**Easy**

1. **Computed column (easy)**

The table given below consists of two columns first name and lastname. Add a third computed column to the table which would display the first name and lastname together as fullname.

|  |  |
| --- | --- |
| ***First name*** | ***Lastname*** |
| Jason | Stuart |
| Mike | Adams |
| Rachael | Adams |

CREATE TABLE tblTestComputed (

Firstname VARCHAR (50),

Lastname VARCHAR(50)

)

INSERT INTO tblTestComputed (Firstname, Lastname)

SELECT 'Jason', 'Stuart'

UNION

SELECT 'mike', 'Adams'

Union

Select ‘Rachael’,’Adams’

SELECT \* FROM tblTestComputed

ALTER TABLE tblTestComputed

ADD Fullname AS (ISNULL (Lastname,'') + ' ' + ISNULL (Firstname,''))

SELECT \* FROM tblTestComputed

1. **Persisted column(easy)**

|  |  |
| --- | --- |
| ***EmployeeID*** | ***Joining\_Date*** |
| 201 | 2005-06-04 |
| 202 | 2007-05-04 |
| 203 | 2007-04-05 |
| 204 | 2010-09-08 |

Add a persisted computed column to the table given above to find the number of years employee worked in the company. Add one more date column

Create table emp1 (EmployeeID int,Dateofjoin datetime)

Insert into emp1 (eno,dateofjoin)

select 1,'2005-05-06'

ALTER TABLE emp1

ADD Yearsworked AS datediff (year, dateofjoin, getdate ())

select \* from emp1

3. **Using top clause (easy)**

Write a query to find the 6th highest employee name and salary from Employee table

Rewrite query

SELECT TOP 1 salary

FROM (

SELECT DISTINCT TOP 6 salary

FROM employee

ORDER BY salary DESC) a

ORDER BY salary

4. **Using top clause (easy)**

Write a query to find the nth highest salary from Employee table

SELECT TOP 1 salary

FROM (

SELECT DISTINCT TOP n salary

FROM employee

ORDER BY salary DESC) a

ORDER BY salary

Where n > 1 (n is always greater than one)

Ignore

5.

Write a Query to retrieve the top two employees who earn the most from employee table in SQL Server

Using a CTE

With NumberedItems As

(

Select EmpId, Salary, Name

, Row\_Number () Over (Order By Salary Desc) As SalaryRank

From Employee

)

Select EmpId, Salary, Name

From NumberedItems

Where SalaryRank <=2

Use rank function

6. **Subquery (easy)**

Write a query to find employee details with the 3rd highest salary (using correlated subQuery)

SELECT \*

FROM EmployeeDetails e1

WHERE (2) = (

SELECT COUNT(DISTINCT(e2.Salary))

FROM EmployeeDetails e2

WHERE e2.Salary > e1.Salary)

7. **Subquery (easy)**

Write a query to find list employees who draw maximum salary in their respective departments

Select \* from e where salary in (select max (salary) from e group by deptno

8. **Join (easy)**

Create the following two tables

CREATE TABLE addresses (

adr\_id INTEGER NOT NULL PRIMARY KEY,

adr\_city VARCHAR(15),

adr\_country VARCHAR(15) NOT NULL);

CREATE TABLE persons (

prs\_id INTEGER NOT NULL PRIMARY KEY,

prs\_father\_id INTEGER,

prs\_mother\_id INTEGER,

prs\_adr\_id INTEGER,

prs\_first\_name VARCHAR(15),

prs\_surname VARCHAR(15),

CONSTRAINT prs\_prs\_father\_fk FOREIGN KEY (prs\_father\_id)

REFERENCES persons(prs\_id),

CONSTRAINT prs\_prs\_mother\_fk FOREIGN KEY (prs\_mother\_id)

REFERENCES persons(prs\_id),

CONSTRAINT prs\_adr\_fk FOREIGN KEY (prs\_adr\_id)

REFERENCES addresses(adr\_id));

INSERT INTO addresses VALUES (1, 'RIGA', 'LATVIA');

INSERT INTO addresses VALUES (2, 'BERLIN', 'GERMANY');

INSERT INTO addresses VALUES (3, 'NEW YORK', 'USA');

INSERT INTO persons VALUES (1, NULL, NULL, NULL, 'JANIS', 'BERZINS');

INSERT INTO persons VALUES (2, 1, NULL, 2, 'PETER', 'BERZINS');

INSERT INTO persons VALUES (3, NULL, NULL, 2, 'ANN', 'SMYTH');

INSERT INTO persons VALUES (4, 2, 3, 2, 'CHARLES', 'BERZINS');

Write a query to display first name, surname and city for every person.

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

INNER JOIN addresses ON prs\_adr\_id = adr\_id;

Sample output

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- --------------------------------

PETER BERZINS BERLIN

9. **aggregate function(easy)**

Using Sales.SalesTaxRate table in adventureworks, Write a query to display total tax rate for each province (using Aggregate function)

select sum(TaxRate) total\_tax,StateProvinceID from sales.SalesTaxRate group by stateprovinceid;

24. Create the following table, and write a query to **calculate the average AVG () for the Sales column. (easy)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Company\_ID** | **CompanyName** | **Address** | **Town** | **Sales** |
| 1 | SQL Sample | 1 Sample St | Hamburg | 10000 |
| 2 | SQL Code Land | 2 Code Rd | Hamburg | 20000 |
| 3 | Sample Code World | 66 SQL St | Curry | 18000 |
| 4 | SQL Reference Ltd | 34 Reference St | Pisa | 17000 |

SELECT AVG(Sales) AS Average\_Sales FROM tblCompany;

**10. (Easy) aggregate function**

Using the Sales.SalesTaxRate table in adventureworks, write a query to calculate lowest (minimum) tax rate.

