# ASSIGNMENT Overview

The goal is to create a company database with multiple interrelated tables. Each group is responsible for creating specific tables and populating them with data. Once all data is in place, groups will then establish relationships and constraints using `ALTER` and `UPDATE` statements.

# 1 - Groups and Responsibilities

1. **Group 1: `Employees` and `Departments`**
    - **Tables**: `Employees`, `Departments`
    - **Tasks**:
        - Create `Employees` table with columns: `EmployeeID` (Primary Key), `FirstName`, `LastName`, `DepartmentID`, `HireDate`, `Position`, `Salary`.
        - Create `Departments` table with columns: `DepartmentID` (Primary Key), `DepartmentName`, `Location`.
        - Populate tables with initial data.
2. **Group 2: Projects and Assignments**
    - **Tables**: `Projects`, `Assignments`
    - **Tasks**:
        - Create `Projects` table with columns: `ProjectID` (Primary Key), `ProjectName`, `StartDate`, `EndDate`, `Budget`.
        - Create `Assignments` table with columns: `AssignmentID` (Primary Key), `EmployeeID`, `ProjectID`, `Role`, `AssignmentDate`.
        - Populate tables with initial data.
3. **Group 3: Customers and Orders**
    - **Tables**: `Customers`, `Orders`
    - **Tasks**:
        - Create `Customers` table with columns: `CustomerID` (Primary Key), `CustomerName`, `ContactNumber`, `Email`, `Address`.
        - Create `Orders` table with columns: `OrderID` (Primary Key), `CustomerID`, `OrderDate`, `TotalAmount`.
        - Populate tables with initial data.
4. **Group 4: Products and OrderDetails**
    - **Tables**: Products, OrderDetails

  - **Tasks**:
    - Create `Products` table with columns: `ProductID` (Primary Key), `ProductName`, `Category`, `Price`, `StockQuantity`.
    - Create `OrderDetails` table with columns: `OrderDetailID` (Primary Key), `OrderID`, `ProductID`, `Quantity`, `UnitPrice`.
    - Populate tables with initial data.

# Additional Requirements

1. **Constraints and Relationships**:
   - Each group will create a second script to establish constraints and relationships after the tables are populated.
   - Use `ALTER TABLE` to add foreign key constraints.
   - Use `UPDATE` to ensure data consistency where necessary.
2. **Integrity**:
   - Ensure that all primary keys are unique.
   - Use appropriate data types for each column.
   - Implement `NOT NULL` constraints where applicable.
   - Use `CHECK` constraints to enforce data integrity (e.g., salary should be positive, hire date should not be in the future).
3. **Validation**:
   - Each group should validate their tables by inserting sample data before running the constraint scripts.
   - Ensure data consistency before establishing relationships.
4. **Documentation**:
   - Each group must document their tables, including column definitions, constraints, and relationships.
   - Provide SQL scripts for creating and populating the tables, and separate scripts for adding constraints and relationships.

# Example Scripts

**Group 1**:

- **Table Creation Script (create_tables.sql)**:

```
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(100) NOT NULL,
    Location VARCHAR(100)
);

CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    DepartmentID INT,
    HireDate DATE NOT NULL,
    Position VARCHAR(50),
    Salary DECIMAL(10, 2) CHECK (Salary > 0)
);

INSERT INTO Departments VALUES (1, 'HR', 'New York');
INSERT INTO Departments VALUES (2, 'IT', 'San Francisco');

INSERT INTO Employees VALUES (1, 'John', 'Doe', 1, '2022-01-10', 'Manager', 80000);
INSERT INTO Employees VALUES (2, 'Jane', 'Smith', 2, '2023-03-15', 'Developer', 90000);
```

- **Constraint and Relationship Script (alter_tables.sql)**:

```
ALTER TABLE Employees
ADD CONSTRAINT FK_Department
FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID);

-- Additional updates for data consistency if needed
UPDATE Employees SET DepartmentID = 1 WHERE EmployeeID = 1;
UPDATE Employees SET DepartmentID = 2 WHERE EmployeeID = 2;
```

## Timeline

- **Day 1**: Introduction to the project, group formation, and initial planning.
- **Day 2**: Design and creation of tables, initial data population.
- **Day 3**: Validation of data, preparation of `ALTER` and `UPDATE` scripts.
- **Day 4**: Execution of `ALTER` and `UPDATE` scripts, final testing and validation.
- **Day 5**: `ONLINE SESSION` Final presentation and review.

This approach ensures that each team carefully checks their work before establishing relationships, fostering a deeper understanding of database integrity and collaboration.

# 2 - Additional Step for the Project

Each group will create an additional script to insert data into all tables in the database, not just the tables they created. This ensures that each group understands the entire database structure and can validate the integrity of the data across all tables.

