

Database assignment – 3

- Creation of database :
 - CREATE DATABASE Assignment_3;

- Creation of Table :
 - CREATE TABLE Department
(pk_department_id INT NOT NULL PRIMARY KEY,
department_name VARCHAR(255) NOT NULL);

 - CREATE TABLE Employee
(pk_employee_id INT NOT NULL PRIMARY KEY,
first_name VARCHAR(255) NOT NULL,
last_name VARCHAR(255) NOT NULL,
salary INT NOT NULL,
DOB DATE,
joining_date DATE,
fk_department_id INT NOT NULL,
last_modified DATETIME NOT NULL,
FOREIGN KEY (fk_department_id) REFERENCES
Department(pk_department_id));

 - CREATE TABLE Project
(pk_project_id INT NOT NULL PRIMARY KEY,
project_title VARCHAR(255) NOT NULL,
project_description VARCHAR(MAX) NOT NULL,
project_start_date DATE,
project_end_date DATE);

- CREATE TABLE Employee_project
 - (pk_emp_project_id INT NOT NULL PRIMARY KEY,
 - fk_employee_id INT NOT NULL,
 - fk_project_id INT NOT NULL,
 - FOREIGN KEY (fk_employee_id) REFERENCES Employee(pk_employee_id),
 - FOREIGN KEY (fk_project_id) REFERENCES Project(pk_project_id));

- CREATE TABLE Tasks
 - (pk_task_id INT NOT NULL PRIMARY KEY,
 - task_description VARCHAR(255) NOT NULL,
 - fk_employee_project_id INT NOT NULL,
 - create_time DATETIME NOT NULL,
 - FOREIGN KEY (fk_employee_project_id) REFERENCES Employee_project(pk_emp_project_id));

- CREATE TABLE Subtasks
 - (pk_subtask_id INT NOT NULL PRIMARY KEY,
 - subtask_description VARCHAR(255) NOT NULL,
 - fk_task_id INT NOT NULL,
 - create_time DATETIME NOT NULL,
 - FOREIGN KEY (fk_task_id) REFERENCES Tasks(pk_task_id));

- CREATE TABLE Employee_Audit
 - (pk_emp_audit_id INT NOT NULL PRIMARY KEY,
 - fk_employee_id INT NOT NULL,
 - delete_time DATETIME DEFAULT GETDATE());

- Insertion of data :
 - INSERT INTO Department VALUES
 - (1, 'Designing'),
 - (2, 'QA'),
 - (3, 'Devloperment');

 - INSERT INTO Employee VALUES
 - (1, 'Akshay', 'Vaghasiya', 15000, '2003-09-15', '2025-01-16', 3, GETDATE()),

```
(2, 'Rushi', 'Sureja', 50000, '2003-08-04', '2025-01-20', 1, GETDATE()),  
(3, 'Abhay', 'Gohel', 40000, '2003-11-09', '2024-12-18', 2, GETDATE()),  
(4, 'Mandar', 'Parekh', 45000, '2003-08-29', '2024-12-23', 3, GETDATE()),  
(5, 'Nikhil', 'Vaghasiya', 120000, '2002-12-31', '2024-03-10', 2,  
GETDATE());
```

➤ INSERT INTO Project VALUES

```
(1, 'Trend loom', 'Ecommerce website for clothing.', '2025-01-20',  
NULL),  
(2, 'Bytebattles', 'Online competitive coding platform.', '2025-01-31',  
'2025-05-01'),  
(3, 'Project Management System', 'Platform for managing projects.',  
'2025-03-01', NULL);
```

➤ INSERT INTO Employee_project VALUES

```
(1, 1, 1),  
(2, 2, 1),  
(3, 3, 1),  
(4, 4, 2),  
(5, 5, 2),  
(6, 2, 2),  
(7, 1, 3),  
(8, 4, 3),  
(9, 5, 3),  
(10, 2, 3);
```

➤ INSERT INTO Tasks VALUES

```
(1, 'Design product page', 2, '2025-01-21 10:14:35'),  
(2, 'Develop product page', 1, '2025-01-23 11:30:09'),  
(3, 'Test product page', 3, '2025-01-31 10:30:59'),  
(4, 'Design home page', 6, '2025-02-01 13:30:20'),  
(5, 'Develop home page', 4, '2025-02-03 15:20:10'),  
(6, 'Test home page', 5, '2025-01-10 12:30:09'),  
(7, 'Design project list page', 10, '2025-03-01 10:30:09'),  
(8, 'Devlop frontend for project list page', 7, '2025-03-03 17:30:09'),  
(9, 'Devlop backend apis for projects', 8, '2025-03-03 11:10:09'),  
(10, 'Test project list page', 9, '2025-03-10 11:30:09');
```

