**NAME :- ANIKET SANJAYKUMAR BIYANI**

**DATA ENGINEERING BATCH -1**

**SQL CODING CHALLENGE**

**Query 1:-** CREATE DATABASE sqlcodingchallenge;

* *EXPLANATION :-*

CREATE DATABASE: This is a SQL statement used to create a new database. It's followed by the name you want to give to your new database.

sqlcodingchallenge: This is the name of the database you're creating

**Query 2:-** USE sqlcodingchallenge;

* *EXPLANATION :-*

The USE statement in SQL is used to select a particular database as the current database

**Query 3:-** Creating tables for Departments ,Employee and Projects

* *EXPLANATION :-*

Employees table:

EmployeeID: Unique identifier

FirstName and LastName: The first and last names of the employee

BirthDate: Date of birth of the employee

DepartmentID: Foreign key referencing the DepartmentID in the Departments table.

Departments table:

DepartmentID: Unique identifier for each department.

DepartmentName: Name of the department.

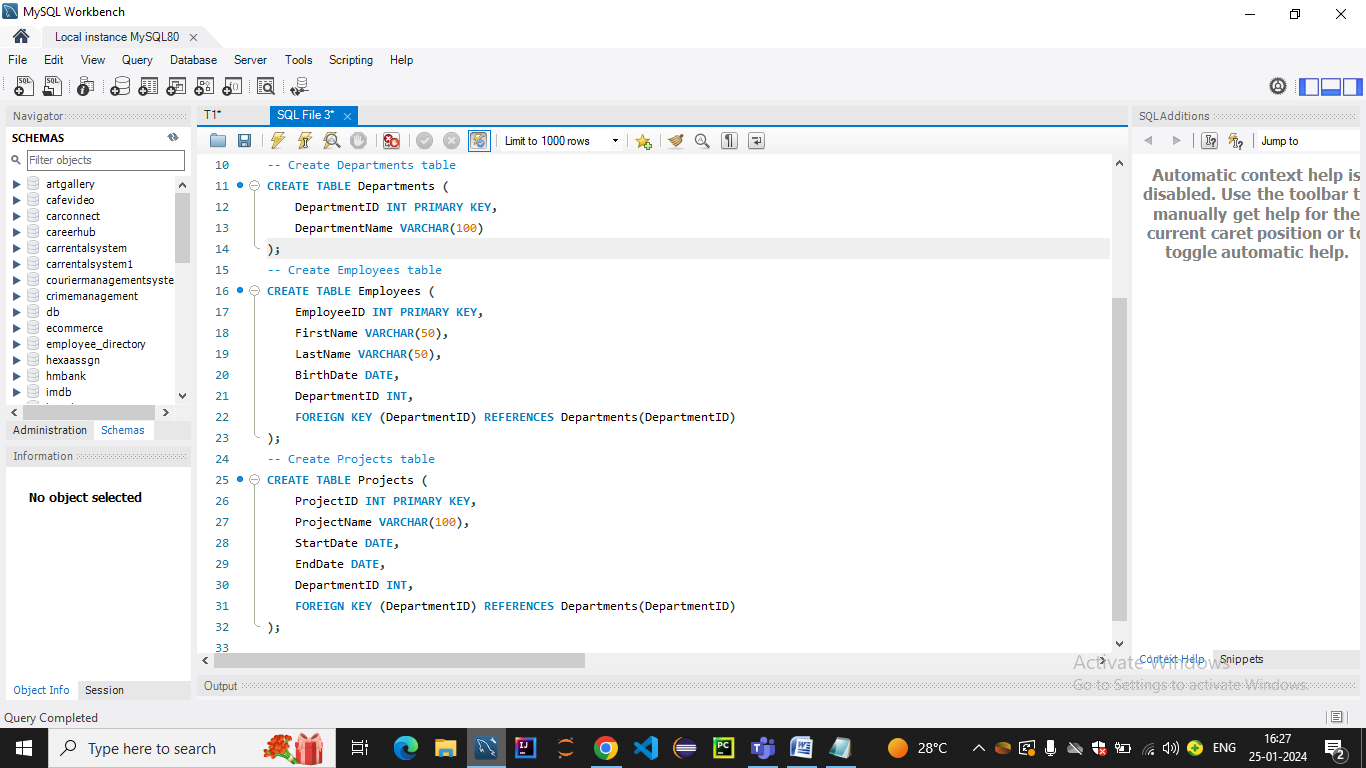
Projects table:

ProjectID: Unique identifier for each project.

