```sql
use CompanyDB2
```

---PART 2

--SQL Server Assignments on Aggregate Queries (SUM, MIN, MAX, AVG)

---These assignments will help you practice using aggregate functions in SQL Server, such as
--SUM(), MIN(), MAX(), and AVG(). Aggregate functions allow you to calculate values from
--multiple rows in a result set, returning a single value.

---Assignment 1 SALES DATABASE

```sql
CREATE TABLE Sales(SaleID INT IDENTITY(1,1) PRIMARY KEY,ProductID INT, ProductName
VARCHAR(50), QuantitySold INT, SaleAmount INT, SaleDate DATE)

INSERT INTO Sales(ProductID, ProductName, QuantitySold, SaleAmount, SaleDate)VALUES
(101,'Laptop',5,5000,'2024-09-01'),
(102,'Mouse',10,250,'2024-09-02'),
(101,'Laptop',3,3000,'2024-09-03'),
(103,'Keyboard',8,800,'2024-09-04'),
(102,'Mouse',6,150,'2024-09-05')
SELECT * FROM Sales
```

----1. Write a query to calculate the total sales amount for all products (SUM).

```sql
SELECT SUM(SaleAmount) AS TotalSalesAmount FROM Sales
```

----2. Write a query to find the maximum quantity sold of any product (MAX).

```sql
SELECT MAX(QuantitySold) AS MaxQuantitySold FROM Sales
```

---3. Write a query to calculate the average sale amount per sale (AVG).

```sql
SELECT AVG(SaleAmount) AS AverageSalesAmount FROM Sales
```

---4. Write a query to find the minimum sale amount in the database (MIN).

```sql
SELECT MIN(SaleAmount) AS MinSalesAmount FROM Sales
```

---5. Write a query to calculate the total quantity sold of a specific product, say "Laptop" (SUM with WHERE clause).

```sql
SELECT SUM(QuantitySold) AS TotalLaptopQuantity FROM Sales WHERE ProductName =
'Laptop'
```

