SQL | WHERE Clause

WHERE keyword is used for fetching **filtered data** in a result set. It is used to fetch data according to particular criteria. **WHERE** keyword can also be used to filter data by matching patterns.

Syntax:

*SELECT **column1,column2** FROM table_name WHERE column_name operator value;*

Parameter Explanation:

1. **column1,column2:** fields in the table
2. **table_name:** name of table
3. **column_name:** name of field used for filtering the data
4. **operator:** operation to be considered for filtering
5. **value:** exact value or pattern to get related data in result

List of Operators that Can be Used with WHERE Clause

Operator	Description
>	Greater Than
>=	Greater than or Equal to

Operator	Description
<	Less Than
<=	Less than or Equal to
=	Equal to
<>	Not Equal to
BETWEEN	In an inclusive Range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Query:

```
CREATE TABLE Emp1(
    EmpID INT PRIMARY KEY,
    Name VARCHAR(50),
    Country VARCHAR(50),
    Age int(2),
  mob int(10)
);
-- Insert some sample data into the Customers table
INSERT INTO Emp1 (EmpID, Name,Country, Age, mob)
VALUES (1, 'Shubham',  'India','23','738479734'),
       (2, 'Aman ',  'Australia','21','436789555'),
       (3, 'Naveen', 'Sri lanka','24','34873847'),
       (4, 'Aditya',  'Austria','21','328440934'),
       (5, 'Nishant', 'Spain','22','73248679');
       Select * from Emp1;
```

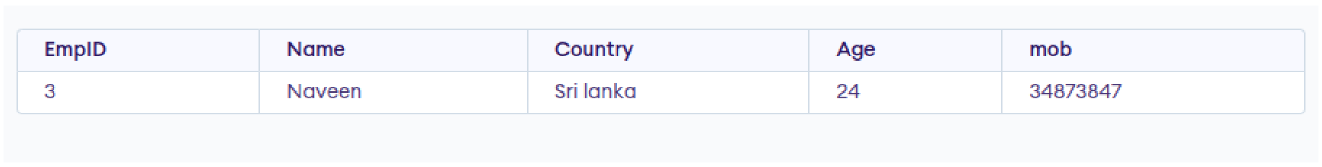
Where Clause with Logical Operators

To fetch records of Employee with ages equal to 24.

Query:

```
SELECT * FROM Emp1 WHERE Age=24;
```

Output:



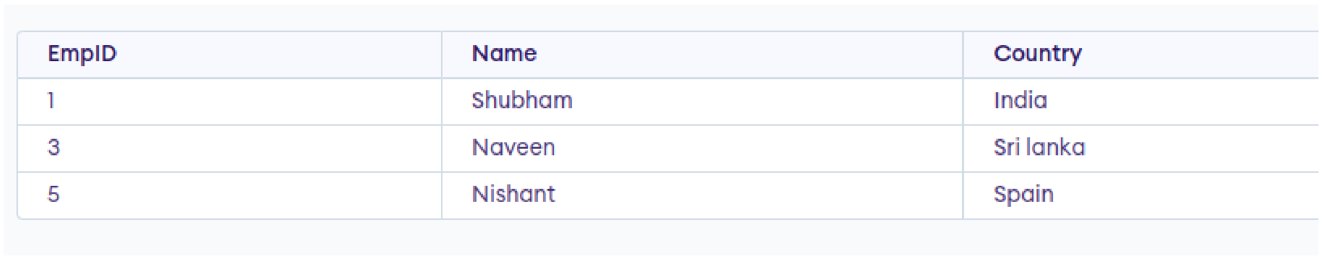
EmpID	Name	Country	Age	mob
3	Naveen	Sri lanka	24	34873847

To fetch the EmpID, Name and Country of Employees with Age greater than 21.

Query:

```
SELECT EmpID, Name, Country FROM Emp1 WHERE Age > 21;
```

Output:



EmpID	Name	Country
1	Shubham	India
3	Naveen	Sri lanka
5	Nishant	Spain

Where Clause with BETWEEN Operator

It is used to fetch filtered data in a given range inclusive of two values.

Syntax:

SELECT column1,column2 FROM table_name

WHERE column_name BETWEEN value1 AND value2;

Parameter Explanation:

1. **BETWEEN:** operator name
2. **value1 AND value2:** exact value from value1 to value2 to get related data in result set.

To fetch records of Employees where Age is between 22 and 24 (inclusive).

Query:

```
SELECT * FROM Emp1 WHERE Age BETWEEN 22 AND 24;
```

Output:

EmpID	Name	Country	Age	mob
1	Shubham	India	23	738479734
3	Naveen	Sri lanka	24	34873847
5	Nishant	Spain	22	73248679

Where Clause with LIKE Operator

It is used to fetch filtered data by searching for a particular pattern in the where clause.

Syntax:

SELECT column1,column2 FROM

table_name WHERE column_name LIKE pattern;

Parameters Explanation:

1. **LIKE:** operator name
2. **pattern:** exact value extracted from the pattern to get related data in the result set.

Note: The character(s) in the pattern is case sensitive.

To fetch records of Employees where Name starts with the letter S.

Query:

```
SELECT * FROM Emp1 WHERE Name LIKE 'S%';
```

The '%'(wildcard) signifies the later characters here which can be of any length and value.

Output:

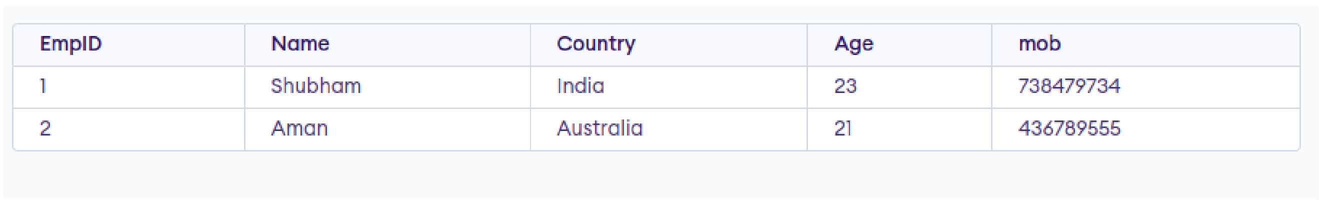
EmpID	Name	Country	Age	mob
1	Shubham	India	23	738479734

To fetch records of Employees where Name contains the pattern 'M'.

Query:

```
SELECT * FROM Emp1 WHERE Name LIKE '%M%';
```

Output:



EmpID	Name	Country	Age	mob
1	Shubham	India	23	738479734
2	Aman	Australia	21	436789555

Where Clause with IN Operator

It is used to fetch the filtered data same as fetched by '=' operator just the difference is that here we can specify multiple values for which we can get the result set.

Syntax:

SELECT column1,column2 FROM table_name WHERE column_name IN (value1,value2,..);

Parameters Explanation:

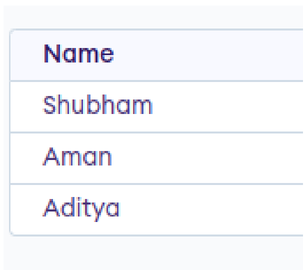
1. **IN:** operator name
2. **value1,value2,..:** exact value matching the values given and get related data in the result set.

To fetch the Names of Employees where Age is 21 or 23.

Query:

```
SELECT Name FROM Emp1 WHERE Age IN (21,23);
```

Output:



Name
Shubham
Aman
Aditya

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 07 May, 2023

SQL | USING Clause

akanshgupta

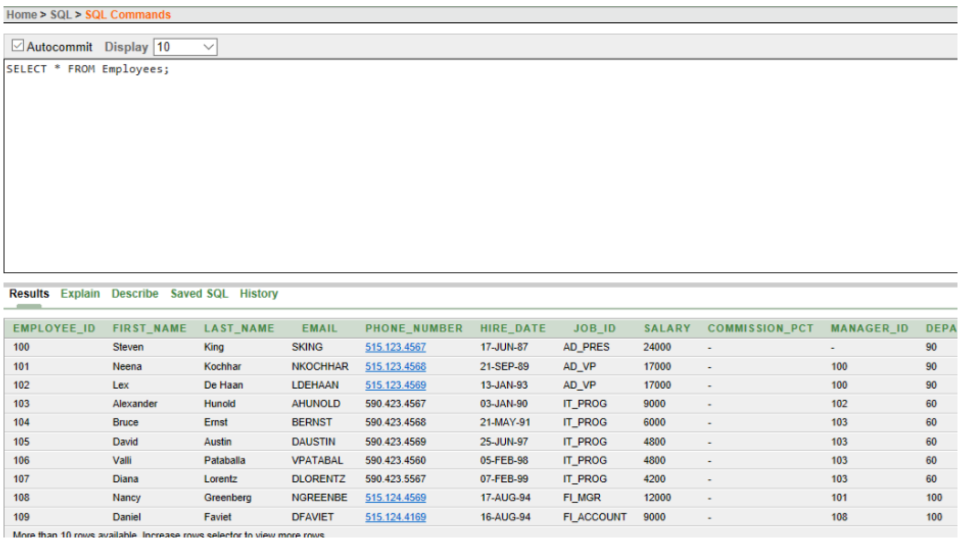
If several columns have the same names but the datatypes do not match, the NATURAL JOIN clause can be modified with the **USING** clause to specify the columns that should be used for an EQUIJOIN.

- USING Clause is used to match only one column when more than one column matches.
- NATURAL JOIN and USING Clause are mutually exclusive.
- It should not have a qualifier(table name or Alias) in the referenced columns.
- NATURAL JOIN uses all the columns with matching names and datatypes to join the tables. The USING Clause can be used to specify only those columns that should be used for an EQUIJOIN.

EXAMPLES:

We will apply the below mentioned commands on the following base tables:

AD


Home > SQL > SQL Commands

Autocommit Display 10

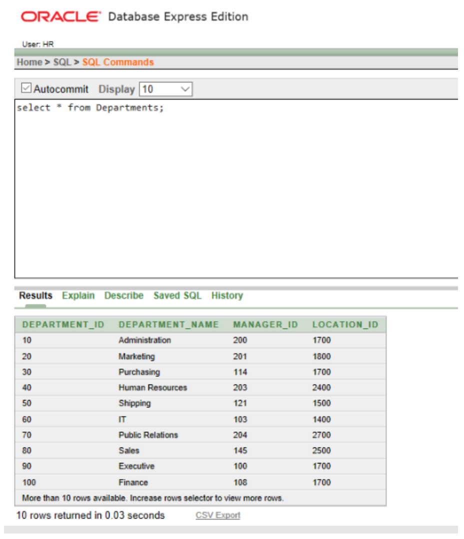
```
SELECT * FROM Employees;
```

Results Explain Describe Saved SQL History

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPA
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	-	-	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000	-	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000	-	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000	-	102	60
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000	-	103	60
105	David	Austin	DAUSTIN	590.423.4569	25-JUN-97	IT_PROG	4800	-	103	60
106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-98	IT_PROG	4800	-	103	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	4200	-	103	60
108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-94	FI_MGR	12000	-	101	100
109	Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-94	FI_ACCOUNT	9000	-	108	100

More than 10 rows available. Increase rows selector to view more rows.

