Assignment Day4 –SQL: Comprehensive practice

Answer following questions

1. What is View? What are the benefits of using views?
   A view is a virtual table whose contents are defined by a query. Like a real table, a view consists of a set of named columns and rows of data.   The types of view are regular view, indexed view and distributed partitioned view.
   For beginners, it is easy to select from a view rather than using join and sub-query. Different views can represent different records from a table. Views can join and simplify multiple tables into a single virtual table. A view also takes little space to store.
2. Can data be modified through views?
   Yes. Once we insert a new record to a view, the record will be inserted into the base table. We can also delete and update records in a view and change the data .
3. What is stored procedure and what are the benefits of using it?
   A stored procedure groups one or more Transact-SQL statements into a logical unit, stored as an object in a SQL Server database.
   Reusing stored procedure for multiple times can save the time compared to user-defined function or views.
   When a stored procedure is executed for the first time SQL determines the most optimal query access plan and stores it in the plan memory cache. SQL Server can then reuse the plan on subsequent executions of this stored procedure.
   Plan reuse allows stored procedures to provide fast and reliable performance compared to non-compiled temporary query equivalents.
4. What is the difference between view and stored procedure?
   A view does not accept parameters but the stored procedure does. A view can be used as a block in large query. SQL Server can reuse the saved plan on subsequent executions of stored procedure, for a view SQL SERVER has to determine the plan every time. A view contains only one select query but a stored procedure can contain multiple ones.
5. What is the difference between stored procedure and functions?
   A function must return a value while a stored procedure may or not return values. A function will allow only select statements and not execute DML statements. A stored procedure can execute DML statements. A function will allow only input parameters, doesn't support output parameters. But a stored procedure can have both input and output parameters. Transactions are not allowed in functions, however available in stored procedures. Functions can be called from select statement but stored procedure can't be. A function can be used in join clause as a result set but stored procedure can't.

6. Can stored procedure return multiple result sets?
   Yes. A stored procedure can include one or more select queries, which leads to one or more result sets.
7. Can stored procedure be executed as part of SELECT Statement? Why?
   No. Because a stored procedure does not return a result set. It just executes several statements.
8. What is Trigger? What types of Triggers are there?
   Triggers are a special type of stored procedure that get executed (fired) when a specific event happens. Triggers are automatically fired on a event.
   DDL Trigger, DML Trigger.
9. What are the scenarios to use Triggers?
   When the user wants to enforce integrity beyond simple referential integrity, implement business rules, maintain audit record of changes and accomplish cascading updates and deletes.
10. What is the difference between Trigger and Stored Procedure?
    Trigger runs automatically when specific events happen, while Stored procedures must be invoked explicitly by the user. A Stored Procedure has input/output parameters but a Trigger doesn't. Transaction statements can not be used in a trigger but can be used in a Stored Procedure.

Write queries for following scenarios

Use Northwind database. All questions are based on assumptions described by the Database Diagram sent to you yesterday. When inserting, make up info if necessary. Write query for each step. Do not use IDE. BE CAREFUL WHEN DELETING DATA OR DROPPING TABLE.

1. Lock tables Region, Territories, EmployeeTerritories and Employees. Insert following information into the database. In case of an error, no changes should be made to DB.
   a. A new region called "Middle Earth";
   b. A new territory called "Gondor", belongs to region "Middle Earth";
   c. A new employee "Aragorn King" who's territory is "Gondor".
```
begin tran
select * from region with(tablock, holdlock)
select * from Territories with(tablock, holdlock)
select * from EmployeeTerritories  with(tablock, holdlock)
select * from Employees with(tablock, holdlock)
insert into Region values(5, 'Middle Earth')
insert into Territories values(99999, 'Gondor', 5)
insert into Employees(FirstName,LastName) values('Aragorn ', 'King')
insert into EmployeeTerritories values(11, 99999)
```
2. Change territory "Gondor" to "Arnor".
```
update Territories set TerritoryDescription = 'Arnor' where TerritoryID = 99999
```