--Assignment 2 EMPLOYEE SALARIES

```sql
CREATE TABLE Employees(EmployeeID INT IDENTITY(1,1) PRIMARY KEY, EmployeeName
VARCHAR(50), Department VARCHAR(50), Salary INT, HireDate DATE)
INSERT INTO Employees(EmployeeName, Department, Salary, HireDate) VALUES
```

```sql
('John Doe','IT','5000','2022-01-10'),
('Jane Smith','HR','6000','2021-03-15'),
('Boby Johnson','Finance','5500','2020-06-20'),
('Alice Brown','IT','6200','2023-02-01'),
('Charlie White','Marketing','4800','2023-03-10')
SELECT * FROM Employees

---1. Write a query to find the total salary expenditure for all employees (SUM).
SELECT SUM(Salary) AS TotalSalaryExpenditures FROM Employees

---2. Write a query to find the highest salary among all employees (MAX).
SELECT MAX(Salary) AS HighestSalary FROM Employees

---3. Write a query to calculate the average salary across all employees (AVG).
SELECT AVG(Salary) AS AverageSalary FROM Employees

---4. Write a query to find the minimum salary in the IT department (MIN with WHERE clause).
SELECT MIN(Salary) AS MinimumSalaryIT FROM Employees WHERE Department = 'IT'

-- 5. Write a query to find the total number of employees in each department (COUNT and
GROUP BY).
SELECT Department,COUNT(*) AS TotalNoOfEmployees FROM Employees GROUP BY
Department

--Assignment 3 INVENTORY MANAGEMENT
CREATE TABLE Inventory(ProductID INT IDENTITY(201,1) PRIMARY KEY, ProductName
VARCHAR(50), QuantityInStock INT, ReorderLevel INT, LastRestockDate DATE)
INSERT INTO Inventory(ProductName, QuantityInStock, ReorderLevel, LastRestockDate)
VALUES
('Monitor',50,10,'2024-08-25'),
('Printer',30,5,'2024-09-01'),
('Mouse',100,15,'2024-09-10'),
('Keyboard',80,20,'2024-09-15'),
('Laptop',25,5,'2024-09-20')
SELECT * FROM Inventory

---1. Write a query to find the total quantity of products in stock (SUM).
SELECT SUM(QuantityInStock) AS TotalQuantity FROM Inventory

--2. Write a query to find the product with the highest quantity in stock (MAX).
SELECT ProductName, QuantityInStock FROM Inventory WHERE QuantityInStock = (SELECT
MAX(QuantityInStock) FROM Inventory)

--3. Write a query to find the average reorder level across all products (AVG).
```

```
SELECT AVG(ReorderLevel) AS AverageReorderLevel FROM Inventory

--4. Write a query to find the product with the lowest reorder level (MIN)
SELECT ProductName, ReorderLevel FROM Inventory WHERE ReorderLevel = (SELECT
MIN(ReorderLevel) FROM Inventory)

--5. Write a query to find the total quantity in stock for products where the quantity is less than
the reorder level (SUM with WHERE clause).
SELECT SUM(QuantityInStock) AS TotalQuantity FROM Inventory WHERE QuantityInStock <
ReorderLevel


--Assignment 4 CUSTOMER ORDER DATABASE
CREATE TABLE CustomerOrders(OrderID INT IDENTITY(501,1)PRIMARY KEY, CustomerID
VARCHAR(50), OrderDate DATE, OrderAmount INT, QuantityOrdered INT)
INSERT INTO CustomerOrders(CustomerID, OrderDate, OrderAmount,
QuantityOrdered)VALUES
('C001','2024-09-10',1500,3),
('C002','2024-09-11',2000,5),
('C001','2024-09-12',1000,2),
('C003','2024-09-13',2500,4),
('C002','2024-09-14',3000,6)
SELECT * FROM CustomerOrders

---1. Write a query to calculate the total order amount for all orders (SUM).
SELECT SUM(OrderAmount) AS TotalOrderAmount FROM CustomerOrders

---2. Write a query to find the average order amount for all customers (AVG).
SELECT AVG(OrderAmount) AS AverageAmount FROM CustomerOrders

---3. Write a query to find the maximum order amount from a single order (MAX).
SELECT MAX(OrderAmount) AS MaximumOrderAmount FROM CustomerOrders

---4. Write a query to find the total quantity ordered across all orders (SUM).
SELECT SUM(QuantityOrdered) AS TotalQuantityOrdered FROM CustomerOrders

---5. Write a query to find the minimum order amount in the database (MIN).
SELECT MIN(OrderAmount) AS MinimumOrderAmount FROM CustomerOrders


---Assignment6 PRODUCT RATINGS DATABASE
CREATE TABLE ProductRatings(RatingID INT IDENTITY(1,1) PRIMARY KEY, ProductID
VARCHAR(50), CustomerID VARCHAR(50), Rating INT, ReviewDate DATE)
INSERT INTO ProductRatings(ProductID, CustomerID, Rating, ReviewDate) VALUES
```

('P001','C001',4,'2024-09-01'),
('P002','C002',5,'2024-09-02'),
('P001','C003',3,'2024-09-03'),
('P003','C001',2,'2024-09-04'),
('P002','C004',4,'2024-09-05')
SELECT * FROM ProductRatings

---1. Write a query to calculate the average rating for each product (AVG with GROUP BY).
SELECT ProductID, AVG(Rating) AS AvrerageRating FROM ProductRatings GROUP BY ProductID

---2. Write a query to find the highest rating given to any product (MAX).
SELECT MAX(Rating) AS HighestRating FROM ProductRatings

---3. Write a query to find the lowest rating given to any product (MIN).
SELECT MIN(Rating) AS LowestRating FROM ProductRatings

--4. Write a query to find the total number of ratings for each product (COUNT and GROUP BY).
SELECT ProductID, COUNT(RatingID) AS TotalRatings FROM ProductRatings GROUP BY ProductID

--5. Write a query to find the total number of products rated (COUNT with DISTINCT).
SELECT COUNT(DISTINCT ProductID) AS TotalNumberOfProducts FROM ProductRatings

<u>--SQL Server assignments and solutions using various SQL operators such as IN,--BETWEEN, LIKE, AND, OR, NOT, etc.</u>

--Assignment 1:
--Retrieve Employees Who Work in Specific Departments Using IN Operator
--Tables:Employees (EmployeeID, EmployeeName, DepartmentID)

