**SQL Server Lab Assignments-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

**use Bank**

**--Assignment 1: Sales Database**

**Scenario:**

**You are given a Sales table containing sales data for various products. The table structure is as**

**follows:**

**create table Sales(**

**SaleID int identity(1,1) primary key not null,**

**ProductID int not null,**

**ProductName varchar(50),**

**QuantitySold int ,**

**SaleAmount int,**

**SaleDate varchar(50)**

**);**

**insert into Sales**

**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**

**--Task:**

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

**select sum(SaleAmount) as SumOFSales from Sales**

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

**select max(QuantitySold) as MaxQuantity from Sales**

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

**select avg(SaleAmount) as AvgOfSales from Sales**

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

**select min(SaleAmount)as MinSaLES from Sales**

**--5. Write a query to calculate the total quantity sold of a specific product, say "Laptop"**

**--(SUM with WHERE clause).**

**select sum(QuantitySold) as QuantityOfSales from Sales**

**where ProductName='Laptop'**

**--Assignment 2: Employee Salaries**

**create table Employees(**

**EmployeeID int identity(1,1) primary key not null,**

**EmployeeName varchar(50),**

**Department varchar(50),**

**Salary decimal(10,2),**

**HireDate varchar(50)**

**);**

**insert into Employees**

**values('JohnDoe','IT', 5000 ,'2022-01-10'),**

**('JaneSmith', 'HR', 6000 ,'2021-03-15'),**

**('BobJohnson',' Finance', 5500,' 2020-06-20'),**

**('AliceBrown', 'IT', 6200 ,'2023-02-01'),**

**('CharlieWhite', 'Marketing', 4800 ,'2023-03-10')**

**select \*from Employees**

**--Task:**

**--1. Write a query to find the total salary expenditure for all employees (SUM).**

**select sum(Salary)as TotalSalary 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 MinSalaryInIT 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 TotalOccur from Employees**

**group by Department;**

**select \*from Employees**

**--Assignment 3: Inventory Management**

**create table Inventory(**

**ProductID int identity(201,1) not null primary key,**

**ProductName varchar(100),**

**QuantityInStock int ,**

**ReorderLevel int,**

**LastRestockDate date**

**);**

**insert into Inventory**

**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 AvgRecorder 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 TotalStockLessQuantity from Inventory**

**where QuantityInStock<ReorderLevel**

**--Assignment 4: Customer Orders Database**

**create table CustomerOrders(**

**OrderID int identity(501,1),**

**CustomerID varchar(50),**

**OrderDate date,**

**OrderAmount int,**

**QuantityOrdered int**

**);**

**insert into CustomerOrders**

**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 AllOrder from CustomerOrders**

**--2. Write a query to find the average order amount for all customers (AVG).**

**select avg(OrderAmount) as avgOrder from CustomerOrders**

**--3. Write a query to find the maximum order amount from a single order (MAX).**

**select max(OrderAmount) as maxOrder from CustomerOrders**

**--4. Write a query to find the total quantity ordered across all orders (SUM).**

**select sum(QuantityOrdered) as QuanityAllOrder from CustomerOrders**

**--5. Write a query to find the minimum order amount in the database (MIN).**

**select min(OrderAmount) as minOrder from CustomerOrders**

**--Assignment 6: Product Ratings Database**

**create table ProductRatings(**

**RatingID int not null identity(1,1),**

**ProductID varchar(50),**

**CustomerID varchar(50),**

**Rating int,**

**ReviewDate date**

**);**

**insert into ProductRatings**

**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 avgRating from ProductRatings**

**group by ProductID;**

**--2. Write a query to find the highest rating given to any product (MAX).**

**select ProductId ,max(Rating) as HighRate from ProductRatings**

**group by ProductID**

**--3. Write a query to find the lowest rating given to any product (MIN).**

**select ProductId ,min(Rating) as HighRate from ProductRatings**

**group by ProductID**

**--4. Write a query to find the total number of ratings for each product (COUNT and GROUP**

**--BY).**

**select ProductID,count(Rating) as TotNoRating 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 ToTNoProducts from ProductRatings**