Employee Table



Database Express Edition

User: HR

Home > SQL > SQL Commands

Autocommit Display 10

```
select * from Departments;
```

Results Explain Describe Saved SQL History

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700

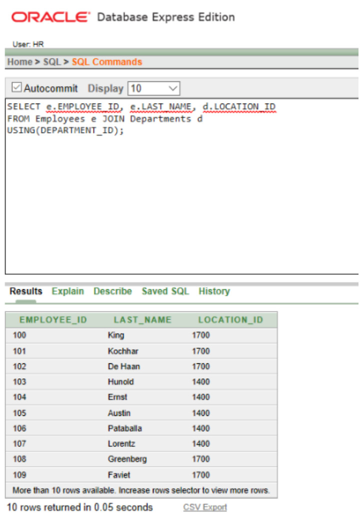
More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.03 seconds CSV Export

Department Table

QUERY 1: Write SQL query to find the working location of the employees. Also give their respective employee_id and last_name?

```
Input : SELECT e.EMPLOYEE_ID, e.LAST_NAME, d.LOCATION_ID
FROM Employees e JOIN Departments d
USING(DEPARTMENT_ID);
Output :
```



Database Express Edition

User: HR

Home > SQL > SQL Commands

Autocommit Display 10

```
SELECT e.EMPLOYEE_ID, e.LAST_NAME, d.LOCATION_ID
FROM Employees e JOIN Departments d
USING(DEPARTMENT_ID);
```

Results Explain Describe Saved SQL History

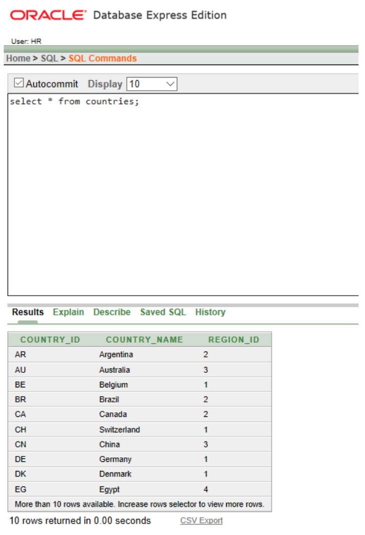
EMPLOYEE_ID	LAST_NAME	LOCATION_ID
100	King	1700
101	Kochhar	1700
102	De Haan	1700
103	Hunold	1400
104	Ernst	1400
105	Austin	1400
106	Pataballa	1400
107	Lorentz	1400
108	Greenberg	1700
109	Faviet	1700

More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.05 seconds CSV Export

Explanation: The example shown joins the DEPARTMENT_ID column in the EMPLOYEES and DEPARTMENTS
tables, and thus shows the location where an employee works.

We will apply the below mentioned commands on the following base tables:


ORACLE Database Express Edition
User: HR
Home > SQL > SQL Commands
Autocommit Display 10
Results Explain Describe Saved SQL History
CSV Export

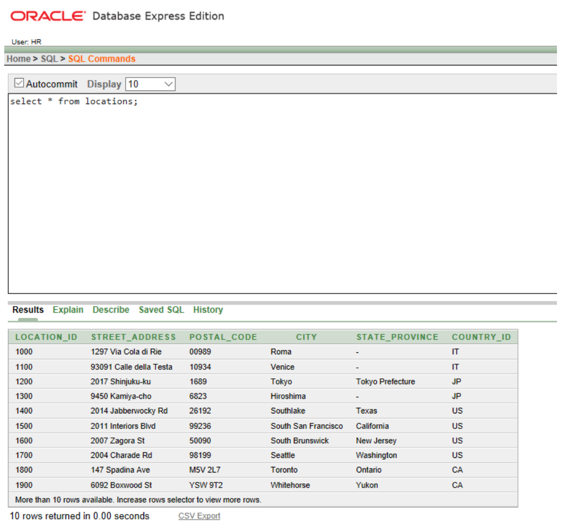
```
select * from countries;
```

COUNTRY_ID	COUNTRY_NAME	REGION_ID
AR	Argentina	2
AU	Australia	3
BE	Belgium	1
BR	Brazil	2
CA	Canada	2
CH	Switzerland	1
CN	China	3
DE	Germany	1
DK	Denmark	1
EG	Egypt	4

More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.00 seconds

Country Table


ORACLE Database Express Edition
User: HR
Home > SQL > SQL Commands
Autocommit Display 10
Results Explain Describe Saved SQL History
CSV Export

```
select * from locations;
```

LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	COUNTRY_ID
1000	1297 Via Cola di Rie	00989	Roma	-	IT
1100	93091 Calle della Testa	10934	Venice	-	IT
1200	2017 Shinjuku-ku	1689	Tokyo	Tokyo Prefecture	JP
1300	9450 Kamiya-cho	6823	Hiroshima	-	JP
1400	2014 Jabberwocky Rd	26192	Southlake	Texas	US
1500	2011 Interiors Blvd	99236	South San Francisco	California	US
1600	2007 Zagora St	50090	South Brunswick	New Jersey	US
1700	2004 Charade Rd	98199	Seattle	Washington	US
1800	147 Spadina Ave	M5V 2L7	Toronto	Ontario	CA
1900	6092 Boxwood St	YSW 9T2	Whitehorse	Yukon	CA

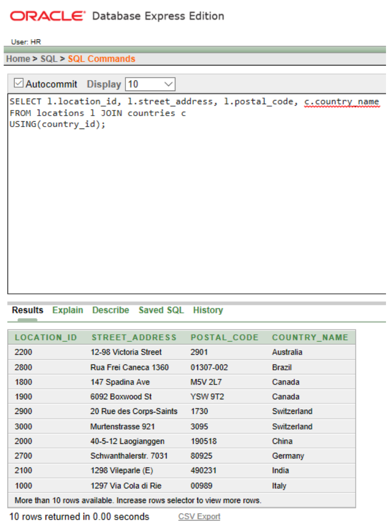
More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.00 seconds

Location Table

QUERY 2: Write SQL query to find the location_id, street_address, postal_code and their respective country name?

```
Input : SELECT l.location_id, l.street_address, l.postal_code, c.country_name
FROM locations l JOIN countries c
USING(country_id);
Output :
```


ORACLE Database Express Edition
User: HR
Home > SQL > SQL Commands
Autocommit Display 10

```
SELECT l.location_id, l.street_address, l.postal_code, c.country_name
FROM locations l JOIN countries c
USING(country_id);
```

Results Explain Describe Saved SQL History

LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	COUNTRY_NAME
2200	12-98 Victoria Street	2901	Australia
2800	Rua Frei Caneca 1360	01307-002	Brazil
1800	147 Spadina Ave	M5V 2L7	Canada
1900	6092 Boxwood St	YSW 9T2	Canada
2900	20 Rue des Corps-Saints	1730	Switzerland
3000	Murtenstrasse 921	3095	Switzerland
2000	40-5-12 Laogianggen	190518	China
2700	Schwanthalerstr. 7031	80925	Germany
2100	1298 Vileparle (E)	490231	India
1000	1297 Via Cola di Rie	00989	Italy

More than 10 rows available. Increase rows selector to view more rows.

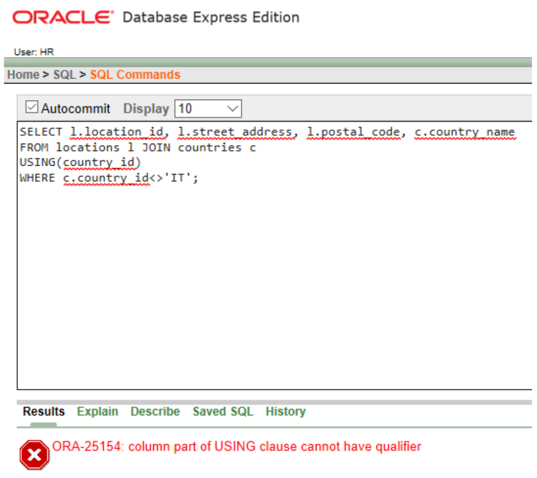
10 rows returned in 0.00 seconds CSV Export

Explanation: The example shown joins the COUNTRY_ID column in the LOCATIONS and COUNTRIES
tables, and thus shows the required details.

NOTE: When we use the USING clause in a join statement, the join column is not qualified with table Alias. Do not Alias it even if the same column is used elsewhere in the SQL statement:

Example:

```
Input: SELECT l.location_id, l.street_address, l.postal_code, c.country_name
FROM locations l JOIN countries c
USING(country_id)
WHERE c.country_id'IT';
Output:
```


ORACLE Database Express Edition
User: HR
Home > SQL > SQL Commands
Autocommit Display 10

```
SELECT l.location_id, l.street_address, l.postal_code, c.country_name
FROM locations l JOIN countries c
USING(country_id)
WHERE c.country_id<>'IT';
```

Results Explain Describe Saved SQL History

ORA-25154: column part of USING clause cannot have qualifier

Explanation: Since the column in USING Clause is used again in WHERE Clause, thus it throws an error to the user.

Last Updated : 21 Mar, 2018

14

Similar Reads

1. Difference between Having clause and Group by clause
2. Combining aggregate and non-aggregate values in SQL using Joins and Over clause
3. SQL Full Outer Join Using Left and Right Outer Join and Union Clause
4. SQL query using COUNT and HAVING clause
5. SQL Full Outer Join Using Union Clause
6. SQL Full Outer Join Using Where Clause
7. Using CASE in ORDER BY Clause to Sort Records By Lowest Value of 2 Columns in SQL
8. Configure SQL Jobs in SQL Server using T-SQL
9. SQL | Distinct Clause
10. SQL | WHERE Clause

Previous Next

Article Contributed By :

akanshgupta
akanshgupta

Vote for difficulty

Current difficulty : Easy

Easy Normal Medium Hard Expert

Article Tags : SQL-Clauses-Operators, SQL

Practice Tags : SQL

SQL | MERGE Statement

Prerequisite – INSERT, UPDATE, DELETE

The **MERGE** command in SQL is actually a combination of three SQL statements: **INSERT, UPDATE and DELETE**. In simple words, the MERGE statement in SQL provides a convenient way to perform all these three operations together which can be very helpful when it comes to handle the large running databases. But unlike INSERT, UPDATE and DELETE statements MERGE statement requires a source table to perform these operations on the required table which is called as target table.

Now we know that the MERGE in SQL requires two tables : one the target table on which we want to perform INSERT, UPDATE and DELETE operations, and the other one is source table which contains the new modified and correct data for target table and is actually compared with the actual target table in order to modify it.

In other words, the MERGE statement in SQL basically merges data from a source result set to a target table based on a condition that is specified. The syntax of MERGE statement can be complex to understand at first but its very easy once you know what it means.So,not to get confused first let's discuss some basics. Suppose you have two tables: source and target, now think if you want to make changes in the required target table with the help of provided source table which consists of latest details.

- When will you need to insert the data in the target table?
  Obviously when there is data in source table and not in target table *i.e* when data not matched with target table.
- When will you need to update the data?
  When the data in source table is matched with target table but any entry other than the primary key is not matched.

- When will you need to delete the data?
  When there is data in target table and not in source table *i.e* when data not matched with source table.

Now, we know when to use INSERT, UPDATE and DELETE statements in case we want to use MERGE statement so there should be no problem for you understanding the syntax given below :

```
//.....syntax of MERGE statement....//

//you can use any other name in place of target
MERGE target_table_name AS TARGET

//you can use any other name in place of source
USING source_table_name AS SOURCE
ON condition (for matching source and target table)
WHEN MATCHED (another condition for updation)

 //now use update statement syntax accordingly
THEN UPDATE
WHEN NOT MATCHED BY TARGET

//now use insert statement syntax accordingly
THEN INSERT
WHEN NOT MATCHED BY SOURCE
THEN DELETE;
```

That's all about the MERGE statement and its syntax.