ProjectName: Name of the project.

StartDate and EndDate: Start and end dates of the project.

DepartmentID: Foreign key referencing the DepartmentID in the Departments table.

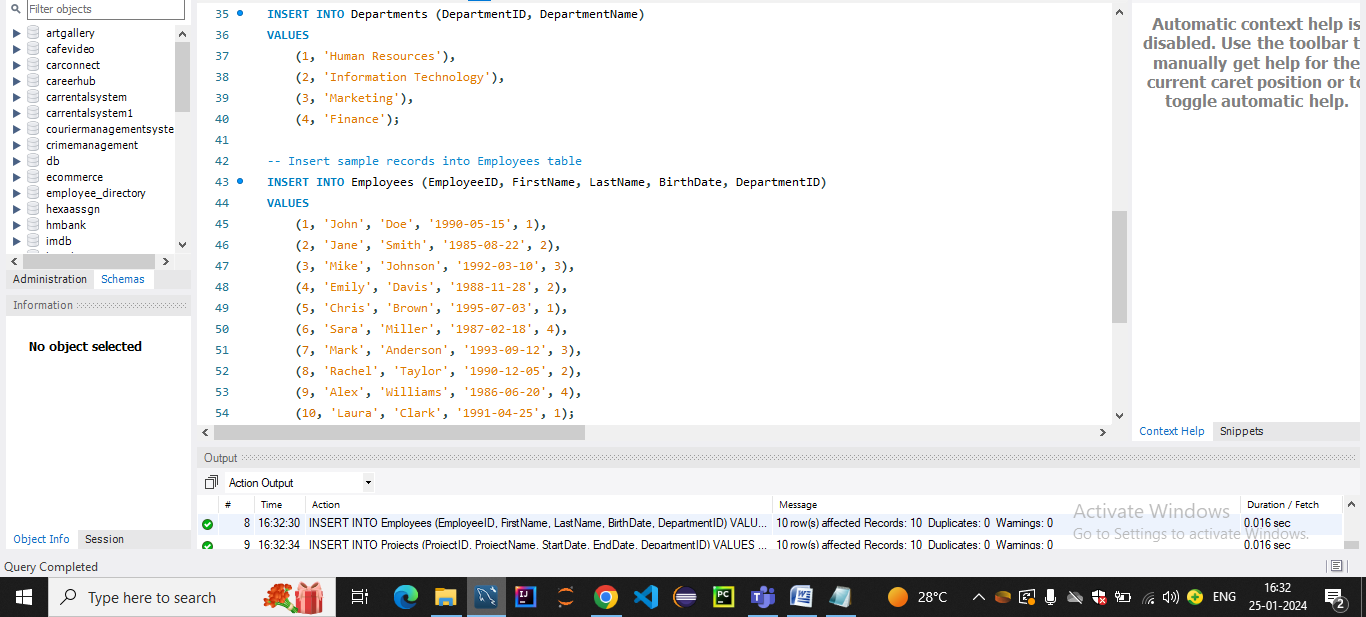


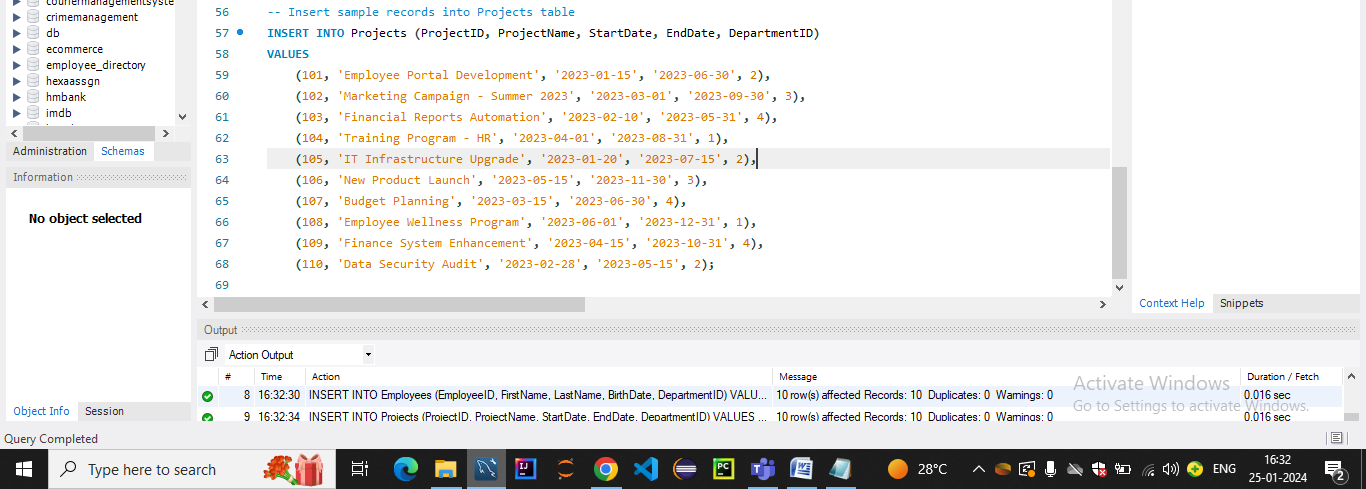