➤ INSERT INTO Subtasks VALUES

```
(1, 'Design particular product detail page', 1, '2025-01-21 10:14:35'),  
(2, 'Develop single product detail page', 2, '2025-01-23 11:30:09'),  
(3, 'Test responsiveness of product page', 3, '2025-01-31 10:30:59'),  
(4, 'Design home page for mobile view', 4, '2025-02-01 13:30:20'),  
(5, 'Develop home page for mobile view', 5, '2025-02-03 15:20:10'),  
(6, 'Test responsiveness of home page', 6, '2025-01-10 12:30:09'),  
(7, 'Design searching and filtering project list', 7, '2025-03-01  
10:30:09'),  
(8, 'Integrate frontend and backend for project list page', 8, '2025-03-  
03 17:30:09'),  
(9, 'Develop backend apis for searching and filtering projects', 9, '2025-  
03-03 11:10:09'),  
(10, 'Test functionality of project list page', 10, '2025-03-10 11:30:09');
```

❖ Stored Procedures :

- Write a stored procedure that retrieves all records from the "Employees" table.

```
➤ CREATE PROCEDURE GetEmployees  
AS  
BEGIN  
    SELECT * FROM Employee;  
END;
```

➤ EXEC GetEmployees;

Results		Messages						
	pk_employee_id	first_name	last_name	salary	DOB	joining_date	fk_department_id	last_modified
1	1	Akshay	Vaghasiya	15000	2003-09-15	2025-01-16	3	2025-01-18 17:26:49.723
2	2	Rushi	Sureja	50000	2003-08-04	2025-01-20	1	2025-01-18 17:26:49.723
3	3	Abhay	Gohel	40000	2003-11-09	2024-12-18	2	2025-01-18 17:26:49.723
4	4	Mandar	Parekh	45000	2003-08-29	2024-12-23	3	2025-01-18 17:26:49.723
5	5	Nikhil	Vaghasiya	120000	2002-12-31	2024-03-10	2	2025-01-18 17:26:49.723

- Create a stored procedure that takes an employee ID as input and returns the corresponding employee's name and department.

```
➤ CREATE PROCEDURE GetEmployeeById
    @EmployeeId INT
AS
BEGIN
    SELECT e.first_name, e.last_name, d.department_name FROM
    Employee e
    INNER JOIN Department d ON e.fk_department_id =
    d.pk_department_id
    WHERE e.pk_employee_id = @EmployeeId;
END;

➤ EXEC GetEmployeeById @EmployeeId=3;
```

Results		Messages	
	first_name	last_name	department_name
1	Abhay	Gohel	QA

- Write a stored procedure that takes an employee's ID and a percentage as input and updates the employee's salary by increasing it by the given percentage. Ensure that the procedure handles transaction management and rollback in case of any errors during the update.

```
➤ CREATE PROCEDURE UpdateEmployeeSalary
    @EmployeeID INT,
    @Percentage FLOAT
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;
        UPDATE Employee SET salary = salary +
        (salary*@Percentage/100)
        WHERE pk_employee_id = @EmployeeID;
        COMMIT TRANSACTION;
    END TRY
```

```

        BEGIN CATCH
            IF @@TRANCOUNT > 0
            BEGIN
                ROLLBACK TRANSACTION;
            END
        END CATCH
    END;

```

- Create a stored procedure that performs a complex join operation across three tables: "Employees," "Departments," and "Projects." The procedure should return the department name, employee name, their corresponding salaries, and a comma-separated list of project names for employees working on multiple projects, filtered by a given range of salary values.

```

➤ CREATE PROCEDURE GetEmployeeBySalaryRange
    @Minsalary INT,
    @Maxsalary INT
AS
BEGIN
    SELECT e.first_name, d.department_name, e.salary,
    STRING_AGG(p.project_title, ', ') AS projects FROM Employee e
    INNER JOIN Department d ON e.fk_department_id =
    d.pk_department_id
    INNER JOIN Employee_project ep ON e.pk_employee_id =
    ep.fk_employee_id
    INNER JOIN Project p ON ep.fk_project_id = p.pk_project_id
    WHERE e.salary BETWEEN @Minsalary AND @Maxsalary
    GROUP BY e.first_name, d.department_name, e.salary;
END;

➤ EXEC GetEmployeeBySalaryRange @Minsalary = 20000, @Maxsalary =
50000;

```

Results		Messages		
	first_name	department_name	salary	projects
1	Abhay	QA	40000	Trend loom
2	Mandar	Devlopement	45000	Bytebattles, Project Management System
3	Rushi	Designing	50000	Trend loom, Bytebattles, Project Management System

❖ Triggers :