**--SQL Server assignments and solutions using various SQL operators such as IN,**

**--BETWEEN, LIKE, AND, OR, NOT, etc**

**--Assignment 1:**

**--Retrieve Employees Who Work in Specific Departments Using IN Operator**

**--Task: Retrieve the names of employees who work in either the 'HR', 'Finance', or 'IT'**

**--departments**

**create table Employees1(**

**EmployeeID int identity(1,1) primary key not null,**

**EmployeeName varchar(100),**

**DepartmentID varchar(100),**

**);**

**drop table Employees1**

**insert into Employees1**

**values('John','IT'),**

**('Jane', 'HR'),**

**('Bob','Finance'),**

**('Alice', 'IT'),**

**('Charlie', 'UX')**

**select \*from Employees1**

**--Task: Retrieve the names of employees who work in either the 'HR', 'Finance', or 'IT'**

**--departments.**

**select \*from Employees1 where DepartmentID in('HR','Finance','IT')**

**--Assignment 2:**

**--Retrieve Products Within a Specific Price Range Using BETWEEN Operator**

**--Task: Retrieve the product names and prices of products that are priced between 100 and 500.**

**create table Products(**

**ProductID int identity(201,1) not null primary key,**

**ProductName varchar(100),**

**Price decimal(10,2)**

**);**

**insert into Products**

**values('Monitor',50),**

**('Printer',300),**

**('Mouse' ,1000),**

**('Keyboard' ,180),**

**('Laptop' ,25)**

**select \*from Products**

**select ProductName,Price from Products where Price between 100 and 500**

**--Assignment 3:**

**--Find Customers Whose Name Starts With 'A' Using LIKE Operator**

**--Task: Retrieve the customer names that start with the letter 'A'.**

**--Tables:**

**-- Customers (CustomerID, CustomerName)**

**create table Customers101(**

**CustomerID int identity(1,1) not null,**

**CustomerName varchar(100),**

**);**

**insert into Customers101**

**values ('Doe'),**

**('Aish'),**

**('Sneha'),**

**('Phool'),**

**('Adithya')**

**select \*from Customers101**

**select \*from Customers101 where CustomerName like 'A%'**

**--Assignment 4:**

**--Retrieve Orders Placed on Specific Dates Using IN Operator**

**--Task: Retrieve the order IDs of orders placed on '2023-01-01', '2023-02-01', and '2023-03-01'.**

**--Tables:**

**-- Orders (OrderID, OrderDate)**

**create table Orders(**

**OrderId int not null identity(1,1),**

**OrderDate date**

**);**

**insert into Orders**

**values ('2023-01-01'),**

**('2023-02-01'),**

**('2023-04-01'),**

**('2023-05-01'),**

**('2023-03-01')**

**select \*from Orders 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**

**--Task: Retrieve the product names and prices of products that are not priced between 100 and**

**--500.**

**--Tables:**

**-- Products (ProductID, ProductName, Price)**

**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**

**--Task: Retrieve the order IDs where the total amount is either greater than 5000 or less than 1000.**

**insert into CustomerOrders**

**values ('C006' ,'2024-09-10', 6000, 3),**

**('C005' ,'2024-09-10', 400, 3)**

**select \*from CustomerOrders**

**select OrderID,OrderAmount from CustomerOrders where OrderAmount>5000 or OrderAmount<1000**

**--Assignment 7:**

**--Retrieve Employees Who Do Not Work in the 'HR' or 'IT' Departments Using NOT IN**

**--Operator**

**--Task: Retrieve the names of employees who do not work in the 'HR' or 'IT' departments.**

**select \*from Employees1 where DepartmentID not in('HR','IT')**

**--Assignment 8:**

**--Retrieve Orders Placed in 2023 Using BETWEEN and AND Operators**

**--Task: Retrieve the order IDs of orders placed between '2023-01-01' and '2023-12-31'**

**insert into CustomerOrders**

**values ('C007' ,'2023-09-10', 400, 3),**

**('C008' ,'2023-09-10', 100, 3)**

**select \*from CustomerOrders**

**select OrderID,OrderDate from CustomerOrders 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**

**--Task: Retrieve the customer names that do not have 'John' in them.**

**--Tables:**

**-- Customers (CustomerID, CustomerName)**

**insert into Customers101**

**values ('John')**

**select \*from Customers101**

**select CustomerID,CustomerName from Customers101 where CustomerName not like 'John%'**

**--Assignment 10:**

**--Retrieve Products That Are Either in Category 'A' or Priced Below 100 Using IN and OR**

**--Operators**

**--Task: Retrieve the product names and prices of products that are either in category 'A' or have a**

**--price less than 100.