SELECT Min(TaxRate)

FROM Sales.SalesTaxRate;

11.**(easy) aggregate function**

Using the production.Product table in adventureworks calculate the average weight.

Substitute the null value columns with a value (isnull function)

SELECT AVG(ISNULL(Weight, 50))

FROM Production.Product;

GO

12**.(easy)system function**

Write a query to find the current system date and time.

SELECT SYSDATETIME()

,SYSDATETIMEOFFSET()

,SYSUTCDATETIME()

,CURRENT\_TIMESTAMP

,GETDATE()

,GETUTCDATE();

13**. (Easy) system function**

Write a query to return the name of the current user

SELECT CURRENT\_USER;

14**. (Easy) system function**

Write a query to create the database role buyers that is owned by user BenMiller

CREATE LOGIN urname

WITH PASSWORD = '1a2b3c4d5e6f7D$';

USE AdventureWorks

GO

CREATE USER BenMiller FOR LOGIN urname;

GO

CREATE ROLE buyers AUTHORIZATION BenMiller;

GO

15**.(easy)system function**

Write a query to create the database role auditors that is owned the db\_securityadmin fixed database role.

CREATE ROLE auditors AUTHORIZATION db\_securityadmin;

GO

16**.constraints (easy)**

Write a query to create the following tables with columns given:

Employee table with columns eid (primary key) ,deptid

Department table with columns eid (foreign key referencing eid in employee table) , deptid

1. After creating both the tables try inserting values in the department table first. What do you get as output.
2. After inserting values in both the table , try deleting a row from employee table referred by department table, what do you see?
3. Now try deleting the employee table, what is the result?

17. **triggers(medium)**

Create employee\_test table and employee\_test\_audit table.

CREATE TABLE Employee\_Test

(

Emp\_ID INT Identity,

Emp\_name Varchar(100),

Emp\_Sal Decimal (10,2)

)

INSERT INTO Employee\_Test VALUES ('Anees',1000);

INSERT INTO Employee\_Test VALUES ('Rick',1200);

INSERT INTO Employee\_Test VALUES ('John',1100);

INSERT INTO Employee\_Test VALUES ('Stephen',1300);

INSERT INTO Employee\_Test VALUES ('Maria',1400);

CREATE TABLE Employee\_Test\_Audit

(

Emp\_ID int,

Emp\_name varchar(100),

Emp\_Sal decimal (10,2),

Audit\_Action varchar(100),

Audit\_Timestamp datetime

)

Create an AFTER INSERT TRIGGER on the employee table which will insert the rows inserted into the employee table into employee audit table. The main purpose of this audit table is to record the changes in the main table.

CREATE TRIGGER trgAfterInsert ON [dbo].[Employee\_Test]

FOR INSERT

AS

declare @empid int;

declare @empname varchar(100);

declare @empsal decimal(10,2);

declare @audit\_action varchar(100);

select @empid=i.Emp\_ID from inserted i;

select @empname=i.Emp\_Name from inserted i;

select @empsal=i.Emp\_Sal from inserted i;

set @audit\_action='Inserted Record -- After Insert Trigger.';

insert into Employee\_Test\_Audit

(Emp\_ID,Emp\_Name,Emp\_Sal,Audit\_Action,Audit\_Timestamp)

values(@empid,@empname,@empsal,@audit\_action,getdate());

PRINT 'AFTER INSERT trigger fired.'

GO

**Medium**

18. **CTE (medium)**

Write a [Query to retrieve the top two employees who earn the most from employee table in SQL Server](http://stackoverflow.com/questions/2445034/query-for-retrieving-top-two-rows-from-a-table-in-sql-server)

Using a CTE

With NumberedItems As

(

Select EmpId, Salary, Name

, Row\_Number () Over (Order By Salary Desc) As SalaryRank

From Employee

)

Select EmpId, Salary, Name

From NumberedItems

Where SalaryRank <=2(repeated)

19. **Subquery (medium)**

Write a query to find salesperson fullname, emailaddress, phone who received 2,000.0 in bonus. Use humanresources.employee, person.contact, sales.salesperson tables from adventureworks database.

Using JOIN

SELECT SalesPerson = c.LastName + ', ' + c.FirstName,

EmailAddress,

Phone

FROM HumanResources.Employee e

INNER JOIN Person.Contact c

ON e.ContactID = c.ContactID

WHERE 2000.00 = (SELECT Bonus

FROM Sales.SalesPerson sp

WHERE e.EmployeeID = sp.SalesPersonID);

20. **Subquery (medium)**

|  |  |  |
| --- | --- | --- |
| ***Employee\_no*** | ***Salary*** | ***Department*** |
| 1 | 5678 | 10 |
| 2 | 9807 | 20 |
| 3 | 6500 | 10 |
| 4 | 6565 | 10 |
| 5 | 9000 | 20 |
| 6 | 1000 | 10 |
| 7 | 8000 | 20 |
| 8 | 5000 | 20 |
| 9 | 3890 | 10 |
| 10 | 8000 | 20 |

|  |  |
| --- | --- |
| ***Department\_id*** | ***Department\_name*** |
| 10 | Marketing |
| 20 | finance |

Using the above 2 tables write a query to find the list of employees (employee number and names) having more salary than the average salary of all employees in that employee's department.

