With professional regards,
Eng. Osama
Senior .NET Full Stack Web Developer & Certified Technical Instructor | Problem Solver
💼 16+ Years of Hands-On Project Experience
Specialized in Advanced Project Analysis & Scalable Solution Design
Building Scalable Solutions with .NET & Expert in ASP.NET Core, C#, SQL Server, REST APIs, Azure
Enterprise Software | API Architectures
📧 osamamu2025@gmail.com | 📍 Egypt

🎯 Passionate about clean code, scalable systems, and team mentorship you are required to design and implement a small yet comprehensive SQL Server database system to simulate a business environment for a company. Your task includes:

📌 1. Database Design:

Create a normalized database consisting of at least **3 related tables**:

- `employees`
- `departments`
- `projects`

Make sure to include proper use of:

- Primary keys & foreign keys
- Constraints (`CHECK`, `NOT NULL`, etc.)
- A **computed column** (e.g., employee age)

📌 2. Populate the Tables:

Insert relevant sample data into all tables with realistic values (at least 3–4 records per table).

📌 3. Write a Series of Diverse and Advanced SQL Queries:

Write **at least 10 advanced SELECT queries** using:

- `JOIN` and `LEFT JOIN`
- `GROUP BY`, `COUNT()`, `AVG()`, `MAX()`, `MIN()`
- `WHERE`, `HAVING`, filtering with functions (`YEAR()`, `MONTH()`, etc.)
- `DISTINCT`, `ORDER BY`
- `WINDOW FUNCTIONS` like `RANK() OVER()`
- `SUBQUERIES`, `NOT EXISTS`

Make sure to comment and explain each query clearly as part of a technical document or embedded in SQL script.

QUERIES SECTION:

1. **Display all employees along with their computed ages.**
   (USE A COMPUTED COLUMN FOR AGE BASED ON BIRTHDATE.)

2. **Display employees along with their department names and locations.**
   (USE INNER JOIN BETWEEN *employees* AND *departments*.)
3. **Display each department along with the number of employees in it.**
   (USE AGGREGATION AND *GROUP BY* WITH *COUNT()* FUNCTION.)
4. **List employees who are working on projects managed by their own departments.**
   (USE *JOIN* ACROSS *employees*, *departments*, AND *projects* TABLES.)
5. **Display projects that end in the year 2024 only.**
   (USE *WHERE* WITH THE *YEAR()* FUNCTION ON *end_date*.)
6. **Find employees who have their birthday today.**
   (MATCH *DAY()* AND *MONTH()* OF BIRTHDATE WITH TODAY'S DATE.)
7. **Calculate average salary grouped by gender.**
   (USE *AVG(salary)* AND *GROUP BY gender*.)
8. **List departments that do not have any projects assigned to them.**
   (USE *NOT EXISTS* OR A *LEFT JOIN* WITH *NULL* CHECK.)
9. **Display salary statistics (min, max, avg) for each department.**
   (USE AGGREGATION FUNCTIONS GROUPED BY DEPARTMENT.)
10. **List employees who have worked on more than one project.**
    (ASSUMING AN INTERMEDIATE TABLE LIKE *employee_projects*, USE *GROUP BY* AND *HAVING COUNT > 1*.)
11. **Rank employees by salary within each department.**
    (USE *RANK()* OR *DENSE_RANK()* WITH *PARTITION BY* IN A WINDOW FUNCTION.)
12. **Display IT department projects that exceed the department's average project budget.**
    (USE A SUBQUERY TO FILTER BASED ON AVERAGE BUDGET.)
13. **Calculate the total company payroll (monthly and annually).**
    (USE *SUM(salary)* AND MULTIPLY FOR ANNUAL PAYROLL ESTIMATE.)

📌 4. Implement Business Logic with Stored Procedures and Functions:

Create at least:

- **3 Stored Procedures**, e.g.:
  - Procedure to add a new employee after validating department existence.
  - Procedure to apply salary bonuses based on gender.
  - Procedure to generate department-wise employee summary.
- **2 Scalar or Table-Valued Functions**, e.g.:
  - Function to calculate employee bonus.
  - Function to calculate project duration or employee tenure.

Explain the business purpose and logic behind each one.

⚙ PROCEDURES & FUNCTIONS SECTION:

1. **Create a scalar function to calculate employee bonus based on gender.**
   (FEMALES RECEIVE 10%, MALES RECEIVE 5% OF SALARY AS BONUS.)
2. **Create a stored procedure to apply bonuses by updating salaries using the bonus function.**
3. **Create a stored procedure to insert a new employee with validation on department existence.**
   (CHECK WHETHER THE PROVIDED *dept_id* EXISTS BEFORE INSERTING THE EMPLOYEE.)

