SoftServe | Empowering your Business through Software Development

SQL Overview

October 2014

What is SQL?

- **SQL** is a database computer language designed for the management and retrieval of data in relational database.

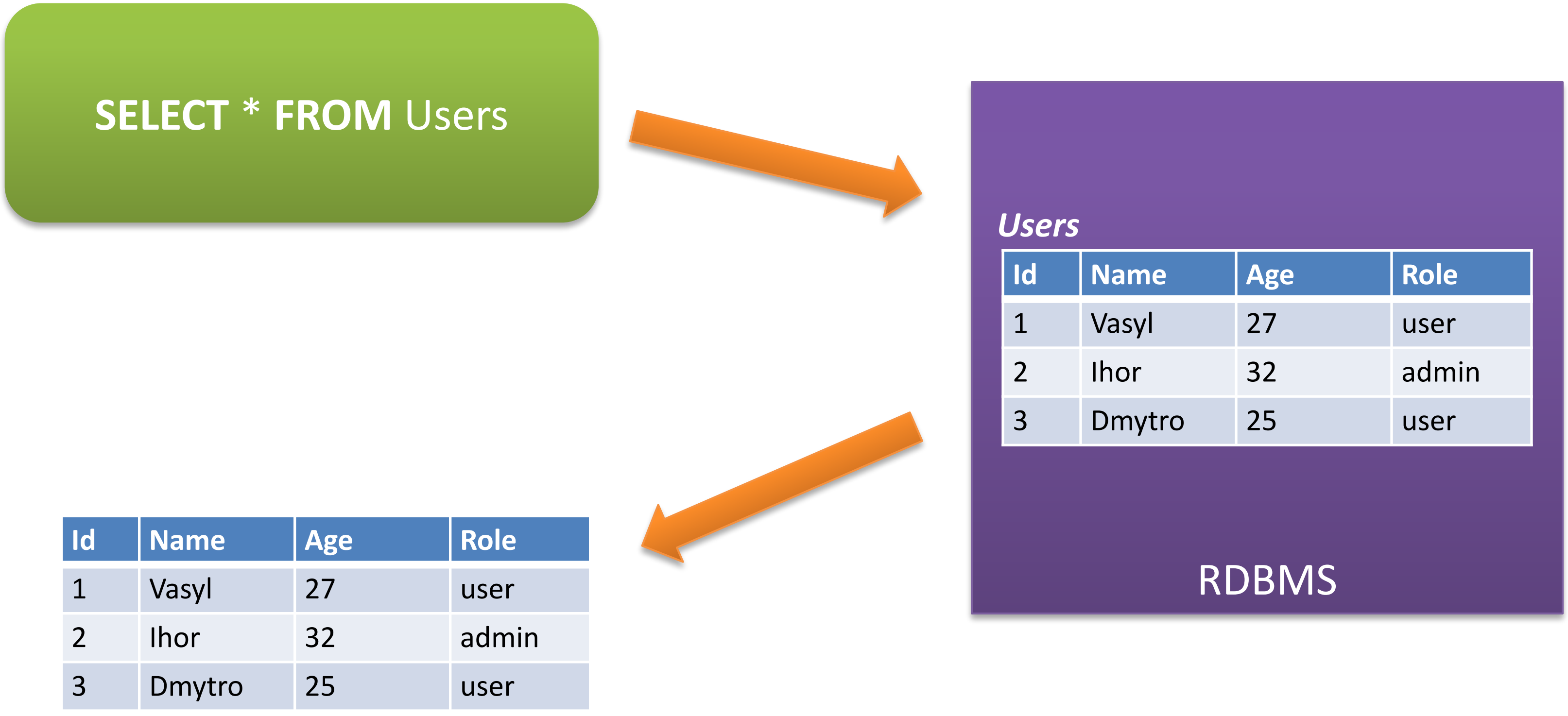
- **SQL** stands for Structured Query Language.

SELECT

SELECT Statement

SQL **SELECT** Statement is used to fetch the data from a database table which returns data in the form of result table. These result tables are called result-sets.