SELECT employee\_number, name  
FROM employee AS e1  
WHERE salary > (SELECT avg(salary)  
    FROM employee  
    WHERE department = e1.department);

21. **Common Table Expression (Medium)**

**Create the following table**

CREATE TABLE dbo.Employees

(

EmployeeID int NOT NULL PRIMARY KEY,

FirstName varchar (50) NOT NULL,

LastName varchar (50) NOT NULL,

ManagerID int NULL

)

GO

INSERT INTO Employees VALUES (101, 'Ken', 'Sánchez', NULL)

INSERT INTO Employees VALUES (102, 'Terri', 'Duffy', 101)

INSERT INTO Employees VALUES (103, 'Roberto', 'Tamburello', 101)

INSERT INTO Employees VALUES (104, 'Rob', 'Walters', 102)

INSERT INTO Employees VALUES (105, 'Gail', 'Erickson', 102)

INSERT INTO Employees VALUES (106, 'Jossef', 'Goldberg', 103)

INSERT INTO Employees VALUES (107, 'Dylan', 'Miller', 103)

INSERT INTO Employees VALUES (108, 'Diane', 'Margheim', 105)

INSERT INTO Employees VALUES (109, 'Gigi', 'Matthew', 105)

INSERT INTO Employees VALUES (110, 'Michael', 'Raheem', 106)

Write a Common Table Expression which will display the below, employee name with respective manager name , and also the employee level

Sample Output:

|  |  |  |
| --- | --- | --- |
| ***fullname*** | ***Employee level*** | ***manager*** |
| Ken Sanchez | 1 | Null |
| Terri duffy | 2 | Ken Sanchez |

WITH

cteReports (EmpID, FirstName, LastName, MgrID, EmpLevel)

AS

(

SELECT EmployeeID, FirstName, LastName, ManagerID, 1

FROM Employees

WHERE ManagerID IS NULL

UNION ALL

SELECT e.EmployeeID, e.FirstName, e.LastName, e.ManagerID,

r.EmpLevel + 1

FROM Employees e

INNER JOIN cteReports r

ON e.ManagerID = r.EmpID

)

SELECT

FirstName + ' ' + LastName AS FullName,

EmpLevel,

(SELECT FirstName + ' ' + LastName FROM Employees

WHERE EmployeeID = cteReports.MgrID) AS Manager

FROM cteReports

ORDER BY EmpLevel, MgrID

22. **Get the names of managers for each employee from the below employee table (using join) (medium)**

|  |  |  |
| --- | --- | --- |
| ***EmpId*** | ***emp\_name*** | ***emp\_manager\_id*** |
| 1 | John | Null |
| 2 | Tom | 1 |
| 3 | Smith | 1 |
| 4 | Albert | 2 |
| 5 | David | 2 |
| 6 | Murphy | 5 |
| 7 | Petra | 5 |

Sample output:

|  |  |
| --- | --- |
| ***manager*** | ***employee*** |
| John | Tom |

Select e1.emp\_name 'manager',e2.emp\_name 'employee'

from employees e1 join employees e2

on e1.emp\_id=e2.emp\_manager\_id

23. Create the below tables using the code given **(join) (medium)**

Create table People

(

PersonID int identity primary key,

PersonName varchar (20)

)

Create table PetTypes

(

PetTypeID int identity primary key,

PetType varchar (10)

)

Create table Pets

(

PetID int identity primary key,

PetTypeID int references PetTypes (PetTypeID) not null,

PetName varchar (10),

OwnerID int references People (PersonID) not null

)

Insert into People (PersonName)

Select 'Fred Flintstone' union all

Select 'Barney Rubble' union all

Select 'George Jetson'

Insert into PetTypes (PetType)

Select 'Dinosaur' union all

select 'Hopparoo'

Insert into Pets (PetTypeID, PetName, OwnerID)

Select 1,'Dino', 1 union all

Select 2,'Hoppy', 2

Write a query to display all People and their pets, if any

select People.PersonName, Pets.PetName

from People

left outer join Pets on Pets.OwnerID = People.PersonID

Expected output:

PersonName PetName

-------------------- ----------

Fred Flintstone Dino

Barney Rubble Hoppy

George Jetson NULL

24. Using the tables created in the above question; write a query which would display the personname, petname and the pettype, if any **(join )(medium)**

select People.PersonName, Pets.PetName, PetTypes.PetType

from People

left outer join Pets on Pets.OwnerID = People.PersonID

inner join PetTypes on Pets.PetTypeID = PetTypes.PetTypeID

Expected output:

PersonName           PetName    PetType      
-------------------- ---------- ----------   
Fred Flintstone      Dino       Dinosaur  
Barney Rubble        Hoppy      Hopparoo  
**George Jetson        NULL       NULL**

25. Using the Production.Product, Purchasing.ProductVendor and Purchasing.Vendor tables in Adventureworks database write a query which would display the ProductName along with their VendorName. (medium)

Expected Output:

ProductName VendorName

-------------------------------------------------------

Thin-Jam Hex Nut 1 Advanced Bicycles

Thin-Jam Hex Nut 10 Advanced Bicycles

Thin-Jam Hex Nut 11 Advanced Bicycles

Thin-Jam Hex Nut 12 Advanced Bicycles

SELECT p.Name, v.Name

FROM Production.Product p

JOIN Purchasing.ProductVendor pv

ON p.ProductID = pv.ProductID

JOIN Purchasing.Vendor v

ON pv.VendorID = v.VendorID

ORDER BY v.Name

26. **join (medium)**

Using HumanResources.Employee, HumanResources.EmployeeDepartmentHistory and

HumanResources.Department from the adventureworks database write a query to return the

EmployeeId, Designation, DepartmentName, GroupName and Startdate

select he.employeeid , he.title designation,hd.name deptname, hd.groupname ,hed.startdate from HumanResources.Employee he

join HumanResources.EmployeeDepartmentHistory hed on

he.EmployeeID = hed.EmployeeID

join HumanResources.Department hd

on hd.departmentid = hed.DepartmentID

Expected Output:

employeeID designation departmentname groupname StartDate

---------------------------------------------------------------------------------------------------------------

