

Database Management System

IT 261

Bachelor of Information Technology

Prepared by Dr. Khaled Eskaf

Lecture 7



Revision

The SELECT DISTINCT statement **retrieves distinct** values from a database table.

-- select the unique ages from the Customers table

```
SELECT DISTINCT age  
FROM Customers;
```

Here, the SQL command selects only the unique values of age from the Customers table.

Syntax of SQL SELECT DISTINCT

```
SELECT DISTINCT column1, column2 ... FROM table;
```

Here,

- column1, column2, ... are the table columns
- table is table name from where we retrieve the distinct columns

selects unique countries from the Customers table.

-- select the **unique countries** from the customers table

```
SELECT DISTINCT country  
FROM Customers;
```

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

SELECT DISTINCT country
FROM Customers;

country
USA
UK
UAE

We can also use SELECT DISTINCT with multiple columns. For example,

Selects rows if combinations of country and first_name are unique. Meaning, the result will include each pair of country and first_name only once.

-- select rows if the first name and country of a customer is unique

```
SELECT DISTINCT country, first_name  
FROM Customers;
```

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT DISTINCT country, first_name  
FROM Customers;
```

country	first_name
USA	John
USA	Robert
UK	David
UK	John
UAE	Betty

We can use SQL DISTINCT with the COUNT() function to count the number of unique rows.

```
-- count the unique countries where customers are from  
SELECT COUNT(DISTINCT country)  
FROM Customers;
```

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

SQL command returns the count of unique countries.

SELECT COUNT(DISTINCT country)
FROM Customers;

COUNT(DISTINCT country)

3

The AS keyword is used to give columns or tables a temporary name that can be used to identify that column or table later.

Here, the SQL command selects the first_name column from the Customers table. However, the column name is changed to name in the result set.

```
SELECT first_name AS name  
FROM Customers;
```

SQL AS Alias Syntax

The syntax of the SQL AS command is:

```
SELECT column_1 AS alias_1,  
       column_2 AS alias_2,  
... .. column_n AS alias_n  
FROM table_name;
```

Here,

- column_1, column_2,...column_n are the table columns
- alias_1, alias_2,...alias_n are the aliases of the table columns

selects the first_name column of Customers.
However, the column name will change to name in the result set.

```
SELECT first_name AS name  
FROM Customers;
```

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT first_name AS name  
FROM Customers;
```

name
John
Robert
David
John
Betty

We can also use aliases with more than one column.

selects **customer_id** as **cid** and **first_name** as **name**.

```
SELECT customer_id AS cid, first_name AS name  
FROM Customers;
```

customer_id	first_name
1	John
2	Jane
3	Doe

cid	name
1	John
2	Jane
3	Doe

We can combine data from multiple columns and represent it in a single column using the `CONCAT()` function. For example,

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name FROM Customers;
```

Customers

first_name	last_name
John	Doe
Jane	Smith

full_name
John Doe
Jane Smith

SQL BETWEEN Syntax

```
SELECT column1, column2, ...  
FROM table  
WHERE column BETWEEN value1 AND value2;
```

- column1, column2, ... are the columns you want to filter
- table is the name of the table
- column is the name of the column where we want to specify a range of values
- BETWEEN is an operator used to specify a range of values for the column
- value1 and value2 are the lower and upper bounds of the range

select item and amount columns from Orders that have amounts between **300** and **500** (including **300** and **500**).

```
SELECT item, amount  
FROM Orders  
WHERE amount BETWEEN 300 AND 500;
```

Table: Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT item, amount  
FROM Orders  
WHERE amount BETWEEN 300 AND 500;
```

item	amount
Keyboard	400
Mouse	300
Keyboard	400

In SQL, the BETWEEN operator with the WHERE clause selects values within a given range.

Here, the SQL command selects all columns from the Orders table where the amount is between **200** and **600** (including **200** and **600**).

```
-- select rows where the amount is between 200 and 600
```

```
SELECT *  
FROM Orders  
WHERE amount BETWEEN 200 AND 600;
```

Orders

OrderID	CustomerID	Amount	OrderDate
1	1001	150	2023-01-01
2	1002	250	2023-01-02
3	1003	600	2023-01-03
4	1004	700	2023-01-04

OrderID	CustomerID	Amount	OrderDate
2	1002	250	2023-01-02
3	1003	600	2023-01-03

The NOT BETWEEN operator is used to exclude the rows that match the values in the range. It returns all the rows except the excluded rows.