CREATE TABLE Employees2(EmployeeID INT IDENTITY(1,1) PRIMARY KEY, EmployeeName VARCHAR(50), DepartmentID INT, DepartmentName VARCHAR(50))
INSERT INTO Employees2(EmployeeName, DepartmentID, DepartmentName) VALUES
('John Doe',001,'HR'),

```sql
('Jane Smith',002,'Finance'),
('Mike Johnson',003,'IT'),
('Alice Joseph',004,'Finance')
SELECT * FROM Employees2

--Retrieve the names of employees who work in either the 'HR', 'Finance', or 'IT' departments.
SELECT EmployeeName FROM Employees2 WHERE DepartmentName IN ('HR','Finance','IT')


--Assignment 2:
--Retrieve Products Within a Specific Price Range Using BETWEEN Operator
--Tables:Products (ProductID, ProductName, Price)
CREATE TABLE Products2(ProductID INT IDENTITY(1,1) PRIMARY KEY, ProductName
VARCHAR(50), Price INT)
INSERT INTO Products2(ProductName, Price) VALUES
('Keyboard',100),
('Mouse',200),
('Joystick',300),
('RAM',1500),
('SSD',600)
SELECT * FROM Products2



--Retrieve the product names and prices of products that are priced between 100 and 500.
SELECT ProductName,Price FROM Products2 WHERE Price BETWEEN 100 AND 500



--Assignment 3:
--Find Customers Whose Name Starts With 'A' Using LIKE Operator
--Tables:Customers (CustomerID, CustomerName)
CREATE TABLE Customers2(CustomerID INT IDENTITY(1,1) PRIMARY KEY, CustomerName
VARCHAR(50))
INSERT INTO Customers2(CustomerName) VALUES
('John Doe'),
('Jane Smith'),
('Mike Johnson'),
('Alice Joseph'),
('Arun Ravi')
SELECT * FROM Customers2
--Retrieve the customer names that start with the letter 'A'.
SELECT CustomerName FROM Customers2 WHERE CustomerName LIKE 'A%'
```

```
--Assignment 4:
--Retrieve Orders Placed on Specific Dates Using IN Operator
--Tables:Orders (OrderID, OrderDate)
CREATE TABLE Orders2(OrdersID INT IDENTITY(1,1) PRIMARY KEY, OrderDate DATE)
INSERT INTO Orders2(OrderDate)VALUES
('2023-01-01'),
('2024-05-01'),
('2023-02-01'),
('2023-01-11'),
('2023-03-01')
SELECT * FROM Orders2

--Retrieve the order IDs of orders placed on '2023-01-01', '2023-02-01', and '2023-03-01'.
SELECT OrdersID FROM Orders2 WHERE OrderDate IN ('2023-01-01', '2023-02-01',
'2023-03-01')

--Assignment 5:
--Retrieve Products That Are Not Priced Between 100 and 500 Using NOT BETWEEN Operator
--Tables:Products (ProductID, ProductName, Price)
CREATE TABLE Products(ProductID INT IDENTITY(1,1) PRIMARY KEY, ProductName
VARCHAR(50), Price INT)
INSERT INTO Products(ProductName, Price) VALUES
('Keyboard',50),
('Mouse',150),
('Joystick',40),
('RAM',520),
('SSD',20)
SELECT * FROM Products
--Retrieve the product names and prices of products that are not priced between 100 and 500.