1 Executive Production Manufacturing 1996-07-31 00:00:00.000

2 Executive Marketing Sales and Marketing 1997-02-26 00:00:00.000

3 Executive science Research and Development1997-12-12 00:00:00.000

27. **Union, intersect, except (medium)**

CREATE VIEW Lunch

AS

SELECT 'Beer' AS item

UNION SELECT 'Olives'

UNION SELECT 'Bread'

UNION SELECT 'Salami'

UNION SELECT 'Calamari'

UNION SELECT 'Coffee';

GO

CREATE VIEW Dinner

AS

SELECT 'Wine' AS item

UNION SELECT 'Olives'

UNION SELECT 'Bread'

UNION SELECT 'Steak'

UNION SELECT 'Aubergines'

UNION SELECT 'Salad'

UNION SELECT 'Coffee'

GO

Using the above views write a query using UNION operator in order to return everything you’ve eaten today

SELECT item FROM Lunch

UNION

SELECT item FROM Dinner;

Using EXCEPT operator write a query which would return only the food you ate (or drank) for lunch, but did not have for dinner

SELECT item FROM Lunch

EXCEPT

SELECT item FROM Dinner;

Using EXCEPT operator write a query which would return those items you had for dinner but not lunch

SELECT item FROM Dinner

EXCEPT

SELECT item FROM Lunch;

Use the INTERSECT operator to return only the food that was eaten at both meals

SELECT item FROM Dinner

INTERSECT

SELECT item FROM Lunch;

Write a query return a list of food that you ate at one of the meals, but not both meals, in other words, the food you ate other than bread, olives, and coffee

SELECT item

FROM

  (

    SELECT item FROM Lunch

    EXCEPT SELECT item FROM Dinner

  ) Only\_Lunch

UNION

SELECT item

FROM

  (

    SELECT item FROM Dinner

    EXCEPT SELECT item FROM Lunch

  ) Only\_Dinner; --Items you only ate once in the day.

28. **Aggregate function (medium)**

Create the below table : company

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Company\_ID** | **CompanyName** | **Address** | **Town** | **Sales** |
| 1 | SQL Sample | 1 Sample St | Hamburg | 10000 |
| 2 | SQL Code Land | 2 Code Rd | Hamburg | 20000 |
| 3 | Sample Code World | 66 SQL St | Curry | 18000 |
| 4 | SQL Reference Ltd | 34 Reference St | Pisa | 17000 |

Write a query to calculate the average sales **for each Town**

SELECT Town, AVG (Sales) FROM Company GROUP BY Town;

29. **functions, over clause(medium)**

Using the MIN, MAX, AVG functions with the OVER clause, write a query to provide aggregated values for each department in the HumanResources.Department table. Refer HumanResources.EmployeePayHistory, HumanResources.EmployeeDepartmentHistory and HumanResources.Department in adventureworks database.

SELECT DISTINCT Name,MIN(Rate) OVER (PARTITION BY edh.DepartmentID) AS MinSalary, MAX(Rate) OVER (PARTITION BY edh.DepartmentID) AS MaxSalary ,AVG(Rate) OVER (PARTITION BY edh.DepartmentID) AS AvgSalary

FROM HumanResources.EmployeePayHistory AS eph

JOIN HumanResources.EmployeeDepartmentHistory AS edh

ON eph.EmployeeID = edh.EmployeeID

JOIN HumanResources.Department AS d

ON d.DepartmentID = edh.DepartmentID

WHERE edh.EndDate IS NULL

ORDER BY Name;

30**.(medium)function**

Write a query to return the standard deviation for all bonus values in the SalesPerson table

SELECT STDEV(Bonus)FROM Sales.SalesPerson;

31**.(medium)function**

Write a query to return the variance for all bonus values in the SalesPerson table

SELECT VAR(Bonus) FROM Sales.salesperson;

32. **(Medium) function**

Using the salesorderdetail in adventureworks and over clause, calculate the sum, avg, count, min, max of orderquantity column.

SELECT SalesOrderID, ProductID, OrderQty

,SUM(OrderQty) OVER(PARTITION BY SalesOrderID) AS Total

,AVG(OrderQty) OVER(PARTITION BY SalesOrderID) AS "Avg"

,COUNT(OrderQty) OVER(PARTITION BY SalesOrderID) AS "Count"

,MIN(OrderQty) OVER(PARTITION BY SalesOrderID) AS "Min"

,MAX(OrderQty) OVER(PARTITION BY SalesOrderID) AS "Max"

FROM Sales.SalesOrderDetail

GO

33. **(Medium) function**

Using the sales.specialoffer table in Adventureworks database, write a query to display description, discountpct, minquantity, maximum quantity.

Substitute the null value in the max quantity column to zero or any value (isnull function)

SELECT Description, DiscountPct, MinQty, ISNULL (MaxQty, 0.00) AS 'Max Quantity'

FROM Sales.specialoffer;

34**.(medium)function**

Using the aggregate functions to calculate the volume of cylinder having radius of 1 inch and a height of 5 inch (hint: use a variable, formula: pie r^2 h)

DECLARE @h float, @r float

SET @h = 5

SET @r = 1

SELECT PI()\* SQUARE(@r)\* @h AS 'Cyl Vol'

35**. (Medium) function**

Create an employee table with firstname, middlename, lastname.