References –
MERGE – docs.microsoft
MERGE – docs.oracle

This article is contributed by **Dimpy Varshni** If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.
Last Updated : 31 Jan, 2019 20

Similar Reads

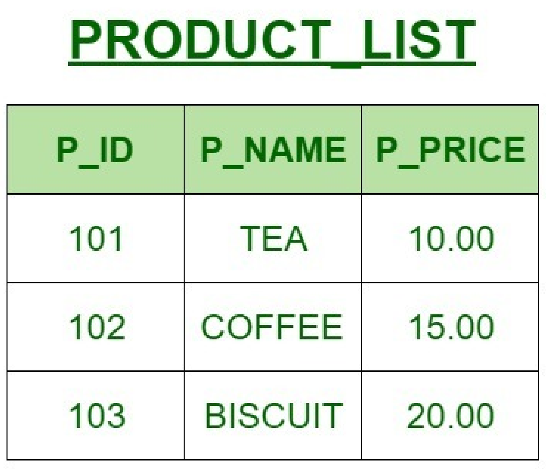
MERGE Statement in SQL Explained

Prerequisite – MERGE Statement
As MERGE statement in SQL, as discussed before in the previous post, is the combination of three INSERT, DELETE and UPDATE statements. So if there is a **Source table** and a **Target table** that are to be merged, then with the help of MERGE statement, all the three operations (INSERT, UPDATE, DELETE) can be performed at once.

A simple example will clarify the use of MERGE Statement.

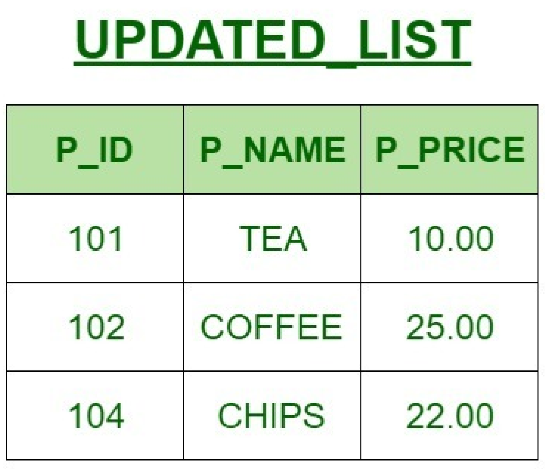
Example:
Suppose there are two tables:

- **PRODUCT_LIST** which is the table that contains the current details about the products available with fields P_ID, P_NAME, and P_PRICE corresponding to the ID, name and price of each product.
- **UPDATED_LIST** which is the table that contains the new details about the products available with fields P_ID, P_NAME, and P_PRICE corresponding to the ID, name and price of each product.



PRODUCT_LIST

P_ID	P_NAME	P_PRICE
101	TEA	10.00
102	COFFEE	15.00
103	BISCUIT	20.00



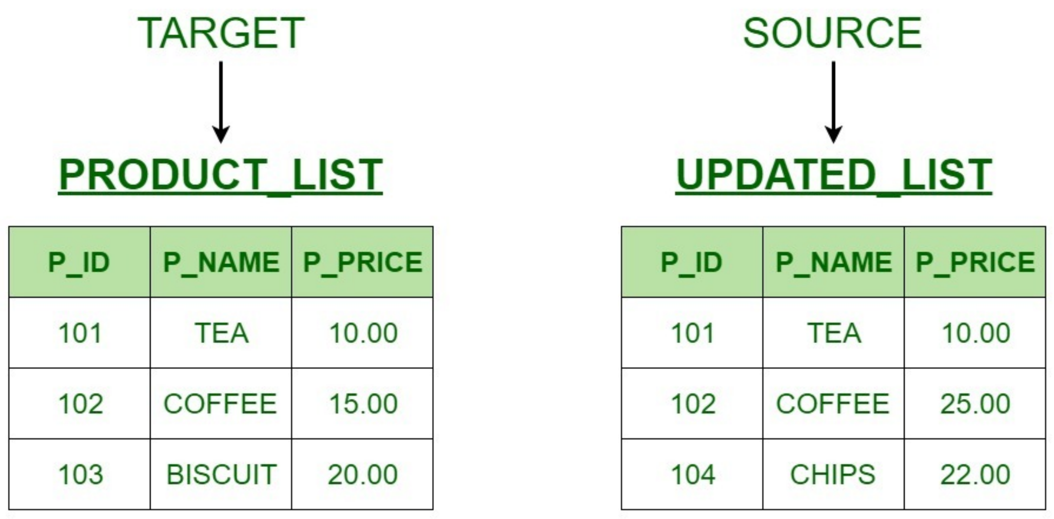
UPDATED_LIST

P_ID	P_NAME	P_PRICE
101	TEA	10.00
102	COFFEE	25.00
104	CHIPS	22.00

The task is to update the details of the products in the PRODUCT_LIST as per the UPDATED_LIST.

Solution
Now in order to explain this example better, let's split the example into steps.

Step 1: Recognise the TARGET and the SOURCE table
So in this example, since it is asked to update the products in the PRODUCT_LIST as per the UPDATED_LIST, hence the PRODUCT_LIST will act as the TARGET and UPDATED_LIST will act as the SOURCE table.


TARGET
SOURCE
PRODUCT_LIST
UPDATED_LIST

P_ID	P_NAME	P_PRICE
101	TEA	10.00
102	COFFEE	15.00
103	BISCUIT	20.00

P_ID	P_NAME	P_PRICE
101	TEA	10.00
102	COFFEE	25.00
104	CHIPS	22.00

Step 2: Recognise the operations to be performed.
Now as it can be seen that there are three mismatches between the TARGET and the SOURCE table, which are:

1. The cost of COFFEE in TARGET is 15.00 while in SOURCE it is 25.00

```
       PRODUCT_LIST
102     COFFEE    15.00

       UPDATED_LIST
102     COFFEE    25.00
```

2. There is no BISCUIT product in SOURCE but it is in TARGET

```
      PRODUCT_LIST
 103     BISCUIT   20.00
```

3. There is no CHIPS product in TARGET but it is in SOURCE

```
      UPDATED_LIST
 104     CHIPS     22.00
```

Therefore, three operations need to be done in the TARGET according to the above discrepancies. They are:

1. UPDATE operation

```
 102     COFFEE    25.00
```

2. DELETE operation

```
 103     BISCUIT   20.00
```

3. INSERT operation

```
 104     CHIPS     22.00
```

Step 3: Write the SQL Query.

> *Note: Refer this post for the syntax of MERGE statement.*

The **SQL query** to perform the above-mentioned operations with the help of **MERGE statement** is:

SQL

```sql
/* Selecting the Target and the Source */
MERGE PRODUCT_LIST AS TARGET
    USING UPDATE_LIST AS SOURCE

    /* 1. Performing the UPDATE operation */

    /* If the P_ID is same,
       check for change in P_NAME or P_PRICE */
    ON (TARGET.P_ID = SOURCE.P_ID)
    WHEN MATCHED
         AND TARGET.P_NAME <> SOURCE.P_NAME
         OR TARGET.P_PRICE <> SOURCE.P_PRICE

    /* Update the records in TARGET */
    THEN UPDATE
```

```
        SET TARGET.P_NAME = SOURCE.P_NAME,
        TARGET.P_PRICE = SOURCE.P_PRICE

    /* 2. Performing the INSERT operation */

    /* When no records are matched with TARGET table
       Then insert the records in the target table */
    WHEN NOT MATCHED BY TARGET
    THEN INSERT (P_ID, P_NAME, P_PRICE)
        VALUES (SOURCE.P_ID, SOURCE.P_NAME, SOURCE.P_PRICE)

    /* 3. Performing the DELETE operation */

    /* When no records are matched with SOURCE table
       Then delete the records from the target table */
    WHEN NOT MATCHED BY SOURCE
    THEN DELETE

 /* END OF MERGE */
```

Output:

```
   PRODUCT_LIST
 P_ID     P_NAME     P_PRICE
 101      TEA        10.00
 102      COFFEE     25.00
 104      CHIPS      22.00
```

So, in this way all we can perform all these three main statements in SQL together with the help of MERGE statement.

Note: Any name other than target and source can be used in the MERGE syntax. They are used only to give you a better explanation.

Last Updated : 09 Sep, 2021 21

Similar Reads

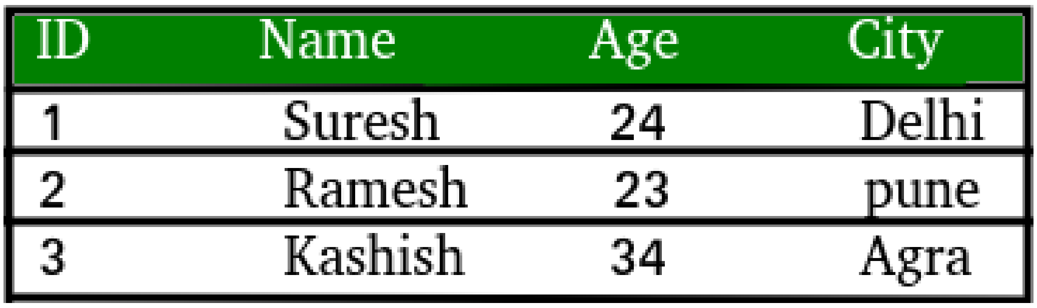
1. SQL | MERGE Statement
2. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)
3. Configure SQL Jobs in SQL Server using T-SQL
4. SQL vs NO SQL vs NEW SQL
5. SQL INSERT INTO Statement
6. SQL | DELETE Statement

SQL | Intersect & Except clause

1. **INTERSECT clause :** As the name suggests, the intersect clause is used to provide the result of the intersection of two select statements. This implies the result contains all the rows which are common to both the SELECT statements. **Syntax :**

```
SELECT column-1, column-2 ……
FROM table 1
WHERE…..

INTERSECT

SELECT column-1, column-2 ……
FROM table 2
WHERE…..
```

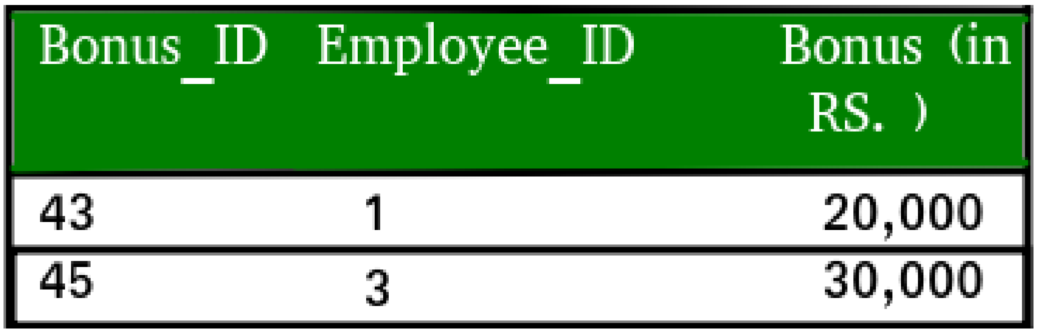
Example : Table 1 containing Employee Details