Example #1



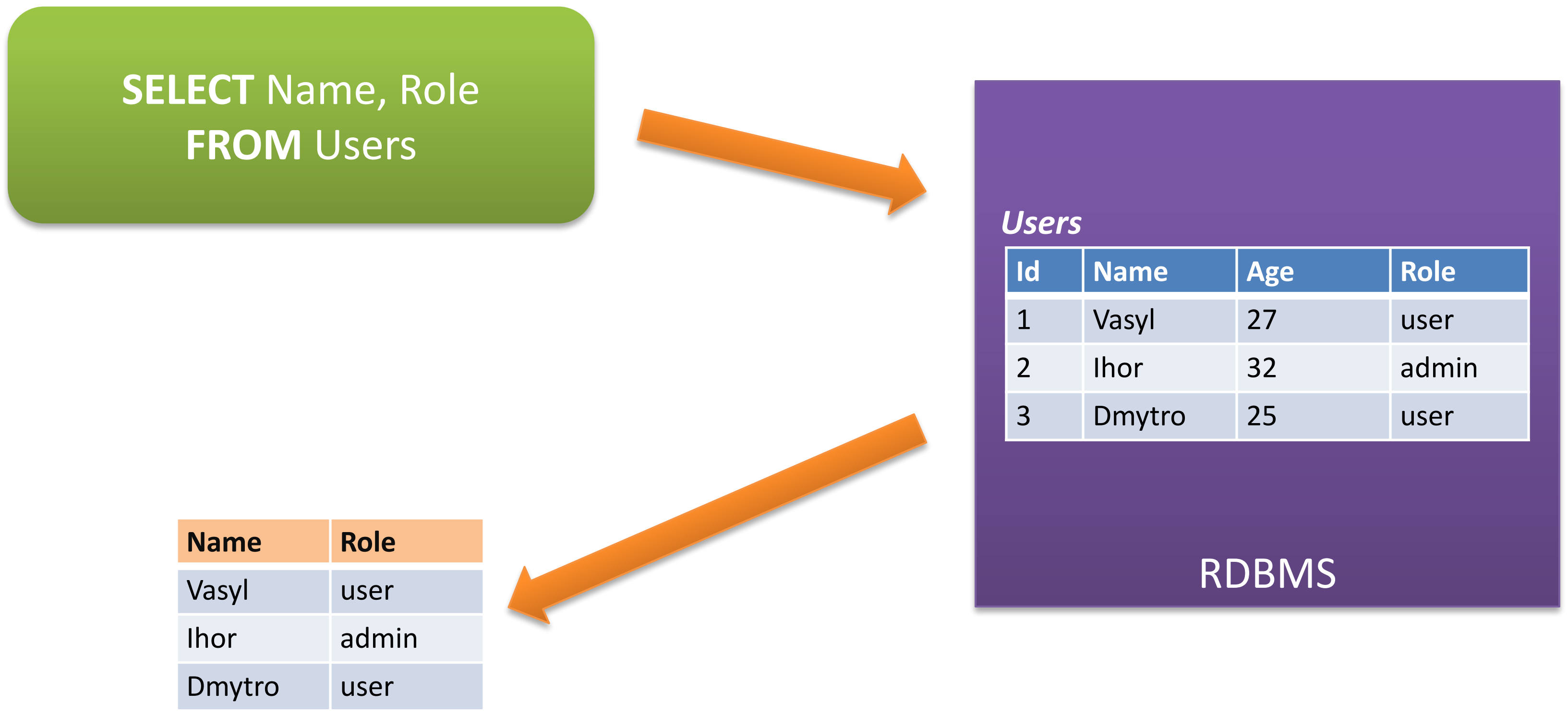
```
SELECT * FROM Users
```

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user

Example #2



```
SELECT Name, Role
FROM Users
```

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user

Name	Role
Vasyl	user
Ihor	admin
Dmytro	user

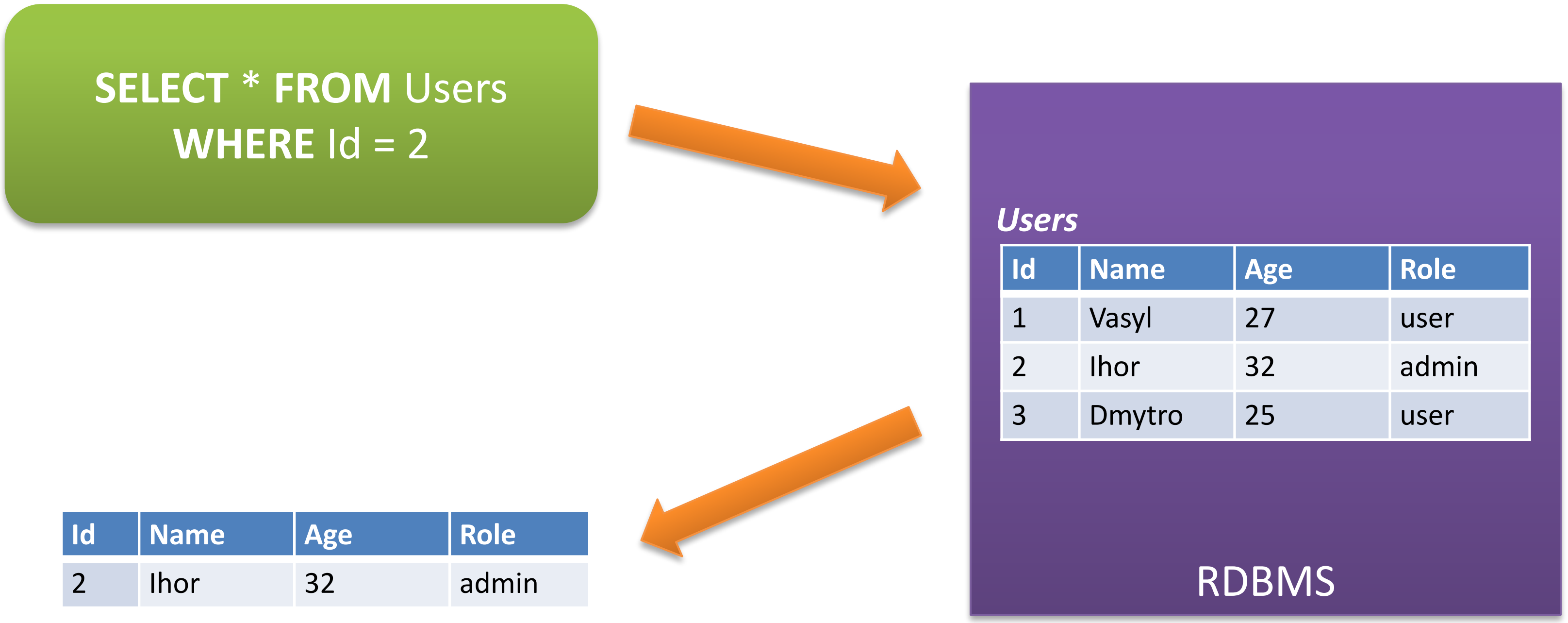
WHERE

- The SQL **WHERE** clause is used to specify a condition while fetching the data from single table or joining with multiple table.
- If the given condition is satisfied then only it returns specific value from the table. You would use WHERE clause to filter the records and fetching only necessary records.

Syntax:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

Example #3



SELECT * FROM Users
WHERE Id = 2

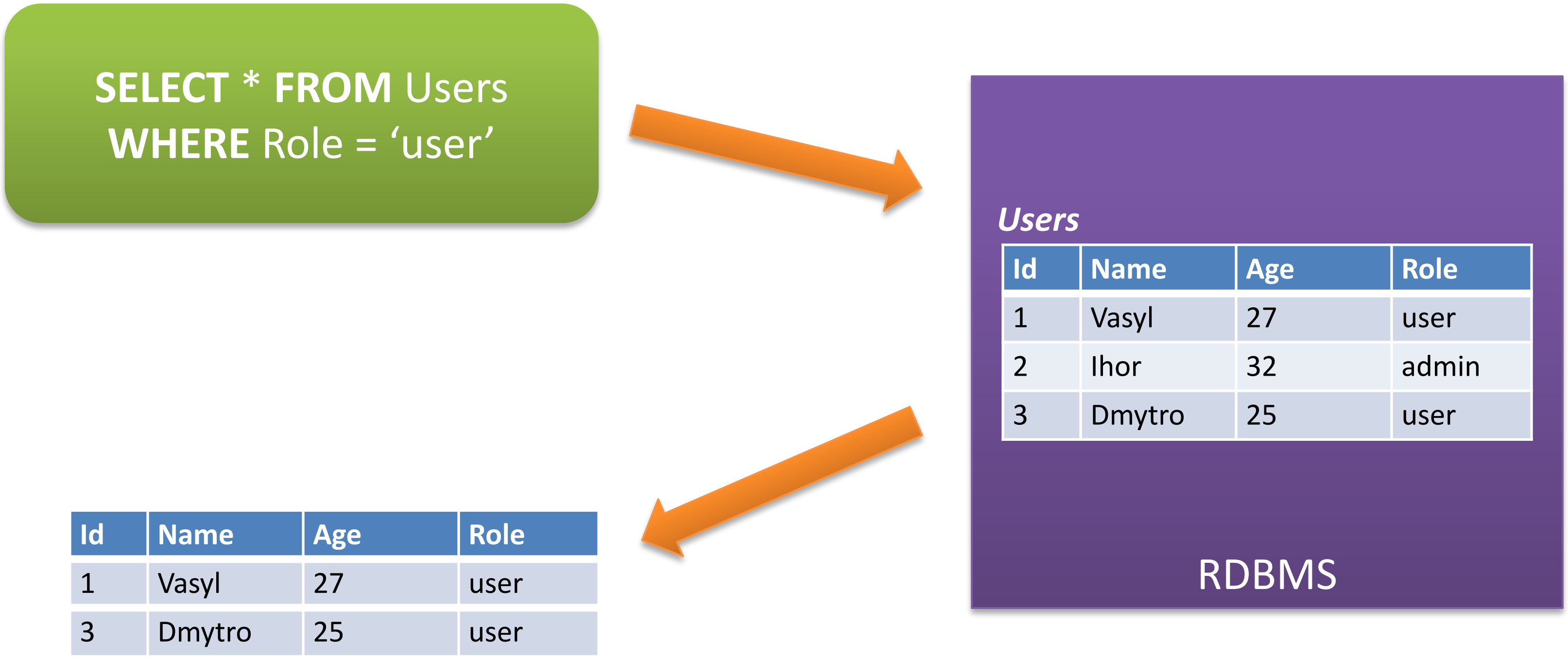
Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user

RDBMS

Id	Name	Age	Role
2	Ihor	32	admin

Example #4

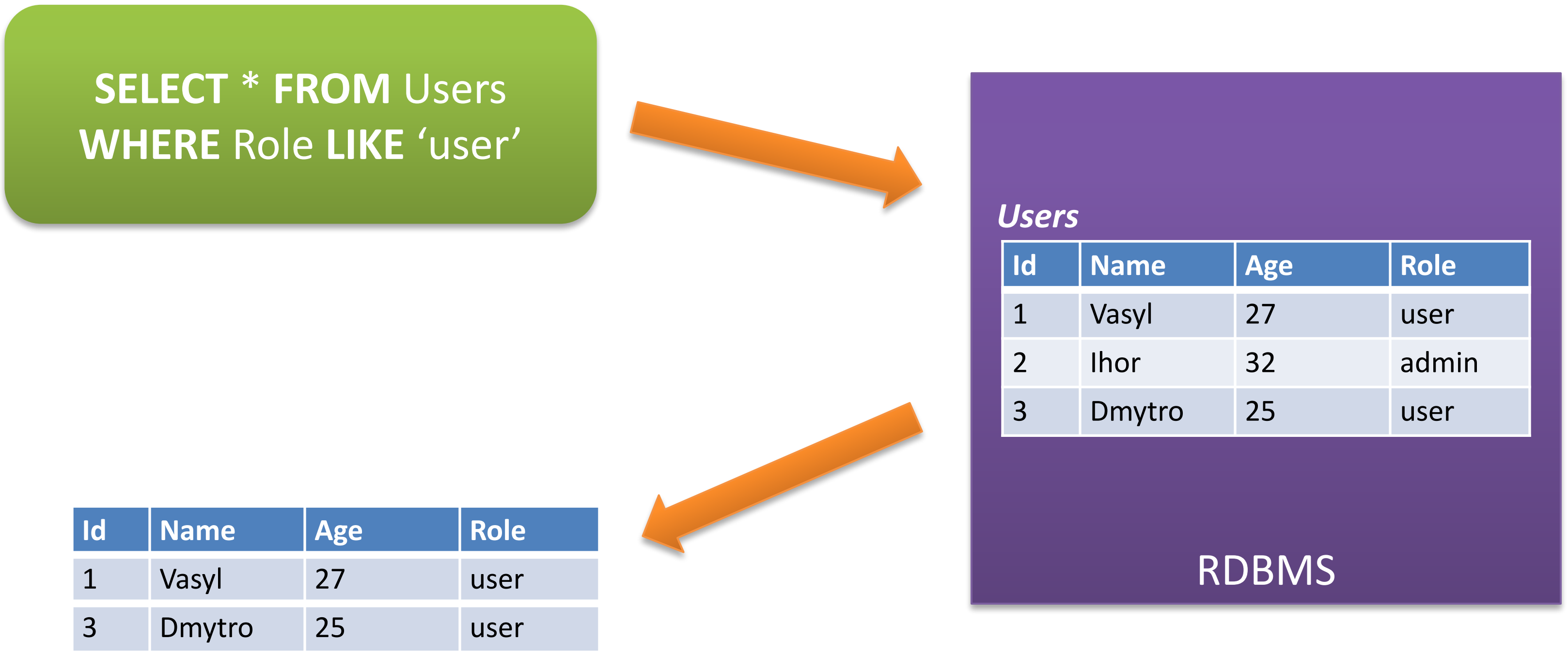


```
SELECT * FROM Users
WHERE Role = ‘user’
```