Use a built in sql server function to display the fullname of the employee (HINT: firstame+middlename+lastname)

CREATE TABLE #temp (

emp\_name nvarchar(200) NOT NULL,

emp\_middlename nvarchar(200) NULL,

emp\_lastname nvarchar(200) NOT NULL

);

INSERT INTO #temp VALUES( 'fredrick', 'joan', 'harow' );

SELECT emp\_name + ' ' + emp\_middlename + ' ' + emp\_LastName

FROM #temp

36. **(Medium) function**

Run the below query

IF EXISTS (SELECT name FROM sys.tables

WHERE name = 't1')

DROP TABLE t1;

GO

CREATE TABLE t1

(

c1 varchar(3),

c2 char(3)

);

GO

INSERT INTO t1 VALUES ('2', '2'), ('37', '37'),('597', '597');

GO

Write a query to left pad the column c1 values to a specified length while converting them from a numeric data type to character.

SELECT REPLICATE('0', 3 - DATALENGTH(c1)) + c1 AS 'Varchar Column',

REPLICATE('0', 3 - DATALENGTH(c2)) + c2 AS 'Char Column'

FROM t1;

37**. (Medium) function**

Create an employee table with columns firstname and lastname.

Write a query to return the last name in one column with only the first initial (from the firstname column), in the other column

SELECT LastName, SUBSTRING(FirstName, 1, 1) AS Initial

FROM employee

ORDER BY LastName;

38. **Intersect (medium)**

Create a table table1 in database1.

Create another table table2 in database2.

Insert values in both the tables, **few rows with same data**.

Write a query which would return the rows with similar data from both the tables table1 and table2 in database1 and database2 databases

(HINT: intersect)

USE [master]

GO

IF EXISTS (SELECT name FROM sys.databases WHERE name = N'MyDBTest')

DROP DATABASE [MyDBTest]

go

create database MyDBTest

go

use MyDBTest

go

create table Employee (id int, Fname varchar(50), Lname varchar(50))

go

Insert into Employee values ( 1, 'Jen' , 'Ambelang' )

Insert into Employee values ( 1, 'Jen' , 'Ambelang' )

Insert into Employee values ( 1, 'Jen' , 'Ambelang' )

Insert into Employee values ( 2, 'Alan' , 'Eechi' )

Insert into Employee values ( 3, 'Steve' , 'Borders' )

Insert into Employee values ( 4, 'Adam' , 'Carlos' )

Insert into Employee values ( 5, 'Walter' , 'Williams' )

Insert into Employee values ( 6, 'Madoka' , 'Kurosawa' )

Insert into Employee values ( 7, 'Jane' , 'Johnson' )

Insert into Employee values ( 7, 'Jane' , 'Johnson' )

Insert into Employee values ( 8, 'Hong' , 'Annie' )

Insert into Employee values ( 9, 'Lily' , 'Chang' )

Insert into Employee values ( 10, 'Frank' , 'Zhao' )

go

USE [master]

GO

IF EXISTS (SELECT name FROM sys.databases WHERE name = N'MyDBTest2')

DROP DATABASE [MyDBTest2]

go

create database MyDBTest2

go

use MyDBTest2

go

create table Employee (id int, Fname varchar(50), Lname varchar(50))

go

Insert into Employee values ( 1, 'Jen' , 'Ambelang' )

Insert into Employee values ( 2, 'Alan' , 'Eechi' )

Insert into Employee values ( 2, 'Alan' , 'Eechi' )

Insert into Employee values ( 13, 'Steve' , 'Borders' )

Insert into Employee values ( 4, 'Adam' , 'Carlos' )

Insert into Employee values ( 5, 'Walter' , 'Williams' )

Insert into Employee values ( 5, 'Walter' , 'Williams' )

Insert into Employee values ( 5, 'Walter' , 'Williams' )

Insert into Employee values ( 11, 'Nui' , 'Wanarasat' )

go

SELECT \* FROM MyDBTest.dbo.Employee

SELECT \* FROM MyDBTest2.dbo.Employee

SELECT \* FROM MyDBTest.dbo.Employee

INTERSECT

SELECT \* FROM MyDBTest2.dbo.Employee

39. **except (medium)**

From the tables created above (Q46)

Write a query which would return the rows which do not have similar data from both the tables, in different databases, by running the query from any one of the databases created.

(HINT: except)

SELECT \* FROM MyDBTest.dbo.Employee

except

SELECT \* FROM MyDBTest2.dbo.Employee

40. CTE(medium)

Write a common table expression to delete duplicate rows from the table below:

CREATE TABLE DuplicateRcordTable (Col1 INT, Col2 INT)

INSERT INTO DuplicateRcordTable

SELECT 1, 1

UNION ALL

SELECT 1, 1 --duplicate

UNION ALL

SELECT 1, 1 --duplicate

UNION ALL

SELECT 1, 2

UNION ALL

SELECT 1, 2 --duplicate

UNION ALL

SELECT 1, 3

UNION ALL

SELECT 1, 4

select \* from DuplicateRcordTable;

WITH CTE (COl1,Col2, DuplicateCount)

AS

(

SELECT COl1,Col2,

ROW\_NUMBER() OVER(PARTITION BY COl1,Col2 ORDER BY Col1) AS DuplicateCount

FROM DuplicateRcordTable

)

DELETE

FROM CTE

WHERE DuplicateCount > 1

GO

select \* from DuplicateRcordTable;

41. **Except (medium)**

Create the following table:

CREATE TABLE EmployeeRecord

(EmpNo INT NOT NULL, EmpName VARCHAR (10),

EmpPost VARCHAR (9), ManagerID INT,

Salary INT, COMM INT, Deptno INT);

INSERT INTO EmployeeRecord

VALUES (7369, 'SMITH', 'CLERK', 7902, 800, NULL, 20);

