SQL

DataWithBaraa

SQL DataWithBaraa

```
-- SELECT The Database
USE MyDatabase;

/*
    ======================
    == SELECT Statement ==
    ======================
    - SELECT is an SQL command used to fetch data from one or more tables in a
database.
    - It returns the result in the form of a result set (table).

    ----------
    - syntax -
    ----------
    SELECT column1, column2, ...
    FROM table_name;
*/

-- select all records on the table
SELECT * FROM customers;

-- select specific columns from the table
SELECT id, first_name
FROM customers;

-- show available databases;
SHOW DATABASES;

-- determine the database
USE salesdb;

SELECT *
FROM orders;

-- select orderid, customerid
SELECT
    orderid,
    customerid
FROM orders;
```

```
/*
    ==================
    == WHERE Clause ==
    ==================
    - The WHERE clause in SQL is used to filter rows in a table based on a specific
condition.

    ------------
    -- syntax --
    ------------
        SELECT column1, column2
        FROM table_name
        WHERE condition;
*/

-- determine the database
USE MyDatabase;
```

```sql
-- select first 3 rows from the table
SELECT *
FROM customers
WHERE id <= 3;

-- select all customers from Germany
SELECT *
FROM customers
WHERE country = 'Germany';

-- select all customers that their score not equal 0
SELECT *
FROM customers
WHERE score != 0;
```

```sql
/*
    ====================
    == ORDER BY Clause ==
    ====================
    - It sorts the rows returned by a SQL query in ascending (ASC) or descending
(DESC) order based on one or more columns.

    ------------
    -- syntax --
    ------------
        SELECT column1, column2
        FROM table_name
        ORDER BY column1 [ASC | DESC];
*/

USE MyDatabase;

-- sort the customers based on its 'id' from the bigest number to smallest number
SELECT *
FROM customers
ORDER BY id DESC;

-- retrieve all customers and sort the results by the heighest score first
SELECT *
FROM customers
ORDER BY score DESC;

-- retrieve all customers and sort the customers name from a-z
SELECT *
FROM customers
ORDER BY first_name ASC;
```

```sql
/*
    ==============================
    == SELECT DISTINCT Statement ==
    ==============================
    - used to removes the duplicates
    ------------
    -- syntax --
    ------------
        SELECT DICTINCT
            column1, column2, ..
        FROM table_name;
*/

USE MyDatabase;

-- removes duplicates names from the customers table
SELECT DISTINCT first_name
FROM customers;

-- return unique list of all countries
SELECT DISTINCT country
FROM customers;
```

```
/*
    =================
    == TOP & LIMIT ==
    =================
    - used to return the first n rows on the table

    TOP: used in the SQL Server
    LIMIT: used in the MySQL

    ------------
    -- syntax --
    ------------
        SELECT column1, column2, ...
        FROM table_name
        LIMIT n;
*/

USE MyDatabase;

-- retrieve only 3 customers
SELECT *
FROM customers
LIMIT 2;

-- retrieve the top 3 customers with the highest score
SELECT *
FROM customers
ORDER BY score DESC
LIMIT 3;


-- get the two most recent orders

-- determine the database
USE salesdb;

-- write the query
SELECT *
```

```sql
FROM orders
ORDER BY orderdate DESC
LIMIT 2;
```

```sql
/*
      ====================================
      == Execution Order VS Coding Order =
      ====================================

      ------------------
      -- Coding Order --
      ------------------
      - syntax
            SELECT DISTINCT TOP <num>
            column1, column2, ...
            FROM table_name
            WHERE condition
            GROUP BY <column name>
            ORDER BY column_name ASC|DESC

      ---------------------
      -- Execution Order --
      ---------------------
      1- FROM Clause
      2- WHERE Clause
      3- GROUP BY Statement
      4- HAVING Clause
      5- SELECT DISTINCT Statement
      6- ORDER BY Keyword
      7- TOP Statement
*/
```

```sql
/*
    ======================
    == CREATE Statement ==
    ======================
    - used to create databases, tables

    ----------
    - syntax -
    ----------
        CREATE TABLE table_name
        (column1 DATA_TYPE CONSTRAINTS, column2 DATA_TYPE CONSTRAINTS, ..);
*/

USE MyDatabase;

-- create a new table called persons with columns: id, person_name, birth_date, and
phone
CREATE TABLE persons (
    id INT NOT NULL,
    person_name VARCHAR(100),
    birth_date DATE,
    phone VARCHAR(50),
    CONSTRAINT pk_persons PRIMARY KEY (id)
);
```

```sql
/*
    =====================
    == ALTER Statement ==
    =====================
    - ALTER TABLE is used to modify an existing table in a database.