**

**--Tables:**

**-- Products (ProductID, ProductName, Price, CategoryID)**

**-- Categories (CategoryID, CategoryName)**

**--These assignments cover various SQL operators to filter data, using IN, BETWEEN, LIKE,**

**--NOT, and combining logical operators like AND/OR**

**create table Products101(**

**ProductID int identity(1,1) not null,**

**ProductName varchar(50),**

**Price decimal(10,2),**

**CategoryID int**

**foreign key(CategoryID) references Categories(CategoryID)**

**);**

**insert into Products101**

**values('Laptop',50,'1'),**

**('Mouse', 500,'2'),**

**('Printer',390,'1')**

**select \*from Products101**

**create table Categories(**

**CategoryID int identity(1,1) not null primary key,**

**CategoryName varchar(50)**

**);**

**insert into Categories**

**values('A'),**

**('B')**

**select \*from Categories**

**select P.ProductName, P.Price**

**from Products101 P**

**join Categories C**

**on P.CategoryID = C.CategoryID**

**where C.CategoryName = 'A' or P.Price < 100;**

**SQL Server assignments focused on using the ALTER TABLE**

**command to perform various table modifications**

**--SQL Server assignments focused on using the ALTER TABLE**

**--command to perform various table modifications**

**--Assignment 1: Add a New Column**

**--Task: Add a column DateOfBirth (data type DATE) to the Employees table to store employees'**

**--dates of birth.**

**create table Employees(**

**EmployeeID int identity(1,1) not null,**

**FirstName varchar(50)**

**);**

**insert into Employees**

**values('Max'),**

**('Joe'),**

**('Imen')**

**select \*from Employees**

**alter table Employees**

**add DateOfBirth date;**

**---**

**select \*from Employees**

**--Assignment 2: Modify Column Data Type**

**--Task: Change the data type of the PhoneNumber column in the Customers table to**

**--VARCHAR(15)**

**create table Customers(**

**CustomerID int identity(1,1) not null,**

**CustomerName varchar(50),**

**PhoneNumber bigint**

**);**

**insert into Customers**

**values('Max', 3456782900),**

**('Mini',1098765748),**

**('Sony', 4563797649)**

**select \*from Customers**

**alter table Customers**

**alter column PhoneNumber varchar(50);**

**--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 not null,**

**Department varchar(50)**

**);**

**insert into Departments**

**values ('IT'),**

**('HR'),**

**('UX/UI')**

**Select \*from Departments**

**alter table Departments**

**add constraint PK\_Departments Primary Key(DepartmentID);**

**Select \*from Departments**

**--Assignment 4: Drop a Column**

**--Task: Remove the MiddleName column from the Employees table.**

**--First lets add middle name and last name to employees**

**alter table Employees**

**add MiddleName varchar(50);**

**alter table Employees**

**add LastName varchar(50);**

**--Now lets drop Middlename**

**alter table Employees**

**drop column MiddleName**

**select \*from Employees**

**--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.**

**-- First lets make EmployeeId in Employee able a primary key**

**alter table Employees**

**add constraint Pk\_Employees primary key(EmployeeID);**

**create table Orders(**

**OrderID int identity not null,**

**EmployeeID int,**

**);**

**insert into Orders**

**values(1),**

**(2)**

**alter table Orders**

**add constraint FK\_Orders Foreign key(EmployeeID) references Employees(EmployeeID);**

**select \*from Orders**

**--Assignment 6: Drop a Foreign Key**

**--Task: Remove the foreign key constraint that references EmployeeID from the Orders table.**

**alter table Orders**

**drop constraint FK\_Orders;**

**--Assignment 7: Rename a Column**

**--Task: Rename the column FullName to EmployeeFullName in the Employees table.**

**--First we will have dd col full name**

**alter table Employees**

**add FullName varchar(50);**

**select \*from Employees**

**Exec sp\_rename 'Employees.FullName','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(20) not null DEFAULT 'Active';**