**Query 4:-**Inserting data into tables

* *EXPLANATION :-*

The INSERT INTO statement is used to add records to the Departments table.

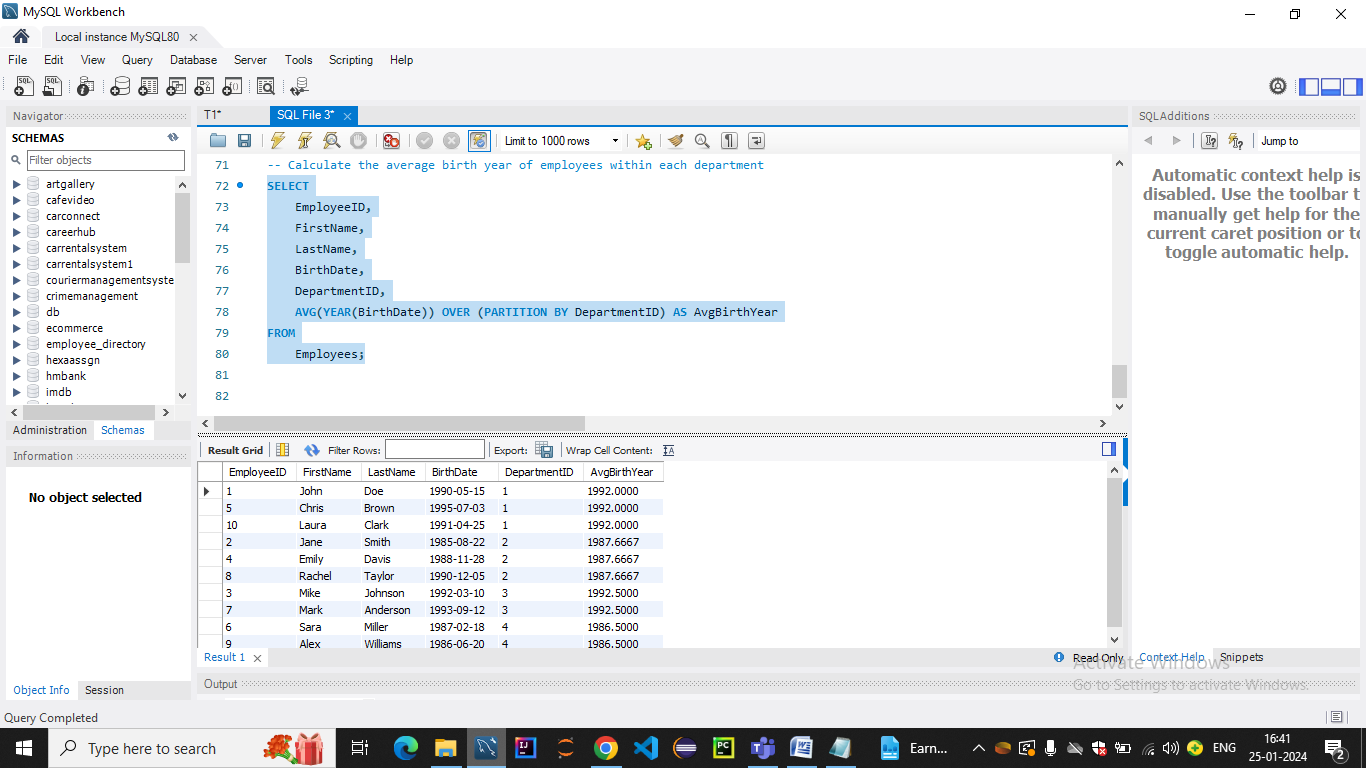
The VALUES clause contains multiple sets of values, each set corresponding to a record being inserted.