INSERT INTO EmployeeRecord

VALUES (7499, 'ALLEN', 'SALESMAN', 7698, 1600, 300, 30);

INSERT INTO EmployeeRecord

VALUES (7521, 'WARD', 'SALESMAN', 7698, 1250, 500, 30);

INSERT INTO EmployeeRecord

VALUES (7566, 'JONES', 'MANAGER', 7839, 2975, NULL, 20);

INSERT INTO EmployeeRecord

VALUES (7654, 'MARTIN', 'SALESMAN', 7698, 1250, 1400, 30);

INSERT INTO EmployeeRecord

VALUES (7698, 'BLAKE', 'MANAGER', 7839, 2850, NULL, 30);

INSERT INTO EmployeeRecord

VALUES (7782, 'CLARK', 'MANAGER', 7839, 2450, NULL, 10);

INSERT INTO EmployeeRecord

VALUES (7788, 'SCOTT', 'ANALYST', 7566, 3000, NULL, 20);

INSERT INTO EmployeeRecord

VALUES (7839, 'KING', 'PRESIDENT', NULL, 5000, NULL, 10);

INSERT INTO EmployeeRecord

VALUES (7844, 'TURNER', 'SALESMAN', 7698, 1500, 0, 30);

INSERT INTO EmployeeRecord

VALUES (7876, 'ADAMS', 'CLERK', 7788, 1100, NULL, 20);

INSERT INTO EmployeeRecord

VALUES (7900, 'JAMES', 'CLERK', 7698, 950, NULL, 30);

INSERT INTO EmployeeRecord

VALUES (7902, 'FORD', 'ANALYST', 7566, 3000, NULL, 20);

INSERT INTO EmployeeRecord

VALUES (7934, 'MILLER', 'CLERK', 7782, 1300, NULL, 10);

SELECT \*FROM EmployeeRecord;

Answer:

SELECT EmpNo, EmpName ,salary FROM EmployeeRecord

WHERE Salary > 1000

EXCEPT

SELECT EmpNo, EmpName , salary FROM EmployeeRecord

WHERE Salary > 2000

ORDER BY EmpName;

Write a query using except which would return the employees with salary greater than 1000 but lesser than 2000.

**Difficult**

42. **Join (difficult)**

Using the tables below, write a query to display the childname (firstname column), along with its father name and mother name (using prs\_father\_id and prs\_mother\_id columns)

CREATE TABLE addresses (

adr\_id INTEGER NOT NULL PRIMARY KEY,

adr\_city VARCHAR(15),

adr\_country VARCHAR(15) NOT NULL);

CREATE TABLE persons (

prs\_id INTEGER NOT NULL PRIMARY KEY,

prs\_father\_id INTEGER,

prs\_mother\_id INTEGER,

prs\_adr\_id INTEGER,

prs\_first\_name VARCHAR(15),

prs\_surname VARCHAR(15),

CONSTRAINT prs\_prs\_father\_fk FOREIGN KEY (prs\_father\_id)

REFERENCES persons(prs\_id),

CONSTRAINT prs\_prs\_mother\_fk FOREIGN KEY (prs\_mother\_id)

REFERENCES persons(prs\_id),

CONSTRAINT prs\_adr\_fk FOREIGN KEY (prs\_adr\_id)

REFERENCES addresses(adr\_id));

INSERT INTO addresses VALUES (1, 'RIGA', 'LATVIA');

INSERT INTO addresses VALUES (2, 'BERLIN', 'GERMANY');

INSERT INTO addresses VALUES (3, 'NEW YORK', 'USA');

INSERT INTO persons VALUES (1, NULL, NULL, NULL, 'JANIS', 'BERZINS');

INSERT INTO persons VALUES (2, 1, NULL, 2, 'PETER', 'BERZINS');

INSERT INTO persons VALUES (3, NULL, NULL, 2, 'ANN', 'SMYTH');

INSERT INTO persons VALUES (4, 2, 3, 2, 'CHARLES', 'BERZINS');

SELECT father.prs\_first\_name "Father Name",

mother.prs\_first\_name "Mother Name",

child.prs\_first\_name "Child Name"

FROM persons child

INNER JOIN persons father ON child.prs\_father\_id = father.prs\_id

INNER JOIN persons mother ON child.prs\_mother\_id = mother.prs\_id

Sample output

Father Name Mother Name Child Name

--------------- --------------- --------------- -----------

PETER ANN CHARLES

43. **CTE (difficult)**

Create the tables using below script

--Create Employees table and insert values.

CREATE TABLE Employees

(

empid int NOT NULL

,mgrid int NULL

,empname varchar(25) NOT NULL

,salary money NOT NULL

CONSTRAINT PK\_Employees PRIMARY KEY(empid)

);

GO

INSERT INTO Employees VALUES(1 , NULL, 'Nancy' , $10000.00);

INSERT INTO Employees VALUES(2 , 1 , 'Andrew' , $5000.00);

INSERT INTO Employees VALUES(3 , 1 , 'Janet' , $5000.00);

INSERT INTO Employees VALUES(4 , 1 , 'Margaret', $5000.00);

INSERT INTO Employees VALUES(5 , 2 , 'Steven' , $2500.00);

INSERT INTO Employees VALUES(6 , 2 , 'Michael' , $2500.00);

INSERT INTO Employees VALUES(7 , 3 , 'Robert' , $2500.00);

INSERT INTO Employees VALUES(8 , 3 , 'Laura' , $2500.00);

INSERT INTO Employees VALUES(9 , 3 , 'Ann' , $2500.00);

INSERT INTO Employees VALUES(10, 4 , 'Ina' , $2500.00);

INSERT INTO Employees VALUES(11, 7 , 'David' , $2000.00);

INSERT INTO Employees VALUES(12, 7 , 'Ron' , $2000.00);

INSERT INTO Employees VALUES(13, 7 , 'Dan' , $2000.00);

INSERT INTO Employees VALUES(14, 11 , 'James' , $1500.00);

GO

--Create Departments table and insert values.