- Write a trigger that automatically updates the "last_modified" timestamp column in the "Employee" table whenever a row in this table is updated.
 - ```
CREATE TRIGGER Update_last_modified
ON Employee
AFTER UPDATE
AS
BEGIN
 UPDATE Employee SET last_modified = GETDATE()
 FROM Employee INNER JOIN inserted ON Employee.pk_employee_id
 = inserted.pk_employee_id;
END;
```
- Create a trigger that logs the deletion of any row from the "Employee" table into a "Employee\_Audit" table, capturing the employee ID and deletion timestamp.
  - ```
CREATE TRIGGER Log_employee_delete
ON Employee
AFTER DELETE
AS
BEGIN
    INSERT INTO Employee_Audit
    SELECT deleted.pk_employee_id, deleted.pk_employee_id,
    GETDATE() FROM deleted;
END;
```

- Write a trigger that ensures no employee can be assigned to more than one project at a time. If an insert or update operation violates this rule, the trigger should prevent the operation and raise an appropriate error message.

```
➤ CREATE TRIGGER Check_single_project
  ON Employee_project
  AFTER INSERT, UPDATE
  AS
  BEGIN
      IF EXISTS (SELECT 1 FROM Employee_project WHERE
        fk_employee_id=
        (SELECT inserted.fk_employee_id FROM inserted)
        AND fk_project_id != (SELECT inserted.fk_project_id FROM
        inserted))
      BEGIN
          RAISERROR('Employee cannot be assigned to more
            than one project at a time',16, 1);
          ROLLBACK;
      END
  END;
END;
```

- Develop a trigger that enforces a cascading delete in a complex hierarchical table structure. For instance, if a record in the "Projects" table is deleted, the trigger should delete all related records in the "Tasks" and "Subtasks" tables, ensuring data integrity across multiple levels.

```
➤ CREATE TRIGGER cascade_delete_project
  ON project
  AFTER DELETE
  AS
  BEGIN
      DELETE FROM Subtasks WHERE fk_task_id IN
      (SELECT pk_task_id FROM Tasks WHERE
      Tasks.fk_employee_project_id IN
```



```
(SELECT pk_emp_project_id FROM Employee_project WHERE
Employee_project.fk_project_id IN
(SELECT deleted.pk_project_id FROM deleted))));
```

```
DELETE FROM Tasks WHERE fk_employee_project_id IN
(SELECT pk_emp_project_id FROM Employee_project WHERE
Employee_project.fk_project_id IN
(SELECT deleted.pk_project_id FROM deleted));
```

```
DELETE FROM Employee_project WHERE fk_project_id IN
(SELECT deleted.pk_project_id FROM deleted);
```

END;

❖ Functions :

- Write a function that takes an employee ID as input and returns the employee's full name (first name and last name concatenated.)

```
➤ CREATE FUNCTION GetEmployeeName(@EmployeeId INT)
RETURNS VARCHAR(255)
AS
BEGIN
    DECLARE @name VARCHAR(255);
    SELECT @name = CONCAT(first_name, ' ', last_name) FROM
    Employee WHERE pk_employee_id = @EmployeeId;
    RETURN @name;
END;
```

```
➤ SELECT dbo.GetEmployeeName(1);
```

Results Messages	
	(No column name)
1	Akshay Vaghasiya

- Create a function that takes an employee ID and returns the employee's age.

```
➤ CREATE FUNCTION GetAge(@EmployeeId INT)
    RETURNS INT
    AS
    BEGIN
        DECLARE @age INT;
        SELECT @age = DATEDIFF(YEAR, DOB, GETDATE()) FROM Employee
        WHERE pk_employee_id = @EmployeeId;
        RETURN @age-1;
    END;

➤ SELECT dbo.GetAge(1);
```

Results Messages	
	(No column name)
1	21

- Write a function that calculates the number of years an employee has been with the company based on their joining date.

```
➤ CREATE FUNCTION GetWorkYear(@EmployeeId INT)
    RETURNS FLOAT
    AS
    BEGIN
        DECLARE @year FLOAT;
        SELECT @year = DATEDIFF(YEAR, joining_date, GETDATE()) FROM
        Employee WHERE pk_employee_id = @EmployeeId;
        RETURN @year;
    END;

➤ SELECT dbo.GetWorkYear(1);
```

Results Messages	
	(No column name)
1	0

- Create a function that takes an employee ID and returns the total number of projects the employee is assigned to.
 - ```
CREATE FUNCTION GetNoOfProject(@EmployeeId INT)
RETURNS INT
AS
BEGIN
 DECLARE @project INT;
 SELECT @project = COUNT(fk_employee_id) FROM Employee_project
 WHERE fk_employee_id = @EmployeeId GROUP BY fk_employee_id;
 RETURN @project;
END;
```
  - ```
SELECT dbo.GetNoOfProject(1);
```