**alter table Employees**

**drop constraint**

**--Assignment 9: Drop a Primary Key**

**--Task: Drop the primary key constraint from the Departments table.**

**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.**

**create table Employee101(**

**EmployeeID int not null identity,**

**EmailID varchar(100)**

**);**

**insert into Employee101**

**values ('asididj@gmail'),('whdbw@yahoo')**

**alter table Employee101**

**add constraint UK\_Email unique(EmailID);**

**--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**

**-- Create the Employee102 table**

**create table Employee102(**

**EmployeeID int not null identity,**

**EmailID varchar(100),**

**Salary bigint**

**);**

**-- Insert sample data into the Employee102 table**

**insert into Employee102**

**values**

**('asididj@gmail', 10000),**

**('whdbw@yahoo', 200);**

**alter table Employee102**

**add constraint CK\_Salary check(Salary>=1000)**

**---drop check**

**Assignment 13: Drop a Check Constraint**

**Task: Remove the check constraint from the Salary column in the Employees table**

**alter table Employee102**

**drop constraint CK\_Salary**

**select \*from Employee102**

These assignments are designed to give you hands-on practice with the ALTER TABLE

statement for modifying tables, adding/removing constraints, changing column properties, and

More.

**SQL Server assignments involving subqueries.**

**--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.**

**create table Employees(**

**EmployeeID int not null identity,**

**FirstName varchar(50),**

**Salary bigint**

**);**

**insert into Employees**

**values('Max',30000),**

**('Ann',20000),**

**('Aish',15000),**

**('Alec',29000)**

**select \*from Employees**

**Select FirstName,Salary from Employees where Salary>(Select avg(Salary) as SalaryAvg from Employees)**

**drop table Employees**

**--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)**

**drop table Employees**

**create table Employees(**

**EmployeeID int not null identity,**

**EmployeeName varchar(50),**

**DepartmentID int**

**foreign key(DepartmentID) references Department(DepartmentID)**

**);**

**create table Department(**

**DepartmentID int identity not null primary key,**

**DepartmentName varchar(50)**

**);**

**insert into Department**

**values('HR'),**

**('Finance'),**

**('UX/UI'),**

**('C#')**

**insert into Employees**

**values ('Max',1),('Noel',2),('Aisha',1),('Divya',1),('Sona',1),('Pari',1),('Deigo',3),('Nafsa',4),('Hadi',3)**

**select DepartmentName from Department where DepartmentID in (**

**select DepartmentID**

**from Employees**

**group by DepartmentID**

**having count(\*)>=5**

**);**

**select \* from Employees**

**select \*from Department**

**drop table Employees**

**drop table Department**

**--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)**

**create table Products(**

**ProductID int not null identity,**

**ProductName varchar(50),**

**Price bigint,**

**CategoryID varchar(10)**

**);**

**insert into Products**

**values('Mouse',400,'A'),**

**('Printer',500,'A'),**

**('LAPTOP',1000,'B')**

**--Task: Retrieve the product names and prices of products that have a price higher than the**

**--maximum price of products in category 'A'**

**select ProductName,Price from Products where Price> (select max(Price) as Aprice from Products where CategoryID = '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)**