CREATE TABLE Departments

(

deptid INT NOT NULL PRIMARY KEY

,deptname VARCHAR(25) NOT NULL

,deptmgrid INT NULL REFERENCES Employees

);

GO

INSERT INTO Departments VALUES(1, 'HR', 2);

INSERT INTO Departments VALUES(2, 'Marketing', 7);

INSERT INTO Departments VALUES(3, 'Finance', 8);

INSERT INTO Departments VALUES(4, 'R&D', 9);

INSERT INTO Departments VALUES(5, 'Training', 4);

INSERT INTO Departments VALUES(6, 'Gardening', NULL);

Using the employees and departments table write a table valued user defined function using Common table expression , which would return empid, emp\_name, mgrid, lvl when supplied with the input parameter @empid to the TVF

Expected output:

empid empname mgrid lvl

2 Andrew 1 0

5 Steven 2 1

CREATE FUNCTION dbo.fn\_getsubtree(@empid AS INT)

RETURNS @TREE TABLE

(

empid INT NOT NULL

,empname VARCHAR(25) NOT NULL

,mgrid INT NULL

,lvl INT NOT NULL

)

AS

BEGIN

WITH Employees\_Subtree(empid, empname, mgrid, lvl)

AS

(

-- Anchor Member (AM)

SELECT empid, empname, mgrid, 0

FROM Employees

WHERE empid = @empid

UNION all

-- Recursive Member (RM)

SELECT e.empid, e.empname, e.mgrid, es.lvl+1

FROM Employees AS e

JOIN Employees\_Subtree AS es

ON e.mgrid = es.empid

)

INSERT INTO @TREE

SELECT \* FROM Employees\_Subtree;

RETURN

END

GO

select \* from dbo.fn\_getsubtree (10)

44. **Join (difficult)**

Use the below tables and write queries using JOIN, to give the output referring to the 5 scenarios given below

CREATE VIEW Lunch

AS

SELECT 'Beer' AS item

UNION SELECT 'Olives'

UNION SELECT 'Bread'

UNION SELECT 'Salami'

UNION SELECT 'Calamari'

UNION SELECT 'Coffee';

GO

CREATE VIEW Dinner

AS

SELECT 'Wine' AS item

UNION SELECT 'Olives'

UNION SELECT 'Bread'

UNION SELECT 'Steak'

UNION SELECT 'Aubergines'

UNION SELECT 'Salad'

UNION SELECT 'Coffee'

GO

Using the above views write a query using UNION operator in order to return everything you’ve eaten today

Using EXCEPT operator write a query which would return only the food you ate (or drank) for lunch, but did not have for dinner

Using EXCEPT operator write a query which would return those items you had for dinner but not lunch

Use the INTERSECT operator to return only the food that was eaten at both meals

Write a query return a list of food that you ate at one of the meals, but not both meals, in other words, the food you ate other than bread, olives, and coffee

SELECT DISTINCT COALESCE(Lunch.item, Dinner.item) AS item

FROM Lunch

  FULL OUTER JOIN Dinner

  ON Dinner.item = Lunch.item

SELECT Lunch.item

FROM Lunch

  LEFT OUTER JOIN Dinner

  ON Dinner.item = Lunch.item

WHERE dinner.item IS NULL;

SELECT dinner.item

FROM dinner

  LEFT OUTER JOIN Lunch

  ON Dinner.item = Lunch.item

WHERE Lunch.item IS NULL;

SELECT Dinner.item

FROM Dinner

  INNER JOIN Lunch

  ON Dinner.item = Lunch.item;

SELECT COALESCE(Dinner.item, Lunch.item) AS item

FROM Dinner

  FULL OUTER JOIN Lunch

  ON Dinner.item = Lunch.item

WHERE Dinner.item IS NULL OR Lunch.item IS NULL;

45. **Cte (difficult)**

Using the production.Product and Production.ProductsubCategory tables from Adventureworks database, write a common table expression to return the product name, category name and unit price.

WITH ProductAndCategoryNamesOverTenDollars (ProductName, CategoryName, UnitPrice) AS

(

SELECT

p.Name ProductName,

c.Name CategoryName,

p.ListPrice UnitPrice

FROM Production.Product p

INNER JOIN Production.ProductSubCategory c ON

c.ProductSubcategoryID = p.ProductSubcategoryID

WHERE p.ListPrice > 10.0

)

SELECT \*

FROM ProductAndCategoryNamesOverTenDollars

ORDER BY CategoryName ASC, UnitPrice ASC, ProductName ASC

46. **Recursive cte (difficult)**

Write a recursive common table expression to list all persons in the company along with their position in the corporate hierarchy, using employees table in adventureworks database.

WITH EmployeeHierarchy (EmployeeID, LastName, FirstName, ReportsTo, HierarchyLevel) AS

(

-- Base case

SELECT

EmployeeID,

LastName,

FirstName,

ManagerID,

1 as HierarchyLevel

FROM Employees

WHERE ManagerID IS NULL

UNION ALL

-- Recursive step

SELECT

e.EmployeeID,

e.LastName,

e.FirstName,

e.ManagerID,

eh.HierarchyLevel + 1 AS HierarchyLevel

FROM Employees e

INNER JOIN EmployeeHierarchy eh ON

e.ManagerID = eh.EmployeeID

)

SELECT \*

FROM EmployeeHierarchy

ORDER BY HierarchyLevel, LastName, FirstName

47.