ID	Name	Age	City
1	Suresh	24	Delhi
2	Ramesh	23	pune
3	Kashish	34	Agra

Table

2 containing details of employees who are provided bonus

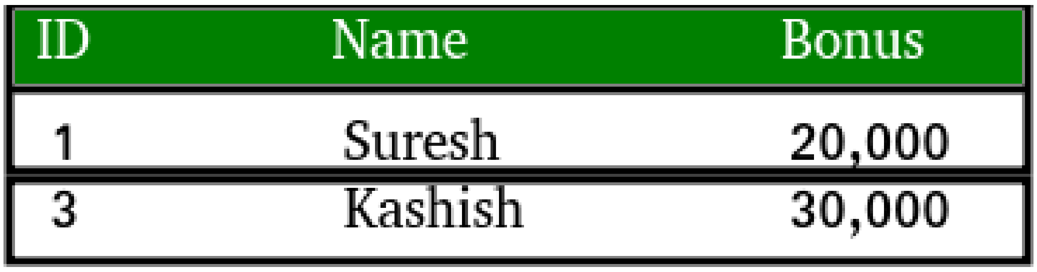


Bonus_ID	Employee_ID	Bonus (in RS.)
43	1	20,000
45	3	30,000

Query :

```
SELECT ID, Name, Bonus
FROM
table1
LEFT JOIN
table2
ON table1.ID = table2.Employee_ID

INTERSECT

SELECT ID, Name, Bonus
FROM
table1
RIGHT JOIN
table2
ON table1.ID = table2.Employee_ID;
```

Result :



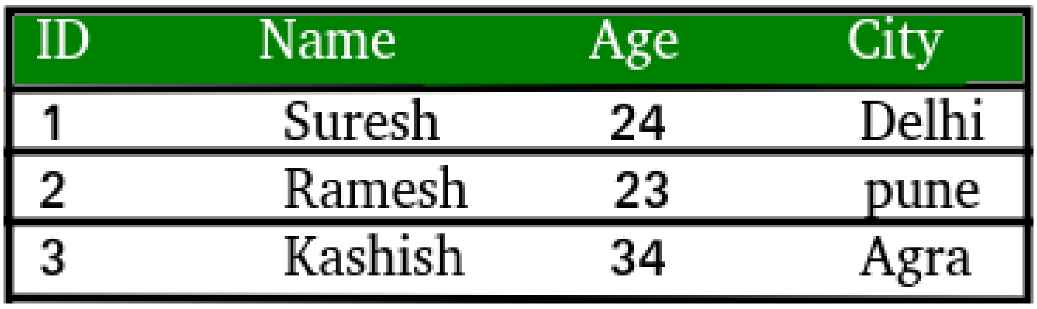
ID	Name	Bonus
1	Suresh	20,000
3	Kashish	30,000

2.

EXCEPT clause : contains all the rows that are returned by the first SELECT operation, and not returned by the second SELECT operation. **Syntax :**

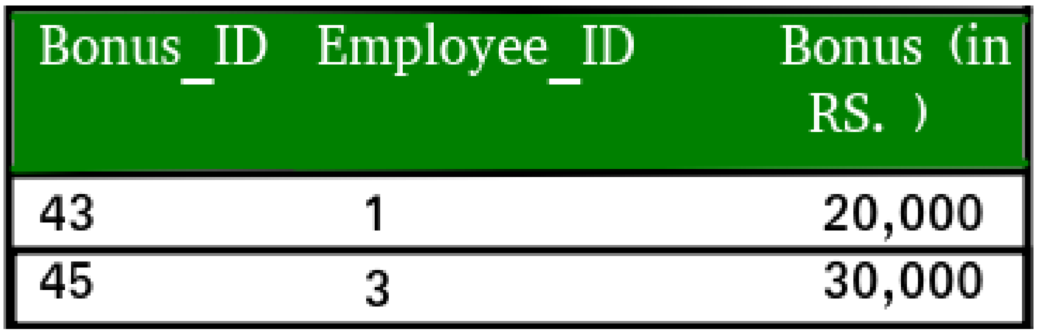
```
SELECT column-1, column-2 ……
FROM table 1
WHERE…..

EXCEPT

SELECT column-1, column-2 ……
FROM table 2
WHERE…..
```

Example : Table 1 containing Employee Details



ID	Name	Age	City
1	Suresh	24	Delhi
2	Ramesh	23	pune
3	Kashish	34	Agra

Table 2 containing details of employees who are provided bonus

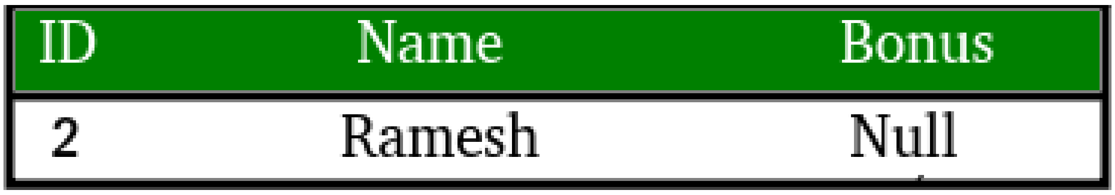


Bonus_ID	Employee_ID	Bonus (in RS.)
43	1	20,000
45	3	30,000

Query :

```
SELECT ID, Name, Bonus
FROM
table1
LEFT JOIN
table2
ON table1.ID = table2.Employee_ID

EXCEPT

SELECT ID, Name, Bonus
FROM
table1
RIGHT JOIN
table2
ON table1.ID = table2.Employee_ID;
```

Result :



ID	Name	Bonus
2	Ramesh	Null

Last Updated : 30 Jun, 2022 23

Similar Reads

1. SQL | Except Clause

SQL | Distinct Clause

The distinct keyword is used in conjunction with the select keyword. It is helpful when there is a need to avoid duplicate values present in any specific columns/table. When we use distinct keywords only the **unique values** are fetched.

Syntax :

SELECT DISTINCT column1, column2

FROM table_name

1. **column1, column2:** Names of the fields of the table.
2. **Table_name:** Table from where we want to fetch the records.

This query will return all the unique combinations of rows in the table with fields column1, and column2.

NOTE: If a distinct keyword is used with multiple columns, the distinct combination is displayed in the result set.

Distinct Operations

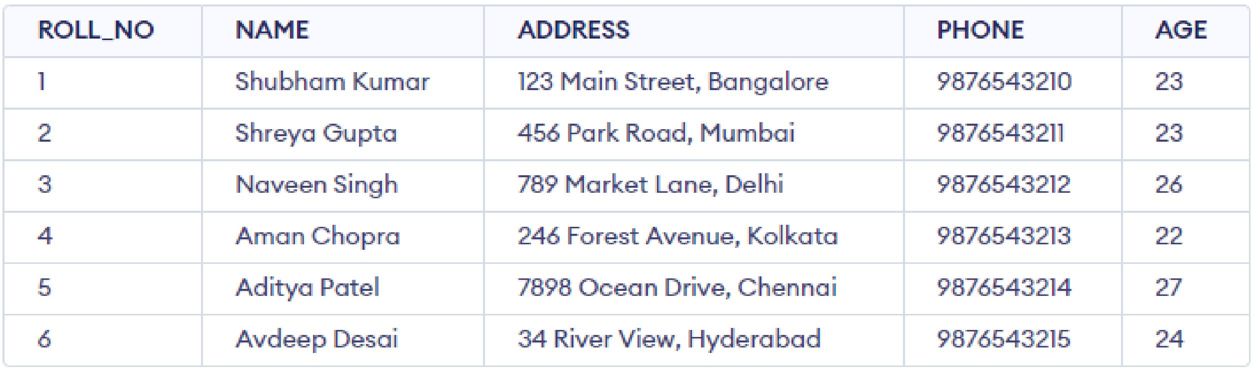
Query:

```
CREATE TABLE students (
  ROLL_NO INT,
  NAME VARCHAR(50),
  ADDRESS VARCHAR(100),
  PHONE VARCHAR(20),
  AGE INT
);
```

Inserting some random data to perform distinct operations.

```
INSERT INTO students (ROLL_NO, NAME, ADDRESS, PHONE, AGE)
VALUES
  (1, 'Shubham Kumar', '123 Main Street, Bangalore', '9876543210', 23),
  (2, 'Shreya Gupta', '456 Park Road, Mumbai', '9876543211', 23),
  (3, 'Naveen Singh', '789 Market Lane, Delhi', '9876543212', 26),
  (4, 'Aman Chopra', '246 Forest Avenue, Kolkata', '9876543213', 22),
  (5, 'Aditya Patel', '7898 Ocean Drive, Chennai', '9876543214', 27),
  (6, 'Avdeep Desai', '34 River View, Hyderabad', '9876543215', 24);
```

Output:



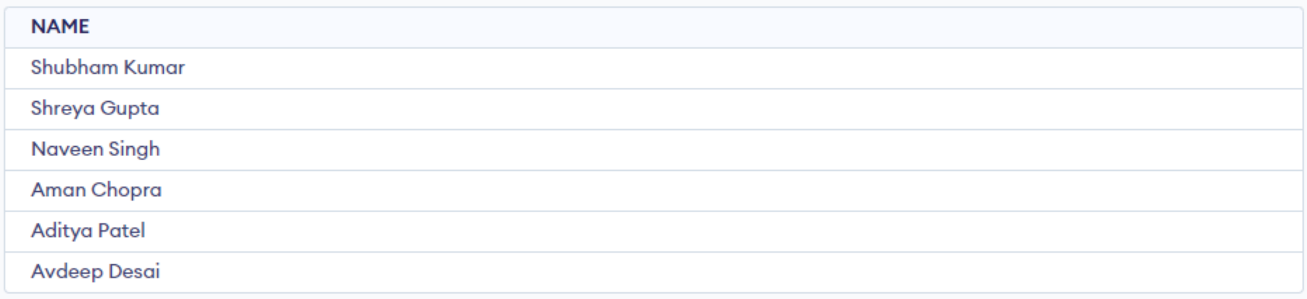
ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	Shubham Kumar	123 Main Street, Bangalore	9876543210	23
2	Shreya Gupta	456 Park Road, Mumbai	9876543211	23
3	Naveen Singh	789 Market Lane, Delhi	9876543212	26
4	Aman Chopra	246 Forest Avenue, Kolkata	9876543213	22
5	Aditya Patel	7898 Ocean Drive, Chennai	9876543214	27
6	Avdeep Desai	34 River View, Hyderabad	9876543215	24

Now, to fetch unique names from the NAME field.

Query:

```
SELECT DISTINCT NAME FROM Student;
```

Output :



NAME
Shubham Kumar
Shreya Gupta
Naveen Singh
Aman Chopra
Aditya Patel
Avdeep Desai

Now, to fetch a unique combination of rows from the whole table.