SELECT ProductName, Price FROM Products WHERE Price NOT BETWEEN 100 AND 500

--Assignment 6:
---Find Orders Where the Total Amount is More Than 5000 or Less Than 1000 Using OR
Operator
CREATE TABLE Orders2(OrderID INT IDENTITY(1,1) PRIMARY KEY, TotalAmount INT)
INSERT INTO Orders2(TotalAmount)VALUES(12200),(3990),(4000),(500),(10000),(2000)
Task: Retrieve the order IDs where the total amount is either greater than 5000 or less than
1000.
SELECT * FROM Orders2 WHERE TotalAmount>5000 OR TotalAmount<1000

--Assignment 7:
--Retrieve Employees Who Do Not Work in the 'HR' or 'IT' Departments Using NOT IN Operator
```

```sql
--Tables:Employees (EmployeeID, EmployeeName, DepartmentID)

--Task: Retrieve the names of employees who do not work in the 'HR' or 'IT' departments.
SELECT * FROM Employees
SELECT EmployeeName FROM Employees WHERE DEPARTMENT NOT IN ('HR','IT')


--Assignment 8:
--Retrieve Orders Placed in 2023 Using BETWEEN and AND Operators
--Tables: Orders (OrderID, OrderDate)
--Task: Retrieve the order IDs of orders placed between '2023-01-01' and '2023-12-31'.
SELECT * FROM Orders2
SELECT * FROM Orders2 WHERE OrderDate BETWEEN '2023-01-01' AND '2023-12-31'


--Assignment 9:Find Customers Who Do Not Have 'John' in Their Name Using NOT LIKE
Operator
--Tables:Customers (CustomerID, CustomerName)
--Task: Retrieve the customer names that do not have 'John' in them.
SELECT * FROM Customers2
SELECT CustomerName FROM Customers2 WHERE CustomerName NOT LIKE 'John%'


--Assignment 10:Retrieve Products That Are Either in Category 'A' or Priced Below 100 Using IN
and OR Operators
--Tables:
----Products (ProductID, ProductName, Price, CategoryID)
--Categories (CategoryID, CategoryName)
--Task: Retrieve the product names and prices of products that are either in category 'A' or have
a price less than 100.
SELECT * FROM Products3
CREATE TABLE Products3(ProductID INT PRIMARY KEY,ProductName VARCHAR(50),Price
INT,CategoryID INT,FOREIGN KEY(CategoryID) REFERENCES Categories(CategoryID))
CREATE TABLE Categories(CategoryID INT PRIMARY KEY, CategoryName VARCHAR(50))
INSERT INTO
Categories(CategoryID,CategoryName)VALUES(101,'A'),(102,'B'),(103,'A'),(104,'A'),(105,'B')
INSERT INTO Products3(ProductID, ProductName, Price, CategoryID)VALUES
(1,'Keyboard',100,101),
(2,'Mouse',200,102),
(3,'Joystick',30,103),
(4,'Keyboard',500,104),
(5,'SSD',30,105)

SELECT Products3.ProductName,Products3.Price FROM Products3 INNER JOIN Categories
```

```
ON Products3.CategoryID = Categories.CategoryID
WHERE Categories.CategoryName = 'A' OR Products3.Price < 100
```