**@@rowcount function (difficult)**

UPDATE the job title column based on any nationalidnumber column in the humanresources.employee table. Use **@@ROWCOUNT** function to detect if any rows were changed or not, by printing ‘success’ or ‘no row updated’ as messages.

create procedure update\_employee (@nationalid int,@title varchar(10))

as

begin

UPDATE HumanResources.Employee

SET Title = @title

WHERE NationalIDNumber = @nationalid

IF @@ROWCOUNT = 0

PRINT 'Warning: No rows updated';

ELSE

PRINT 'success rows updated'

end

select \* from HumanResources.Employee;

exec update\_employee @title = ceo , @nationalid = 15517807

48.

**Function (difficult)**

Using if else statement, check if a particular database has a particular table, return ‘table exists’ or ‘table does not exist’ messages based on the result.

create procedure check\_table (@tablename varchar(20))

as

begin

if db\_name() = 'AdventureWorks' and

(select count(\*) from sysobjects

where name = @tablename) = 1

print 'Table' + ' ' + @tablename + ' ' + 'Exists'

else

print 'Table' + ' ' + @tablename + ' ' + 'does not Exist'

end

exec check\_table @tablename = employees

49. **Pivot, unpivot (difficult)**

Run the below queries to create product table and insert values in it.

-- Creating Test Table

CREATE TABLE Product (Cust VARCHAR (25), Product VARCHAR (20), QTY INT)

GO

-- Inserting Data into Table

INSERT INTO Product (Cust, Product, QTY)

VALUES ('KATE','VEG', 2)

INSERT INTO Product (Cust, Product, QTY)

VALUES ('KATE','SODA', 6)

INSERT INTO Product (Cust, Product, QTY)

VALUES ('KATE','MILK', 1)

INSERT INTO Product (Cust, Product, QTY)

VALUES ('KATE','BEER', 12)

INSERT INTO Product (Cust, Product, QTY)

VALUES ('FRED','MILK', 3)

INSERT INTO Product (Cust, Product, QTY)

VALUES ('FRED','BEER', 24)

INSERT INTO Product (Cust, Product, QTY)

VALUES ('KATE','VEG', 3)

GO

-- Selecting and checking entires in table

SELECT \*

FROM Product

GO

1. Pivot table ordered by PRODUCT
2. Pivot table ordered by cust
3. Unpivot table ordered by cust

Expected outputs:

PRODUCT FRED KATE

--------------------------------------------

BEER 24 12

MILK 3 1

SODA NULL 6

VEG NULL 5

CUST VEG SODA MILK BEER CHIPS

----------------------------------------------------------------

FRED NULL NULL 3 24 NULL

KATE 5 6 1 12 NULL

CUST PRODUCT QTY

-----------------------------------------

FRED MILK 3

FRED BEER 24

KATE VEG 5

KATE SODA 6

KATE MILK 1

KATE BEER 12

-- Pivot Table ordered by PRODUCT  
SELECT PRODUCT, FRED, KATE  
FROM (  
SELECT CUST, PRODUCT, QTY  
FROM Product) up  
PIVOT (SUM(QTY) FOR CUST IN (FRED, KATE)) AS pvt  
ORDER BY PRODUCT  
GO  
-- Pivot Table ordered by CUST  
SELECT CUST, VEG, SODA, MILK, BEER, CHIPS  
FROM (  
SELECT CUST, PRODUCT, QTY  
FROM Product) up  
PIVOT (SUM(QTY) FOR PRODUCT IN (VEG, SODA, MILK, BEER, CHIPS)) AS pvt  
ORDER BY CUST  
GO  
-- Unpivot Table ordered by CUST  
SELECT CUST, PRODUCT, QTY  
FROM  
(  
SELECT CUST, VEG, SODA, MILK, BEER, CHIPS  
FROM (  
SELECT CUST, PRODUCT, QTY  
FROM Product) up  
PIVOT  
( SUM(QTY) FOR PRODUCT IN (VEG, SODA, MILK, BEER, CHIPS)) AS pvt) p  
UNPIVOT  
(QTY FOR PRODUCT IN (VEG, SODA, MILK, BEER, CHIPS)  
) AS Unpvt  
GO

50.

**Apply (difficult)**

Create a query using CROSS APPLY operator to merge rows from a **table valued function** returning the top 5 customers. Use Production.Product, Sales.SalesOrderDetail, Sales.SalesOrderHeader tables in the Adventureworks database.

The function should display the quantity sold for each product for a given store, on providing the input parameter @storeid of TVF with any value.

CREATE FUNCTION Sales.ufn\_QtyByStore (@storeid int)

RETURNS TABLE

AS

RETURN

(

SELECT P.ProductID, P.Name, SUM(SD.OrderQty) AS 'Quantity'

FROM Production.Product AS P

JOIN Sales.SalesOrderDetail AS SD ON SD.ProductID = P.ProductID

JOIN Sales.SalesOrderHeader AS SH ON SH.SalesOrderID = SD.SalesOrderID

WHERE SH.CustomerID = @storeid

GROUP BY P.ProductID, P.Name

)

Executing the function:

select \* from Sales.ufn\_QtyByStore (2)

cross apply:

select S.Name,QbyS.Name ProductName,QbyS.Quantity from

(select top(5) Name, CustomerId from AdventureWorks.Sales.Store) S

CROSS APPLY

(select top(2) Name, Quantity from AdventureWorks.Sales.ufn\_QtyByStore(S.CustomerId)

order by Quantity desc) QbyS