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user

Id	Name	Age	Role
1	Vasyl	27	user
3	Dmytro	25	user

Example #5



```
SELECT * FROM Users
WHERE Role LIKE 'user'
```

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user

Id	Name	Age	Role
1	Vasyl	27	user
3	Dmytro	25	user

LIKE operator

- The SQL **LIKE** operator is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator:
  - The percent sign (%)
  - The underscore (_)
- The percent sign represents zero, one, or multiple characters. The underscore represents a single number or character. The symbols can be used in combinations.

LIKE Example

1. `WHERE NOTE LIKE '200%`‘

   Finds any values that start with 200

2. `WHERE NOTE LIKE '%200%`‘

   Finds any values that have 200 in any position

3. `WHERE NOTE LIKE '_00%`‘

   Finds any values that have 00 in the second and third positions

4. `WHERE NOTE LIKE '2_%_%`‘

   Finds any values that start with 2 and are at least 3 characters in length

5. `WHERE NOTE LIKE '_2%3`‘

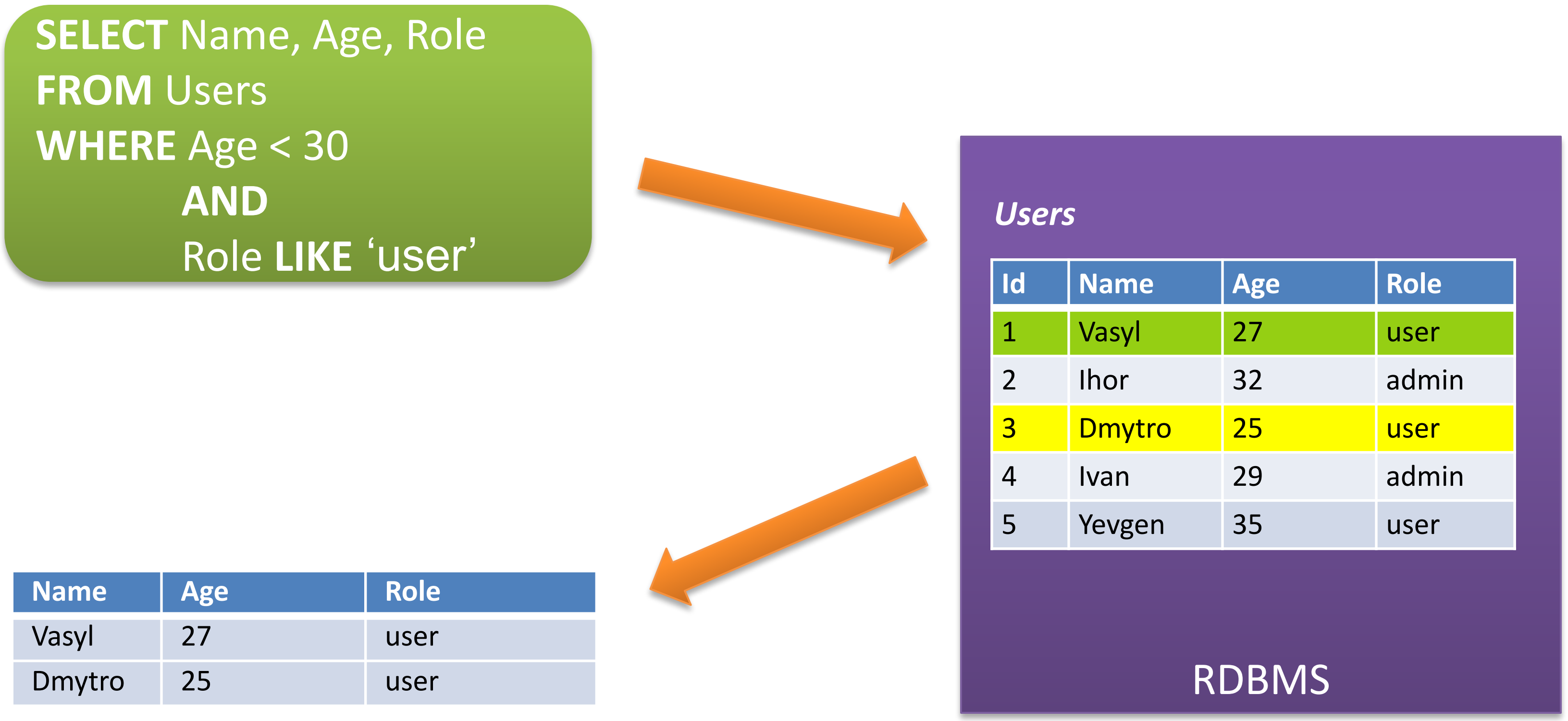
   Finds any values that have a 2 in the second position and end with a 3

`WHERE NOTE LIKE '2   3`‘

AND & OR operators

- The SQL **AND** and **OR** operators are used to combine multiple conditions to narrow data in an SQL statement.
- These two operators are called conjunctive operators.
- These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

Example #6



```
SELECT Name, Age, Role
FROM Users
WHERE Age < 30
         AND
         Role LIKE 'user'
```