<u>--SQL Server assignments focused on using the ALTER TABLE command to perform various table modifications</u>

```
--Assignment 1: Add a New Column
--Task: Add a column DateOfBirth (data type DATE) to the Employees table to store employees'
--dates of birth.
ALTER TABLE Employees
ADD DateOfBirth DATE
SELECT * FROM Employees

--Assignment 2: Modify Column Data Type
CREATE TABLE Customers(CustomerID INT IDENTITY(201,1) PRIMARY KEY, CustomerName
VARCHAR(100), EMail VARCHAR(150), PhoneNumber VARCHAR(15))

--INSERTING VALUES
INSERT INTO Customers(CustomerName, EMail, PhoneNumber) VALUES
('Alice Johnson', 'alice@example.com', '555-1234'),
('Bob Smith', 'bob@example.com', '555-5678'),
('Charlie Brown','charlie@example.com','555-8765')

--Task: Change the data type of the PhoneNumber column in the Customers table to
--VARCHAR(15).
ALTER TABLE Customers
ALTER COLUMN PhoneNumber VARCHAR(15)

--Assignment 3: Add a Primary Key
--Task: Add a primary key to the DepartmentID column in the Departments table.
CREATE TABLE Departments(DepartmentID INT IDENTITY(201,1), Name VARCHAR(100))
ALTER TABLE Departments
ADD CONSTRAINT PK_Departments PRIMARY KEY(DepartmentID)

--Assignment 4: Drop a Column
CREATE TABLE Employees3(EmployeeID INT IDENTITY(1,1) PRIMARY KEY, FName
VARCHAR(50),MiddleName VARCHAR(50),LastName VARCHAR(50), DepartmentID INT,
DepartmentName VARCHAR(50))
INSERT INTO Employees3(FName,MiddleName,LastName, DepartmentID, DepartmentName)
VALUES
('John','M', 'Doe',001,'HR'),
('Jane','Kylie', 'Smith',002,'Finance'),
('Mike','Sur','Johnson',003,'IT'),
('Alice','T','Joseph',004,'Finance')
--Task: Remove the MiddleName column from the Employees table.
ALTER TABLE Employees3
DROP COLUMN MiddleName
SELECT * FROM Employees3

---Assignment 5: Add a Foreign Key
---Task: Add a foreign key to the EmployeeID column in the Orders table that references the
---EmployeeID column in the Employees table.
ALTER TABLE Orders2
ADD EmployeeID INT

ALTER TABLE Orders2
ADD CONSTRAINT fk_employee
FOREIGN KEY(EmployeeID)
REFERENCES Employees(EmployeeID)

SELECT * FROM Orders2
SELECT * FROM Employees


---Assignment 6: Drop a Foreign Key
---Task: Remove the foreign key constraint that references EmployeeID from the Orders table.
ALTER TABLE Orders2
DROP CONSTRAINT fk_employee



---Assignment 7: Rename a Column
---Task: Rename the column FullName to EmployeeFullName in the Employees table.
SELECT * FROM Employees
EXEC sp_rename 'Employees.EmployeeName','EmployeeFullname','COLUMN'
SELECT * FROM Employees


--Assignment 8: Add a Default Value
---Task: Add a default value of 'Active' to the Status column in the Employees table.
ALTER TABLE Employees
ADD Status VARCHAR(50)

ALTER TABLE Employees
ADD CONSTRAINT df_status DEFAULT 'Active' FOR Status
```

```
SELECT * FROM Employees


---Assignment 9: Drop a Primary Key
---Task: Drop the primary key constraint from the Departments table.
SELECT * FROM Departments
ALTER TABLE Departments
DROP CONSTRAINT PK_Departments

---Assignment 10: Add a Unique Constraint
---Task: Ensure the Email column in the Employees table is unique by adding a unique
constraint.

ALTER TABLE Employees2
ADD CONSTRAINT UQ_Email UNIQUE(Email)

SELECT * FROM Employees2

--Assignment 11: Add a Check Constraint
--Task: Add a check constraint to the Salary column in the Employees table to ensure that no
--salary is less than 1000.

ALTER TABLE Employees
ADD CONSTRAINT chk_salary CHECK(Salary >= 1000)

SELECT * FROM Employees

--Assignment 13: Drop a Check Constraint
--Task: Remove the check constraint from the Salary column in the Employees table.
ALTER TABLE Employees
DROP CONSTRAINT chk_salary
```

