**1. Create a Table**

**Exercise:**  
Create a table Employees with the following columns:

* EmployeeID (Primary Key, int)
* Name (nvarchar(50))
* Position (nvarchar(50))
* Salary (decimal(10,2))
* HireDate (datetime).

**Solution:**

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name NVARCHAR(50),

Position NVARCHAR(50),

Salary DECIMAL(10, 2),

HireDate DATETIME

);

**2. Insert Records**

**Exercise:**  
Insert the following records into the Employees table:

1. John Doe, Manager, 75000.00, 2020-03-01
2. Jane Smith, Developer, 65000.00, 2021-07-15
3. Emily Davis, Analyst, 55000.00, 2022-01-10

**Solution:**

INSERT INTO Employees (EmployeeID, Name, Position, Salary, HireDate)

VALUES

(1, 'John Doe', 'Manager', 75000.00, '2020-03-01'),

(2, 'Jane Smith', 'Developer', 65000.00, '2021-07-15'),

(3, 'Emily Davis', 'Analyst', 55000.00, '2022-01-10');

**3. Select All Records**

**Exercise:**  
Write a query to fetch all records from the Employees table.

**Solution:**

SELECT \* FROM Employees;

**4. Filter Records**

**Exercise:**  
Write a query to fetch all employees with a salary greater than 60000.

**Solution:**

SELECT \*

FROM Employees

WHERE Salary > 60000;

**5. Update a Record**

**Exercise:**  
Update Jane Smith's position to 'Senior Developer'.

**Solution:**

UPDATE Employees

SET Position = 'Senior Developer'

WHERE Name = 'Jane Smith';

**6. Delete a Record**

**Exercise:**  
Delete the employee record with the name 'Emily Davis'.

**Solution:**

DELETE FROM Employees

WHERE Name = 'Emily Davis';

**7. Aggregate Functions**

**Exercise:**  
Find the total salary of all employees.

**Solution:**

SELECT SUM(Salary) AS TotalSalary

FROM Employees;

**8. Count Records**

**Exercise:**  
Count the total number of employees.

**Solution:**

SELECT COUNT(\*) AS EmployeeCount

FROM Employees;

**9. Group By**

**Exercise:**  
Group employees by position and find the average salary for each position.

**Solution:**

SELECT Position, AVG(Salary) AS AverageSalary

FROM Employees

GROUP BY Position;

**10. Join Tables**

**Exercise:**  
Create a Departments table with the following data:

* DepartmentID (Primary Key, int)
* DepartmentName (nvarchar(50))
* EmployeeID (Foreign Key, int)

Insert sample records and write a query to join Employees and Departments to display employee names along with their department names.

**Solution:**

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName NVARCHAR(50),

EmployeeID INT FOREIGN KEY REFERENCES Employees(EmployeeID)

);

INSERT INTO Departments (DepartmentID, DepartmentName, EmployeeID)

VALUES

(1, 'IT', 2),

(2, 'HR', 1);

SELECT e.Name, d.DepartmentName

FROM Employees e

JOIN Departments d

ON e.EmployeeID = d.EmployeeID;

**11. Subquery**

**Exercise:**  
Find employees who earn more than the average salary.

**Solution:**

SELECT Name, Salary

FROM Employees

WHERE Salary > (SELECT AVG(Salary) FROM Employees);

**12. Common Table Expressions (CTE)**

**Exercise:**  
Use a CTE to find all employees hired after 2021.

**Solution:**

WITH RecentHires AS (

SELECT \*

FROM Employees

WHERE HireDate > '2021-01-01'

)

SELECT \* FROM RecentHires;

**13. Ranking Function**

**Exercise:**  
Rank employees by their salary in descending order.

**Solution:**

SELECT Name, Salary,

RANK() OVER (ORDER BY Salary DESC) AS Rank

FROM Employees;

**14. Transactions**

**Exercise:**  
Insert a new employee and then roll back the transaction.

**Solution:**

BEGIN TRANSACTION;

INSERT INTO Employees (EmployeeID, Name, Position, Salary, HireDate)

VALUES (4, 'Mark Brown', 'Consultant', 58000.00, '2023-05-20');

ROLLBACK TRANSACTION;

**15. Stored Procedure**

**Exercise:**  
Create a stored procedure to fetch employees by position.

**Solution:**

CREATE PROCEDURE GetEmployeesByPosition

@Position NVARCHAR(50)

AS

BEGIN

SELECT \*

FROM Employees

WHERE Position = @Position;

END;

**16. View**

**Exercise:**  
Create a view that shows employee names and their positions.

**Solution:**

CREATE VIEW EmployeeDetails AS

SELECT Name, Position

FROM Employees;

**17. Index**

**Exercise:**  
Create an index on the Salary column.

**Solution:**

CREATE INDEX idx\_Salary ON Employees(Salary);

**18. Error Handling**

**Exercise:**  
Write a TRY-CATCH block to handle errors during an update.

**Solution:**

BEGIN TRY

UPDATE Employees

SET Salary = -50000

WHERE EmployeeID = 1;

END TRY

BEGIN CATCH

PRINT 'An error occurred: ' + ERROR\_MESSAGE();

END CATCH;

**19. Pivot Table**

**Exercise:**  
Pivot the data to display the total salary by position.

**Solution:**

SELECT \*

FROM (

SELECT Position, Salary

FROM Employees

) AS SourceTable

PIVOT (

SUM(Salary) FOR Position IN ([Manager], [Developer], [Analyst])

) AS PivotTable;

**20. Dynamic SQL**

**Exercise:**  
Write a dynamic SQL query to fetch employees by position.

**Solution:**

DECLARE @Position NVARCHAR(50) = 'Manager';

DECLARE @SQL NVARCHAR(MAX);

SET @SQL = 'SELECT \* FROM Employees WHERE Position = ''' + @Position + '''';

EXEC sp\_executesql @SQL;