# GROUP BY

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

* GROUP BY clause is used with the SELECT statement.
* In the query, GROUP BY clause is placed after the WHERE clause.
* In the query, GROUP BY clause is placed before ORDER BY clause if used any.

**Syntax**:

SELECT column1, function\_name(column2)

FROM table\_name

WHERE condition

GROUP BY column1, column2

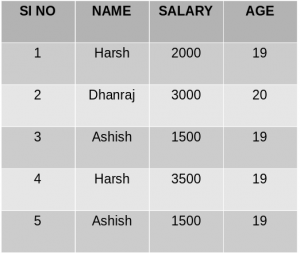
ORDER BY column1, column2;

**function\_name**: Name of the function used for example, SUM() , AVG().

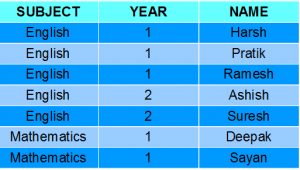
**table\_name**: Name of the table.

**condition**: Condition used.

**Sample Table:**



Student Table



**Example:**

* **Group By single column**: Group By single column means, to place all the rows with same value of only that particular column in one group. Consider the query as shown below:
* SELECT NAME, SUM(SALARY) FROM Employee
* GROUP BY NAME;

The above query will produce the below output:  
[](https://media.geeksforgeeks.org/wp-content/uploads/table_out.png)

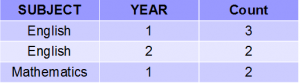
* As you can see in the above output, the rows with duplicate NAMEs are grouped under same NAME and their corresponding SALARY is the sum of the SALARY of duplicate rows. The SUM() function of SQL is used here to calculate the sum.

**Group By multiple columns**: Group by multiple column is say for example, **GROUP BY column1, column2**. This means to place all the rows with same values of both the columns **column1** and **column2** in one group. Consider the below query:

* SELECT SUBJECT, YEAR, Count(\*)
* FROM Student

GROUP BY SUBJECT, YEAR;

**Output**:



**HAVING Clause**

**Syntax**:

SELECT column1, function\_name(column2)

FROM table\_name

WHERE condition

GROUP BY column1, column2

HAVING condition

ORDER BY column1, column2;

**function\_name**: Name of the function used for example, SUM() , AVG().

**table\_name**: Name of the table.

**condition**: Condition used.

**Example**:

SELECT NAME, SUM(SALARY) FROM Employee

GROUP BY NAME

HAVING SUM(SALARY)>3000;

**Output**:



**Cross Join**

When each row of first table is combined with each row from the second table, known as Cartesian join or cross join. In general words we can say that SQL CROSS JOIN returns the Cartesian product of the sets of rows from the joined table.

**We can specify a CROSS JOIN in two ways:**

1. Using the JOIN syntax.
2. the table in the FROM clause without using a WHERE clause.

**SYNTAX of SQL Cross Join**

**SELECT** \* **FROM** [TABLE1] CROSS JOIN [TABLE2]

OR

**SELECT** \* **FROM** [ TABLE1] , [TABLE2]

Let us take an example of two tables,

|  |  |  |
| --- | --- | --- |
| **Player** | **Department\_id** | **Goals** |
| Franklin | 1 | 2 |
| Alan | 1 | 3 |
| Priyanka | 2 | 2 |
| Rajesh | 3 | 5 |

**Table1 - MatchScore**

**Table2 - Departments**

|  |  |
| --- | --- |
| **Department\_id** | **Department\_name** |
| 1 | IT |
| 2 | HR |
| 3 | Marketing |

**SQL Statement:**

**SELECT** \* **FROM** MatchScore CROSS JOIN Departments

After executing this query , you will find the following result:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Player** | **Department\_id** | **Goals** | **Depatment\_id** | **Department\_name** |
| Franklin | 1 | 2 | 1 | IT |
| Alan | 1 | 3 | 1 | IT |
| Priyanka | 2 | 2 | 1 | IT |
| Rajesh | 3 | 5 | 1 | IT |
| Franklin | 1 | 2 | 2 | HR |
| Alan | 1 | 3 | 2 | HR |
| Priyanka | 2 | 2 | 2 | HR |
| Rajesh | 3 | 5 | 2 | HR |
| Franklin | 1 | 2 | 3 | Marketing |
| Alan | 1 | 3 | 3 | Marketing |
| Priyanka | 2 | 2 | 3 | Marketing |
| Rajesh | 3 | 5 | 3 | Marketing |

# 

**Student**

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/table1-3.png)

**StudentCourse**

[](https://media.geeksforgeeks.org/wp-content/uploads/table5.png)

**INNER JOIN:**

 The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be same.  
**Syntax**:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

INNER JOIN table2

ON table1.matching\_column = table2.matching\_column;

**table1**: First table.

**table2**: Second table

**matching\_column**: Column common to both the tables.

We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.