--SQL Server assignments involving subqueries.

```
--Assignment 1:
--Retrieve Employees Who Earn More Than the Average Salary
--Task: Retrieve the names and salaries of employees whose salary is greater than the average
--salary of all employees in the company.
--Tables:
--Employees (EmployeeID, EmployeeName, Salary)
SELECT * FROM Employees
SELECT EmployeeFullname, Salary FROM Employees WHERE SALARY > (SELECT
AVG(Salary) FROM Employees)

--x--Assignment 2:
--Find Departments with More Than 5 Employees
--Task: Retrieve the department names that have more than 5 employees.
--Tables:
--Employees (EmployeeID, EmployeeName, DepartmentID)

SELECT * FROM Employees
SELECT * FROM Departments2



CREATE TABLE Departments2(DepartmentID VARCHAR(50) PRIMARY KEY,
DepartmentName VARCHAR(50))
INSERT INTO Departments2(DepartmentID,DepartmentName)VALUES
('101','IT'),
('102','HR'),
('103','Finance'),
('104','IT'),
('105','Marketing'),
('106','IT'),
('107','IT'),
('108','IT')


ALTER TABLE Employees
ADD CONSTRAINT fk_employee2
FOREIGN KEY(DepartmentID) REFERENCES Departments2(DepartmentID)

SELECT DISTINCT Department FROM Employees WHERE Department = (
SELECT Department FROM Employees GROUP BY Department HAVING
COUNT(Department) >= 5
)
```

```
--Assignment 3:
--Retrieve Products with a Price Higher Than the Maximum Price of Category 'A'
--Task: Retrieve the product names and prices of products that have a price higher than the
--maximum price of products in category 'A'.
--Tables:
--Products (ProductID, ProductName, Price, CategoryID)
ALTER TABLE Products
ADD CONSTRAINT fk_Cid
FOREIGN KEY(CategoryID) REFERENCES Categories(CategoryID)

UPDATE Products
SET Price = 2005
WHERE ProductID = 5

SELECT * FROM Products
SELECT * FROM Categories

SELECT ProductName, Price FROM Products WHERE Price > (SELECT MAX(Price) FROM
Products JOIN Categories ON Products.CategoryID = Categories.CategoryID
WHERE Categories.CategoryName = 'A')

--Assignment 4:
--Retrieve Employees Who Work in Departments with Average Salary Higher Than 50,000
--Task: Retrieve the names of employees who work in departments where the average salary is
--higher than 50,000.
--Tables:
--Employees (EmployeeID, EmployeeName, Salary, DepartmentID)
--Departments (DepartmentID, DepartmentName)

SELECT * FROM Employees
SELECT * FROM Departments2


SELECT e.EmployeeFullname
FROM Employees e
WHERE e.DepartmentID IN (
   SELECT e2.DepartmentID
   FROM Employees e2
   GROUP BY e2.DepartmentID
   HAVING AVG(e2.Salary) > 50000
)
```

```
--Assignment 5:
--Find Employees Who Earn More Than Their Department's Average Salary
--Task: Retrieve the names of employees who earn more than the average salary of their
--department.
--Tables:
--Employees (EmployeeID, EmployeeName, Salary, DepartmentID)


SELECT e.EmployeeFullname
FROM Employees e
WHERE e.Salary >= (
   SELECT AVG(e2.Salary)
   FROM Employees e2
   WHERE e2.DepartmentID = e.DepartmentID
)


--Assignment 6:
--Find Customers Who Have Not Placed Any Orders
--Task: Retrieve the names of customers who have not placed any orders.
--Tables:
--Customers (CustomerID, CustomerName)
SELECT * FROM Customers
SELECT * FROM Orders2

ALTER TABLE Orders2
ADD CONSTRAINT fk_cons
FOREIGN KEY(CustomerID) REFERENCES Customers(CustomerID)

SELECT c.CustomerName
FROM Customers c
WHERE c.CustomerID NOT IN (
   SELECT o.CustomerID
   FROM Orders2 o
   WHERE o.CustomerID IS NOT NULL
);

--Assignment 7:
--Retrieve Top 3 Highest-Paid Employees in Each Department
--Task: Retrieve the top 3 highest-paid employees in each department.
--Tables:
--Employees (EmployeeID, EmployeeName, Salary, DepartmentID)
```

```
SELECT * FROM Employees


--Assignment 8:
--Find Products That Have Never Been Ordered
--Task: Retrieve the names of products that have never been ordered.
--Tables:
--Products (ProductID, ProductName)
--OrderDetails (OrderID, ProductID)
SELECT * FROM Products
SELECT * FROM OrderDetails
CREATE TABLE OrderDetails(OrderID INT, ProductID INT, Quantity INT, PRIMARY
KEY(OrderID),FOREIGN KEY(ProductID) REFERENCES Products(ProductID))
--Inserting values
INSERT INTO OrderDetails(OrderID, ProductID, Quantity) VALUES
(101,1,8),
(102,1,6),
(103,3,2),
(104,4,5),
(105,3,9),
(106,4,1)

SELECT P.ProductName FROM Products P WHERE P.ProductID NOT IN(SELECT
O.ProductID FROM OrderDetails O)
```