    - used to add, delete, modify the columns and data types

    ------------
    -- syntax --
    ------------
        ALTER TABLE table_name
        ALTER COLUMN column_name new_data_type [NULL | NOT NULL];
*/

USE NTI;

CREATE TABLE courses (id INT, name VARCHAR(100), price DECIMAL(10, 2));

SELECT * FROM courses;

-- add new column
ALTER TABLE courses
ADD description VARCHAR(255);

-- modify the column data type
ALTER TABLE courses
ALTER COLUMN price DECIMAL(12, 2);

-- drop a column
ALTER TABLE courses
DROP COLUMN description;

-- add primary key constraints
ALTER TABLE courses
ADD CONSTRAINT PK_courses_id PRIMARY KEY (id);


-- add unique constraints
ALTER TABLE courses
ADD CONSTRAINT UQ_courses_name UNIQUE (name);


INSERT INTO courses (id, name, price, description)
VALUES
(1, 'Python', 10.50, 'Learn Python programming'),
(2, 'C++', 25.00, 'Learn C++ programming'),
(3, 'C', 15.75, 'Learn C programming'),
(4, 'PHP', 30.20, 'Learn PHP programming'),
(5, 'MySQL', 8.99, 'Learn MySQL database');
```

```
    ====================
    == DROP Statement ==
    ====================
    - used to delete the table from the database

    DROP TABLE table_name;
*/

USE MyDatabase;

-- show all tables available
SHOW TABLES;

-- delete the table perons from the database
DROP TABLE persons;

-- show all tables available
SHOW TABLES;
```

```
/*
    ===========================
    == INSERT INTO Statement ==
    ===========================
    - used to append new data to the table

    ------------
    -- syntax --
    ------------
        INSERT INTO table_name (c1, c2, c3, ...) VALUES (v1, v2, v3, ...);
*/

SELECT * FROM users;

-- insert one record into the table
INSERT INTO users (id, name, address, email)
    VALUES (1, 'osama', 'cairo', 'osama@gmail.com');

-- insert multiple records into the table
INSERT INTO users (id, name, address, email)
    VALUES
    (2, 'ahmed', 'cairo', 'ahmed@gmail.com'),
    (3, 'khaled', 'giza', 'khaled@gmail.com'),
    (4, 'ali', 'ayat', 'ali@gmail.com');

/*
    =========================
    == INSERT using SELECT ==
    =========================
    - must the output of the SELECT statement equal to the INSERT statement
    - means number of columns must be equal in both tables
*/
```

```sql
/*
    ======================
    == UPDATE Statement ==
    ======================
    - used to update exist rows from the table

    ------------
    -- syntax --
    ------------
        UPDATE table_name
        SET column_name new_value;
        WHERE condition;

    -- note that: do not forget to write the WHERE condition
    -- if not determined the WHERE condition will change all rows
*/

USE MyDatabase;

SELECT * FROM customers;

INSERT INTO customers VALUES (6, 'youssef', 'cairo', 450);


-- change the country of youssef to Egypt
UPDATE customers
SET country = 'Egypt'
WHERE first_name = 'youssef' AND country = 'cairo';


-- change the name of youssef to Ali
UPDATE customers
SET first_name = 'Ali'
WHERE first_name = 'youssef';


-- update all customers with a NULL score by setting their score = 0
UPDATE customers
SET score = 0
WHERE score IS NULL;
```

```sql
/*
    ======================
    == DELETE Statement ==
    ======================
    - used to delete specific rows from the table

    ------------
    -- syntax --
    ------------
        DELETE FROM customers
        WHERE condition;

    -- note that: do not forget the WHERE condition
    -- if not determine the WHERE condition will delete all rows like TRUNCATE
Statement
```

```sql
*/

USE MyDatabase;

SELECT * FROM customers;

-- remove all customers from Egypt
DELETE FROM customers
WHERE country = 'Egypt';

/*
    --------------
    -- TRUNCATE --
    --------------
    -- exactly like 'DELETE FROM table_name' without any conditions
    -- used to delete all rows from the table

    ------------
    -- syntax --
    ------------
        TRUNCATE TABLE table_name;
*/

TRUNCATE TABLE customers;
```

```sql
/*
    ===================
    == HAVING Clause ==
    ===================
    - used to filter aggregated data
    - filter data after aggregation

    ----------
    - syntax -
    ----------

*/

USE MyDatabase;

SELECT
    country,
    SUM(score) AS total_score
FROM customers
GROUP BY country
HAVING SUM(score) > 750;

/*
    find the average score for each country considering only customers with a score
not equal 0
    and return only those countries with an average score greater than 430