3. Delete Region “Middle Earth”. (tip: remove referenced data first) (Caution: do not forget WHERE or you will delete everything.) In case of an error, no changes should be made to DB. Unlock the tables mentioned in question 1.
```
delete from EmployeeTerritories where EmployeeID = 11
delete from Territories where TerritoryID = 99999
delete from Region where RegionID = 5
commit tran
```
4. Create a view named “view_product_order_[your_last_name]”, list all products and total ordered quantity for that product.
```
create view view_product_order_LI as
select p.ProductID, p.ProductName, sum(od.Quantity) TotalQTY from Products p
join [Order Details] od on od.ProductID = p.ProductID
group by p.ProductID, p.ProductName
select top 100 percent * from view_product_order_LI
```
5. Create a stored procedure “sp_product_order_quantity_[your_last_name]” that accept product id as an input and total quantities of order as output parameter.
```
create proc sp_product_order_quantity_LI
@pid int,
@total_qty int out
as
begin
select @total_qty = count(distinct od.orderid)
from [Order Details] od
join Products p on p.ProductID = od.ProductID
where p.ProductID = @pid
group by p.ProductID
end

declare @qty int
exec sp_product_order_quantity_LI 1, @qty out
print @qty
```
6. Create a stored procedure “sp_product_order_city_[your_last_name]” that accept product name as an input and top 5 cities that ordered most that product combined with the total quantity of that product ordered from that city as output.
```
create proc sp_product_order_city_LI
@pname nvarchar(40)
as
begin
select top 5 o.ShipCity, sum(od.Quantity) from [Order Details] od
join Orders o on o.OrderID = od.OrderID
join Products p on p.ProductID = od.ProductID
where p.ProductName = @pname
group by o.ShipCity
order by sum(od.Quantity) desc
end

exec sp_product_order_city_LI 'Chai'
```
7. Lock tables Region, Territories, EmployeeTerritories and Employees. Create a stored procedure “sp_move_employees_[your_last_name]” that automatically find all employees in territory “Tory”; if more than 0 found, insert a new territory “Stevens

Point" of region "North" to the database, and then move those employees to "Stevens Point".

```
create proc sp_move_employees_LI
as
begin
declare @rows int
select @rows = count(*) from Territories t
join EmployeeTerritories et on et.TerritoryID = t.TerritoryID
where t.TerritoryDescription = 'Tory'
if @rows > 0
begin
       insert into Territories values(99999, 'Stevens Point', 3)
       declare @eid int, @tid int
       while @rows > 0
       begin
       select @rows = @rows - 1
       select @eid = et.EmployeeID, @tid = et.TerritoryID from Territories t
       join EmployeeTerritories et on et.TerritoryID = t.TerritoryID
       where t.TerritoryDescription = 'Tory'
       update EmployeeTerritories set TerritoryID = 99999
       where EmployeeID = @eid and TerritoryID = @tid
       end
end
end
exec sp_move_employees_LI
```

8. Create a trigger that when there are more than 100 employees in territory "Stevens Point", move them back to Troy. (After test your code,) remove the trigger. Move those employees back to "Troy", if any. Unlock the tables.

```
create trigger trigger_LI
on EmployeeTerritories
after update
as
begin
declare @rows int
select @rows = count(*) from Territories t
join EmployeeTerritories et on et.TerritoryID = t.TerritoryID
where t.TerritoryDescription = 'Stevens Point'
if @rows > 100
begin
    declare @eid int, @tid int
    while @rows > 0
    begin
    select @rows = @rows - 1
    select @eid = et.EmployeeID, @tid = et.TerritoryID from Territories t
    join EmployeeTerritories et on et.TerritoryID = t.TerritoryID
    where t.TerritoryDescription = 'Stevens Point'
    update EmployeeTerritories set TerritoryID = 48084
    where EmployeeID = @eid and TerritoryID = @tid
    end
end
end
drop trigger trigger_LI
```