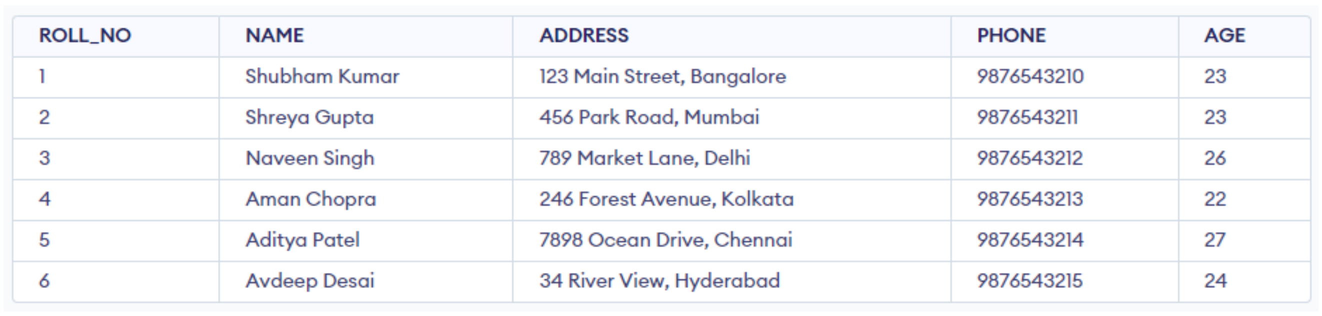
Syntax:

*SELECT DISTINCT * FROM Table_name;*

Query:

```
SELECT DISTINCT * FROM students;
```

Output :



ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	Shubham Kumar	123 Main Street, Bangalore	9876543210	23
2	Shreya Gupta	456 Park Road, Mumbai	9876543211	23
3	Naveen Singh	789 Market Lane, Delhi	9876543212	26
4	Aman Chopra	246 Forest Avenue, Kolkata	9876543213	22
5	Aditya Patel	7898 Ocean Drive, Chennai	9876543214	27
6	Avdeep Desai	34 River View, Hyderabad	9876543215	24

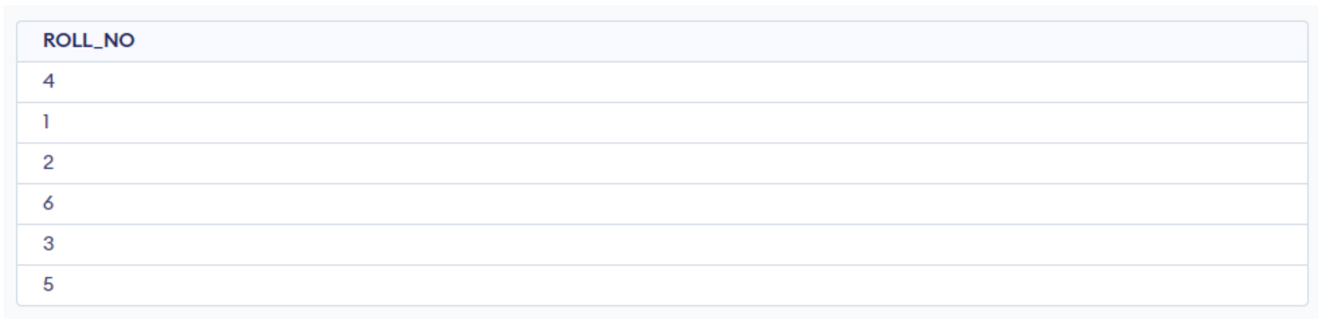
Using Distinct Clause with Order By

Here, we will check the order by clause with a Distinct clause which will filter out the data on the basis of the order by clause.

Query:

```
SELECT DISTINCT ROLL_NO FROM Students ORDER BY AGE;
```

Output:



ROLL_NO
4
1
2
6
3
5

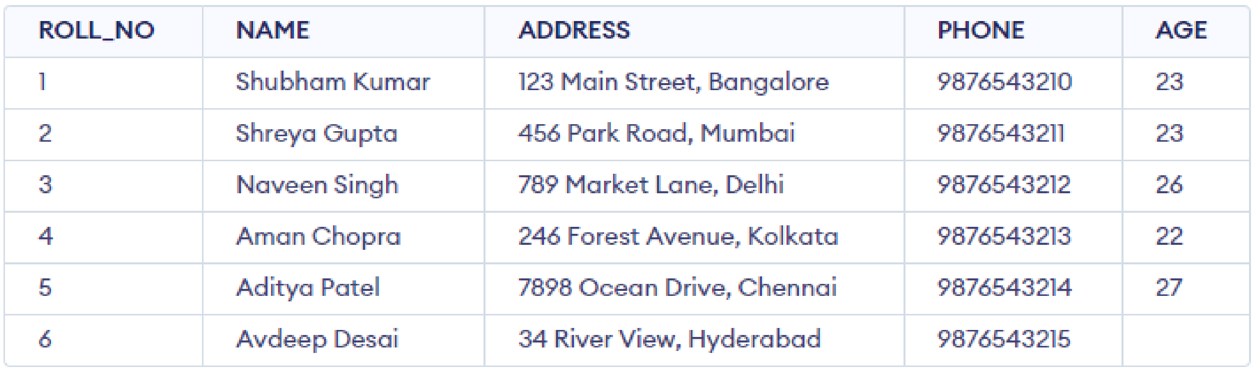
How the DISTINCT Clause Handles NULL Values?

Finally, does the DISTINCT clause considers a NULL to be a unique value in SQL? The answer is yes.

CREATE TABLE:

```
CREATE TABLE students (
  ROLL_NO INT,
  NAME VARCHAR(50),
  ADDRESS VARCHAR(100),
  PHONE VARCHAR(20),
  AGE INT
);
INSERT INTO students (ROLL_NO, NAME, ADDRESS, PHONE, AGE)
VALUES
  (1, 'Shubham Kumar', '123 Main Street, Bangalore', '9876543210', 23),
  (2, 'Shreya Gupta', '456 Park Road, Mumbai', '9876543211', 23),
  (3, 'Naveen Singh', '789 Market Lane, Delhi', '9876543212', 26),
  (4, 'Aman Chopra', '246 Forest Avenue, Kolkata', '9876543213', 22),
  (5, 'Aditya Patel', '7898 Ocean Drive, Chennai', '9876543214', 27),
  (6, 'Avdeep Desai', '34 River View, Hyderabad', '9876543215', NULL);
```

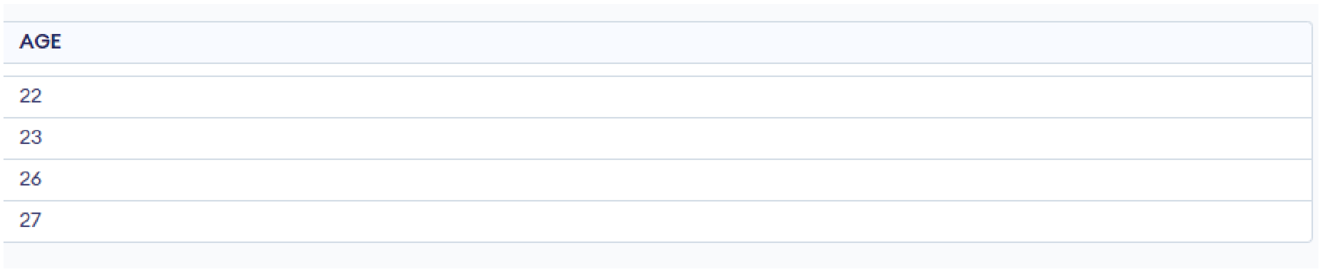
Output:



ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	Shubham Kumar	123 Main Street, Bangalore	9876543210	23
2	Shreya Gupta	456 Park Road, Mumbai	9876543211	23
3	Naveen Singh	789 Market Lane, Delhi	9876543212	26
4	Aman Chopra	246 Forest Avenue, Kolkata	9876543213	22
5	Aditya Patel	7898 Ocean Drive, Chennai	9876543214	27
6	Avdeep Desai	34 River View, Hyderabad	9876543215	

Query:

```
  SELECT DISTINCT AGE
  FROM students;
```



AGE
22
23
26
27

Note: Without the keyword distinct in both the above examples 6 records would have been fetched instead of 4, since in the original table there are 6 records with the duplicate values.

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to

Last Updated : 10 May, 2023

33

Similar Reads

1. Difference between Having clause and Group by clause
2. Distinct clause in MS SQL Server
3. SQL | WHERE Clause
4. Top Clause in Microsoft SQL Server
5. SQL | Union Clause
6. SQL | WITH clause
7. SQL | Except Clause
8. SQL | OFFSET-FETCH Clause
9. Having vs Where Clause in SQL
10. SQL | LIMIT Clause

Previous Next

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Basic

Easy Normal Medium Hard Expert

Improved By : Anshika Goyal, shubhamthakur05

SQL | LIMIT Clause

SQL limit clause is very useful in some scenarios where we really need the data in some sorted manner suppose if there are a large number of tuples satisfying the query conditions, it might be resourceful to view only a handful of them at a time.

Important points to remember:

- The LIMIT clause is used to set an upper limit on the number of tuples returned by SQL.
- It is important to note that this clause is not supported by all SQL versions.
- The LIMIT clause can also be specified using the SQL 2008 OFFSET/FETCH FIRST clauses.
- The limit/offset expressions must be a non-negative integer.

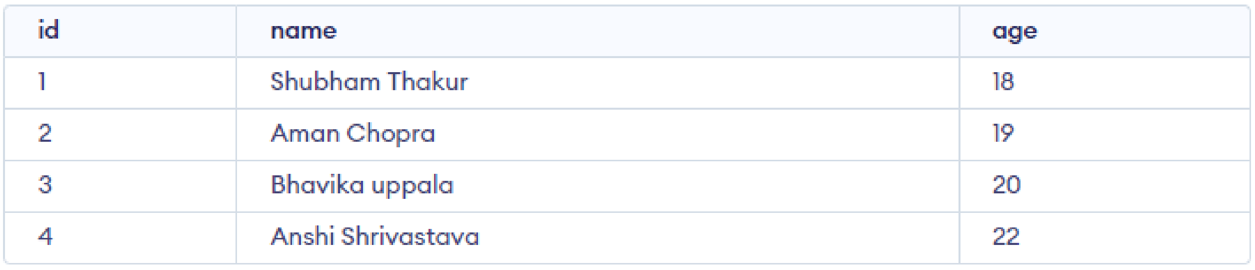
Example:

Let's assume that we have one sample table name Student and to understand better we will see some query about limit clause.Say we have a relationship, Student.

Student Table:

```
CREATE TABLE student (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  age INT
);

INSERT INTO student (id, name, age)
VALUES (1, 'Shubham Thakur', 18),
       (2, 'Aman Chopra', 19),
       (3, 'Bhavika uppala', 20),
       (4,'Anshi Shrivastava',22);
```

Output:

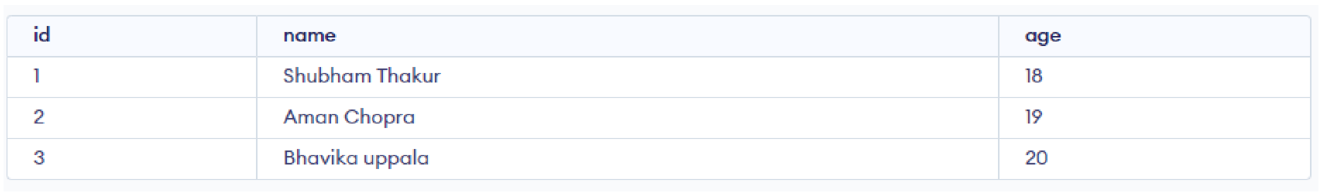


id	name	age
1	Shubham Thakur	18
2	Aman Chopra	19
3	Bhavika uppala	20
4	Anshi Shrivastava	22

Queries:

```
SELECT *
FROM student
LIMIT 3;
```

Output:

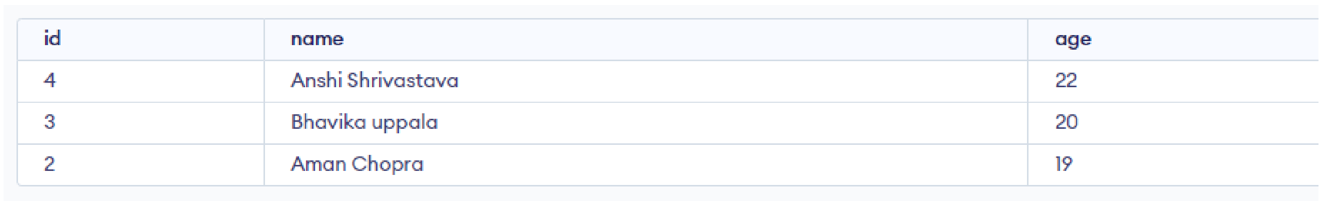


id	name	age
1	Shubham Thakur	18
2	Aman Chopra	19
3	Bhavika uppala	20

Other Query to check with ORDER BY Clause:

```
SELECT *
FROM Student
ORDER BY Grade DESC
LIMIT 3;
```

Output:



id	name	age
4	Anshi Shrivastava	22
3	Bhavika uppala	20
2	Aman Chopra	19

The LIMIT operator can be used in situations such as the above, where we need to find the top 3 students in a class and do not want to use any conditional statements.