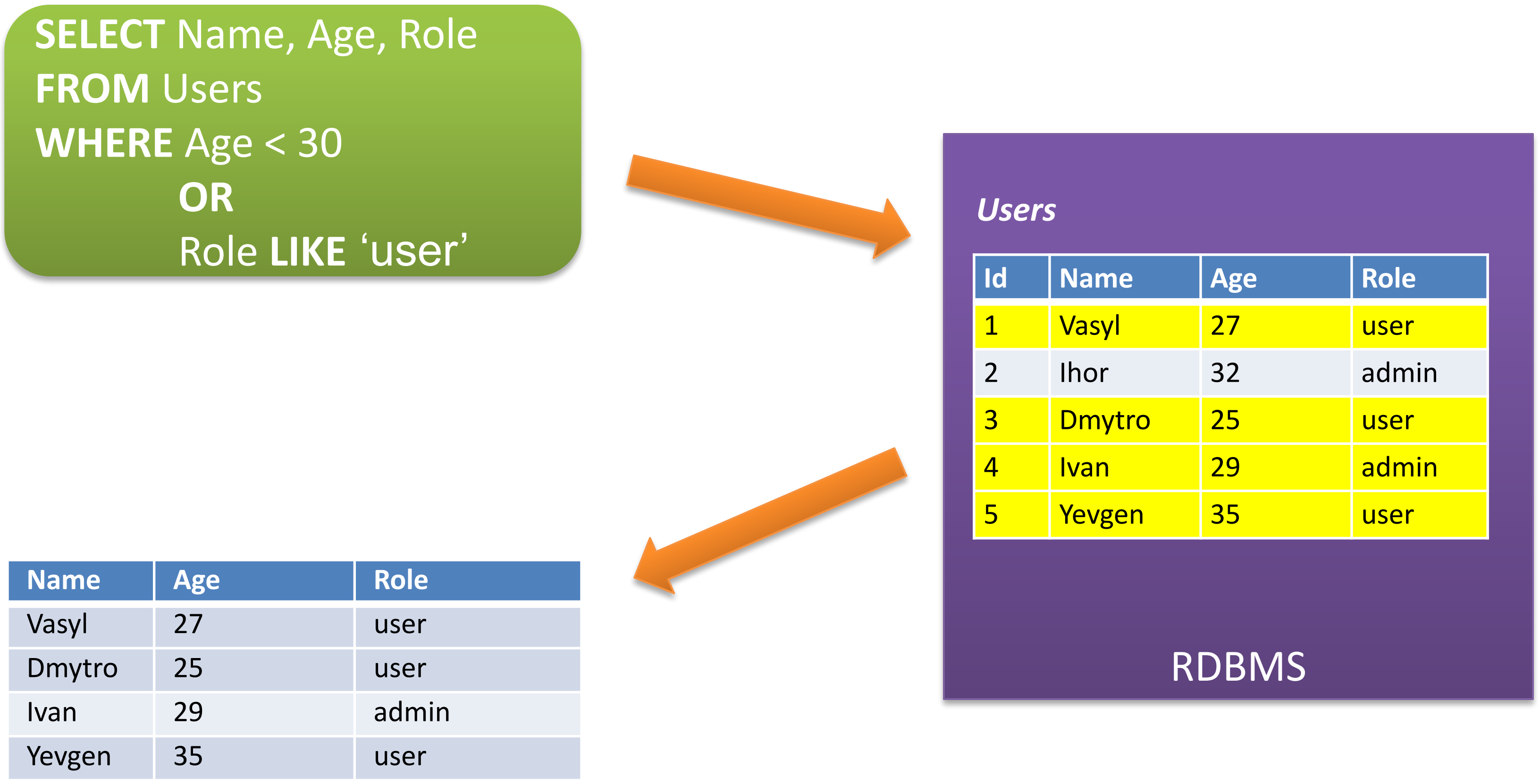
Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user
4	Ivan	29	admin
5	Yevgen	35	user

RDBMS

Name	Age	Role
Vasyl	27	user
Dmytro	25	user

Example #7



```
SELECT Name, Age, Role
FROM Users
WHERE Age < 30
          OR
          Role LIKE 'user'
```

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user
4	Ivan	29	admin
5	Yevgen	35	user

RDBMS

Name	Age	Role
Vasyl	27	user
Dmytro	25	user
Ivan	29	admin
Yevgen	35	user

TOP clause

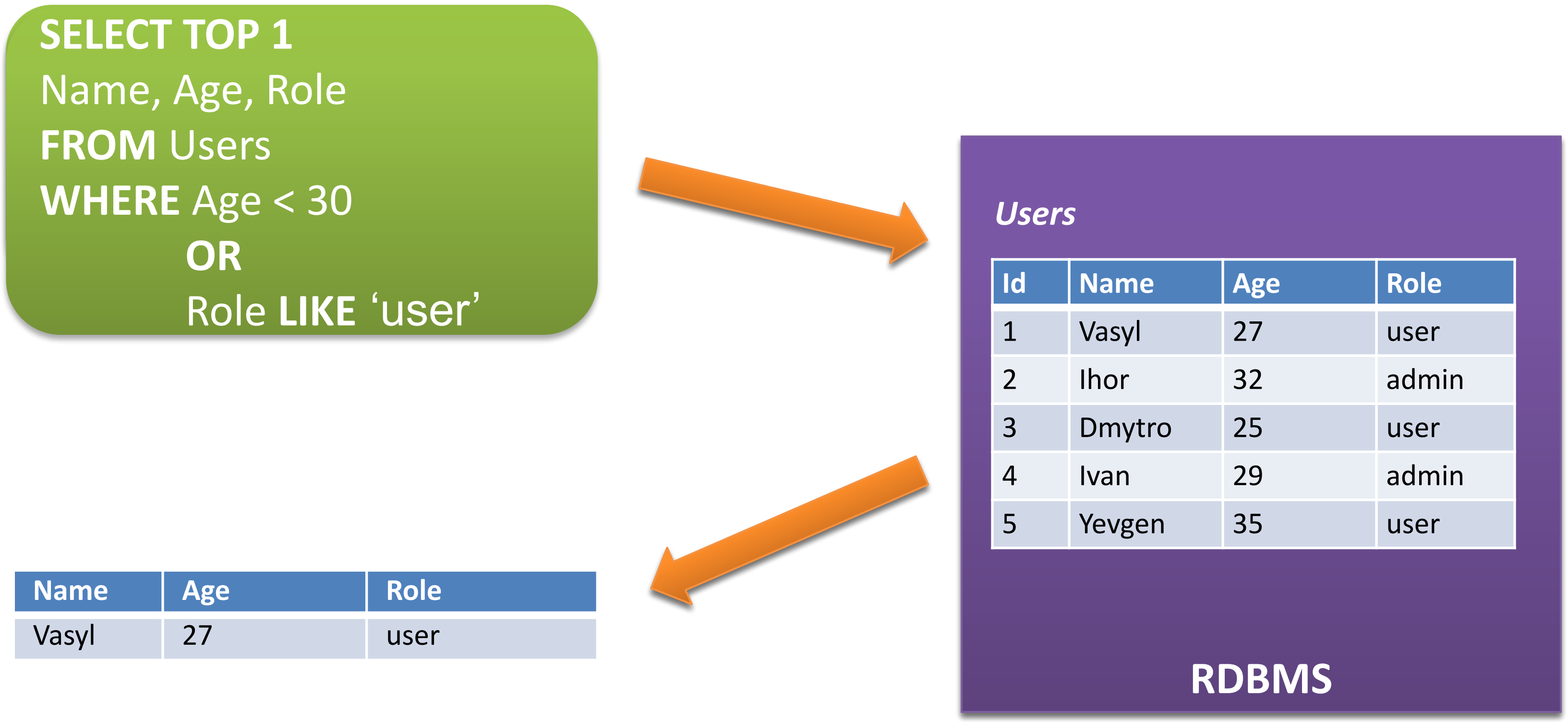
- The SQL **TOP** clause is used to fetch a TOP N number or X percent records from a table.

Note: *All the databases do not support TOP clause. For example MySQL supports* ***LIMIT*** *clause to fetch limited number of records and Oracle uses* ***ROWNUM*** *to fetch limited number of records.*

Syntax:

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE [condition]
```

Example #8



