ASSIGNMENT 03

**1) Does ANSI recognize the data type DATE?**

Yes, ANSI (American National Standards Institute) **does recognize the data type DATE** as part of the SQL standard. The ANSI SQL standard defines DATE as a data type used to store calendar dates. It typically includes the **year, month, and day**, but not the time of day.

Here’s a detailed breakdown of how the DATE data type works in ANSI SQL:

* **Format**: The DATE data type typically follows the **YYYY-MM-DD** format, which stands for:
  + **YYYY**: Year (e.g., 2024)
  + **MM**: Month (e.g., 10 for October)
  + **DD**: Day (e.g., 11 for the 11th day)
* **Storage**: It stores dates without any time or time zone information. For example, if you enter a date like 2024-10-11, only the year, month, and day are stored.
* **Usage**: The DATE data type is used in SQL to store birthdates, event dates, transaction dates, and so on. You can perform various operations on DATE fields, such as comparing dates, adding or subtracting intervals, or extracting specific parts (year, month, day) from a date.

Example of creating a table with a DATE field:

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

EmployeeName VARCHAR(255),

HireDate DATE

);

Inserting a value into the DATE field:

INSERT INTO Employees (EmployeeID, EmployeeName, HireDate)

VALUES (1, 'John Doe', '2024-10-11');

Thus, ANSI SQL does indeed recognize the DATE type, and it is supported by most relational database systems, such as MySQL, PostgreSQL, SQL Server, and Oracle.

**2) Which subdivision of SQL is used to insert values in tables?**

The subdivision of SQL that is used to insert values into tables is part of **DML** (**Data Manipulation Language**).

**What is Data Manipulation Language (DML)?**

* **DML** is a subset of SQL that is used to manage and manipulate the data stored in a database. It includes statements that allow users to:
  + **Insert** new data into tables.
  + **Update** existing data.
  + **Delete** rows of data.
  + **Select** data from tables.

The key **DML statement** for inserting values into a table is the **INSERT** statement.

**Detailed Breakdown of the INSERT Statement**

The INSERT statement is used to add new records (rows) to a table. Here’s a detailed breakdown of how it works:

1. **Basic Syntax**:

INSERT INTO table\_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);

* + **table\_name**: This is the name of the table where you want to insert the new data.
  + **column1, column2, ...**: These are the column names where the values will be inserted. You specify which columns the values correspond to.
  + **value1, value2, ...**: These are the actual values you want to insert into the columns.

1. **Example**: Let’s assume you have a Customers table with columns CustomerID, CustomerName, and ContactNumber.

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CustomerName VARCHAR(255),

ContactNumber VARCHAR(15)

);

To insert a new customer into the Customers table, you would use:

INSERT INTO Customers (CustomerID, CustomerName, ContactNumber)

VALUES (1, 'John Doe', '555-1234');

This inserts a single row with the values 1 for CustomerID, 'John Doe' for CustomerName, and '555-1234' for ContactNumber.

1. **Inserting Multiple Rows**: SQL allows you to insert multiple rows in a single INSERT statement by separating the sets of values with commas:

INSERT INTO Customers (CustomerID, CustomerName, ContactNumber)

VALUES

(2, 'Jane Smith', '555-5678'),

(3, 'Mike Johnson', '555-8765');

This inserts two rows into the Customers table at once.

1. **Inserting Values Without Specifying Column Names**: If you want to insert values into **all** columns of the table and you know the order of the columns, you can omit the column names and only specify the values:

INSERT INTO Customers

VALUES (4, 'Emily Davis', '555-4321');

However, this requires that you provide values for every column in the correct order as defined in the table schema.

1. **Handling NULL Values**: If some columns allow NULL values and you don't want to provide data for those columns, you can simply leave them out in your INSERT statement, and SQL will automatically insert NULL into those fields if they are allowed to be NULL:

INSERT INTO Customers (CustomerID, CustomerName)

VALUES (5, 'Steve Rogers');

Here, the ContactNumber column would be set to NULL for this row, assuming it allows NULL values.

1. **Error Handling**:
   * If you attempt to insert a row that violates any **constraints** (such as a duplicate value in a PRIMARY KEY column or inserting a NULL in a column that is marked as NOT NULL), SQL will raise an error, and the insert will fail.
   * For example, this would cause an error if CustomerID is a primary key and 1 already exists:

INSERT INTO Customers (CustomerID, CustomerName, ContactNumber)

VALUES (1, 'Duplicate Entry', '555-9999');

To handle such cases, some databases support the **ON DUPLICATE KEY UPDATE** clause or similar options to control how duplicates are managed.

1. **Inserting Data from Another Table**: In some cases, you may want to insert data from another table into a new table. You can do this using a variation of the INSERT statement:

INSERT INTO NewCustomers (CustomerID, CustomerName)

SELECT CustomerID, CustomerName FROM Customers;

This inserts data into the NewCustomers table by selecting it from the Customers table.

**Summary of SQL Subdivisions for DML:**

* **DML** commands are responsible for modifying data in the database.
* The **INSERT** command is used to add new rows of data into a table.
* Other important DML commands include:
  + **UPDATE**: Modifies existing data.
  + **DELETE**: Removes data from the table.
  + **SELECT**: Retrieves data from the table, although this is more of a data retrieval operation.

By using **DML** and specifically the INSERT statement, users can efficiently manage data within their SQL-based database systems.