*/
SELECT
    country,
    AVG(score) AS average_score
FROM customers
```

```sql
WHERE score != 0
GROUP BY country
HAVING AVG(score) > 430;
```

```sql
/*
    ==========================
    == Comparison Operators ==
    ==========================
*/

USE MyDatabase;

SELECT * FROM customers;

-- find customer with id = 1
SELECT *
FROM customers
WHERE id = 1;


-- find customers with score > 700
SELECT *
FROM customers
WHERE score > 700;


-- find all customers with score < 700
SELECT *
FROM customers
WHERE score < 700;


-- find all customers that are not from USA
SELECT *
FROM customers
WHERE country <> 'USA';

-- find all customers that are not from USA
SELECT *
FROM customers
WHERE country != 'USA';
```

```sql
/*
    ======================
    == Logical Operator ==
    ======================
    - AND: all conditions must be true
    - OR: one of the conditions must be true
    - NOT: get the opposite of the condition
*/

Use MyDatabase;

-- find all customers that from USA and have an ID smaller than 3
SELECT *
```

```
FROM customers
WHERE country = 'USA' AND id < 3;

-- find all customers that from USA or have an ID smaller than 3
SELECT *
FROM customers
WHERE country = 'USA' OR id < 3;

-- find all customers that are not from USA
SELECT *
FROM customers
WHERE NOT country = 'USA';
```

```
      /*
    ======================
    == BETWEEN Operator ==
    ======================
    - check if a value is within a range
*/
USE MyDatabase;

-- get the first 4 customers
SELECT *
FROM customers
WHERE id BETWEEN 1 AND 4;

-- find all customers that have score between 200 to 750
SELECT *
FROM customers
WHERE score BETWEEN 200 AND 750;
```

```
/*
    ==========================
    == Membership Operators ==
    ==========================
*/

USE MyDatabase;

-- find all customers that from USA and UK
SELECT *
FROM customers
WHERE country IN ("USA", 'UK');

-- find all customers that have specific score 500, and 750
SELECT *
FROM customers
WHERE score IN (500, 750);

-- find all customers that are not from USA and UK
SELECT *
FROM customers
WHERE country NOT IN ('USA', 'UK');
```

```sql
/*
	==================
	== Like Operator ==
	==================
	- LIKE: used to search about sub-string, number, ... on the field
	- There are two wildcards often used in conjunction with the LIKE operator:
		1- The percent sign % represents zero, one, or multiple characters
		2- The underscore sign _ represents one, single character
*/

USE MyDatabase;


-- search about customers from usa or USA
SELECT *
FROM customers
WHERE country LIKE 'USA';


-- search about counties that start with G
SELECT *
FROM customers
WHERE country LIKE 'G%';


-- Find customers whose names start with "A":
SELECT first_name, country, score
FROM customers
WHERE first_name LIKE 'A%';


-- Find customers whose names contain the letter "o"
SELECT first_name, country, score
FROM customers
WHERE first_name LIKE '%o%';



-- Find customers whose names end with "a"
SELECT first_name, country, score
FROM customers
WHERE first_name LIKE '%a';


-- Find customers whose names have exactly 4 characters
SELECT first_name, country, score
FROM customers
WHERE first_name LIKE '____';  -- Four underscores represent four characters


-- Find customers from countries whose name starts with "E"
SELECT first_name, country, score
FROM customers
WHERE country LIKE 'E%';
```

```sql
-- Find customers whose names have "a" as the second lette
SELECT first_name, country, score
FROM customers
WHERE first_name LIKE '_a%';


-- Find customers whose names have "a" as the second-to-last letter:
SELECT first_name, country, score
FROM customers
WHERE first_name LIKE '%a_';


-- Find customers whose names start with any letter between "A" and "D"
SELECT first_name, country, score
FROM customers
WHERE first_name LIKE '[A-D]%';


-- Find customers whose names contain either "a" or "e" and are from a country
starting with "E":
SELECT first_name, country, score
FROM customers
WHERE (first_name LIKE '%a%' OR first_name LIKE '%e%')
AND country LIKE 'E%';


-- Find customers whose names start with "A" or "B" and their score is greater than
750
SELECT first_name, country, score
FROM customers
WHERE (first_name LIKE 'A%' OR first_name LIKE 'B%')
AND score > 750;


-- Find customers whose names start with "S" or "N" and their country ends with "a"
or "e":
SELECT first_name, country, score
FROM customers