Using LIMIT along with OFFSET

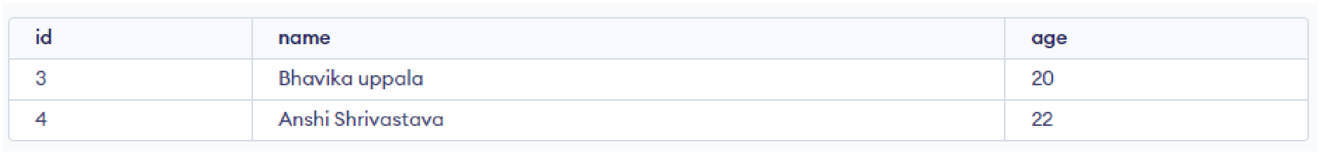
LIMIT x OFFSET y simply means skip the first y entries and then return the next x entries. OFFSET can only be used with the ORDER BY clause. It cannot be used on its own. OFFSET value must be greater than or equal to zero. It cannot be negative, else returns an error.

Queries:

```
SELECT *
FROM Student
ORDER BY ROLLNO LIMIT 5 OFFSET 2;

or

SELECT *
FROM Student
ORDER BY ROLLNO LIMIT 2,5; # it skips the
first 2 values and then return the next 5 entries
```

The first query and second query return the same results. In the second query, limit is followed by two values. LIMIT X, Y The first value X is the offset value (skips X number of entries) and the second value Y is the limit (it returns the next Y number of entries).

Output:



id	name	age
3	Bhavika uppala	20
4	Anshi Shrivastava	22

SQL LIMIT to Get the nth Highest or Lowest Value

Now we will look for LIMIT use in finding highest or lowest value we need to retrieve the rows with the nth highest or lowest value. In that situation, we can use the subsequent MySQL LIMIT clause to obtain the desired outcome.

Syntax:

SELECT column_list

FROM table_name

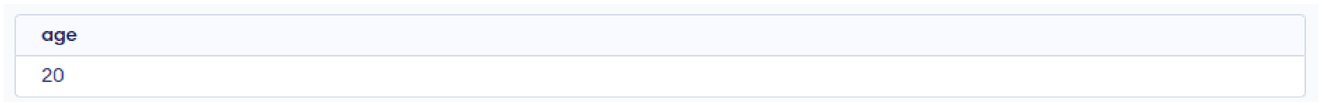
ORDER BY expression

LIMIT n-1, 1;

Query:

```
SELECT age FROM Student
ORDER BY age  LIMIT 2, 1;
```

Output:



age
20

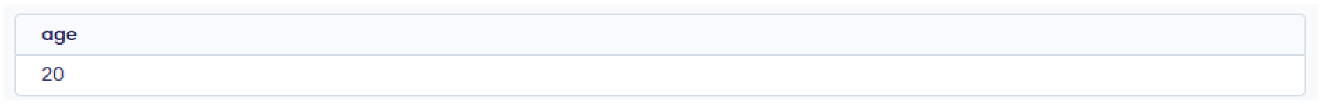
The Limit in MySQL with Where

The WHERE clause can also be used with MySQL Limit. It produces the rows that matched the condition after checking the specified condition in the table.

Query:

```
SELECT age
FROM Student
WHERE id<4
ORDER BY age
LIMIT 2, 1;
```

Output:



age
20

Restrictions on the LIMIT clause

The LIMIT clause's limitations. The following situations do not allow the LIMIT clause to be used:

- With regard to defining a view
- The use of nested SELECT statements
- Except for subqueries with table expressions specified in the FROM clause.
- Embedded SELECT statements are used as expressions in a singleton SELECT (where max = 1) within an SPL routine where embedded SELECT statements are used as expressions.

This article is contributed by **Anannya Uberoi**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

SQL | Except Clause

In SQL, EXCEPT returns those tuples that are returned by the first SELECT operation, and not returned by the second SELECT operation.

This is the same as using a subtract operator in relational algebra.

Example:
Say we have two relations, Students and TA (Teaching Assistant). We want to return all those students who are not teaching assistants. The query can be formulated as:

Students Table:

StudentID	Name	Course
1	Rohan	DBMS
2	Kevin	OS
3	Mansi	DBMS
4	Mansi	ADA
5	Rekha	ADA
6	Megha	OS

TA Table:

StudentID	Name	Course
1	Kevin	TOC
2	Sita	IP
3	Manik	AP
4	Rekha	SNS

```
SELECT Name
       FROM Students
EXCEPT
SELECT NAME
       FROM TA;
```

Output:

```
Rohan
Mansi
Megha
```

To retain duplicates, we must explicitly write **EXCEPTALL** instead of EXCEPT.

```
SELECT Name
       FROM Students
EXCEPTALL
SELECT Name
       FROM TA;
```

Output:

```
Rohan
Mansi
Mansi
Megha
```

Difference between EXCEPT and NOT IN Clause

EXCEPT automatically removes all duplicates in the final result, whereas NOT IN retains duplicate tuples. It is also important to note that EXCEPT is not supported by MySQL.

This article is contributed by **Anannya Uberoi**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article

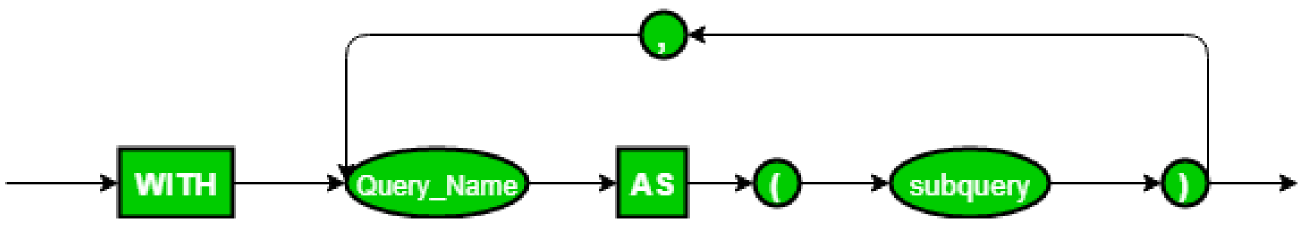
SQL | WITH clause

The SQL WITH clause was introduced by Oracle in the Oracle 9i release 2 database. The SQL WITH clause allows you to give a sub-query block a name (a process also called sub-query refactoring), which can be referenced in several places within the main SQL query.

- The clause is used for defining a temporary relation such that the output of this temporary relation is available and is used by the query that is associated with the WITH clause.
- Queries that have an associated WITH clause can also be written using nested sub-queries but doing so add more complexity to read/debug the SQL query.
- WITH clause is not supported by all database system.
- The name assigned to the sub-query is treated as though it was an inline view or table
- The SQL WITH clause was introduced by Oracle in the Oracle 9i release 2 database.

Syntax:

```
WITH temporaryTable (averageValue) as
    (SELECT avg(Attr1)
    FROM Table)
    SELECT Attr1
    FROM Table, temporaryTable
    WHERE Table.Attr1 > temporaryTable.averageValue;
```



In this query, WITH clause is used to define a temporary relation temporaryTable that has only 1 attribute averageValue. averageValue holds the average value of column Attr1 described in relation Table. The SELECT statement that follows the WITH clause will produce only those tuples where the value of Attr1 in relation Table is greater than the average value obtained from the WITH clause statement.

AD

Note: When a query with a WITH clause is executed, first the query mentioned within the clause is evaluated and the output of this evaluation is stored in a temporary relation. Following this, the main query associated with the WITH clause is finally executed that would use the temporary relation produced.

Queries

Example 1: Find all the employee whose salary is more than the average salary of all employees.
Name of the relation: **Employee**

EmployeeID	Name	Salary
100011	Smith	50000
100022	Bill	94000
100027	Sam	70550

100845	Walden	80000
115585	Erik	60000
1100070	Kate	69000

SQL Query:

```
WITH temporaryTable(averageValue) as
    (SELECT avg(Salary)
    from Employee)
        SELECT EmployeeID,Name, Salary
        FROM Employee, temporaryTable
        WHERE Employee.Salary > temporaryTable.averageValue;
```

Output:

EmployeeID	Name	Salary
100022	Bill	94000
100845	Walden	80000

Explanation: The average salary of all employees is 70591. Therefore, all employees whose salary is more than the obtained average lies in the output relation.

Example 2: Find all the airlines where the total salary of all pilots in that airline is more than the average of total salary of all pilots in the database.

Name of the relation: **Pilot**

EmployeeID	Airline	Name	Salary
70007	Airbus 380	Kim	60000
70002	Boeing	Laura	20000
10027	Airbus 380	Will	80050
10778	Airbus 380	Warren	80780
115585	Boeing	Smith	25000
114070	Airbus 380	Katy	78000

SQL Query:

```
WITH totalSalary(Airline, total) as
    (SELECT Airline, sum(Salary)
    FROM Pilot
    GROUP BY Airline),
    airlineAverage(avgSalary) as
    (SELECT avg(Salary)
    FROM Pilot )
    SELECT Airline
    FROM totalSalary, airlineAverage
    WHERE totalSalary.total > airlineAverage.avgSalary;
```

Output:

Airline
Airbus 380

Explanation: The total salary of all pilots of Airbus 380 = 298,830 and that of Boeing = 45000. Average salary of all pilots in the table Pilot = 57305. Since only the total salary of all pilots of Airbus 380 is greater than the average salary obtained, so Airbus 380 lies in the output relation.

Important Points:

- The SQL WITH clause is good when used with complex SQL statements rather than simple ones
- It also allows you to break down complex SQL queries into smaller ones which make it easy for debugging and processing the complex queries.
- The SQL WITH clause is basically a drop-in replacement to the normal sub-query.

This article is contributed by **Mayank Kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 13 Aug, 2021 111

Similar Reads

1. Difference between Having clause and Group by clause

SQL | With Ties Clause

classyallrounder

This post is a continuation of SQL Offset-Fetch Clause

Now, we understand that how to use the Fetch Clause in Oracle Database, along with the Specified Offset and we also understand that Fetch clause is the newly added clause in the Oracle Database 12c or it is the new feature added in the Oracle database 12c.