This query will show the names and age of students enrolled in different courses.

SELECT StudentCourse.COURSE\_ID, Student.NAME, Student.AGE FROM Student

INNER JOIN StudentCourse

ON Student.ROLL\_NO = StudentCourse.ROLL\_NO;

**Output**:  
[](https://media.geeksforgeeks.org/wp-content/uploads/table22.png)

**LEFT JOIN**:

This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN.**Syntax:**

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

LEFT JOIN table2

ON table1.matching\_column = table2.matching\_column;

table1: First table.

table2: Second table

matching\_column: Column common to both the tables.

We can also use LEFT OUTER JOIN instead of LEFT JOIN, both are same.

SELECT Student.NAME,StudentCourse.COURSE\_ID

FROM Student

LEFT JOIN StudentCourse

ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**Output**:  
[](https://media.geeksforgeeks.org/wp-content/uploads/table31.png)

**RIGHT JOIN:**

RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on left side, the result-set will contain *null*. RIGHT JOIN is also known as RIGHT OUTER JOIN.

**Syntax:**

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

RIGHT JOIN table2

ON table1.matching\_column = table2.matching\_column;

table1: First table.

table2: Second table

matching\_column: Column common to both the tables.

 We can also use RIGHT OUTER JOIN instead of RIGHT JOIN, both are same.

SELECT Student.NAME,StudentCourse.COURSE\_ID

FROM Student

RIGHT JOIN StudentCourse

ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**Output:**  
[](https://media.geeksforgeeks.org/wp-content/uploads/table6.png)

**FULL JOIN:**

 FULL JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain *NULL* values.

**Syntax:**

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

FULL JOIN table2

ON table1.matching\_column = table2.matching\_column;

table1: First table.

table2: Second table

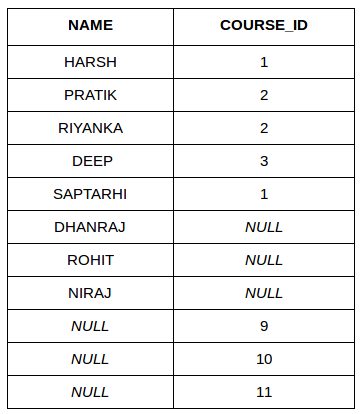
matching\_column: Column common to both the tables.

SELECT Student.NAME,StudentCourse.COURSE\_ID

FROM Student

FULL JOIN StudentCourse

ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**Output:**  
[](https://media.geeksforgeeks.org/wp-content/uploads/table7.png)

# Sub Queries

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

## **Subqueries with the SELECT Statement**

Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows −

SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

WHERE column\_name OPERATOR

(SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

[WHERE])

### **Example**

Consider the CUSTOMERS table having the following records −

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

| ID | NAME | AGE | ADDRESS | SALARY |

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

| 1 | Ramesh | 35 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

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

Now, let us check the following subquery with a SELECT statement.

SQL> SELECT \*

FROM CUSTOMERS

WHERE ID IN (SELECT ID

FROM CUSTOMERS

WHERE SALARY > 4500) ;

This would produce the following result.

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

| ID | NAME | AGE | ADDRESS | SALARY |

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

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

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

## **Subqueries with the INSERT Statement**

Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery can be modified with any of the character, date or number functions.

The basic syntax is as follows.

INSERT INTO table\_name [ (column1 [, column2 ]) ]

SELECT [ \*|column1 [, column2 ]

FROM table1 [, table2 ]

[ WHERE VALUE OPERATOR ]

### **Example**

Consider a table CUSTOMERS\_BKP with similar structure as CUSTOMERS table. Now to copy the complete CUSTOMERS table into the CUSTOMERS\_BKP table, you can use the following syntax.