Select all the items from Orders except the rows that have amounts between 300 and 500 (300 and 500 are also excluded).

-- exclude rows with amount between 300 and 500

```
SELECT item, amount  
FROM Orders  
WHERE amount NOT BETWEEN 300 AND 500;
```

Table: Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT item, amount  
FROM Orders  
WHERE amount NOT BETWEEN 300 AND 500;
```

item	amount
Monitor	12000
Mousepad	250

The BETWEEN operator also works with text data.

selects all orders where the item names begin with letters between I and L.

-- select rows where items begin with letters between 'I' and 'L'
-- exclude all items beginning with 'L' followed by other characters

```
SELECT item, amount  
FROM Orders  
WHERE item BETWEEN 'I' AND 'L';
```

Table: Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT item, amount  
FROM Orders  
WHERE item BETWEEN 'I' AND 'L';
```

item	amount
Keyboard	400
Keyboard	400

In SQL, we can also use BETWEEN to filter data between two dates.

selected the teams who registered between 2021-01-01 and 2022-11-01.

```
-- get the records of those teams  
-- who registered between given dates
```

```
SELECT *  
FROM Teams  
WHERE registered BETWEEN '2021-01-01' AND '2022-11-01';
```

In SQL, IS NULL and IS NOT NULL are used to check if a column in a table contains a NULL value or not.

IS NULL Syntax

In SQL, the IS NULL condition is used to select rows if the specified field is NULL. It has the following syntax:

```
SELECT column1, column2, ...  
FROM table  
WHERE column_name IS NULL;
```

Here,

column1, column2, ... are the table columns

table is the table name from where we select the data

column_name is the name of the column you want to check for NULL

-- select rows with NULL email values

```
SELECT *  
FROM Employee  
WHERE email IS NULL;
```

Table: Employee

employee_id	first_name	last_name	department	email
1	Peter	Doe	Operations	peter@gmail.com
2	Megan	Morel	Finance	NULL
3	Rose	Bailey	Operations	rose@gmail.com
4	Linda	Bailey	Finance	NULL
5	Mary	Doe	Sales	NULL

SELECT *
FROM Employee
WHERE email IS NULL;

employee_id	first_name	last_name	department	email
2	Megan	Morel	Finance	NULL
4	Linda	Bailey	Finance	NULL
5	Mary	Doe	Sales	NULL

In SQL, the IS NOT NULL condition is used to select rows if the specified field is NOT NULL. It has the following syntax:

```
SELECT column1, column2, ...  
FROM table  
WHERE column_name IS NOT NULL;
```

Here,

column1, column2, ... are the table columns

table is the table name from where we select the data

column_name is the name of the column you want to check for NOT NULL

-- select rows where email is not NULL

```
SELECT *  
FROM Employee  
WHERE email IS NOT NULL;
```

Table: Employee

employee_id	first_name	last_name	department	email
1	Peter	Doe	Operations	peter@gmail.com
2	Megan	Morel	Finance	NULL
3	Rose	Bailey	Operations	rose@gmail.com
4	Linda	Bailey	Finance	NULL
5	Mary	Doe	Sales	NULL

SELECT *
FROM Employee
WHERE email IS NOT NULL;

employee_id	first_name	last_name	department	email
1	Peter	Doe	Operations	peter@gmail.com
3	Rose	Bailey	Operations	rose@gmail.com

We can use the COUNT() function with IS NULL to count the number of rows with an empty field. For example,

```
SELECT COUNT(*)  
FROM Employee  
WHERE email IS NULL;
```

Table: Employee

employee_id	first_name	last_name	department	email
1	Peter	Doe	Operations	peter@gmail.com
2	Megan	Morel	Finance	NULL
3	Rose	Bailey	Operations	rose@gmail.com
4	Linda	Bailey	Finance	NULL
5	Mary	Doe	Sales	NULL

SELECT COUNT(*)
FROM Employee
WHERE email IS NULL;

COUNT(*)

3

CONTINUE



In SQL,

The MAX() function returns the maximum value of a column.
The MIN() function returns the minimum value of a column.