Now consider the below example:

Suppose we a have a table named **myTable** with below data:

```
ID    NAME        SALARY
-----------------------------
1    Geeks      10000
4    Finch      10000
2    RR         6000
3    Dhoni      16000
5    Karthik    7000
6    Watson     10000
```

Now, suppose we want the first three rows to be Ordered by Salary in descending order, then the below query must be executed:

```
Query:
SELECT * from myTable
order by salary desc
fetch first 3 rows only;
```

```
Output:
We got only first 3 rows order by Salary in Descending Order

ID    NAME    SALARY
--------------------------
3    Dhoni    16000
1    Geeks    10000
4    Finch    10000
```

Note: In the above result we got first 3 rows, ordered by Salary in Descending Order, but we have one more row with same salary i.e, the row with name **Watson** and Salary **10000**, but it didn't came up, because we restricted our output to first three rows only. But this is not optimal, because most of the time in live applications we will be required to display the tied rows also.

Real Life Example – Suppose we have 10 Racers running, and we have only 3 prizes i.e, first, second, third, but suppose, Racers 3 and 4 finished the race together in same time, so in this case we have a tie between 3 and 4 and that's why both are holder of Position 3.

With Ties

So, to overcome the above problem, Oracle introduces a clause known as **With Ties** clause. Now, let's see our previous example using With Ties clause.

```
Query:
SELECT * from myTable
order by salary desc
fetch first 3 rows With Ties;

Output:
See we get only first 3 rows order by Salary in Descending Order along with Tied
Row also

ID    NAME       SALARY
--------------------------
3    Dhoni     16000
1    Geeks     10000
6    Watson    10000 // We get Tied Row also
4    Finch     10000
```

Now, see we got the **tied row** also, which we were not getting previously.

Note: We **get** the tied row in our output, only when we use the **order by** clause in our Select statement. Suppose, if we won't use order by clause, and still we are using **with ties** clause, then we won't get the tied row in our output and the query behaves same as, if we are using **ONLY** clause **instead** of With Ties clause.

Example – Suppose we execute the below query(without using order by clause) :

```
Query:
SELECT * from myTable
fetch first 3 rows With Ties;

Output:
See we won't get the tied row because we didn't use order by clause

ID    NAME      SALARY
--------------------------
1    Geeks    10000
4    Finch    10000
2    RR       6000
```

In the above result we won't get the tied row and we get only first 3 rows. So **With Ties** is **tied** with **order by** clause, i.e, we get the tied row in output if and only if we use With Ties along with Order by clause.

Note: Please make sure that, you run these queries in Oracle Database 12c, because Fetch clause is the newly added feature in Oracle 12c, also With Ties, runs only in Oracle Database 12c, these queries **won't** run in below versions of 12c like 10g or 11g.

References: About Fetch Clause as well as With Ties Clause, Performing SQL Queries Online

Last Updated : 21 Mar, 2018

33

Similar Reads

1. Difference between Having clause and Group by clause
2. SQL | Distinct Clause
3. SQL | WHERE Clause
4. Top Clause in Microsoft SQL Server
5. SQL | Union Clause
6. SQL | WITH clause
7. SQL | Except Clause

SQL | OFFSET-FETCH Clause

OFFSET and FETCH Clause are used in conjunction with SELECT and ORDER BY clause to provide a means to retrieve a range of records.

OFFSET

The OFFSET argument is used to identify the starting point to return rows from a result set. Basically, it exclude the first set of records.

Note:

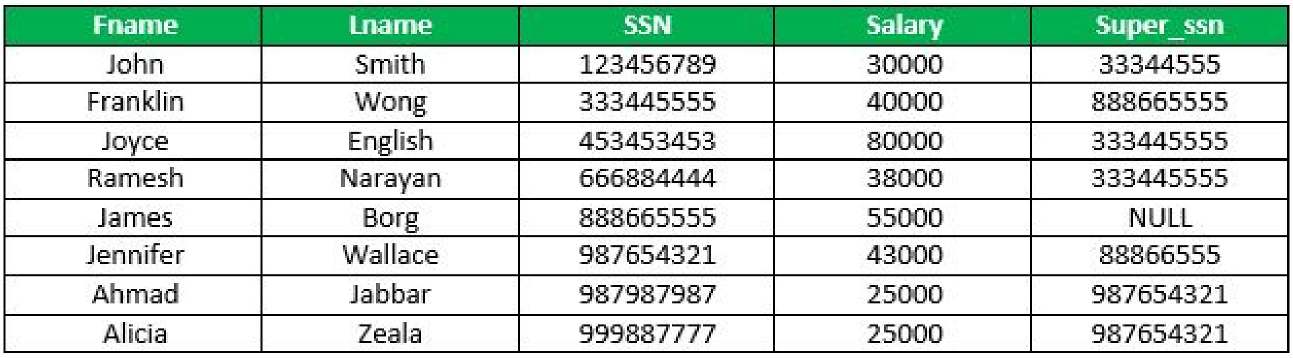
- OFFSET can only be used with ORDER BY clause. It cannot be used on its own.
- OFFSET value must be greater than or equal to zero. It cannot be negative, else return error.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
ORDER BY column_name
OFFSET rows_to_skip ROWS;
```

Examples:

Consider the following Employee table,

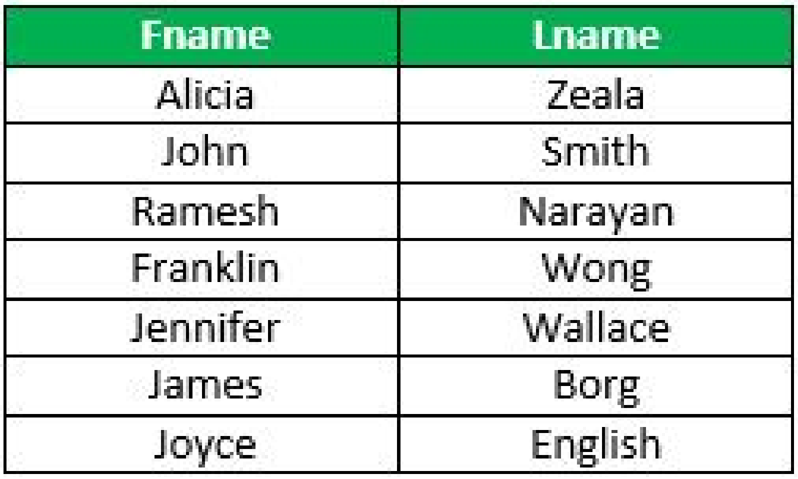


Fname	Lname	SSN	Salary	Super_ssn
John	Smith	123456789	30000	33344555
Franklin	Wong	333445555	40000	888665555
Joyce	English	453453453	80000	333445555
Ramesh	Narayan	666884444	38000	333445555
James	Borg	888665555	55000	NULL
Jennifer	Wallace	987654321	43000	88866555
Ahmad	Jabbar	987987987	25000	987654321
Alicia	Zeala	999887777	25000	987654321

- Print Fname, Lname of all the Employee except the employee having lowest salary.

```
SELECT Fname, Lname
FROM Employee
ORDER BY Salary
OFFSET 1 ROWS;
```

Output:



Fname	Lname
Alicia	Zeala
John	Smith
Ramesh	Narayan
Franklin	Wong
Jennifer	Wallace
James	Borg
Joyce	English

FETCH

The FETCH argument is used to return a set of number of rows. FETCH can't be used itself, it is used in conjunction with OFFSET.
Syntax:

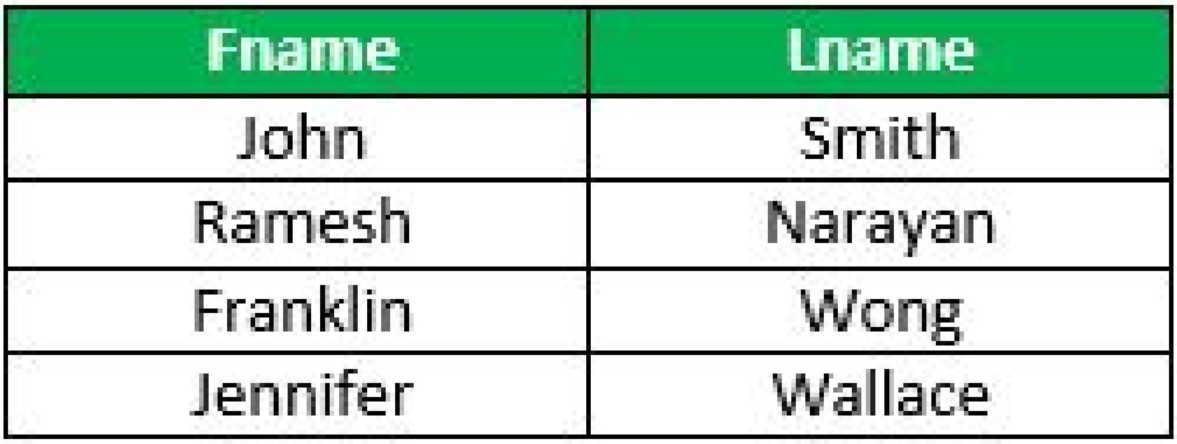
```
SELECT column_name(s)
FROM table_name
ORDER BY column_name
OFFSET rows_to_skip
FETCH NEXT number_of_rows ROWS ONLY;
```

Example:

- Print the Fname, Lname from 3rd to 6th tuple of Employee table when sorted according to the Salary.

```
SELECT Fname, Lname
FROM Employee
ORDER BY Salary
OFFSET 2 ROWS
FETCH NEXT 4 ROWS ONLY;
```

Output:

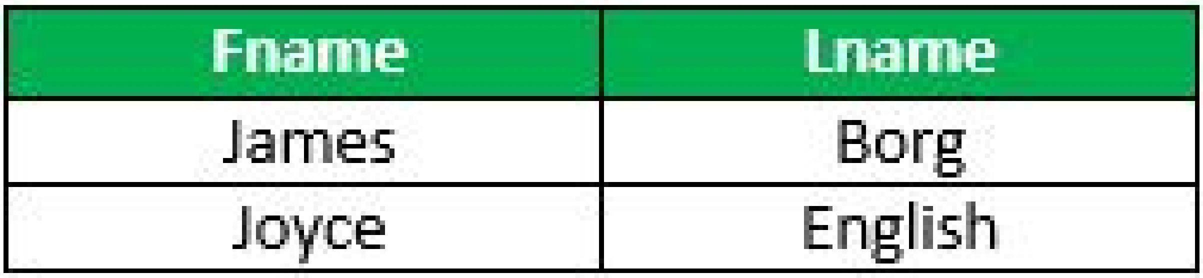


Fname	Lname
John	Smith
Ramesh	Narayan
Franklin	Wong
Jennifer	Wallace

- Print the bottom 2 tuples of Employee table when sorted by Salary.

