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

This document demonstrates how to perform CRUD (Create, Read, Update, Delete) operations on an Employee table using ADO.NET Disconnected 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\_Disconnected Class

using System;  
using System.Data;  
using System.Data.SqlClient;  
  
class EmployeeCRUD\_Disconnected  
{  
 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))  
 {  
 SqlDataAdapter adapter = new SqlDataAdapter("SELECT \* FROM Employee", conn);  
 SqlCommandBuilder builder = new SqlCommandBuilder(adapter);  
  
 DataSet ds = new DataSet();  
 adapter.Fill(ds, "Employee");  
  
 DataRow newRow = ds.Tables["Employee"].NewRow();  
 newRow["Name"] = name;  
 newRow["Department"] = department;  
 newRow["Salary"] = salary;  
  
 ds.Tables["Employee"].Rows.Add(newRow);  
 adapter.Update(ds, "Employee");  
  
 Console.WriteLine("Employee added successfully (Disconnected).");  
 }  
 }  
  
 // READ  
 public void GetEmployees()  
 {  
 using (SqlConnection conn = new SqlConnection(connectionString))  
 {  
 SqlDataAdapter adapter = new SqlDataAdapter("SELECT \* FROM Employee", conn);  
 DataSet ds = new DataSet();  
 adapter.Fill(ds, "Employee");  
  
 foreach (DataRow row in ds.Tables["Employee"].Rows)  
 {  
 Console.WriteLine($"ID: {row["Id"]}, Name: {row["Name"]}, Department: {row["Department"]}, Salary: {row["Salary"]}");  
 }  
 }  
 }  
  
 // UPDATE  
 public void UpdateEmployee(int id, string name, string department, decimal salary)  
 {  
 using (SqlConnection conn = new SqlConnection(connectionString))  
 {  
 SqlDataAdapter adapter = new SqlDataAdapter("SELECT \* FROM Employee", conn);  
 SqlCommandBuilder builder = new SqlCommandBuilder(adapter);  
  
 DataSet ds = new DataSet();  
 adapter.Fill(ds, "Employee");  
  
 DataRow row = ds.Tables["Employee"].Rows.Find(id);  
 if (row != null)  
 {  
 row["Name"] = name;  
 row["Department"] = department;  
 row["Salary"] = salary;  
 adapter.Update(ds, "Employee");  
 Console.WriteLine("Employee updated successfully (Disconnected).");  
 }  
 else  
 {  
 Console.WriteLine("Employee not found.");  
 }  
 }  
 }  
  
 // DELETE  
 public void DeleteEmployee(int id)  
 {  
 using (SqlConnection conn = new SqlConnection(connectionString))  
 {  
 SqlDataAdapter adapter = new SqlDataAdapter("SELECT \* FROM Employee", conn);  
 SqlCommandBuilder builder = new SqlCommandBuilder(adapter);  
  
 DataSet ds = new DataSet();  
 adapter.Fill(ds, "Employee");  
  
 DataRow row = ds.Tables["Employee"].Rows.Find(id);  
 if (row != null)  
 {  
 row.Delete();  
 adapter.Update(ds, "Employee");  
 Console.WriteLine("Employee deleted successfully (Disconnected).");  
 }  
 else  
 {  
 Console.WriteLine("Employee not found.");  
 }  
 }  
 }  
}

# Notes

- Replace "YOUR\_SERVER" and "YOUR\_DB" in the connection string with actual server and database names.  
- Use SqlCommandBuilder only for simple CRUD scenarios.  
- Consider using DataTable.PrimaryKey to define keys for DataRow.Find() if needed.  
- Always include proper exception handling and validation in production code.