SQL> INSERT INTO CUSTOMERS\_BKP

SELECT \* FROM CUSTOMERS

WHERE ID IN (SELECT ID

FROM CUSTOMERS) ;

## **Subqueries with the UPDATE Statement**

The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

The basic syntax is as follows.

UPDATE table

SET column\_name = new\_value

[ WHERE OPERATOR [ VALUE ]

(SELECT COLUMN\_NAME

FROM TABLE\_NAME)

[ WHERE) ]

### **Example**

Assuming, we have CUSTOMERS\_BKP table available which is backup of CUSTOMERS table. The following example updates SALARY by 0.25 times in the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

SQL> UPDATE CUSTOMERS

SET SALARY = SALARY \* 0.25

WHERE AGE IN (SELECT AGE FROM CUSTOMERS\_BKP

WHERE AGE >= 27 );

This would impact two rows and finally CUSTOMERS table would have the following records.

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

| ID | NAME | AGE | ADDRESS | SALARY |

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

| 1 | Ramesh | 35 | Ahmedabad | 125.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 2125.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

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

## **Subqueries with the DELETE Statement**

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

The basic syntax is as follows.

DELETE FROM TABLE\_NAME

[ WHERE OPERATOR [ VALUE ]

(SELECT COLUMN\_NAME

FROM TABLE\_NAME)

[ WHERE) ]

### **Example**

Assuming, we have a CUSTOMERS\_BKP table available which is a backup of the CUSTOMERS table. The following example deletes the records from the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

SQL> DELETE FROM CUSTOMERS

WHERE AGE IN (SELECT AGE FROM CUSTOMERS\_BKP

WHERE AGE >= 27 );

This would impact two rows and finally the CUSTOMERS table would have the following records.

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

| ID | NAME | AGE | ADDRESS | SALARY |

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

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

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

**Corelated queries**

Sample table: orders, agents

**SQL Code:**

SELECT a.ord\_num,a.ord\_amount,a.cust\_code,a.agent\_code

FROM orders a

WHERE a.agent\_code=(

SELECT b.agent\_code

FROM agents b WHERE b.agent\_name='Alex');

**Output:**

ORD\_NUM ORD\_AMOUNT CUST\_CODE AGENT\_CODE

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

200127 2500 C00015 A003

200100 1000 C00015 A003

The inner of the above query returns the 'agent\_code' A003.

The simplified form of above code is:

**SQL Code:**

SELECT a.ord\_num,a.ord\_amount,a.cust\_code,a.agent\_code

FROM orders a

WHERE a.agent\_code='A003';

**Using EXISTS with a correlated subquery**

We have already used the EXISTS operator to check the existence of a result of a subquery. EXISTS operator can be used in correlated subqueries also. Using EXISTS the following query display the employee\_id, manager\_id, first\_name and last\_name of those employees who manage other employees.

Sample table: employees

**SQL Code:**

SELECT employee\_id, manager\_id, first\_name, last\_name

FROM employees a

WHERE EXISTS

(SELECT employee\_id

FROM employees b

WHERE b.manager\_id = a.employee\_id)

**Output:**

EMPLOYEE\_ID MANAGER\_ID FIRST\_NAME LAST\_NAME

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

100 Steven King

101 100 Neena Kochhar

102 100 Lex De Haan

103 102 Alexander Hunold

108 101 Nancy Greenberg

114 100 Den Raphaely

120 100 Matthew Weiss

121 100 Adam Fripp

122 100 Payam Kaufling

123 100 Shanta Vollman

124 100 Kevin Mourgos

145 100 John Russell

146 100 Karen Partners

147 100 Alberto Errazuriz

148 100 Gerald Cambrault

149 100 Eleni Zlotkey

201 100 Michael Hartstein

205 101 Shelley Higgins

## **Using NOT EXISTS with a correlated subquery**

## NOT EXISTS is logically opposite of EXISTS operator. NOT EXISTS is used when we need to check if rows do not exist in the results returned by a subquery. Using NOT EXISTS the following query display the employee\_id, manager\_id, first\_name and last\_name of those employees who have no manager status. This query is opposite to the previous one.