SQL MAX() Function

The syntax of the SQL MAX() function is:

```
SELECT MAX(columnn)  
FROM table;  
Here,
```

column is the name of the column you want to filter
table is the name of the table to fetch the data from

```
SELECT MAX(age)  
FROM Customers;
```

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT MAX(age)  
FROM Customers;
```

MAX(age)

31

The syntax of the SQL MIN() function is:

```
SELECT MIN(columnn)
```

```
FROM table;
```

Here,

column is the name of the column you want to filter

table is the name of the table to fetch the data from

```
SELECT MIN(age)  
FROM Customers;
```

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT MIN(age)  
FROM Customers;
```

MIN(age)

22

The SQL COUNT() function returns the number of records returned by a query.

Example

--returns the number of rows in the Customers table

```
SELECT COUNT(*)  
FROM Customers;
```

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

SELECT COUNT(*)
FROM Customers;

COUNT(*)

5

In SQL, the SUM() and AVG() functions are used to calculate total and average values in numeric columns.

--select the sum of amount from Orders table
SELECT SUM(amount) AS total_sales
FROM Orders;

Table: Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

SELECT SUM(amount) AS total_sales
FROM Orders;

total_sales

13350

--select the sum of the amount of customer_id 4 from orders
SELECT SUM(amount) AS total_of_cus4
FROM Orders
WHERE customer_id = 4;

Table: Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

SELECT SUM(amount) AS total_of_cus4
FROM Orders
WHERE customer_id = 4;

total_of_cus4

700

```
-- get average age of customers  
SELECT AVG(age) AS average_age  
FROM Customers;
```

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT AVG(age) AS average_age  
FROM Customers;
```

average_age
25.6

The ORDER BY clause in SQL is used to sort the result set in ascending or descending order.

Example

```
-- orders all rows from Customers in ascending order by country  
SELECT *  
FROM Customers  
ORDER BY first_name;
```

Here, the SQL command selects all rows from the Customers table and then sorts them in ascending order by first_name.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

SELECT *
FROM Customers
ORDER BY first_name;

customer_id	first_name	last_name	age	country
5	Betty	Doe	28	UAE
3	David	Robinson	22	UK
1	John	Doe	31	USA
4	John	Reinhardt	25	UK
2	Robert	Luna	22	USA

We can use the **ASC** keyword to explicitly sort selected records in ascending order. For example,

```
-- orders all rows from Customers in ascending order by age  
SELECT *  
FROM Customers  
ORDER BY age ASC;
```

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

SELECT *
FROM Customers
ORDER BY age ASC;

customer_id	first_name	last_name	age	country
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE
1	John	Doe	31	USA

We use the **DESC** keyword to sort the selected records in descending order. For example,

```
-- order all rows from Customers in descending order by age
SELECT *
FROM Customers
ORDER BY age DESC;
```

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT *
FROM Customers
ORDER BY age DESC;
```

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
5	Betty	Doe	28	UAE
4	John	Reinhardt	25	UK
2	Robert	Luna	22	USA
3	David	Robinson	22	UK

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

In SQL, we use the GROUP BY clause to **group rows based on the value of columns.**

Example

-- count the number of customers in each country

```
SELECT country, COUNT(*) AS number  
FROM Customers  
GROUP BY country;
```

SELECT country, COUNT(*) AS number
FROM Customers
GROUP BY country;

country	number
UAE	1
UK	2
USA	2

Table: Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

Example: GROUP BY Amount Spent By Each Customer

-- calculate the total amount spent by each customer

```
SELECT customer_id, SUM(amount) AS total  
FROM Orders  
GROUP BY customer_id;
```

```
SELECT customer_id, SUM(amount) AS total  
FROM Orders  
GROUP BY customer_id;
```

customer_id	total
1	400
2	250
3	12000
4	700

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

We use the SQL LIKE operator with the WHERE clause to get a result set that matches the given string pattern.

