CRUD Operations using ADO.NET Connected Architecture (C#)

This document demonstrates how to perform CRUD (Create, Read, Update, Delete) operations on an Employee table using ADO.NET Connected Architecture in C#.

# SQL Table Structure

CREATE TABLE Employee (  
 Id INT PRIMARY KEY IDENTITY,  
 Name NVARCHAR(100),  
 Department NVARCHAR(50),  
 Salary DECIMAL(10, 2)  
);

# C# Code - EmployeeCRUD Class

using System;  
using System.Data.SqlClient;  
  
class EmployeeCRUD  
{  
 string connectionString = "Server=YOUR\_SERVER;Database=YOUR\_DB;Trusted\_Connection=True;";  
  
 // CREATE  
 public void AddEmployee(string name, string department, decimal salary)  
 {  
 using (SqlConnection conn = new SqlConnection(connectionString))  
 {  
 string query = "INSERT INTO Employee (Name, Department, Salary) VALUES (@Name, @Department, @Salary)";  
 SqlCommand cmd = new SqlCommand(query, conn);  
 cmd.Parameters.AddWithValue("@Name", name);  
 cmd.Parameters.AddWithValue("@Department", department);  
 cmd.Parameters.AddWithValue("@Salary", salary);  
 conn.Open();  
 cmd.ExecuteNonQuery();  
 Console.WriteLine("Employee added successfully.");  
 }  
 }  
  
 // READ  
 public void GetEmployees()  
 {  
 using (SqlConnection conn = new SqlConnection(connectionString))  
 {  
 string query = "SELECT \* FROM Employee";  
 SqlCommand cmd = new SqlCommand(query, conn);  
 conn.Open();  
 SqlDataReader reader = cmd.ExecuteReader();  
  
 while (reader.Read())  
 {  
 Console.WriteLine($"ID: {reader["Id"]}, Name: {reader["Name"]}, Department: {reader["Department"]}, Salary: {reader["Salary"]}");  
 }  
  
 reader.Close();  
 }  
 }  
  
 // UPDATE  
 public void UpdateEmployee(int id, string name, string department, decimal salary)  
 {  
 using (SqlConnection conn = new SqlConnection(connectionString))  
 {  
 string query = "UPDATE Employee SET Name = @Name, Department = @Department, Salary = @Salary WHERE Id = @Id";  
 SqlCommand cmd = new SqlCommand(query, conn);  
 cmd.Parameters.AddWithValue("@Id", id);  
 cmd.Parameters.AddWithValue("@Name", name);  
 cmd.Parameters.AddWithValue("@Department", department);  
 cmd.Parameters.AddWithValue("@Salary", salary);  
 conn.Open();  
 int rows = cmd.ExecuteNonQuery();  
 Console.WriteLine($"{rows} employee(s) updated.");  
 }  
 }  
  
 // DELETE  
 public void DeleteEmployee(int id)  
 {  
 using (SqlConnection conn = new SqlConnection(connectionString))  
 {  
 string query = "DELETE FROM Employee WHERE Id = @Id";  
 SqlCommand cmd = new SqlCommand(query, conn);  
 cmd.Parameters.AddWithValue("@Id", id);  
 conn.Open();  
 int rows = cmd.ExecuteNonQuery();  
 Console.WriteLine($"{rows} employee(s) deleted.");  
 }  
 }  
}

# Main Method Example

class Program  
{  
 static void Main()  
 {  
 EmployeeCRUD crud = new EmployeeCRUD();  
  
 crud.AddEmployee("Alice", "HR", 50000);  
 crud.GetEmployees();  
 crud.UpdateEmployee(1, "Alice Smith", "HR", 52000);  
 crud.DeleteEmployee(1);  
 }  
}

# Notes

- Replace "YOUR\_SERVER" and "YOUR\_DB" in the connection string with actual server and database names.  
- Always use parameterized queries to avoid SQL Injection.  
- Add proper exception handling in production code.