**Q1 .a) Execute OVER and PARTITION BY Clause in SQL Queries**

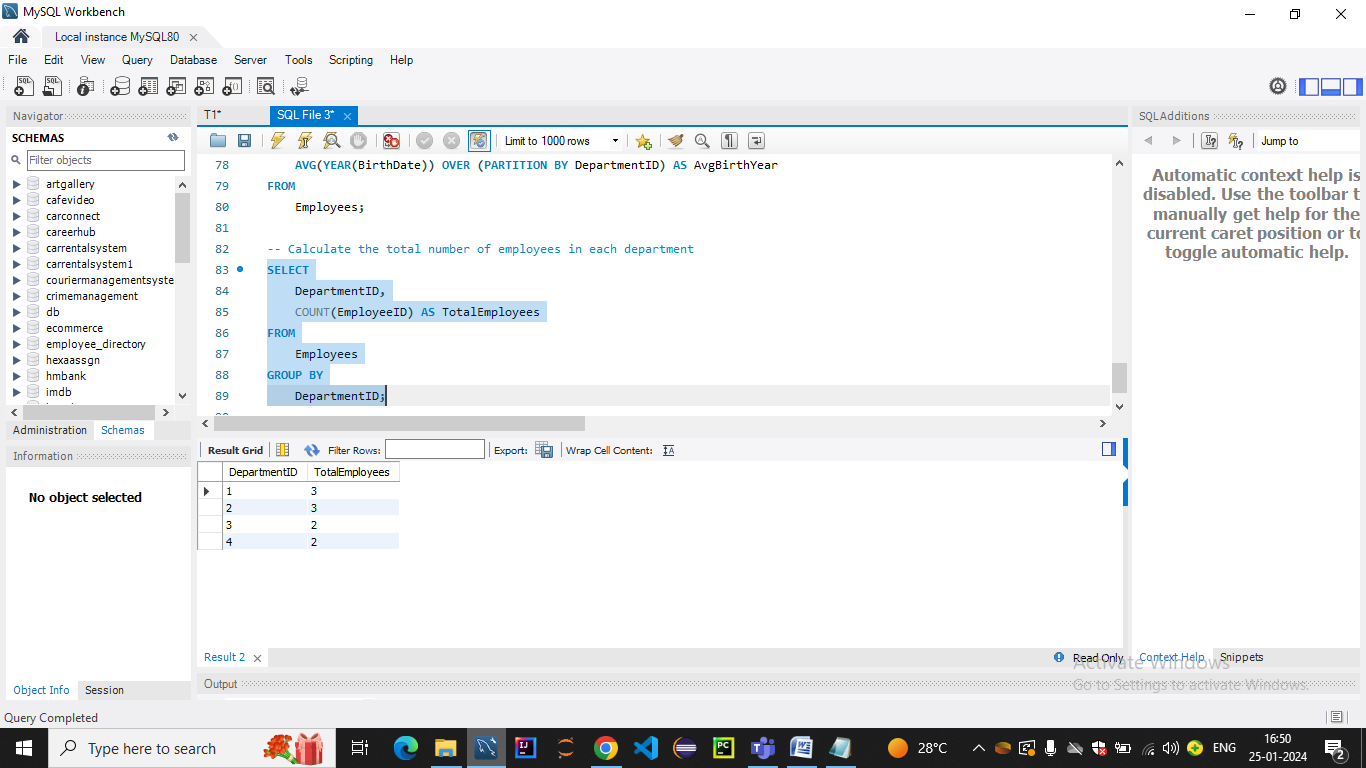
* ***THEORY :-***
* The OVER and PARTITION BY clauses in SQL are used together with window functions to perform calculations across a specified range of rows related to the current row.
* These clauses are commonly used in analytical queries for tasks such as calculating running totals, ranking, and aggregating values within specific partitions.
* **PARTITION BY Clause:**
* The PARTITION BY clause divides the result set into partitions to which the window function is applied.
* It essentially defines the grouping of rows for the window function. Each partition is treated separately, and the window function is applied within each partition.
* **OVER Clause:**
* The OVER clause is used in conjunction with window functions to define a window or a subset of rows within the result set over which the window function operates.
* The OVER clause can include the PARTITION BY clause, which further divides the result set into partitions based on one or more columns.



* *EXPLANATION :-*
* The AVG(YEAR(BirthDate)) calculates the average birth year for each employee based on their BirthDate.
* The OVER (PARTITION BY DepartmentID) part is where the magic happen
* PARTITION BY DepartmentID divides the result set into partitions, and the window function is applied separately to each partition.
* In this case, we're calculating the average birth year for each department separately.
* This query will return the original employee information along with an additional column, AvgBirthYear, which represents the average birth year for employees within each department.