```
SELECT TOP 1
Name, Age, Role
FROM Users
WHERE Age < 30
        OR
        Role LIKE ‘user’
```

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user
4	Ivan	29	admin
5	Yevgen	35	user

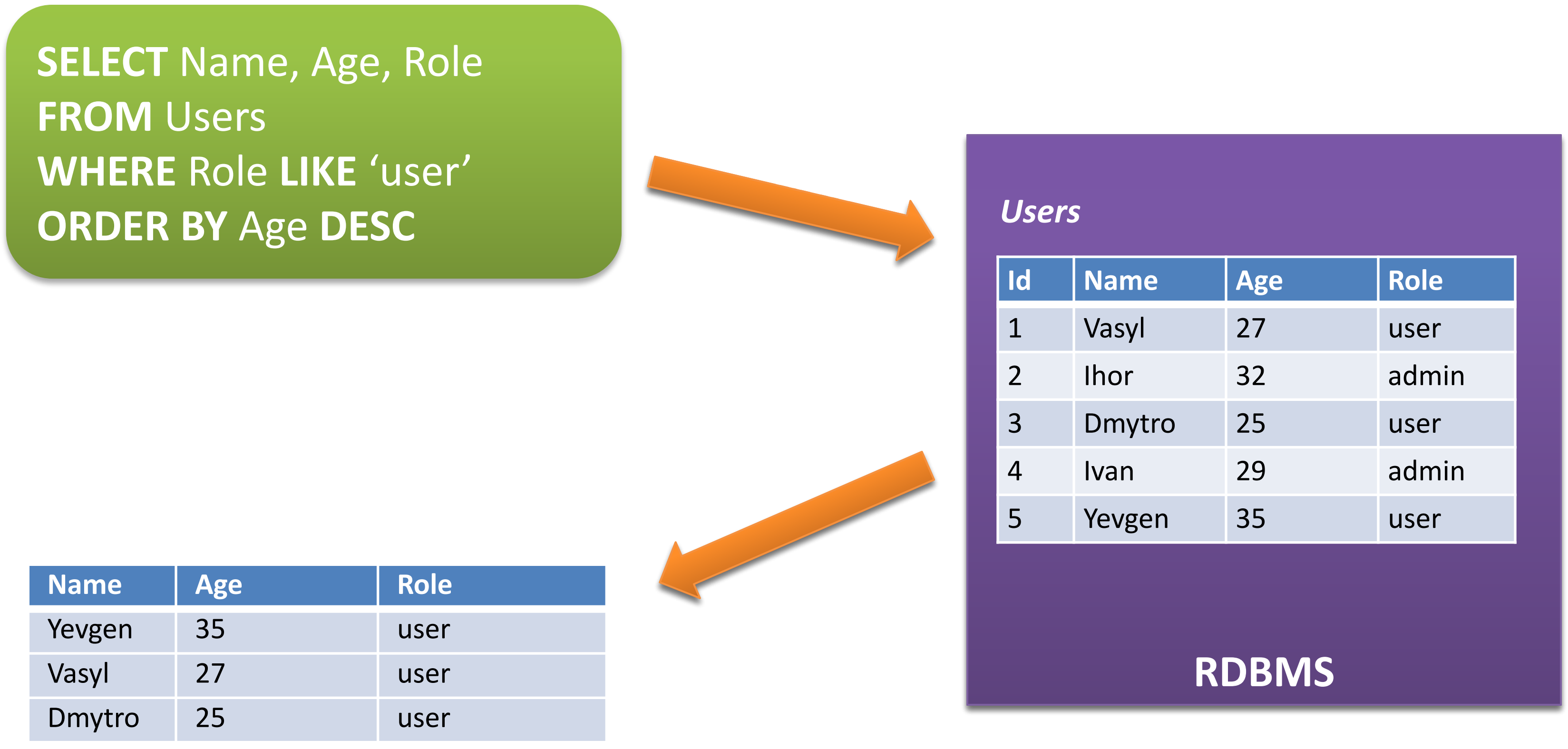
RDBMS

Name	Age	Role
Vasyl	27	user

ORDER BY

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. Some database sorts query results in ascending order by default.

Example #9



```
SELECT Name, Age, Role
FROM Users
WHERE Role LIKE 'user'
ORDER BY Age DESC
```

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user
4	Ivan	29	admin
5	Yevgen	35	user

RDBMS

Name	Age	Role
Yevgen	35	user
Vasyl	27	user
Dmytro	25	user

Example #10

```
SELECT Name, Age, Role
FROM Users
WHERE Role LIKE ‘user’
ORDER BY Age DESC,
         Name ASC
```

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user
4	Ivan	29	admin
5	Yevgen	35	user
6	Andriy	27	user

RDBMS

Name	Age	Role
Yevgen	35	user
Andriy	27	user
Vasyl	27	user
Dmytro	25	user

Aggregate functions

Aggregate functions perform a calculation on a set of values and return a single value

- **SUM** – returns the sum
- **COUNT** – returns the number of rows
- **AVG** – returns the average value
- **MIN** – returns the smallest value
- **MAX** – returns the largest value

Example #11

```
SELECT MAX(Age), MIN(Age)
FROM Users
```

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user
4	Ivan	29	admin
5	Yevgen	35	user

RDBMS

MAX OF Age	MIN OF Age
35	25

GROUP BY

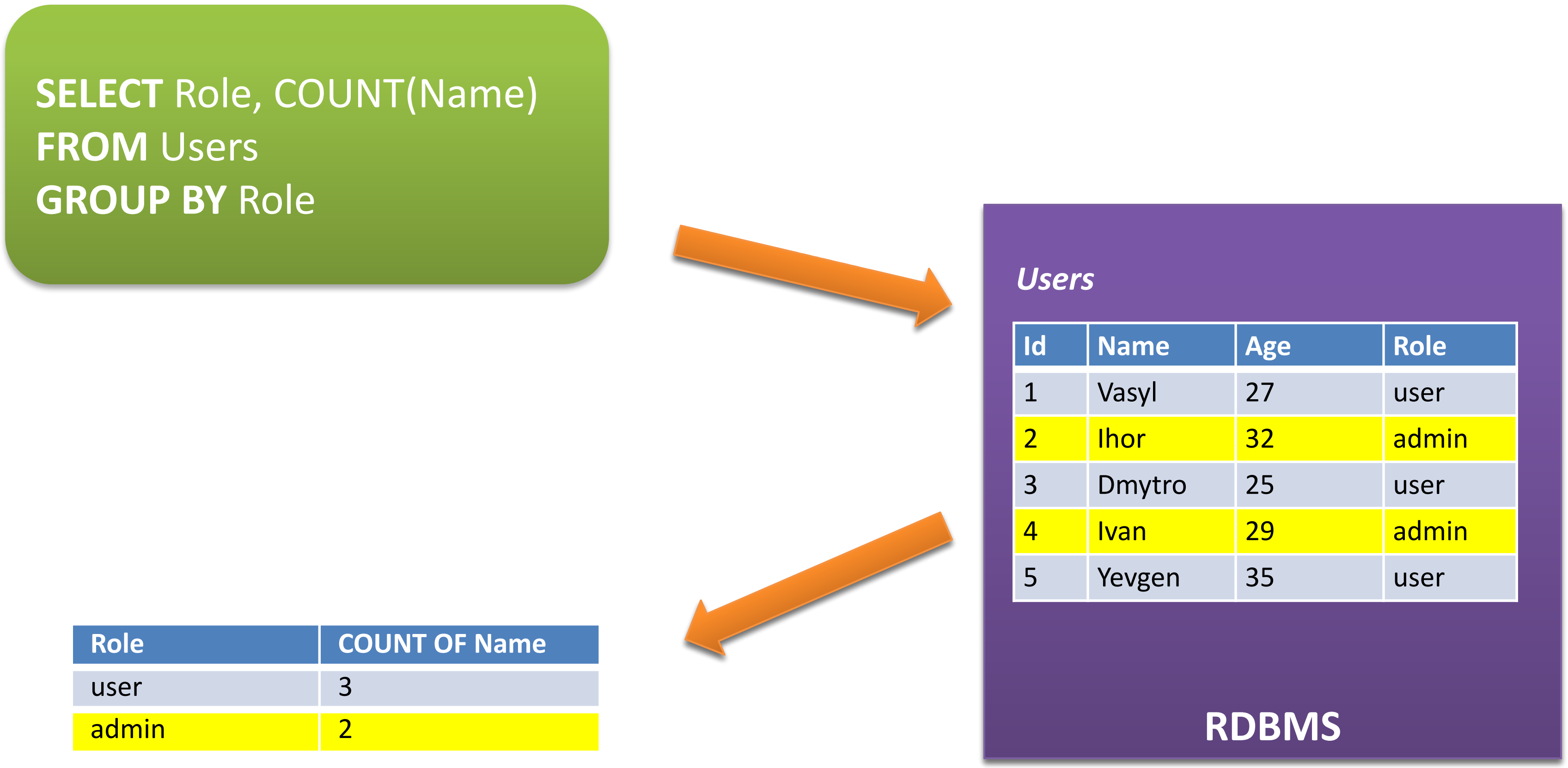
- The **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups.
- The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

GROUP BY Syntax

- The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

Example #12