--SQL Server assignments that focus on different types of joins (INNER--JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN, CROSS JOIN)

```
--Assignment 1:
--Retrieve Employees and Their Department Names (INNER JOIN)
--Task: Retrieve the employee names and their corresponding department names.
--Tables:
--Employees (EmployeeID, EmployeeName, DepartmentID)
--Departments (DepartmentID, DepartmentName)

SELECT * FROM Departments2
SELECT * FROM Employees
SELECT E.EmployeeFullname,D.DepartmentName FROM Employees E INNER JOIN
Departments2 D ON
E.DepartmentID = D.DepartmentID

--Assignment 2:
--Retrieve All Employees and Their Department Names, Including Those Without
--Departments (LEFT JOIN)
--Task: Retrieve the employee names and their corresponding department names. Include
--employees who are not assigned to any department.
--Tables:
-- Employees (EmployeeID, EmployeeName, DepartmentID)


SELECT E.EmployeeFullname, D.DepartmentName FROM Departments2 D LEFT JOIN
Employees E ON E.DepartmentID = D.DepartmentID

--Assignment 3:
--Retrieve All Departments and the Employees Working in Them (RIGHT JOIN)
--Task: Retrieve all departments and their respective employees. Include departments even if
they
--don’t have any employees.
--Tables:
--Employees (EmployeeID, EmployeeName, DepartmentID)

 SELECT * FROM Employees
 SELECT * FROM Departments2

SELECT D.DepartmentName,E.EmployeeFullname FROM Departments2 D RIGHT JOIN
Employees E ON D.DepartmentID = E.DepartmentID

--Assignment 4:
--Retrieve All Employees and Departments, Including Those Without Matches (FULL
--OUTER JOIN)
--Task: Retrieve all employees and all departments, including employees without a department
--and departments without employees.
--Tables:
--Employees (EmployeeID, EmployeeName, DepartmentID)
--Departments (DepartmentID, DepartmentName)
SELECT E.EmployeeFullname, D.DepartmentName FROM Employees E FULL OUTER JOIN
Departments2 D
ON E.DepartmentID = D.DepartmentID

--Assignment 5:
--Retrieve Orders and the Customers Who Placed Them (INNER JOIN)
```

```sql
--Task: Retrieve order IDs and customer names for all orders.
--Tables:
--Orders (OrderID, CustomerID)

SELECT * FROM Customers
SELECT * FROM Orders2

SELECT O.OrdersID, C.CustomerName FROM Orders2 O INNER JOIN Customers C ON
C.CustomerID = O.CustomerID


--Assignment 6:
--Retrieve Orders and Customers, Including Customers Without Orders (LEFT JOIN)
--Task: Retrieve all customers and their respective orders. Include customers even if they
haven't
--placed any orders.
--Tables:
--Orders (OrderID, CustomerID)
--Customers (CustomerID, CustomerName)

SELECT C.CustomerName,O.OrdersID FROM Customers C LEFT JOIN Orders2 O ON
C.CustomerID = O.CustomerID ORDER BY C.CustomerName


--Assignment 7:
--Retrieve Products and Their Categories (INNER JOIN)
--Task: Retrieve product names and their respective category names.
--Tables:
--Products (ProductID, ProductName, CategoryID)

SELECT * FROM Products
SELECT * FROM Categories

SELECT P.ProductName, C.CategoryName FROM Products P INNER JOIN Categories C ON
P.CategoryID = C.CategoryID


--Assignment 8:
--Retrieve All Categories and Products, Including Categories Without Products (RIGHT
--JOIN)
--Task: Retrieve all categories and the products in each category. Include categories that don't
--have any products.
--Tables:
--Products (ProductID, ProductName, CategoryID)
```

```
--Categories (CategoryID, CategoryName)

SELECT * FROM Products
SELECT * FROM Categories
SELECT C.CategoryName,P.ProductName FROM Products P RIGHT JOIN Categories C ON
C.CategoryID = P.CategoryID


--Assignment 9:
--Retrieve Employees and Their Managers (Self-Join)
--Task: Retrieve the employee names and the names of their managers.
--Tables:
--Employees (EmployeeID, EmployeeName, ManagerID)

CREATE TABLE Employees2(
EmployeeID INT PRIMARY KEY,
EmployeeName VARCHAR(100),
ManagerID INT,
FOREIGN KEY (ManagerID) REFERENCES Employees2(EmployeeID)
)
INSERT INTO Employees2 (EmployeeID, EmployeeName, ManagerID)
VALUES
(1, 'John Doe', NULL),
(2, 'Jane Smith', 1),
(3, 'Boby Johnson', 1),
(4, 'Alice Brown', 2),
(5, 'Charlie White', 2),
(6, 'David M', 3),
(7, 'Tim Lee', 3),
(8, 'Mike Jonson',2)

SELECT e.EmployeeName AS Employee, m.EmployeeName AS Manager
FROM Employees2 e
INNER JOIN Employees2 m
ON e.ManagerID = m.EmployeeID
ORDER BY e.EmployeeID

--Assignment 10:
--Get All Possible Combinations of Products and Orders (CROSS JOIN)
--Task: Retrieve all possible combinations of products and orders (cartesian product).
--Tables:
--Products (ProductID, ProductName)
--Orders (OrderID)
```

```
SELECT * FROM Products
SELECT * FROM Orders2


SET IDENTITY_INSERT Products ON
INSERT INTO Products (ProductID, ProductName, CategoryID)
SELECT 7, ProductName, CategoryID
FROM Products
WHERE ProductID = 1003

DELETE FROM Products WHERE ProductID IN(1002,1003)

SELECT P.ProductID, P.ProductName, O.OrdersID
FROM Products P
CROSS JOIN Orders2 O;
```

