A005

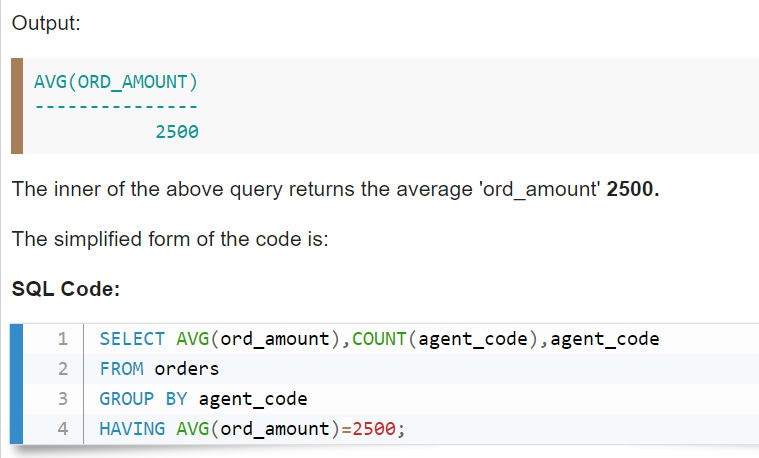
200117 800 200 20-OCT-08 C00014 A001

200102 2000 300 25-MAY-08 C00012 A012

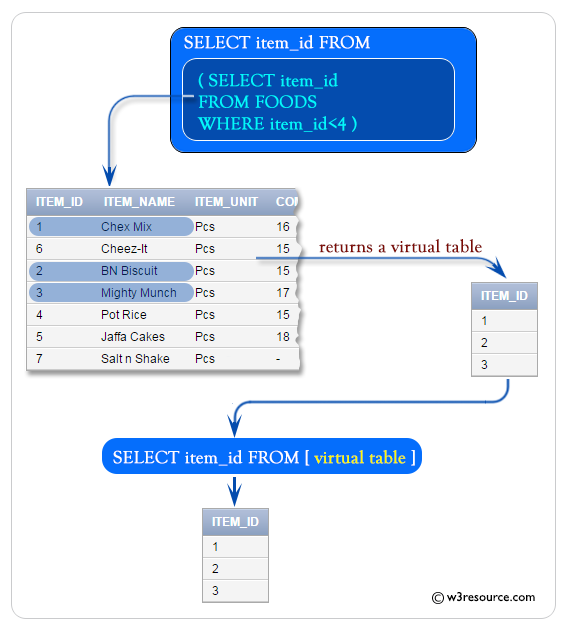


**Using having**

****



**USING FROM clause**



## Subquery using FROM clause

SELECT MAX(rating) FROM (SELECT \* FROM movie WHERE country = ‘India’) as temp;

## Subquery using SELECT

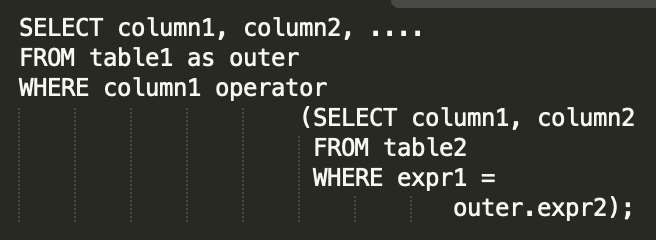
SELECT (SELECT column\_list(s) FROM T\_name WHERE condition), columnList(s) FROM T2\_name WHERE condition;

## Derived Subquery

SELECT columnLists(s) FROM (SELECT columnLists(s) FROM table\_name WHERE [condition]) as new\_table\_name;

## Co-related sub-queries

INNER QUERY REFER OUTER QUERY

With a normal nested subquery, the inner SELECT query runs first and executes once, returning values to be used by the main query. A correlated subquery, however, executes once for each candidate row considered by the outer query. In other words, the inner query is driven by the outer query.

# JOIN VS SUB-QUERIES

|  |  |
| --- | --- |
| JOINS | SUBQUERIES |
| Faster | Slower |
| Joins maximise calculation burden on DBMS | Keeps responsibility of calculation on user. |
| Complex, diﬃcult to understand and implemenT | t Comparatively easy to understand and implemENT |
| Choosing optimal join for optimal use case is diﬃcult | Easy. |

# MySQL VIEWS

1. A view is a database object that has no values. Its contents are based on the base table. It contains rows and columns similar to the real table.
2. In MySQL, the View is a **virtual table** created by a query by joining one or more tables. It is operated similarly to the base table but does not contain any data of its own.
3. The View and table have one main difference that the views are definitions built on top of other tables (or views). If any

changes occur in the underlying table, the same changes reflected in the View also.

1. CREATE VIEW view\_name AS SELECT columns FROM tables [WHERE conditions];
2. ALTER VIEW view\_name AS SELECT columns FROM table WHERE conditions;
3. DROP VIEW IF EXISTS view\_name;
4. CREATE VIEW Trainer AS SELECT c.course\_name, c.trainer, t.email FROM courses c, contact t WHERE c.id = t.id; (View using Join clause).

NOTE: We can also import/export table schema from files (.csv or json).

# Backup of any table

CREATE TABLE NEW\_TABLE\_NAME SELECT \* FROM OLD\_TABLE\_NAME