```
SELECT Role, COUNT(Name)
FROM Users
GROUP BY Role
```

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user
4	Ivan	29	admin
5	Yevgen	35	user

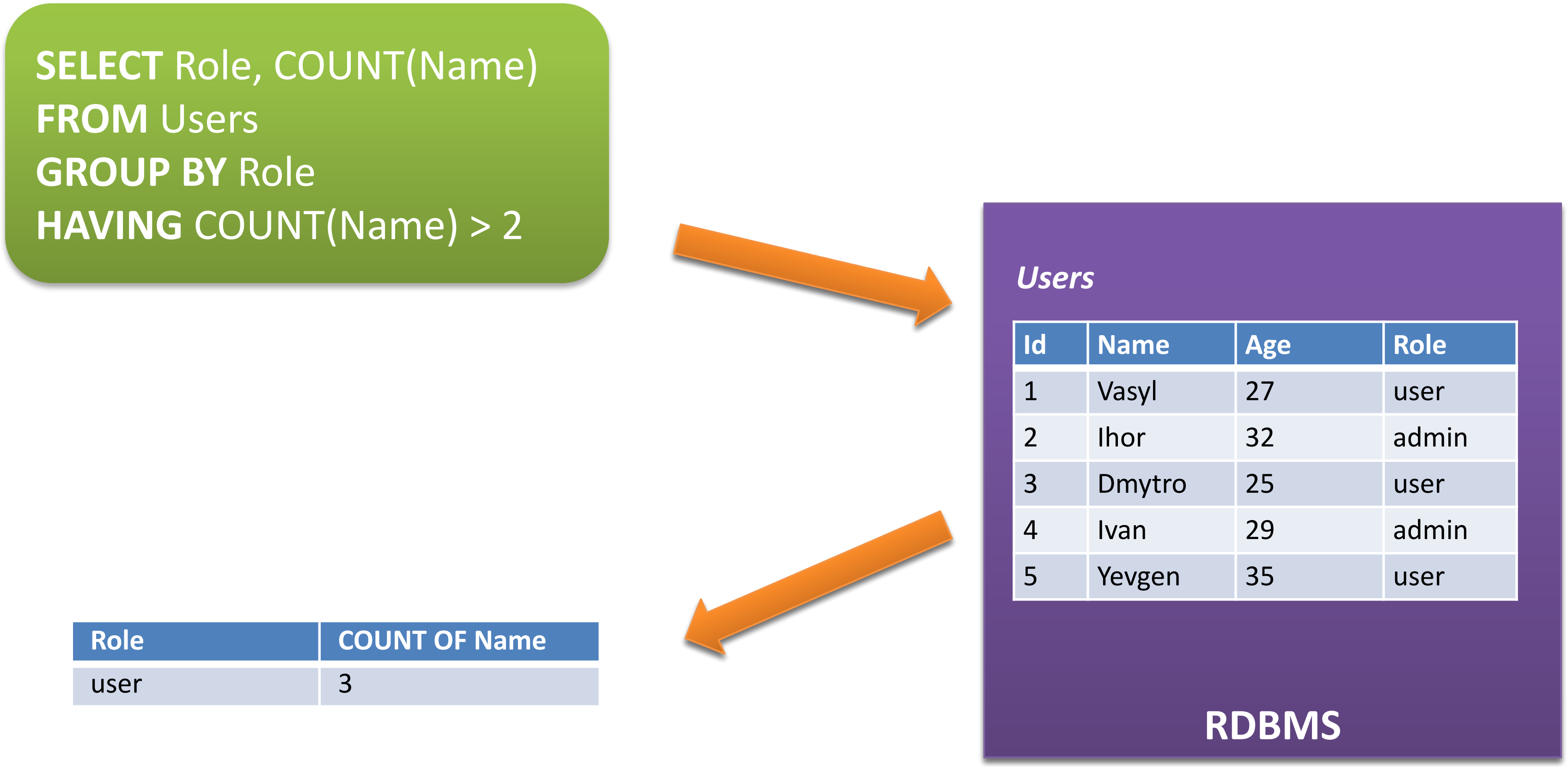
RDBMS

Role	COUNT OF Name
user	3
admin	2

HAVING

- The HAVING clause enables you to specify conditions that filter which group results appear in the final results.
- The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

Example #13



```
SELECT Role, COUNT(Name)
FROM Users
GROUP BY Role
HAVING COUNT(Name) > 2
```

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user
4	Ivan	29	admin
5	Yevgen	35	user

Role	COUNT OF Name
user	3

DISTINCT

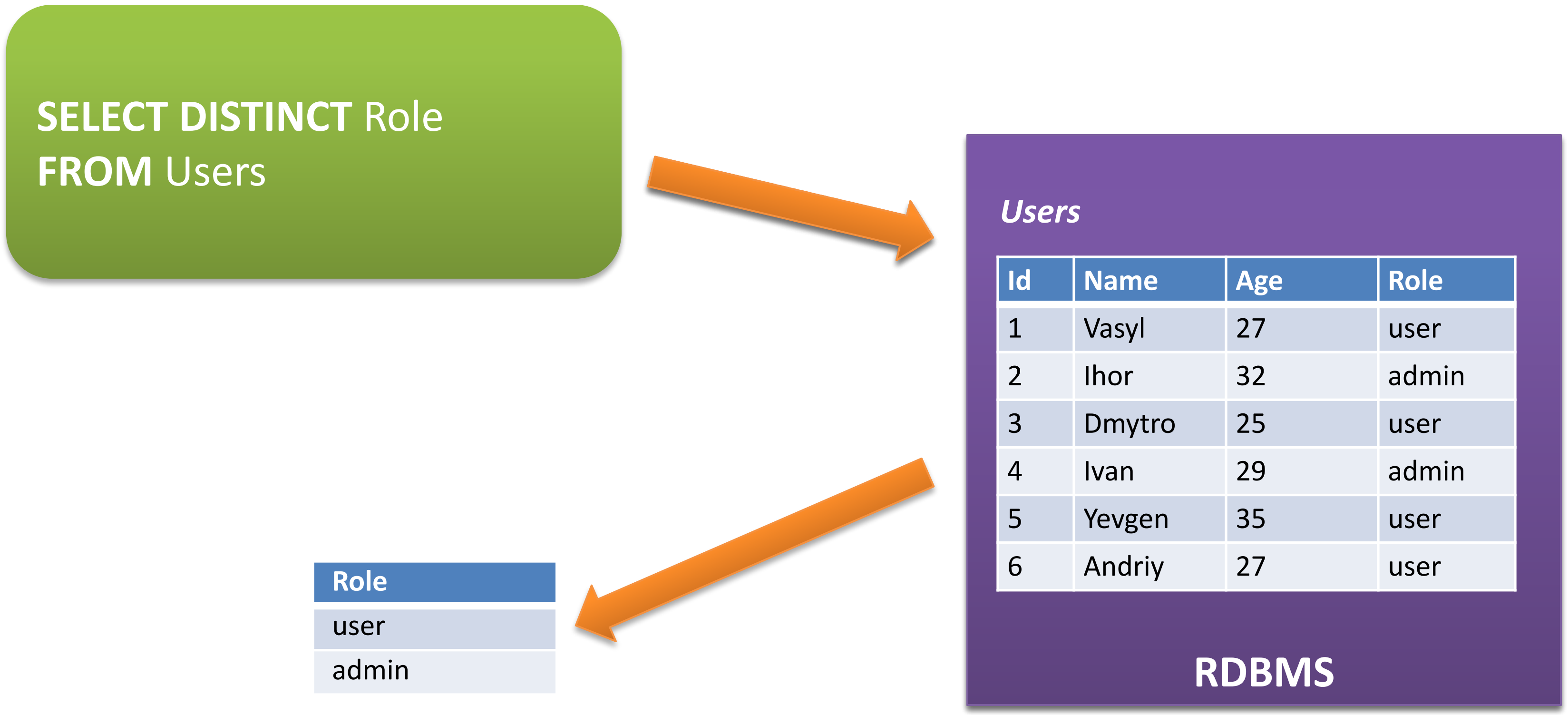
- The SQL **DISTINCT** keyword is used in conjunction with SELECT statement to eliminate all the duplicate records and fetching only unique records.

- There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only unique records instead of fetching duplicate records.

Syntax:

```
SELECT DISTINCT column_name1,column_name2
  FROM table_name
```

Example #14



SELECT DISTINCT Role
FROM Users

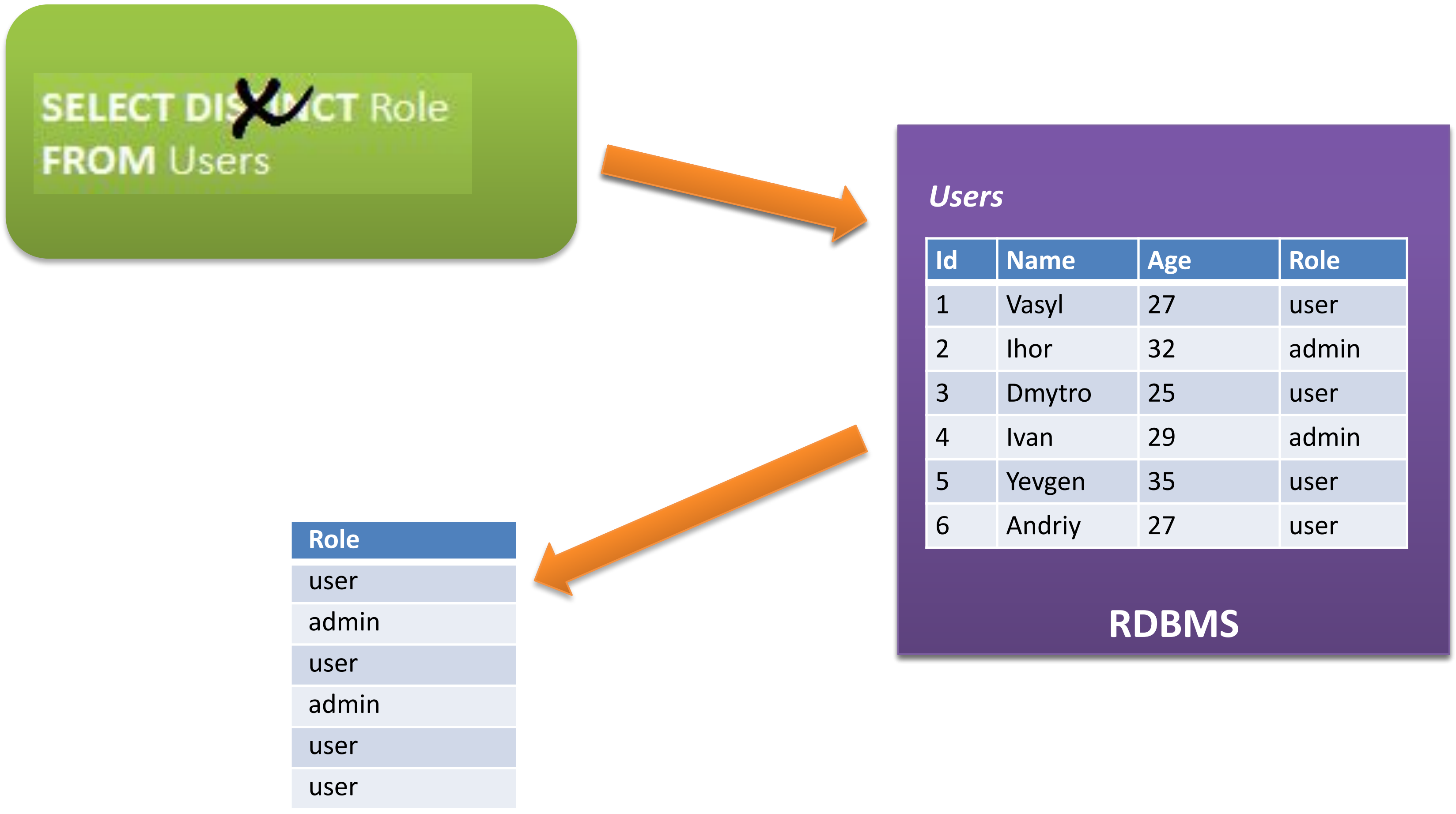
Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user
4	Ivan	29	admin
5	Yevgen	35	user
6	Andriy	27	user

RDBMS

Role
user
admin

Example #15



```
SELECT DIS~~TIN~~CT Role
FROM Users
```

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user
4	Ivan	29	admin
5	Yevgen	35	user
6	Andriy	27	user

RDBMS

Role
user
admin
user
admin
user
user

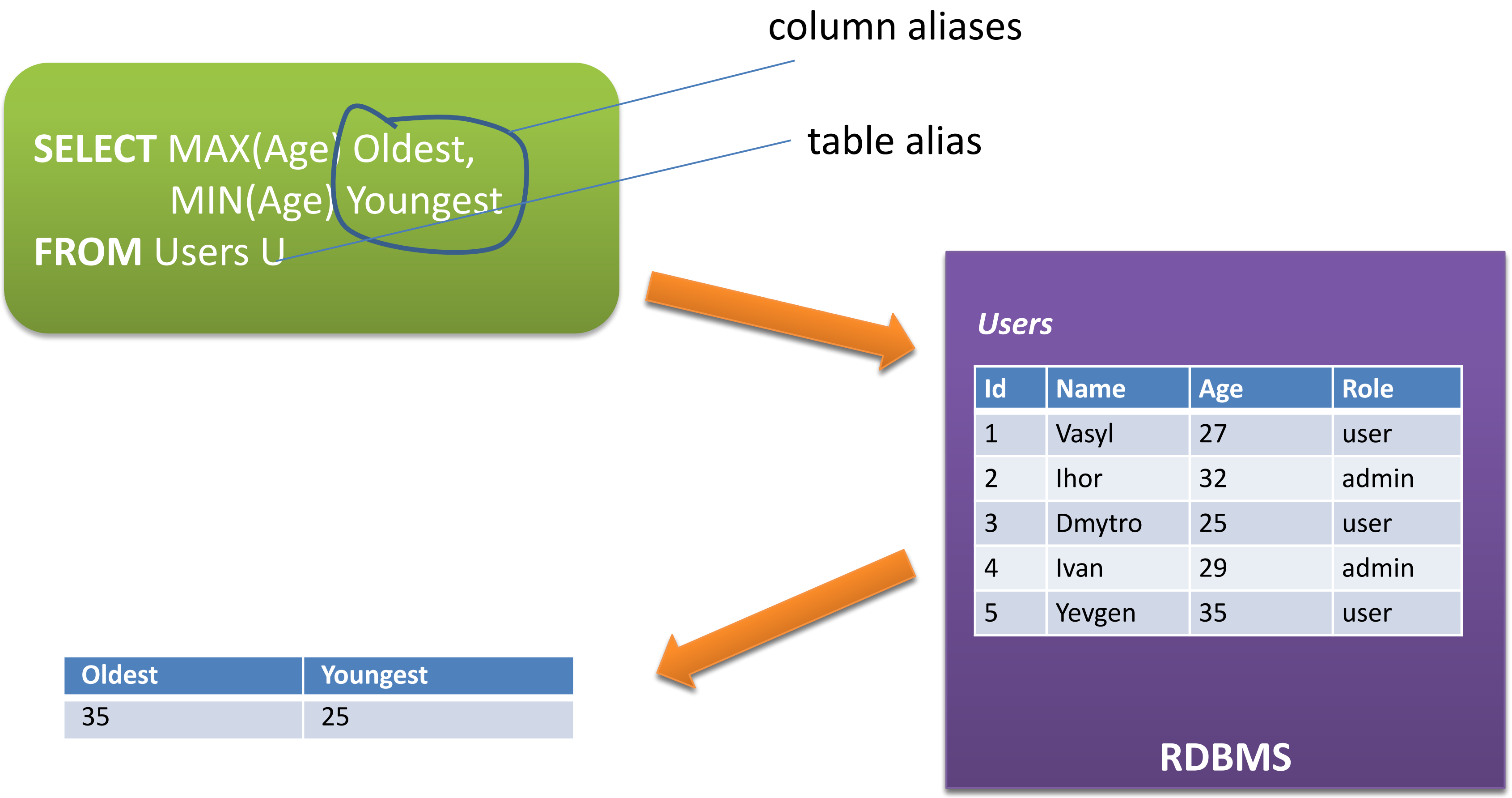
Using Aliases

The readability of a SELECT statement can be improved by giving a table an alias:

- **table_name** AS *table alias*
- **table_name** *table_alias*

You can also create aliases for column names to make it easier to work with column names, calculations, and summary values

Example #16



```
SELECT MAX(Age) Oldest,
       MIN(Age) Youngest
FROM Users U
```

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user
4	Ivan	29	admin
5	Yevgen	35	user

Oldest	Youngest
35	25

Subqueries and Union

Subqueries

- A Subquery, or Inner query, or 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.

Rules for using subqueries

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery; however, the BETWEEN can be used within the subquery.

Example #17

