

# IT2140: Database Design and Development

Follow For more:

**GitHub - Saviru Kashmira Atapattu**

*([github.com/Saviru](https://github.com/Saviru))*

## Assignment Schema: Corporate Database

You will create and use the following database schema for **all** tasks.

- **Department**
  - did (INT, Primary Key)
  - dname (VARCHAR(50), Unique, Not Null)
  - budget (FLOAT, Check > 0)
  - managerId (INT, Foreign Key to Employee)
- **Employee**
  - eid (INT, Primary Key)
  - ename (VARCHAR(100), Not Null)
  - age (INT, Check > 18)
  - salary (FLOAT, Default 30000)
  - did (INT, Foreign Key to Department)
- **Project**
  - pid (INT, Primary Key)
  - pname (VARCHAR(100), Unique)
  - deadline (DATE)
- **Works\_On**
  - eid (INT, Foreign Key to Employee)
  - pid (INT, Foreign Key to Project)

- hours\_worked (INT, Check > 0)
  - (Composite Primary Key: eid, pid)
  - **Salary\_Audit** (For Trigger)
    - logId (INT, Primary Key, Identity)
    - eid (INT)
    - old\_salary (FLOAT)
    - new\_salary (FLOAT)
    - change\_date (DATETIME)
- 

## Part 1: Data Definition Language (DDL) [Based on Lecture 02]

Write the CREATE TABLE statements for all five tables in the **Corporate Database** schema.

- You must create the tables in the correct logical order to avoid foreign key errors.
- Implement all specified constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, CHECK, DEFAULT) .
- **Note:** The Employee(managerId) foreign key references the Employee(eid) table. You may need to add this constraint using ALTER TABLE *after* both tables are created to avoid a circular dependency error.

## Part 2: Data Manipulation Language (DML) [Based on Lecture 02]

Write the SQL statements to populate the tables you created in Part 1.

### 1. INSERT :

- Insert two departments: ('Sales', 500000) and ('Engineering', 1000000).
- Insert three employees:
  - One manager for 'Sales' (e.g., 'Alice', age 40, salary 90000).
  - One manager for 'Engineering' (e.g., 'Bob', age 45, salary 110000).
  - One employee in 'Engineering' (e.g., 'Charlie', age 28, salary 70000).
- *After* inserting the employees, UPDATE the Department table to set the managerId for 'Sales' and 'Engineering' to the eids of Alice and Bob.

- Insert two projects: ('Project Alpha', '2025-12-31') and ('Project Beta', '2026-06-30').
  - Assign employees to projects:
    - Charlie works on 'Project Alpha' for 40 hours.
    - Bob works on 'Project Alpha' for 10 hours.
    - Alice works on 'Project Beta' for 20 hours.
2. **UPDATE** : Give 'Charlie' a 10% raise.
  3. **DELETE** : Add a temporary project 'Project Gamma' and then write a DELETE statement to remove it.

### Part 3: Data Query Language (DQL) [Based on Lectures 02 & 03]

Write SELECT statements to answer the following questions.

1. **Basic SELECT & WHERE**: Find the ename and age of all employees who earn more than 80,000 .
2. **LIKE & ORDER BY**: Find the pname of all projects that contain the word 'Alpha'. Sort the results alphabetically .
3. **BETWEEN**: Find the ename and salary of employees whose salary is BETWEEN 60,000 and 100,000 .
4. **IS NULL**: Find all employees (eid, ename) who have not yet been assigned to a department (did is NULL) .
5. **Aggregate Functions**: Find the average (AVG) salary, total (SUM) budget of all departments, and the MAX age of any employee .
6. **GROUP BY & HAVING**: Show the did and COUNT of employees in each department. Only include departments HAVING more than 5 employees .
7. **INNER JOIN**: List the employee name (ename) and the department name (dname) they work for .
8. **3-Table JOIN**: List the names of employees (ename), the names of the projects (pname) they are working on, and the hours\_worked for each assignment.

9. **LEFT OUTER JOIN:** List **all** employees (ename) and the pname of any project they are on. Employees with no projects should still appear in the list (with a NULL pname) .
10. **RIGHT OUTER JOIN:** List **all** projects (pname) and the ename of any employee working on them. Projects with no employees assigned should still appear in the list.
11. **Subquery (IN):** Find the ename and salary of all employees who work in the 'Engineering' department (use a subquery on the Department table to find the did).
12. **Subquery (> ANY or > MIN):** Find the ename of all employees who earn more than *at least one* manager.

#### Part 4: Views [Based on Lecture 07]

Write the SQL statements to create and use views.

1. **CREATE VIEW (Simple):** Create a view named v\_Engineering\_Employees that shows the eid, ename, and salary of all employees in the 'Engineering' department .
2. **CREATE VIEW (Complex):** Create a view named v\_Project\_Summary that lists each pid, pname, and a new column TotalHours (which is the SUM of hours\_worked for that project) .
3. **Query View:** Write a SELECT query on your v\_Engineering\_Employees view to find employees earning less than 75,000 .
4. **DROP VIEW:** Delete the v\_Engineering\_Employees view .

#### Part 5: T-SQL Database Programming [Based on Lecture 07]

Implement business logic using T-SQL.

1. **CREATE FUNCTION (Scalar):**
  - Create a function fn\_GetTotalHours that takes an @eid (INT) as input.
  - The function should RETURN the total hours\_worked for that employee across all projects .
2. **CREATE PROCEDURE (IN Parameters):**

- Create a procedure sp\_AssignProject that takes @employeeID (INT) and @projectID (INT) as input parameters.
- The procedure should INSERT a new record into the Works\_On table with a default hours\_worked of 10 .

### 3. CREATE PROCEDURE (OUT Parameters):

- Create a procedure sp\_GetDeptStats that takes a @dname (VARCHAR) as an input parameter.
- The procedure must use OUTPUT parameters to return the did, budget, and the total COUNT of employees for that department .

### 4. AFTER Trigger:

- Create a trigger named trg\_AuditSalary on the Employee table that fires AFTER UPDATE .
- If the salary column was updated, the trigger should INSERT the eid, old\_salary (from the deleted table), new\_salary (from the inserted table), and the current date into the Salary\_Audit table .

### 5. INSTEAD OF Trigger:

- Create a trigger named trg\_SafeDeleteDept on the Department table that fires INSTEAD OF DELETE .
- Instead of deleting the department, the trigger should:
  1. UPDATE all employees in that department to have a NULL did.
  2. DELETE the department from the Department table.
  3. Print a message 'Department removed and employees unassigned.'

### 6. Business Rule Trigger:

- Create a trigger on the Employee table that fires AFTER INSERT, UPDATE.
- This trigger must enforce the rule: **"No employee can have a salary greater than their manager."**
- You will need to join the inserted table with the Employee table (to find the manager) and ROLLBACK the transaction if the rule is violated.