## Groups and Additional Responsibilities

1. **Group 1: Employees and Departments**
    - **Additional Task**:
        - Create a script `group1_insert_data.sql` to insert data into `Departments`, `Employees`, `Projects`, `Assignments`, `Customers`, `Orders`, `Products`, and `OrderDetails`.
2. **Group 2: Projects and Assignments**
    - **Additional Task**:
        - Create a script `group2_insert_data.sql` to insert data into `Departments`, `Employees`, `Projects`, `Assignments`, `Customers`, `Orders`, `Products`, and `OrderDetails`.
3. **Group 3: Customers and Orders**
    - **Additional Task**:
        - Create a script `group3_insert_data.sql` to insert data into `Departments`, `Employees`, `Projects`, `Assignments`, `Customers`, `Orders`, `Products`, and `OrderDetails`.
4. **Group 4: Products and OrderDetails**
    - **Additional Task**:
        - Create a script `group4_insert_data.sql` to insert data into `Departments`, `Employees`, `Projects`, `Assignments`, `Customers`, `Orders`, `Products`, and `OrderDetails`.

## Example of `group1_insert_data.sql` Script

**Group 1**:

```
-- Insert data into Departments
INSERT INTO Departments VALUES (1, 'HR', 'New York');
INSERT INTO Departments VALUES (2, 'IT', 'San Francisco');

-- Insert data into Employees
INSERT INTO Employees VALUES (1, 'John', 'Doe', 1, '2022-01-10', 'Manager', 80000);
INSERT INTO Employees VALUES (2, 'Jane', 'Smith', 2, '2023-03-15', 'Developer', 90000);

-- Insert data into Projects
INSERT INTO Projects VALUES (1, 'Project A', '2023-01-01', '2023-12-31', 500000);
INSERT INTO Projects VALUES (2, 'Project B', '2023-02-01', '2023-11-30', 300000);

-- Insert data into Assignments
INSERT INTO Assignments VALUES (1, 1, 1, 'Lead', '2023-01-01');
INSERT INTO Assignments VALUES (2, 2, 2, 'Developer', '2023-03-01');

-- Insert data into Customers
INSERT INTO Customers VALUES (1, 'Alice', '1234567890', 'alice@example.com', '123 Main St');
INSERT INTO Customers VALUES (2, 'Bob', '0987654321', 'bob@example.com', '456 Elm St');

-- Insert data into Orders
INSERT INTO Orders VALUES (1, 1, '2023-05-15', 250.75);
INSERT INTO Orders VALUES (2, 2, '2023-06-20', 150.50);

-- Insert data into Products
INSERT INTO Products VALUES (1, 'Product X', 'Category 1', 19.99, 100);
INSERT INTO Products VALUES (2, 'Product Y', 'Category 2', 29.99, 200);

-- Insert data into OrderDetails
INSERT INTO OrderDetails VALUES (1, 1, 1, 2, 19.99);
INSERT INTO OrderDetails VALUES (2, 2, 2, 1, 29.99);
```

## Timeline Update

- **Day 1**: IS THE SAME DAY OF THE `DAY 5` IN `TASK 1:18-7-2024`
- **Day 2**: Design and creation of tables, initial data population.
- **Day 3**: Validation of data, preparation of `ALTER` and `UPDATE` scripts, creation of `groupX_insert_data.sql` scripts.
- **Day 4**: Execution of `ALTER` and `UPDATE` scripts, execution of `groupX_insert_data.sql` scripts, final testing and validation.
- **Day 5**: Thursday `18-7-20204` Final presentation and review.

This additional step ensures that each group comprehensively understands the database and can contribute to its overall integrity and functionality.

# 3 - Task Plan: Using T-SQL Functions, Joins, and Subqueries

## Objectives

1. Apply T-SQL functions to manipulate and retrieve data.
2. Use joins to combine data from multiple tables.
3. Implement subqueries for complex data retrieval.

## Questions

1. **Question 1: Employee Details with Function Manipulation**

    - Retrieve a list of employees where the first name is converted to uppercase, the last name is converted to lowercase, and the length of their position title is calculated. Additionally, include the department name by joining with the `Departments` table.
    - **Hint**: Use `UPPER`, `LOWER`, `LEN`, and `JOIN`.

2. **Question 2: Department Budget Summary**

    - List all departments with the total salary expenditure rounded to the nearest thousand, and the number of employees in each department. Order the results by total salary expenditure in descending order.
    - **Hint**: Use `ROUND`, `SUM`, `COUNT`, and `GROUP BY`.

3. **Question 3: Project Assignments**

    - Retrieve a list of projects along with the names of employees assigned to each project, and include the role of the employee. Ensure the project names are in uppercase and employee names are concatenated as "FirstName LastName".
    - **Hint**: Use `UPPER`, `CONCAT`, and `JOIN`.

4. **Question 4: Customer Order Analysis**

    - List all customers who have placed orders, along with the total number of orders they have placed and the total amount spent. Ensure customer names are in lowercase.
    - **Hint**: Use `LOWER`, `COUNT`, `SUM`, and `JOIN`.

5. **Question 5: Product Details Extraction**

    - Retrieve a list of products where the product name is truncated to the first 10 characters, and the product category is extracted from the first 2 characters of the product name. Include the total quantity ordered for each product.

   - **Hint**: Use `LEFT`, `SUM`, and `JOIN`.

6. **Question 6: High Salary Employees in Specific Departments**
   - Find employees with salaries above the average salary of their respective departments. Include the employee's name, salary, and department name.
   - **Hint**: Use `AVG`, `JOIN`, and a subquery to calculate average salary per department.