Results Messages	
	(No column name)
1	2

❖ Cursors :

- Write a cursor that iterates over all rows in the "Employee" table and prints each employee ID and employee joining date.
 - ```
DECLARE emp_cursor CURSOR FOR SELECT pk_employee_id, joining_date
FROM Employee;

DECLARE @EmployeeId INT
DECLARE @JoiningDate DATE;

OPEN emp_cursor;

FETCH NEXT FROM emp_cursor INTO @EmployeeId, @JoiningDate;

WHILE @@FETCH_STATUS = 0
BEGIN
 PRINT 'EmployeeId : ' + CAST(@EmployeeId AS VARCHAR(5)) +
```

```
, JoinginDate : ' + CAST(@JoiningDate AS VARCHAR(10));
```

```
FETCH NEXT FROM emp_cursor INTO @EmployeeId, @JoiningDate;
END;
```

```
CLOSE emp_cursor;
```

#### Messages

```
EmployeeId : 1, JoinginDate : 2025-01-16
EmployeeId : 2, JoinginDate : 2025-01-20
EmployeeId : 3, JoinginDate : 2024-12-18
EmployeeId : 4, JoinginDate : 2024-12-23
EmployeeId : 5, JoinginDate : 2024-03-10
```

- Create a cursor that loops through the "Employees" table and increments the salary of each employee by 5%.
  - DECLARE emp\_cursor CURSOR FOR SELECT pk\_employee\_id, salary FROM Employee;

```
DECLARE @EmployeeId INT
DECLARE @CurrSalary INT;
```

```
OPEN emp_cursor;
```

```
FETCH NEXT FROM emp_cursor INTO @EmployeeId, @CurrSalary;
```

```
WHILE @@FETCH_STATUS = 0
BEGIN
```

```
 UPDATE Employee SET salary = @CurrSalary * 1.05
 WHERE pk_employee_id = @EmployeeId;
```

```
 FETCH NEXT FROM emp_cursor INTO @EmployeeId, @CurrSalary;
END;
```

```
CLOSE emp_cursor;
```

| Results Messages |                |            |           |        |            |              |                  |                         |
|------------------|----------------|------------|-----------|--------|------------|--------------|------------------|-------------------------|
|                  | pk_employee_id | first_name | last_name | salary | DOB        | joining_date | fk_department_id | last_modified           |
| 1                | 1              | Akshay     | Vaghasiya | 15750  | 2003-09-15 | 2025-01-16   | 3                | 2025-01-18 23:05:57.927 |
| 2                | 2              | Rushi      | Sureja    | 52500  | 2003-08-04 | 2025-01-20   | 1                | 2025-01-18 23:05:57.927 |
| 3                | 3              | Abhay      | Gohel     | 42000  | 2003-11-09 | 2024-12-18   | 2                | 2025-01-18 23:05:57.927 |
| 4                | 4              | Mandar     | Parekh    | 47250  | 2003-08-29 | 2024-12-23   | 3                | 2025-01-18 23:05:57.927 |
| 5                | 5              | Nikhil     | Vaghasiya | 126000 | 2002-12-31 | 2024-03-10   | 2                | 2025-01-18 23:05:57.927 |

## ❖ Views :

- Write a view that combines the "Employees" and "Departments" tables to show employee names along with their department names.

➤ CREATE VIEW Employee\_department AS  
 SELECT e.first\_name, e.last\_name, d.department\_name FROM Employee e  
 INNER JOIN Department d ON e.fk\_department\_id = d.pk\_department\_id;

| Results Messages |            |           |                 |
|------------------|------------|-----------|-----------------|
|                  | first_name | last_name | department_name |
| 1                | Akshay     | Vaghasiya | Devlopement     |
| 2                | Rushi      | Sureja    | Designing       |
| 3                | Abhay      | Gohel     | QA              |
| 4                | Mandar     | Parekh    | Devlopement     |
| 5                | Nikhil     | Vaghasiya | QA              |

- Write a view that displays the average salary of employees in each department. The view should include the DepartmentID, DepartmentName, and AverageSalary.

➤ CREATE VIEW Department\_avg\_salary AS  
 SELECT d.pk\_department\_id, d.department\_name, AVG(e.salary) AS  
 avg\_salary FROM Department d INNER JOIN Employee e ON  
 d.pk\_department\_id = e.fk\_department\_id GROUP BY d.department\_name,  
 d.pk\_department\_id;

| Results |                  | Messages        |            |
|---------|------------------|-----------------|------------|
|         | pk_department_id | department_name | avg_salary |
| 1       | 1                | Designing       | 52500      |
| 2       | 2                | QA              | 84000      |
| 3       | 3                | Devloperment    | 31500      |