**The 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);

**Customers Table:**

**create table Customers(CustomerID INT NOT NULL AUTO\_INCREMENT, Name VARCHAR(80) NOT NULL, PhoneNumber VARCHAR(20) NOT NULL, Address TINYBLOB NOT NULL, PostalCode INT NOT NULL, COUNTRY VARCHAR(20) NOT NULL, PRIMARY KEY(CustomerID));**

Let’s suppose, this is how, the customer table is created.

The following insertion queries are done:

**insert into tables(CustomerID, Name,PhoneNumber, Address, PostalCode, COUNTRY) values(1, "Alfreds Futterkiste", "492180185611", "Obere Str. 57, Berlin:-12209", 12209,"India");**

**insert into Customers(CustomerID, Name,PhoneNumber, Address, PostalCode, COUNTRY) values(2, "Sayak Haldar", "9674465435", " J2,102/B, DDA Flats, Kalkaji, Delhi-110019 ", 110019,"India");**

**insert into Customers(CustomerID, Name,PhoneNumber, Address, PostalCode, COUNTRY) values(3,"Sayantan Pandit", "8697359734", "Andul Mouri", 711302,"India");**

Now, the table looks like the following:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CustomerID** | **Name** | **Phone Number** | **Address** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | 492180185611 | Obere Str. 57, Berlin:-12209 | 12209 | Germany |
| 2 | Sayak Haldar | 9674465435 | J2,102/B, DDA Flats, Kalkaji, Delhi-110019 | 110019 | India |
| 3 | Sayantan  Pandit | 8697359734 | Andul Mouri | 711302 | India |

Now, SELECT COUNT(CustomerID), Country FROM Customers Group by Country;

This results as the following:

|  |  |
| --- | --- |
| **COUNT(CustomerID)** | **Country** |
| 1 | Germany |
| 2 | India |

Now,

**Another Group By Clause Query:**

Select COUNT(CUSTOMERID) As CountryPerHeadCount, Country From Customers Group By Country Order by CountryPerHeadCount DESC;

**Join Query With Group By Clause:**

create table Orders(OrderID INT NOT NULL AUTO\_INCREMENT, CustomerID INT NOT NULL, EmployeeID INT NOT NULL, OrderDate DATE NOT NULL, ShipperID INT NOT NULL, PRIMARY KEY(OrderID));

And,

create table ShipperInformation(ShipperID INT NOT NULL AUTO\_INCREMENT PRIMARY KEY, ShipperName VARCHAR(80) NOT NULL);

Now, some data is inserted into both tables:

**insert into Orders(OrderID, CustomerID, EmployeeID, OrderDate, ShipperID) Values(1,1,1,"2017-05-20",1);**

**insert into Orders(OrderID, CustomerID, EmployeeID, OrderDate, ShipperID) Values(2,1,1,"2017-05-21",2);**

**insert into Orders(OrderID, CustomerID, EmployeeID, OrderDate, ShipperID) Values(3,2,2,"2017-05-22",3);**

In Orders table.

**insert into ShipperInformation(ShipperID,ShipperName)VALUES(1,"Fedex Express");**

**insert into ShipperInformation(ShipperID,ShipperName)VALUES(2,"ECARTL");**

**insert into ShipperInformation(ShipperID,ShipperName)VALUES(3,"GigaExpress");**

In **ShipperInformation** table.

And finally join query with group by clause:

**SELECT ShipperInformation.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders LEFT JOIN ShipperInformation ON Orders.ShipperID = ShipperInformation.ShipperID GROUP BY ShipperName;**

It would generate the following result:

|  |  |
| --- | --- |
| **ShipperName** | **NumberOfOrders** |
| ECARTL | 1 |
| Fedex Express | 1 |
| GigaExpress | 1 |

**Having Clause:**

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

**Generic Syntax:**

SELECT column\_name(s) FROM table\_name WHERE condition

GROUP BY column\_name(s) HAVING condition ORDER BY column\_name(s);

Consider the following example:

Suppose, a table named Customers has the following table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **NULL** | **Key** | **Default** | **Extra** |
| CustomerID | INT(11) | NO | PRI | NULL | AUTO\_  INCREMENT |
| Name | VARCHAR(80) | NO |  | NULL |  |
| PhoneNumber | VARCHAR(20) | NO |  | NULL |  |
| Address | TINYBLOB | NO |  | NULL |  |
| PostalCode | INT | NO |  | NULL |  |
| Country | VARCHAR(20) | NO |  | NULL |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Customer ID** | **Name** | **Phone Number** | **Address** | **Postal Code** | **Country** |
| 1 | Alfreds Futterkiste | 492180185611 | Obere Str. 57, Berlin:-12209 | 12209 | Germany |
| 2 | Sayak Haldar | 974465435 | J2, 102/B, DDA Flats, Kalkaji | 110019 | India |
| 3 | Sayantan Pandit | 8697359734 | Andul Mouri | 711302 | India |
| 4 | Suman Banerjee | 9231837241 | Andul PurboPara, Andul Mouri, Howrah | 711302 | India |
| 5 | Saptarshi Nag | 8013362856 | Bandra, West Mumbai | 400050 | India |
| 6 | Ankur Debnath | 9734991000 | J2, 102/B, DDA Flats, Kalkaji | 110019 | India |
| 7 | Arinjoy  Basak | 5408381704 | Blacksburg, VA 24061, USA | 0 | USA |

**There is one big flaw in choosing INT datatype for postal code. Some, postal code, starts with 0.**

However, let’s complete the example:

Now, A simple HeadCountPerCountry would require the query:

select Count(CustomerID) as HeadCountPerCountry, Country from Customers Group By Country;

|  |  |
| --- | --- |
| **HeadCountPerCountry** | **Country** |
| 1 | Germany |
| 5 | India |
| 1 | USA |

**Now, suppose, it is required to display the countries in which we have high customerBase (Suppose, the criteria is where HeadCountPerCountry >=5):**

For that, the following query would be required:

**select Count(CustomerID) as HeadCountPerCountry, Country from Customers Group By Country Having HeadCountPerCountry>=5;**

This would display the following result:

|  |  |
| --- | --- |
| **HeadCountPerCountry** | **Country** |
| 5 | India |

However, here only one result So, Order By won’t display proper result:

Though the query is given:

**select Count(CustomerID) as HeadCountPerCountry, Country from Customers Group By Country Having HeadCountPerCountry>=5 Order By HeadCountPerCountry;**

**MySql Any And All Operator:**

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

The ANY operator returns true if any of the sub-query values meet the condition.

The ALL operator returns true if all of the sub-query 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);

**All Syntax:**

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

**Note:** The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Product**  **ID** | **Product Name** | **Supplier**  **ID** | **Category ID** | **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 |

|  |  |  |  |
| --- | --- | --- | --- |
| **OrderDetailID** | **OrderID** | **ProductID** | **Quantity** |
| 1 | 10248 | 11 | 12 |
| 2 | 10248 | 42 | 10 |
| 3 | 10248 | 72 | 5 |
| 4 | 10249 | 14 | 9 |
| 5 | 10249 | 51 | 40 |

## SQL ANY Examples:

The ANY operator returns TRUE if any of the subquery values meet the condition.

The following SQL statement returns TRUE and lists the productnames if it finds ANY records in the OrderDetails table that quantity = 10;

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

**SQL ALL Examples:**

The ALL operator returns TRUE if all of the subquery values meet the condition.

The following SQL statement returns TRUE and lists the productnames if ALL the records in the OrderDetails table has quantity = 10;

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

**Another Fine Example of SQL ANY Example:**

Suppose, you have to make a query to display employees who earn more than the lowest salary from a Employee of DNO=3:

**Select ENAME, SAL, DNO from EMP WHERE SAL>ANY(SELECT DISTINCT SAL FROM EMP WHERE DNO=3);**

How this will work?

Now, the inner query **SELECT DISTINCT SAL FROM EMP WHERE DNO=30**

will fetch all distinct salary.

|  |  |  |  |
| --- | --- | --- | --- |
| **EID** | **ENAME** | **SAL** | **DNO** |
| 1 | Peter | 3,00,00,000 | 1 |
| 2 | Sayan | 1,00,00,000 | 3 |
| 3 | Suman | 1,20,00,000 | 3 |
| 4 | Sayantan | 1,10,00,000 | 3 |
| 5 | Kenedy | 4,50,00,000 | 1 |
| 6 | Albert | 2,00,00,000 | 2 |
| 7 | Parisa | 3,20,00,000 | 2 |
| 8 | Hilary | 80,00,000 | 3 |

Now, inner query would fetch results in a set containing (12000000,10000000,11000000,8000000) Now, every row of the table will be compared against the elements of the inner query result set for generating outer query result set.

Now, Consider 1st row. Peter’s salary is 3,00,00,000. If it is greater than any of the mentioned salaries (12000000,10000000, 11000000,8000000) it will be listed as outer query result set.

Now, you probably understand that how this query will work.

**Now, A Fine Example of All Query:**

Suppose, you have to make a query to display employees who earn more than any Employee of DNO=3:

**Select ENAME, SAL, DNO from EMP WHERE SAL>ALL(SELECT DISTINCT SAL FROM EMP WHERE DNO=3);**