WHERE (first_name LIKE 'S%' OR first_name LIKE 'N%')
AND (country LIKE '%a' OR country LIKE '%e');


-- Find customers whose name contains "i" or "o" and their score is between 700 and
900:
SELECT first_name, country, score
FROM customers
WHERE (first_name LIKE '%i%' OR first_name LIKE '%o%')
AND score BETWEEN 700 AND 900;


-- Find customers from "Egypt" or "USA" whose names contain the letter "a" at any
position:
SELECT first_name, country, score
FROM customers
WHERE (country LIKE 'Egypt' OR country LIKE 'USA')
AND first_name LIKE '%a%';
```

```sql
-- Find customers whose names start with "E", "F", or "G" and score is greater than
750
SELECT first_name, country, score
FROM customers
WHERE (first_name LIKE 'E%' OR first_name LIKE 'F%' OR first_name LIKE 'G%')
AND score > 750;


-- Find customers whose names start with a vowel ("A", "E", "I", "O", or "U") and
are from the USA or Canada:
SELECT first_name, country, score
FROM customers
WHERE (first_name LIKE 'A%' OR first_name LIKE 'E%' OR first_name LIKE 'I%'
    OR first_name LIKE 'O%' OR first_name LIKE 'U%')
AND (country LIKE 'USA' OR country LIKE 'Canada');


-- Find customers whose names contain "a" or "o", their country contains "a", and
their score is between 700 and 900:
SELECT first_name, country, score
FROM customers
WHERE (first_name LIKE '%a%' OR first_name LIKE '%o%')
AND country LIKE '%a%'
AND score BETWEEN 700 AND 900;


-- Find customers whose names contain "o" in the second or second-to-last position:
SELECT first_name, country, score
FROM customers
WHERE first_name LIKE '_o%' OR first_name LIKE '%o_';
```

```sql
/*
    ================
    == INNER JOIN ==
    ================
    - we need a column that exists on the left table and right table
    - return only matches rows from both tables

    ------------
    -- syntax --
    ------------
        SELECT column1, column2, ...
        FROM table1
        INNER JOIN table2
        ON table1.common_column = table2.common_column;

*/

USE salesdb;
SHOW TABLES;

SELECT * FROM customers;
SELECT * FROM orders;

-- get all customers along with their orders, but only for customers who have placed
orde
```

```sql
SELECT
    c.firstname,
    c.country,
    o.orderid,
    o.orderdate,
    o.shipaddress
FROM customers AS c
INNER JOIN orders as o
ON o.orderid = c.customerid;


-- get all customers name, order id, order date and the sales of the orders, but
only the customers who have placed on
SELECT
    c.firstname,
    o.orderid,
    o.orderdate,
    o.sales
FROM customers AS c
INNER JOIN orders AS o
ON o.orderid = c.customerid;
```

```sql
/*
    ===============
    == LEFT JOIN ==
    ===============
    - LEFT JOIN (also called LEFT OUTER JOIN) combines rows from two tables.
        It keeps every row from the left table.
    - the left table is the primary source of your data

    ------------
    -- syntax --
    ------------
        SELECT column1, column2, ...
        FROM table1
        LEFT JOIN table2
        ON table1.common_column = table2.common_column;

*/

    ----------------------------------------------------------
    -- note that the orders of the table is very important --
    ----------------------------------------------------------
USE salesdb;

SELECT
    c.firstname,
    c.country,
    o.orderid,
    o.orderdate,
    o.sales
FROM customers AS c
LEFT JOIN orders AS o
ON o.orderid = c.customerid;
```

```sql
SELECT
    c.firstname,
    c.country,
    o.orderid,
    o.orderdate,
    o.sales
FROM orders AS o
LEFT JOIN customers AS c
ON o.orderid = c.customerid;
```

```sql
/*
    ================
    == RIGHT JOIN ==
    ================
    - RIGHT JOIN (or RIGHT OUTER JOIN) combines two tables such that:
        Every row from the right table appears.
        Only matching rows from the left table appear; unmatched ones show NULL.

    ------------
    -- syntax --
    ------------
        SELECT column1, column2, ...
        FROM table1
        RIGHT JOIN table2
        ON table1.common_column = table2.common_column;

*/

USE salesdb;

SELECT * FROM customers;
SELECT * FROM orders;
SELECT
    c.firstname,
    c.score,
    o.orderid,
    o.orderdate
FROM customers AS c
RIGHT JOIN orders AS o
ON c.customerid = o.orderid;
```

```sql
USE MyDatabase;
/*
    ===============
    == FULL JOIN ==
    ===============
    - A FULL JOIN (or FULL OUTER JOIN) is a type of SQL join that returns all rows
from both tables, whether they have matching records or not.