**create table Employees(**

**EmployeeID int not null identity,**

**EmployeeName varchar(50),**

**Salary bigint,**

**DepartmentID int**

**foreign key(DepartmentID) references Department(DepartmentID)**

**);**

**create table Department(**

**DepartmentID int identity not null primary key,**

**DepartmentName varchar(50)**

**);**

**insert into Department**

**values('HR'),**

**('Finance'),**

**('UX/UI'),**

**('C#')**

**insert into Employees**

**values ('Max',11000,1),**

**('Noel',5900,2),**

**('Aisha',676999,1),**

**('Divya',23000,1),**

**('Sona',89765,1),**

**('Pari',25000,1),**

**('Deigo',870,3),**

**('Nafsa',35000,4),**

**('Hadi',657,3)**

**select EmployeeName from Employees where DepartmentID in**

**(select DepartmentID**

**from Employees**

**group by DepartmentID**

**having avg(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 EmployeeName from Employees e where Salary>**

**(SELECT AVG(Salary)**

**FROM Employees**

**WHERE 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)**

**create table Customers(**

**CustomerID int not null identity primary key,**

**CustomerName varchar(50)**

**)**

**drop table Customers**

**insert into Customers**

**values('Max'),('Aisha'),('Malavika'),('Nife')**

**create table Orders(**

**OrderID int identity not null ,**

**CustomerID int,**

**foreign key(CustomerID) references Customers(CustomerID)**

**)**

**insert into Orders**

**values(1),(3),(4)**

**select \*from Orders**

**select \*from Customers**

**drop table Orders**

**select CustomerName from Customers where CustomerID not in(select CustomerID from Orders)**

**—-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)**

**create table Products(**

**ProductID int not null identity primary key,**

**ProductName varchar(50)**

**);**

**insert into Products**

**values ('Mouse'),**

**('MOvie'),**

**('Phone'),**

**('Laptop')**

**create table OrderDetails(**

**OrderID int not null identity,**

**ProductID int**

**foreign key(ProductID) references Products(ProductID)**

**);**

**insert into OrderDetails**

**values(2),**

**(3)**

**select ProductName from Products where ProductID not in(select ProductID from OrderDetails)**

**—-These SQL subquery-based examples help reinforce key concepts like —-correlated and non-**

**—-correlated subqueries, aggregation, and filtering**

**SQL Server assignments that focus on different types of joins (INNER**

**JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN, CROSS JOIN)**

Explanation of Joins Used:

INNER JOIN: Retrieves records that have matching values in both tables.

LEFT JOIN (LEFT OUTER JOIN): Retrieves all records from the left table and

matched records from the right table, and NULL if there is no match.

RIGHT JOIN (RIGHT OUTER JOIN): Retrieves all records from the right table and

matched records from the left table, and NULL if there is no match.

FULL OUTER JOIN: Retrieves all records when there is a match in either left or right

table, and NULL if there is no match.

CROSS JOIN: Retrieves all possible combinations of records from two tables (cartesian

product).

SELF JOIN: A table joins with itself, typically used to find hierarchical relationships.

--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)

create table Employees(

EmployeeID int identity not null,

EmployeeName varchar(50),

DepartmentID int

foreign key(DepartmentID) references Departments(DepartmentID)

);

insert into Employees

values('Aisha',2),

('Imen',4),

('Sunny',1),

('Siddi',1)

insert into Employees

values ('Noel',null)

create table Departments(

DepartmentID int identity not null primary key,

DepartmentName varchar(50)

);

select \*from Employees

select \*from Departments

drop table Employees

drop table Departments

insert into Departments

values('HR'),

('UX'),

('AI'),

('Data')

select Employees.EmployeeName ,Departments.DepartmentName from Employees

inner join Departments on Employees.DepartmentID=Departments.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 \*from Employees

select \*from Departments

select Employees.EmployeeName ,Departments.DepartmentName from Employees

left outer join Departments on Employees.DepartmentID=Departments.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 Departments

select Employees.EmployeeName ,Departments.DepartmentName from Employees

right outer join Departments on Employees.DepartmentID=Departments.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 \*from Employees

select \*from Departments

select Employees.EmployeeName ,Departments.DepartmentName from Employees

full outer join Departments on Employees.DepartmentID=Departments.DepartmentID

--Assignment 5:

--Retrieve Orders and the Customers Who Placed Them (INNER JOIN)

--Task: Retrieve order IDs and customer names for all orders.

--Tables:

-- Orders (OrderID, CustomerID)

create table Customers(

CustomerID int not null identity primary key,

CustomerName varchar(50)

)

drop table Customers

insert into Customers

values('Max'),('Aisha'),('Malavika'),('Nife')

create table Orders(

OrderID int identity not null ,

CustomerID int,

foreign key(CustomerID) references Customers(CustomerID)

)

insert into Orders

values(1),(3),(4)

select \*from Orders

select \*from Customers

drop table Orders

select Customers.CustomerName,Orders.OrderID from Customers

inner join Orders on Customers.CustomerID = Orders.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 \*from Orders

select \*from Customers

select Customers.CustomerName,Orders.OrderID from Customers

left outer join Orders on Customers.CustomerID = Orders.CustomerID

--Assignment 7:

--Retrieve Products and Their Categories (INNER JOIN)

--Task: Retrieve product names and their respective category names.