## **SQL Code:**

SELECT employee\_id, manager\_id, first\_name, last\_name

FROM employees a

WHERE NOT EXISTS

(SELECT employee\_id

FROM employees b

WHERE b.manager\_id = a.employee\_id);

**Sample table: employees**

**Output:**

EMPLOYEE\_ID MANAGER\_ID FIRST\_NAME LAST\_NAME

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

104 103 Bruce Ernst

105 103 David Austin

106 103 Valli Pataballa

107 103 Diana Lorentz

109 108 Daniel Faviet

110 108 John Chen

111 108 Ismael Sciarra

112 108 Jose Manuel Urman

113 108 Luis Popp

115 114 Alexander Khoo

116 114 Shelli Baida

117 114 Sigal Tobias

118 114 Guy Himuro

119 114 Karen Colmenares

125 120 Julia Nayer

126 120 Irene Mikkilineni

127 120 James Landry

128 120 Steven Markle

129 121 Laura Bissot

130 121 Mozhe Atkinson

131 121 James Marlow

........

.......

**UNION:**

The Union Clause is used to combine two separate select statements and produce the result set as a union of both the select statements.  
**NOTE:**

1. The fields to be used in both the select statements must be in same order, same number and same data type.
2. The Union clause produces distinct values in the result set, to fetch the duplicate values too UNION ALL must be used instead of just UNION.

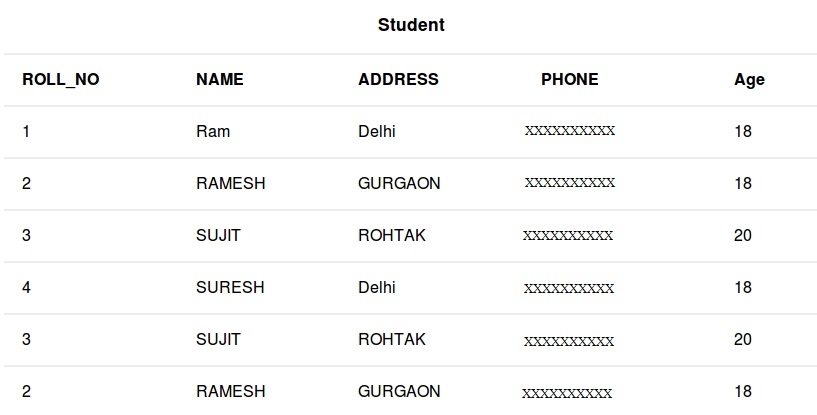
Syntax:

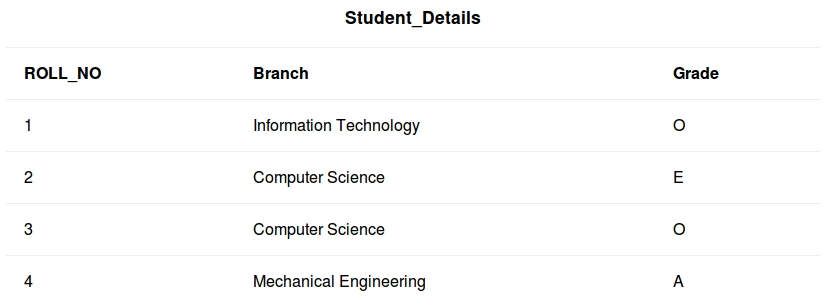
SELECT column\_name(s) FROM table1 UNION SELECT column\_name(s) FROM table2;

Resultant set consists of distinct values.

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

Resultant set consists of duplicate values too.

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/table11.jpg)



To fetch distinct ROLL\_NO from Student and Student\_Details table.

SELECT ROLL\_NO FROM Student UNION SELECT ROLL\_NO FROM Student\_Details;

Output:

| **ROLL\_NO** |
| --- |
| 1 |
| 2 |
| 3 |
| 4 |

To fetch ROLL\_NO , NAME from Student table WHERE ROLL\_NO is greater than 3 and ROLL\_NO , Branch from Student\_Details table WHERE ROLL\_NO is less than 3 , including duplicate values and finally sorting the data by ROLL\_NO.

SELECT ROLL\_NO,NAME FROM Student WHERE ROLL\_NO>3

UNION ALL

SELECT ROLL\_NO,Branch FROM Student\_Details WHERE ROLL\_NO<3

ORDER BY 1;

Output:

| **ROLL\_NO** | **NAME** |
| --- | --- |
| 1 | Information Technology |
| 2 | Computer Science |
| 4 | SURESH |