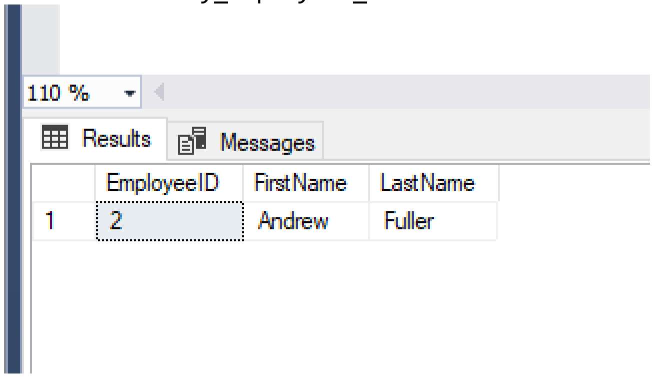
```
    commit tran
```

9. Create 2 new tables “people_your_last_name” “city_your_last_name”. City table has two records: {Id:1, City: Seattle}, {Id:2, City: Green Bay}. People has three records: {id:1, Name: Aaron Rodgers, City: 2}, {id:2, Name: Russell Wilson, City:1}, {Id: 3, Name: Jody Nelson, City:2}. Remove city of Seattle. If there was anyone from Seattle, put them into a new city “Madison”. Create a view “Packers_your_name” lists all people from Green Bay. If any error occurred, no changes should be made to DB. (after test) Drop both tables and view.

```
begin tran
create table city_LI(id int primary key,city varchar(20))
create table people_LI(
id int primary key, name varchar(20),
city int foreign key references city_LI(id) on delete set null
)
insert into city_LI values(1,'Seattle')
insert into city_LI values(2,'Green Bay')
insert into people_LI values(1,'Aaron Rodgers',2)
insert into people_LI values(2,'Russell Wilson',1)
insert into people_LI values(3,'Jody Nelson',2)
select * from city_LI
select * from people_LI
delete from city_LI where city = 'Seattle'
declare @rows int
select @rows = count(id) from people_LI where city is null
print @rows
if @rows > 0
begin
       declare @pid int
       insert into city_LI values (3, 'Madison')
       while @rows > 0
       begin
              select @rows = @rows - 1
              select @pid = id from people_LI where city is null
              update people_LI set city = 3 where id = @pid
       end
end
create view Packers_LI as
select p.name from people_LI p
join city_LI c on c.id = p.city
where c.city = 'Green Bay'
select * from Packers_LI
drop view Packers_LI
drop table people_LI, city_LI
commit tran
```

10. Create a stored procedure “sp_birthday_employees_[you_last_name]” that creates a new table “birthday_employees_your_last_name” and fill it with all employees that have a birthday on Feb. (Make a screen shot) drop the table. Employee table should not be affected.

```
create proc sp_birthday_employees_LI
as
begin
select EmployeeID, FirstName, LastName into birthday_employees_LI from Employees
where DATEPART(m,BirthDate) = 2
end
sp_birthday_employees_LI
select * from birthday_employees_LI
```



110 %

Results | Messages

	EmployeeID	FirstName	LastName
1	2	Andrew	Fuller

11. Create a stored procedure named “sp_your_last_name_1” that returns all cites that have at least 2 customers who have bought no or only one kind of product. Create a stored procedure named “sp_your_last_name_2” that returns the same but using a different approach. (sub-query and no-sub-query).

```
create proc sp_LI_1
as
begin
select c.City, count(distinct od.ProductID) from orders o
join [Order Details] od
on od.OrderID = o.OrderID
right join Customers c
on c.CustomerID = o.CustomerID
group by c.City, c.CustomerID
having count(distinct od.ProductID) < 2 and count( c.CustomerID) > 1
end

create proc sp_LI_2
as
begin
select c1.City from Customers c1
where c1.CustomerID in
(select c.CustomerID from orders o
join [Order Details] od
on od.OrderID = o.OrderID
right join Customers c
on c.CustomerID = o.CustomerID
group by c.CustomerID
having count(distinct od.ProductID) < 2)
group by c1.city
having count(c1.CustomerID) > 1
end
```

12. How do you make sure two tables have the same data?

Imagine we have two tables a and b. They have field names f1, f2, f3, f4.

```
select *
from a
where not exists
(select 1 from b where a.f1=b.f1 and a.f2=b.f2 and a.f3=b.f3 and a.f4=b.f4)
union all
select *
from b
where not exists
(select 1 from a where a.f1=b.f1 and a.f2=b.f2 and a.f3=b.f3 and a.f4=b.f4)
```

14.

First Name	Last Name	Middle Name
John	Green	
Mike	White	M

Output should be

Full Name
John Green
Mike White M.

Note: There is a dot after M when you output.

```
create table temp1([First name] varchar(20),
[Last name] varchar(20), [Middle name] varchar(20))
create table temp2([Full name] varchar(40)

insert into temp1 values('John','Green', null)
insert into temp1 values('Mike','White', 'M')
select * from temp1
declare @rows int, @first varchar(20), @last varchar(20), @middle varchar(20)
select @rows = count(*) from temp1
print @rows
while @rows > 0
begin
       select @rows = @rows - 1
       select @first = [First name], @last = [Last name], @middle = [Middle name]
from temp1
       if @middle is null
       insert into temp2 values(concat(@first, ' ', @last))
       if @middle is not null
       insert into temp2 values(concat(@first, ' ', @last, ' ', @middle,'.'))

       delete from temp1 where [FIRST name] = @first
end
```

15.

Student	Marks	Sex
Ci	70	F
Bob	80	M
Li	90	F
Mi	95	M

Find the top marks of Female students.

If there are to students have the max score, only output one.

```
select top 1 Marks from table1 where sex = 'F'
order by Marks desc
```

16.

Student	Marks	Sex
Li	90	F
Ci	70	F
Mi	95	M
Bob	80	M

How do you out put this?

```
select top 1 Marks from table2 where sex = 'F'
order by Marks desc
```

GOOD LUCK.