--SQLSQL Server assignments using the GROUP BY and HAVING clauses

--Assignment 1:
--Find the Average Salary in Each Department
--Task: Retrieve the department name and the average salary of employees in each department.
--Only display departments where the average salary is greater than 50,000.

```
SELECT * FROM Departments2
SELECT * FROM Employees

SELECT D.DepartmentName,AVG(E.Salary) AS AvgSalary FROM Employees E JOIN
Departments2 D ON E.DepartmentID = D.DepartmentID
GROUP BY
D.DepartmentName HAVING AVG(E.Salary) > 50000
```

--Assignment 2:
--Count the Number of Employees in Each Department
--Task: Retrieve the department name and the total number of employees in each department.

```
--Only display departments that have more than 5 employees.
--Tables:
--Employees (EmployeeID, EmployeeName, DepartmentID)

SELECT D.DepartmentName, Count(E.EmployeeID) AS EmployeeCount FROM Employees E
JOIN Departments2 D
ON E.DepartmentID = D.DepartmentID
GROUP BY
D.DepartmentName HAVING COUNT(E.EmployeeID) > 5


--Assignment 3:
--Find the Maximum and Minimum Salary in Each Department
--Task: Retrieve the department name, maximum salary, and minimum salary for each
--department. Only include departments where the minimum salary is greater than 30,000.
--Tables:
--Employees (EmployeeID, EmployeeName, Salary, DepartmentID)

SELECT D.DepartmentName, MAX(E.Salary) AS MaximumSalary, MIN(E.Salary) AS
MinimumSalary FROM Employees E JOIN Departments2 D
ON E.DepartmentID = D.DepartmentID GROUP BY D.DepartmentName HAVING
MIN(E.Salary) > 30000


--Assignment 4:
--Find the Total Sales by Each Salesperson
--Task: Retrieve the employee name and total sales made by each salesperson. Only display
--salespersons who have made total sales of more than 100,000.
--Tables:
--Sales (SalesID, EmployeeID, Amount)
CREATE TABLE Employees4(EmployeeID INT PRIMARY KEY, EmployeeName
VARCHAR(100))
INSERT INTO Employees4(EmployeeID, EmployeeName) VALUES
(101,'John Doe'),
(102,'Jacob Sam'),
(103, 'Mike Johnson'),
(104, 'Alice Joseph'),
(105, 'Anu James')

ALTER TABLE Sales
ADD EmployeeID INT

ALTER TABLE Sales
ADD CONSTRAINT fk_employeeid
```

```sql
FOREIGN KEY(EmployeeID) REFERENCES Employees4(EmployeeID)

SELECT * FROM Employees4
SELECT * FROM Sales

SELECT E.EmployeeName, SUM(S.SaleAmount) AS TotalSales FROM Sales S JOIN
Employees4 E
ON S.EmployeeID = E.EmployeeID
GROUP BY E.EmployeeName
HAVING SUM(S.SaleAmount) > 100000



--Assignment 5:
--Find the Number of Orders by Each Customer
--Task: Retrieve the customer name and the total number of orders placed by each customer.
Only
--display customers who have placed more than 3 orders.
--Tables:
--Orders (OrderID, CustomerID)
--Customers (CustomerID, CustomerName)


SELECT * FROM Orders2
SELECT * FROM Customers

SELECT C.CustomerName, COUNT(O.OrdersID) AS TotalNumberOfOrders FROM Customers
C JOIN Orders2 O
ON C.CustomerID = O.CustomerID
GROUP BY C.CustomerName
HAVING COUNT(O.OrdersID) > 3
```