```
SELECT Fname, Lname
FROM Employee
ORDER BY Salary
OFFSET (SELECT COUNT(*) FROM EMPLOYEE) - 2 ROWS
FETCH NEXT 2 ROWS;
```

Output:



Fname	Lname
James	Borg
Joyce	English

Important Points:

1. OFFSET clause is mandatory with FETCH. You can never use, ORDER BY … FETCH.
2. TOP cannot be combined with OFFSET and FETCH.
3. The OFFSET/FETCH row count expression can be only be any arithmetic, constant, or parameter expression which will return an integer value.
4. ORDER BY is mandatory to be used with OFFSET and FETCH clause.
5. OFFSET value must be greater than or equal to zero. It cannot be negative, else return error.

This article is contributed by **Anuj Chauhan**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 27 Dec, 2021 32

Similar Reads

1. Offset-Fetch in MS SQL Server
2. SQL - TOP, LIMIT, FETCH FIRST Clause
3. Difference between Having clause and Group by clause
4. SQL Query to find the Nth Largest Value in a Column using Limit and Offset
5. FETCH in SQL
6. How to execute an SQL query and fetch results using PHP ?
7. SQL | Distinct Clause
8. SQL | WHERE Clause
9. Top Clause in Microsoft SQL Server
10. SQL | Union Clause

Previous Next

Article Contributed By :

SQL | Union Clause

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

NOTE:

1. The fields to be used in both the select statements must be in the 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 for UNION:

SELECT column_name(s) FROM table1

UNION

*SELECT column_name(s) FROM table2**;***

Syntax for UNION ALL:

The resultant set consists of distinct values.

SELECT column_name(s) FROM table1

UNION ALL

SELECT column_name(s) FROM table2;

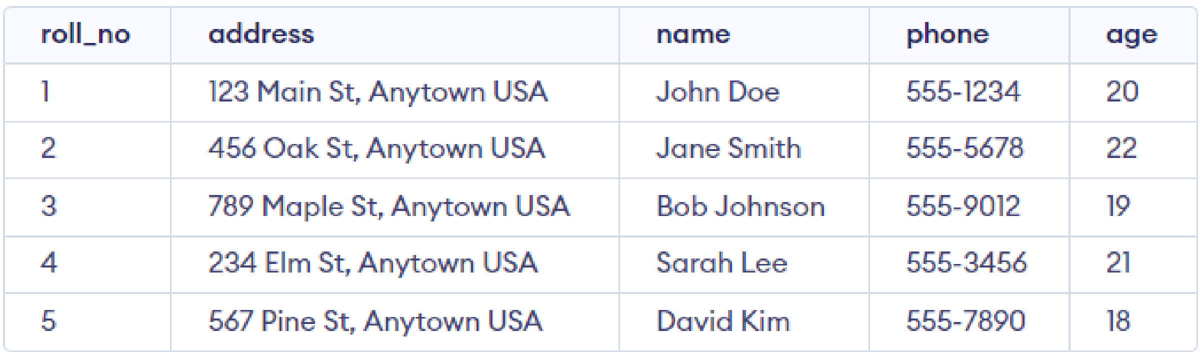
The resultant set consists of duplicate values too.

Consider we have two tables name **Student** and **Student_Details.** Suppose we want to do a union operation to find the common roll no from both tables. Let's first create a table with the name student and insert some random data similarly, we will create another table with the name Student_details keeping in mind that there will be at least one column common here we will take roll_no as a common column.

CREATE :

```
CREATE TABLE students (
  roll_no INT,
  address VARCHAR(255),
  name VARCHAR(255),
  phone VARCHAR(20),
  age INT
);
INSERT INTO students (roll_no, address, name, phone, age)
VALUES
  (1, '123 Main St, Anytown USA', 'John Doe', '555-1234', 20),
  (2, '456 Oak St, Anytown USA', 'Jane Smith', '555-5678', 22),
  (3, '789 Maple St, Anytown USA', 'Bob Johnson', '555-9012', 19),
  (4, '234 Elm St, Anytown USA', 'Sarah Lee', '555-3456', 21),
  (5, '567 Pine St, Anytown USA', 'David Kim', '555-7890', 18);
```

Output:



roll_no	address	name	phone	age
1	123 Main St, Anytown USA	John Doe	555-1234	20
2	456 Oak St, Anytown USA	Jane Smith	555-5678	22
3	789 Maple St, Anytown USA	Bob Johnson	555-9012	19
4	234 Elm St, Anytown USA	Sarah Lee	555-3456	21
5	567 Pine St, Anytown USA	David Kim	555-7890	18

Let's create a second table with the name Student details here it will contain three columns roll_no, branch, and grade.

CREATE :

```
CREATE TABLE student_details (
  roll_no INT,
  branch VARCHAR(50),
  grade VARCHAR(2)
);
INSERT INTO student_details (roll_no, branch, grade)
VALUES
  (1, 'Computer Science', 'A'),
  (2, 'Electrical Engineering', 'B'),
  (3, 'Mechanical Engineering', 'C');
```

Output:

roll_no	branch	grade
1	Computer Science	A
2	Electrical Engineering	B
3	Mechanical Engineering	C

UNION Clause

To fetch distinct ROLL_NO from Student and Student_Details table.

Query:

```
SELECT ROLL_NO FROM Students UNION
SELECT ROLL_NO FROM Student_Details;
```

Output:

roll_no
1
2
3
4
5

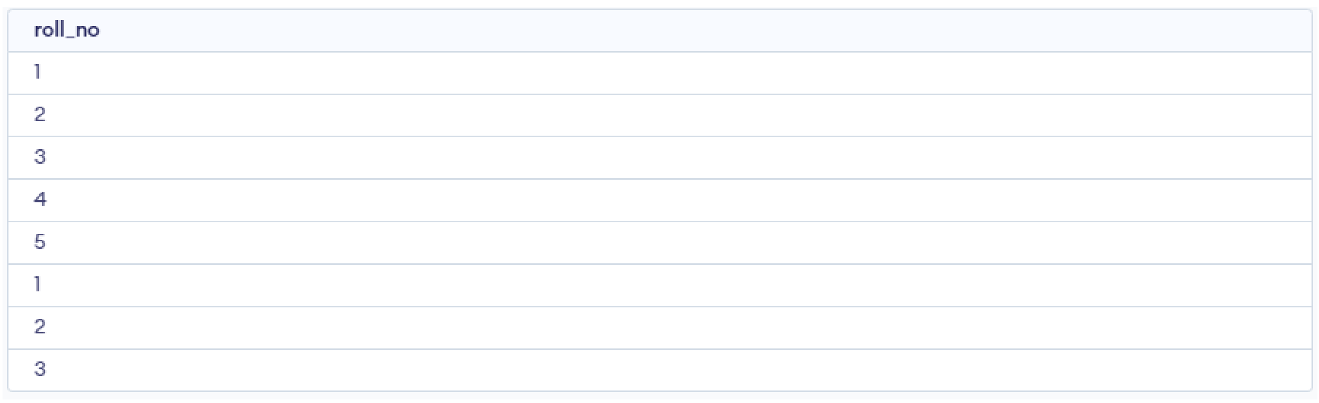
The UNION ALL Clause

To fetch ROLL_NO from Student and Student_Details table including duplicate values.

Query:

```
SELECT ROLL_NO FROM Students UNION ALL
SELECT ROLL_NO FROM Student_Details;
```

Output:



roll_no
1
2
3
4
5
1
2
3

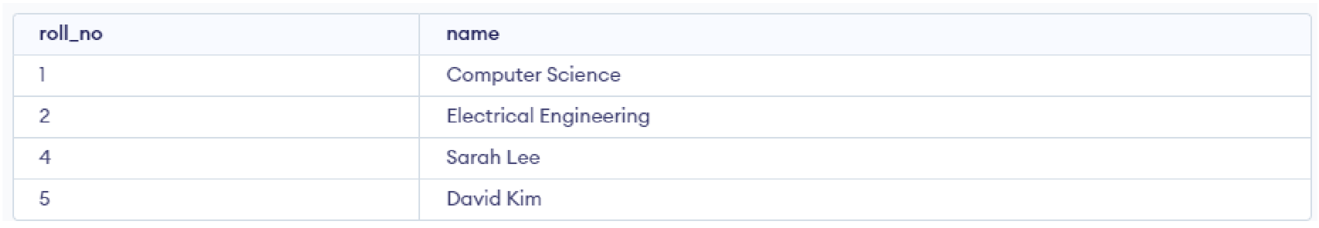
The UNION ALL Clause with Where Condition

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.

Query:

```
SELECT ROLL_NO,NAME FROM Students 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	Computer Science
2	Electrical Engineering
4	Sarah Lee
5	David Kim

Note: The column names in both the select statements can be different but the data type must be same.And in the result set the name of column used in the first select statement will appear.

This article is contributed by **Pratik Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Top Clause in Microsoft SQL Server

THE SELECT TOP clause is used to fetch a limited number of rows from a database. This clause is very useful while dealing with large databases. The top Clause will be useful for fetching the data records in larger datasets as it will drastically reduce the complexity.

Syntax

> *SELECT TOP value column1,column2 FROM table_name;*
>
> *value: number of rows to return from top*

> *column1 , column2 fields in the table*
>
> *table_name: name of table*

Syntax Using Percent

> *SELECT TOP value PERCENT column1,column2 FROM table_name;*
>
> *value: percentage of number of rows to return from top*
>
> *column1 , column2: fields in the table*
>
> *table_name: name of table*

Parameter Explanation

1. **TOP:** Clause is used for fetching the top records from a huge dataset.

Lets us see examples for Top Clause in Microsoft SQL Server, for this we create a database.

Query:

```
CREATE TABLE Customer(
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(50),
    LastName VARCHAR(50),
    Country VARCHAR(50),
    Age int(2),
    Phone int(10)
);

-- Insert some sample data into the Customers table
INSERT INTO Customer (CustomerID, CustomerName, LastName, Country, Age, Phone)
VALUES (1, 'Shubham', 'Thakur', 'India','23','xxxxxxxxxx'),
       (2, 'Aman ', 'Chopra', 'Australia','21','xxxxxxxxxx'),
       (3, 'Naveen', 'Tulasi', 'Sri lanka','24','xxxxxxxxxx'),
       (4, 'Aditya', 'Arpan', 'Austria','21','xxxxxxxxxx'),
       (5, 'Nishant. Salchichas S.A.', 'Jain', 'Spain','22','xxxxxxxxxx');
```

Output:

```
CustomerID      CustomerName    LastName        Country
Age     Phone
1       Shubham Thakur  India   23      45267
2       Aman    Chopra  Australia       21      43627
3       Naveen  Tulasi  Sri lanka       24      36278
4       Aditya  Arpan   Austria 21      92837
5       Nishant. Salchichas S.A.        Jain    Spain
22      98763
```

Query:

To fetch the first two data sets from the Customer table.

```
SELECT TOP 2 * FROM Customer;
```

Output

```
CustomerID      CustomerName    LastName        Country
Age      Phone
1        Shubham Thakur  India   23      45267
2        Aman    Chopra  Australia       21      43627
```

Add WHERE Clause in SQL Server

We can fetch data records by using a where clause with some condition was well.

Query:

```
SELECT TOP 1 * FROM Customers
WHERE Country='Spain';
```

Output:

```
CustomerID      CustomerName    LastName        Country
Age      Phone
5        Nishant. Salchichas S.A.        Jain    Spain
22       98763
```

Note:

To get the same functionality on MySQL and Oracle databases there is a bit of difference in the basic syntax;

- **For MySQL databases:**

```
SELECT column1,column2 FROM table_name LIMIT value;
column1 , column2: fields int the table
table_name: name of table
value: number of rows to return from top
```

- **For Oracle databases:**

```
SELECT column1,column2 FROM table_name WHERE ROWNUM <= value;
column1 , column2: fields int the table
table_name: name of table
value: number of rows to return from top
```

This article is contributed by **Pratik Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page