Example

-- select customers who live in the UK

```
SELECT *  
FROM Customers  
WHERE country LIKE 'UK';
```

SELECT *
FROM Customers
WHERE country LIKE 'UK';

customer_id	first_name	last_name	age	country
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

Note: Although the LIKE operator behaves similarly to the = operator in this example, they are not the same. The = operator is used to check **equality**, whereas the LIKE operator is used to **match string patterns only**.

We can also invert the working of the LIKE operator by using the **NOT** operator with it. This returns a result set that doesn't match the given string pattern.

-- select customers who **don't** live in the USA

```
SELECT *  
FROM Customers  
WHERE country NOT LIKE 'USA';
```

SQL LIKE With Multiple Values

We can use the LIKE operator with **multiple string patterns** using the **OR operator**. For example,

```
-- select customers whose last_name starts with R and ends with t  
-- or customers whose last_name ends with e
```

```
SELECT *  
FROM Customers  
WHERE last_name LIKE 'R%t' OR last_name LIKE '%e';
```

Here, the SQL command selects customers whose last_name starts with **R** and ends with **t** or customers whose last_name ends with **e**.

Table: Customers

```
-- select rows where the last name  
-- of customers start with R  
SELECT *  
FROM Customers  
WHERE last_name LIKE 'R%';
```

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT *  
FROM Customers  
WHERE last_name LIKE 'R%';
```

customer_id	first_name	last_name	age	country
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

In SQL, we use the INSERT INTO statement to insert new row(s) into a database table.

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

- table_name is the table where the new row(s) are inserted
- column1, column2, column3, ... are the columns where the values are to be inserted
- value1, value2, value3, ... are the values to be inserted

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

INSERT INTO Customers(customer_id, first_name,
last_name, age, country)
VALUES (5, 'Harry', 'Potter', 31, 'USA');

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Harry	Potter	31	USA

Insert Data Only in Specified Column

If we skip column names during row insertion, the values of those columns will be NULL.

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
INSERT INTO Customers(first_name, last_name, age)
VALUES
('Brad', 'Pitt', 58);
```

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE
6	Brad	Pitt	58	NULL

```
INSERT INTO Customers(first_name, last_name, age, country)
VALUES
('Harry', 'Potter', 31, 'USA'),
('Chris', 'Hemsworth', 43, 'USA'),
('Tom', 'Holland', 26, 'UK');
```

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

```
INSERT INTO Customers(first_name, last_name, age,
country)
VALUES
('Harry', 'Potter', 31, 'USA'),
('Chris', 'Hemsworth', 43, 'USA'),
('Tom', 'Holland', 26, 'UK');
```

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Harry	Potter	31	USA
6	Chris	Hemsworth	43	USA
7	Tom	Holland	26	UK

In SQL, the UPDATE statement is used to modify **existing** records in a database table.

```
UPDATE table_name  
SET column1 = value1, column2 = value2, ...  
[WHERE condition];
```

- **table_name** is the name of the table to be modified
- **column1, column2, ...** are the names of the columns to be modified
- **value1, value2, ...** are the values to be set to the respective columns
- **[WHERE condition]** is an optional clause specifying which rows should be updated

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

UPDATE Customers
SET first_name = 'Johnny'
WHERE customer_id = 1;

customer_id	first_name	last_name	age	country
1	Johnny	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Harry	Potter	31	USA

-- update multiple values in the given row

```
UPDATE Customers  
SET first_name = 'Johnny', last_name = 'Depp'  
WHERE customer_id = 1;
```

the SQL command changes the value of the first_name column to **Johnny** and last_name to **Depp** if customer_id is equal to **1**.

In SQL, the DELETE clause is used to delete row(s) from a database table.

Example

```
DELETE FROM Customers  
WHERE customer_id = 4;
```

Here, the SQL command will delete a row from the Customers table if its customer_id is 4.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

DELETE FROM Customers
WHERE customer_id = 5;

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

The WHERE clause determines which rows to delete. However, we can delete all rows at once if we omit the WHERE clause. For example,

```
DELETE FROM Customers;
```



Thank You