```
SELECT RegistrationDate
FROM Profiles
WHERE UserId IN
        (
        SELECT Id
        FROM Users
        WHERE Age < 30
        )
```

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user
4	Ivan	29	admin
5	Yevgen	35	user

Profiles

Id	RegistrationDate	UserId
10	05/03/12	2
11	05/05/12	1
12	05/29/12	3
13	05/29/12	5
14	06/01/12	6

RDBMS

RegistrationDate
05/05/12
05/29/12

UNION CLAUSE

- The SQL **UNION** clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.
- To use UNION, each SELECT must have the same number of columns selected, the same number of column expressions, the same data type, and have them in the same order but they do not have to be the same length.

UNION Syntax

```
SELECT column1, column2
 FROM table_name
 WHERE [ conditions ]
UNION [ ALL ]
SELECT column1, column2
 FROM table_name
 WHERE [ conditions ];
```

Any duplicate records are automatically removed unless UNION ALL is used. And sometimes UNION ALL may be much faster than plain UNION.

JOINS

Using Joins

The **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Example #18

```
SELECT Name, Age, RegistrationDate
FROM Users, Profiles
WHERE Users.Id = Profiles. UserId
```

or, using aliases

```
SELECT U,Name, U.Age, P.RegistrationDate
FROM Users U, Profiles P
WHERE U.Id = P. UserId
```

Name	Age	RegistrationDate
Vasyl	27	05/05/12
Ihor	32	05/03/12
Dmytro	25	05/29/12
Yevgen	35	05/29/12

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user
4	Ivan	29	admin
5	Yevgen	35	user

Profiles

Id	RegistrationDate	UserId
10	05/03/12	2
11	05/05/12	1
12	05/29/12	3
13	05/29/12	5
14	06/01/12	6

RDBMS

Example #19

```
SELECT Name, Age, RegistrationDate
FROM Users INNER JOIN Profiles
   ON Users.Id = Profiles. UserId
```

or, using aliases

```
SELECT U.Name, U.Age, P.RegistrationDate
FROM Users U INNER JOIN Profiles P
   ON U.Id = P. UserId
```

Name	Age	RegistrationDate
Vasyl	27	05/05/12
Ihor	32	05/03/12
Dmytro	25	05/29/12
Yevgen	35	05/29/12

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user
4	Ivan	29	admin
5	Yevgen	35	user

Profiles

Id	RegistrationDate	UserId
10	05/03/12	2
11	05/05/12	1
12	05/29/12	3
13	05/29/12	5
14	06/01/12	6

RDBMS

Example #20

```
SELECT Name, Age, RegistrationDate
FROM Users INNER JOIN Profiles
    ON Users.Id = Profiles. UserId
WHERE User.Age < 30
```

or, using aliases

```
SELECT U.Name, U.Age, P.RegistrationDate
FROM Users U INNER JOIN Profiles P
    ON U.Id = P. UserId
WHERE User.Age < 30
```

Name	Age	RegistrationDate
Vasyl	27	05/05/12
Dmytro	25	05/29/12

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user
4	Ivan	29	admin
5	Yevgen	35	user

Profiles

Id	RegistrationDate	UserId
10	05/03/12	2
11	05/05/12	1
12	05/29/12	3
13	05/29/12	5
14	06/01/12	6

RDBMS

SQL Join Types

- **INNER JOIN** (or just **JOIN**): returns rows when there is a match in both tables.
- **LEFT JOIN**: returns all rows from the left table, even if there are no matches in the right table.
- **RIGHT JOIN**: returns all rows from the right table, even if there are no matches in the left table.
- **FULL JOIN**: returns rows when there is a match in one of the tables.

Example #21

```
SELECT Name, Age, RegistrationDate
FROM Users LEFT JOIN Profiles
    ON Users.Id = Profiles. UserId
```

Name	Age	RegistrationDate
Vasyl	27	05/05/12
Ihor	32	05/03/12
Dmytro	25	05/29/12
Ivan	29	NULL
Yevgen	35	05/29/12

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user
4	Ivan	29	admin
5	Yevgen	35	user

Profiles

Id	RegistrationDate	UserId
10	05/03/12	2
11	05/05/12	1
12	05/29/12	3
13	05/29/12	5
14	06/01/12	6

RDBMS

Example #22

```
SELECT Name, Age, RegistrationDate
FROM Users RIGHT JOIN Profiles
   ON Users.Id = Profiles. UserId
```

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user
4	Ivan	29	admin
5	Yevgen	35	user

Profiles

Id	RegistrationDate	UserId
10	05/03/12	2
11	05/05/12	1
12	05/29/12	3
13	05/29/12	5
14	06/01/12	6

RDBMS

Name	Age	RegistrationDate
Vasyl	27	05/05/12
Ihor	32	05/03/12
Dmytro	25	05/29/12
Yevgen	35	05/29/12
NULL	NULL	06/01/12

Example #23

```
SELECT Name, Age, RegistrationDate
FROM Users FULL JOIN Profiles
    ON Users.Id = Profiles. UserId
```

Users

Id	Name	Age	Role
1	Vasyl	27	user
2	Ihor	32	admin
3	Dmytro	25	user
4	Ivan	29	admin
5	Yevgen	35	user

Profiles

Id	RegistrationDate	UserId
10	05/03/12	2
11	05/05/12	1
12	05/29/12	3
13	05/29/12	5
14	06/01/12	6

RDBMS

Name	Age	RegistrationDate
Vasyl	27	05/05/12
Ihor	32	05/03/12
Dmytro	25	05/29/12
Ivan	29	NULL
Yevgen	35	05/29/12
NULL	NULL	06/01/12

NULL Values

- The SQL NULL is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank.

- A field with a NULL value is a field with no value. It’s very important to understand that a NULL value is different than a zero value or a field that contains spaces.

Example #24

```
SELECT  Id, Name, Age, Role
    FROM Users
    WHERE Role IS NOT NULL;
```

UPDATE
INSERT
DELETE

UPDATE Statement

SQL **UPDATE** statement is used to change existing data in a table.

Syntax:

```
UPDATE table_name
SET
   column1 = value,
 column2 = value2,
 ...
WHERE [condition]
```

Example #25

```
UPDATE Users
SET Role = ‘admin’
WHERE id = 3
```

```
UPDATE Users
SET Age = Age + 1
```

INSERT Statement

SQL **INSERT** statement is used to insert new data into a table.

Syntax:

```
INSERT INTO table_name
 (column1, column2, column3,...)
VALUES
 (value1, value2, value3,...)
```

or

```
INSERT INTO table_name
 (column1, column2, column3,...)
[SELECT statement]
```

Example #26

```
INSERT INTO Users
    (Name, Age, Role)
VALUES
    (‘Alan’, 42, ‘boss’)
```

```
INSERT INTO Users
    (Name, Age, Role)
SELECT Name, Age, ‘trainee’
FROM Candidates
WHERE Age > 18
```

DELETE Statement

SQL **DELETE** Statement is used to delete some data from a table.

Syntax:

```
DELETE FROM table_name
WHERE [condition]
```

Example #27

```
DELETE FROM Users
WHERE Role LIKE ‘looser’
```

```
DELETE FROM Users
WHERE Age > 60 OR Age < 18
```

Cautions for UPDATE & DELETE

Be careful when using UPDATE and DELETE statements especially if you are a beginner with SQL. If you make a mistake, you can lose your data ☹.

- Execute an appropriate SELECT statement before executing an UPDATE or DELETE statement and verify the count of rows to be affected.
- Never use UPDATE and DELETE without WHERE clause, otherwise the whole table will be changed (emptied).

More information

- http://www.w3schools.com/sql/default.asp
- http://www.firstsql.com/tutor2.htm
- http://beginner-sql-tutorial.com/sql-select-statement.htm

SoftServe | *Empowering your Business through Software Development*

Thank you

US OFFICES
Austin, TX
Fort Myers, FL
Boston, MA
Newport Beach, CA
Salt Lake City, UT

EUROPE OFFICES
United Kingdom
Germany
The Netherlands
Ukraine
Bulgaria

EMAIL
info@softserveinc.com

WEBSITE:
www.softserveinc.com

USA TELEPHONE
Toll-Free: 866.687.3588
Office: 239.690.3111

UK TELEPHONE
Tel: 0207.544.8414

GERMAN TELEPHONE
Tel: 0692.602.5857