4. • **Create a stored procedure to generate a department-level report.**
   This report should include the total number of employees per department and the average salary in each department.
   (Use `JOIN`, `COUNT()`, `AVG()`, and `GROUP BY`.)
5. • **Create a scalar function to calculate employee tenure (years of service).**
   The function should return the number of full years between two dates: a start date (e.g., project start or hire date) and an end date (or today if still active).

📌 5. Implement Triggers:

Implement at least:

- A **trigger that logs salary changes** into a separate audit table.
- A **trigger to alert/log when a new employee is added**.

TRIGGERS SECTION:

1. **Create a trigger to display an alert when a new employee is inserted.**
   THIS TRIGGER SHOULD EXECUTE AFTER A NEW ROW IS INSERTED INTO THE `employees` TABLE AND PRINT A MESSAGE WITH THE EMPLOYEE'S NAME AND DEPARTMENT ID.
   (USE THE `INSERTED` PSEUDO-TABLE AND `PRINT` STATEMENT.)
2. **Create a table named `salary_changes` to store salary modification history.**
   THIS AUDIT TABLE SHOULD STORE: EMPLOYEE ID, OLD SALARY, NEW SALARY, DATE OF CHANGE, AND OPTIONALLY THE USER WHO MADE THE CHANGE.
3. **Create a trigger to log every salary update into the `salary_changes` table.**
   THIS TRIGGER SHOULD FIRE AFTER ANY UPDATE ON THE `salary` COLUMN IN THE `employees` TABLE, AND INSERT A NEW ROW INTO `salary_changes` FOR EACH SALARY MODIFICATION.
   (USE BOTH `INSERTED` AND `DELETED` PSEUDO-TABLES TO COMPARE VALUES.)

📌 6. Create Useful Views:

Create at least:

- One `View` to display a summarized employee profile (`EmployeeSummary`).
- Additional `View` ideas (optional): Projects by department, or active projects only.

VIEWS SECTION:

1. **Create a view named `EmployeeSummary` to display a detailed overview of each employee.**
   THE VIEW SHOULD INCLUDE: EMPLOYEE ID, NAME, GENDER, SALARY, COMPUTED AGE, DEPARTMENT NAME, AND DEPARTMENT LOCATION.
   (USE A `JOIN` BETWEEN `employees` AND `departments`.)

TRACK THE USER RESPONSIBLE FOR SALARY CHANGES

To enhance auditing and traceability, store the **username of the person (or system account)** who performed the salary update.

You can achieve this by:

✓ 1. MODIFYING THE AUDIT TABLE

Add a new column to the `salary_changes` table to store the username or system user

📌 7. (Optional Enhancements):

If you want to go further, add:

- A trigger for employee deletion that archives the record.
- A history table (`employee_audit`).
- A View combining all three main tables.
- Integration of user/system metadata (e.g., `SYSTEM_USER` in triggers).

💼 *INDEXES SECTION* – ADVANCED SQL TASKS

🔍 TASK 1: CREATE A NON-CLUSTERED INDEX TO IMPROVE QUERY PERFORMANCE

Create a non-clustered index on the `salary` column in the `employees` table to optimize queries that filter or sort by salary.

💡 THIS INDEX WILL HELP IMPROVE PERFORMANCE WHEN RUNNING QUERIES

✓ **Required:**
Write the SQL statement to create the index, and explain when and why you would use a non-clustered index.

🔍 TASK 2: CREATE A COMPOSITE INDEX ON MULTIPLE COLUMNS

Create a composite index on the `project_name` and `end_date` columns in the `projects` table to optimize multi-column search queries.

✓ **Required:**

- Create the index using T-SQL
- Explain the order of columns and how it affects index usage

🔍 TASK 3: ANALYZE AND TEST THE EFFECT OF AN INDEX ON QUERY PERFORMANCE

Compare the performance of a query before and after applying an index.

✓ **Required:**

- Choose a query that performs a full scan (e.g., filtering by `birthdate` or `dept_id`)
- Use `SET STATISTICS IO ON` and `SET STATISTICS TIME ON` before and after adding an index
- Show the difference in execution cost or logical reads

🔧 BONUS: Create an index on `dept_id` in the `employees` table and observe how a `JOIN` on departments improves in speed.

With professional regards,
Eng. Osama
Senior .NET Full Stack Web Developer & Certified Technical Instructor | Problem Solver
💼 **16+ Years of Hands-On Project Experience**
Specialized in Advanced Project Analysis & Scalable Solution Design
Building Scalable Solutions with .NET & Expert in ASP.NET Core, C#, SQL Server, REST APIs, Azure
Enterprise Software | API Architectures
📧 **osamamu2025@gmail.com |** 📍 **Egypt**