--Tables:

-- Products (ProductID, ProductName, CategoryID)

create table Products11(

ProductID int not null identity,

ProductName varchar(50),

Price bigint,

CategoryID varchar(10)

foreign key(CategoryID) references Categories(CategoryID)

);

create table Categories(

CategoryID varchar(10) primary key,

CategoryName varchar(50)

);

insert into Categories

values ('A','HMI'),

('B','Desktop')

insert into Categories

values ('C','Software')

insert into Products11

values('Mouse',400,'A'),

('Printer',500,'A'),

('LAPTOP',1000,'B')

select \*from Products11

select \*from Categories

select Products11.ProductName,Categories.CategoryName from Products11

inner join Categories on ProductS11.CategoryID = Categories.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 Orders

select \*from Customers

select Products11.ProductName,Categories.CategoryName from Products11

right outer join Categories on ProductS11.CategoryID = Categories.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 Employees101(

EmployeeID int identity primary key,

EmployeeName varchar(50),

ManagerID int null

);

insert into Employees101

values('Ravi',null),

('Raj',4),

('Sachin',2),

('SAJIN',1)

select e1.EmployeeName as Namee , e2.EmployeeName as Manager from Employees101 e1

inner join Employees101 e2 on e1.ManagerID = e2.EmployeeID

drop table Employees101

select \*from Employees101

--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)

create table Products(

ProductID int not null identity primary key,

ProductName varchar(50)

);

insert into Products

values ('Mouse'),

('MOvie'),

('Phone'),

('Laptop')

create table OrderDetails(

OrderID int not null

);

insert into OrderDetails

values(1),

(2),

(3)

drop table OrderDetails

select \*from OrderDetails

select \*from Products

select Products.ProductName, OrderDetails.OrderID from Products

cross join OrderDetails

-----

--SQL 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.

create table Employees102(

EmployeeID int not null identity,

EmployeeName varchar(50),

Salary bigint,

DepartmentID int

foreign key(DepartmentID) references Department102(DepartmentID)

);

create table Department102(

DepartmentID int identity not null primary key,

DepartmentName varchar(50)

);

insert into Department102

values('HR'),

('Finance'),

('UX/UI'),

('C#')

insert into Employees102

values ('Max',11000,1),

('Noel',5900,2),

('Aisha',676999,1),

('Divya',23000,1),

('Sona',89765,1),

('Pari',25000,1),

('Deigo',870,3),

('Nafsa',35000,4),

('Hadi',657,3)

select \*from Employees102

select \*from Department102

select D.DepartmentName,avg(E.Salary) as SalAverage from Employees102 E inner join Department102 D

on D.DepartmentID = E.DepartmentID group by D.DepartmentName having avg(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.DepartmentID) as Totall from Employees102 E inner join Department102 D

on D.DepartmentID = E.DepartmentID group by D.DepartmentName having count(E.DepartmentID)>=5;

select count(DepartmentID) as Tot from Employees102 group by DepartmentID

--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, min(E.Salary) as SalMin,max(E.Salary) as SalMax from Employees102 E inner join Department102 D

on D.DepartmentID= E.DepartmentID group by D.DepartmentName having min(E.Salary)>30000;

select \*from Employees102

select \*from Department102

--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 Sales(

SalesID int identity not null,

EmployeeID varchar(10),

Amount bigint,

foreign key(EmployeeID) references Employ(EmployeeID)

);

insert into Sales

values('A',30000),

('B',20977),

('B',50000),

('C',90000),

('B',40000)

select \*from Sales

drop table Sales

create table Employ(

EmployeeID varchar(10) not null primary Key ,

EmployeeName varchar(100)

);

insert into Employ

values('A','Max'),

('B','Noel'),

('C','Jessy')

select E.EmployeeName,count(S.EmployeeID) as TotSales from Sales S inner join Employ E

on E.EmployeeID = S.EmployeeID group by E.EmployeeName having sum(S.Amount)>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)

create table Orders103(

OrderID int identity not null,

CustomerID int

foreign key(CustomerID) references Customers103(CustomerID)

);

insert into Orders103

values(2),(2),(2),(3),(4)

create table Customers103(

CustomerID int identity not null primary key,

CustomerName varchar(10)

);

insert into Customers103

values('Max'),

('Joe'),

('Sheela'),

('Maryam'),

('Aisha')

select \*from Orders103

select \*from Customers103

select C.CustomerName,count(O.CustomerID) as TotORDER from Orders103 O inner join Customers103 C

on C.CustomerID= O.CustomerID group by C.CustomerName having count(O.CustomerID)>=3