    -- If a row in Table A has a matching row in Table B, the result shows a single
combined row.
    -- If a row in Table A does not have a match in Table B, the Table B columns are
filled with NULL.
    -- If a row in Table B does not have a match in Table A, the Table A columns are
filled with NULL.

    What happens in a FULL JOIN:
```

```
    - LEFT JOIN part: All customers appear; if they have no orders, the order
columns are NULL.
        Carol has no order → orderid, orderdate, sales = NULL
    - RIGHT JOIN part: All orders appear; if the customer doesn’t exist, the
customer columns are NULL.
        Order 103 has customerid = 4 → firstname, country = NULL
*/

USE MyDatabase;

/*
    What This Does
        - Returns all customers and all orders.
        - If a customer has no orders, OrderID, OrderDate, and sales will be NULL.
        -   If an order has no matching customer (like customer_id = 6 in your
orders table),
            CustomerID, CustomerName, etc. will be NULL.
*/
SELECT
    c.id AS CustomerID,
    c.first_name AS CustomerName,
    c.country,
    c.score,
    o.order_id AS OrderID,
    o.order_date AS OrderDate,
    o.sales
FROM customers AS c
FULL OUTER JOIN orders AS o
ON c.id = o.customer_id;
```

```
/*
    ====================
    == FULL ANTI JOIN ==
    ====================
    - Returns rows that DO NOT have matching records in the other table.
    - In other words: only unmatched data from both tables.
    - SQL Server does not have FULL ANTI JOIN keyword, so we simulate it using:
        FULL JOIN + WHERE (left_key IS NULL OR right_key IS NULL)
*/

USE MyDatabase;

-- Find:
-- 1) Customers who have NO orders
-- 2) Orders that have NO matching customers
SELECT
    c.id AS customer_id,
    c.first_name,
    o.order_id,
    o.sales
FROM orders AS o
FULL JOIN customers AS c
    ON c.id = o.customer_id
WHERE c.id IS NULL            -- orders with no matching customer
   OR o.customer_id IS NULL;  -- customers with no orders
```

```
/*
    ======================================
    == SIMULATING INNER JOIN (WITHOUT USING INNER JOIN)
    ======================================
    - Goal: Get only customers who have placed at least one order.
    - LEFT JOIN + WHERE right_key IS NOT NULL behaves like an INNER JOIN.
*/

SELECT
    c.id AS customer_id,
    c.first_name,
    o.order_id,
    o.sales
FROM customers AS c
LEFT JOIN orders AS o
    ON c.id = o.customer_id
WHERE o.customer_id IS NOT NULL;   -- only customers WITH orders
```

```
/*
    ====================
    == RIGHT ANTI JOIN ==
    ====================
    - Returns rows from the RIGHT table that have NO matching rows in the LEFT
table.
    - Excludes rows that do have matches.
    - SQL Server/MySQL do not have "RIGHT ANTI JOIN" keyword; we simulate it using:
        RIGHT JOIN + WHERE left_table.key IS NULL
*/

USE MyDatabase;

-- Get all orders that do NOT have a matching customer
SELECT
    o.order_id,
    o.sales
FROM customers AS c
RIGHT JOIN orders AS o
    ON c.id = o.customer_id
WHERE c.id IS NULL;
```

```
/*
    ====================
    == LEFT ANTI JOIN ==
    ====================
    - Returns rows from the LEFT table that have NO matching rows in the RIGHT
table.
    - It does NOT return any rows that have matches.
    - Most commonly used in data analysis and ETL tasks.
    - SQL Server/MySQL do not have a keyword "LEFT ANTI JOIN", but we simulate it
using:
        LEFT JOIN + WHERE right_table.key IS NULL
*/

USE MyDatabase;
```

```sql
-- Get all customers who have NOT placed any orders
SELECT
    c.id AS customer_id,
    c.first_name
FROM customers AS c
LEFT JOIN orders AS o
    ON c.id = o.customer_id
WHERE o.customer_id IS NULL;  -- only customers with NO matching orders
```

```sql
/*
      ================
      == CROSS JOIN ==
      ================
      - combines every row from the left with every row from the right

      ------------
      -- syntax --
      ------------
            SELECT *
                  FROM A
                  CROSS JOIN B;
*/

USE MyDatabase;

-- generate all possible combinations of customers and orders
SELECT *
FROM customers
CROSS JOIN orders;
```