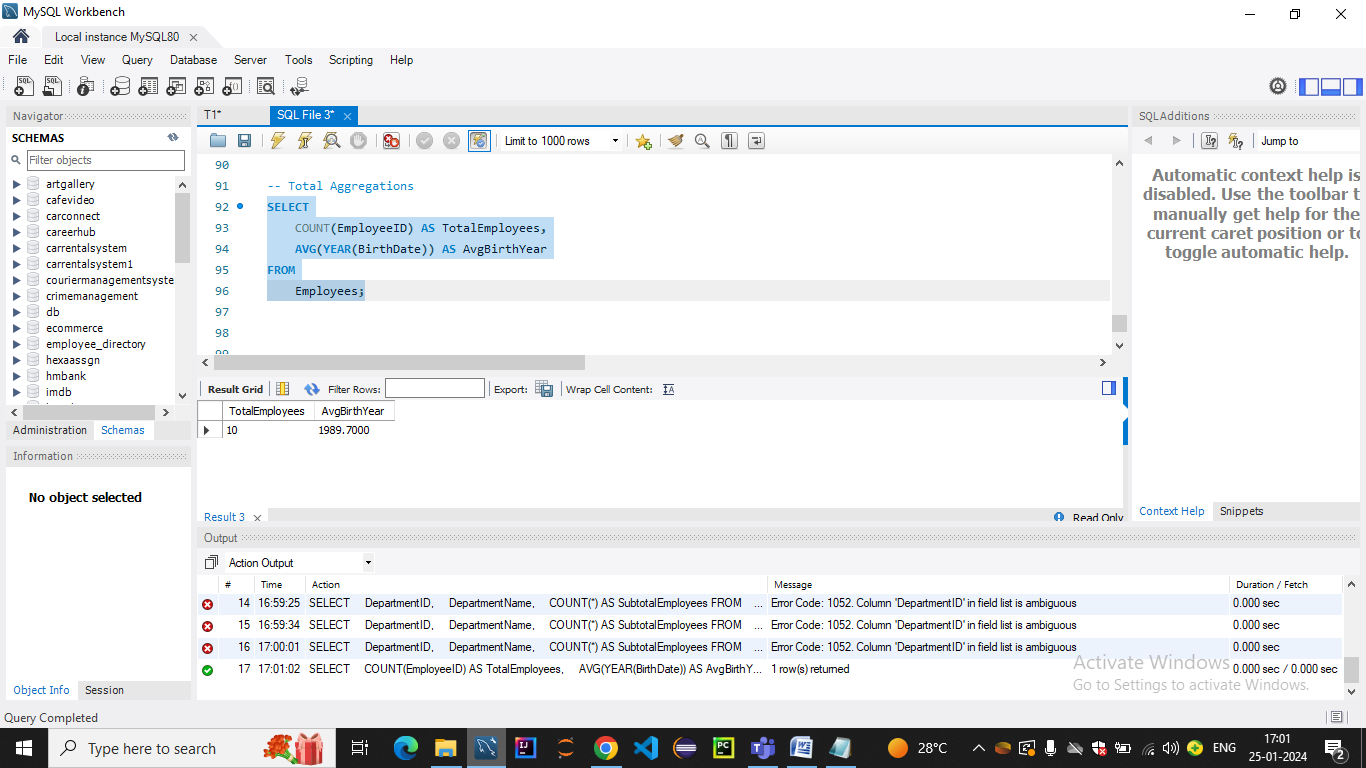
**Q 1.b) Create subtotals Aggregations using SQL Queries.**

* Creating subtotals and aggregations in SQL involves using aggregate functions along with the GROUP BY clause to group rows based on certain criteria.
* Subtotals are essentially summary values for each group created by the GROUP BY clause

****

* *EXPLANATION :-*
* The COUNT(EmployeeID) is an aggregate function that counts the number of rows (employees) in each group.
* The GROUP BY DepartmentID clause groups the result set by the DepartmentID column.
* The result is a list of DepartmentID values along with the total number of employees in each department.

**Q 1.c) Create totals Aggregations using SQL Queries.**



* *EXPLANATION :-*
* COUNT(EmployeeID) AS TotalEmployees: This part calculates the total count of employees in the Employees table.
* AVG(YEAR(BirthDate)) AS AvgBirthYear: This part calculates the average birth year of all employees. The AVG function is applied to the birth year extracted from the BirthDate.
* query calculates total aggregations for the entire Employees table