Assignment Day4 –SQL: Comprehensive practice

# Answer following questions

1. What is View? What are the benefits of using views?

A:

view: virtual table that reference to data from one or multiple tables. It does not hold any data and does not exist physically in the database. Similar to a SQL table, the view name should be unique in a database. In a VIEW, we can also control user security for accessing the data from the database tables. We can allow users to get the data from the VIEW, and the user does not require permission for each table or column to fetch data.

Benefits of using views:

* Views can represent a subset of the data contained in a table. Consequently, a view can limit the degree of exposure of the underlying tables to the outer world: a given user may have permission to query the view, while denied access to the rest of the base table.
* Views can [join](https://en.wikipedia.org/wiki/Join_(SQL)) and simplify multiple tables into a single virtual table.
* Views can act as aggregated tables, where the [database engine](https://en.wikipedia.org/wiki/Database_engine) aggregates data ([sum](https://en.wikipedia.org/wiki/Summation), [average](https://en.wikipedia.org/wiki/Average), etc.) and presents the calculated results as part of the data.
* Views can hide the complexity of data. For example, a view could appear as Sales2000 or Sales2001, transparently [partitioning](https://en.wikipedia.org/wiki/Partition_(database)) the actual underlying table.
* Views take very little space to store; the database contains only the definition of a view, not a copy of all the data that it presents.
* Depending on the [SQL](https://en.wikipedia.org/wiki/SQL) engine used, views can provide extra security.

1. Can data be modified through views?

A:

Yes, users can modify the actual table from where the view is referencing or edited the view itself. Either way, the data in actual table will be modified. But there are read-only views which will not allow users to update or insert data into read-only views.

1. What is stored procedure and what are the benefits of using it?

A:

Stored procedure is used for preparing sql query that we can save and reuse over and over again.

1. What is the difference between view and stored procedure?

View is a virtual table which contains data. SP is stored a block of sql codes.

1. What is the difference between stored procedure and functions?

A:

The **function must return a value** but in Stored Procedure it is optional. Even a procedure can return zero or n values. Functions can have only input parameters for it whereas Procedures can have input or output parameters. Functions can be called from Procedure whereas Procedures cannot be called from a Function.

1. Can stored procedure return multiple result sets?

A: Yes, it could.

1. Can stored procedure be executed as part of SELECT Statement? Why?

A: No, SP relies on “EXEC SP\_Name” to be called. So, SELECT statement is not able to call SP.

1. What is Trigger? What types of Triggers are there?

A:

A trigger is a special type of stored procedure in database that automatically invokes/runs/fires when an event occurs in the database server. A trigger uses the special table to keep a copy of the row which we have just inserted, deleted or modified.

There are many types of triggers in SQL Server:

DDL Trigger (Data Definition Language events are CREATE, ALTER and DROP statements.)

DML Trigger (Data Manipulation Language (DML) events, such as INSERT, UPDATE or DELETE action)

**BEFORE triggers**

Run before an update, or insert. Values that are being updated or inserted can be modified before the database is actually modified.

**BEFORE DELETE triggers**

Run before a delete. Checks values (a raises an error, if necessary).

**AFTER triggers**

Run after an update, insert, or delete.

**INSTEAD OF triggers**

Describe how to perform insert, update, and delete operations against views that are too complex to support these operations natively. They allow applications to use a view as the sole interface for all SQL operations (insert, delete, update and select).

Logon Trigger (Logon triggers fire stored procedures in response to a LOGON event. This event is raised when a user session is established with an instance of SQL Server. Logon triggers fire after the authentication phase of logging in finishes, but before the user session is actually established. Therefore, all messages originating inside the trigger that would typically reach the user, such as error messages and messages from the PRINT statement, are diverted to the SQL Server error log. Logon triggers do not fire if authentication fails.)

1. What are the scenarios to use Triggers?

**BEFORE triggers**

Run before an update, or insert. Values that are being updated or inserted can be modified before the database is actually modified. You can use triggers that run before an update or insert in several ways:

* To check or modify values before they are actually updated or inserted in the database. This is useful if you must transform data from the way the user sees it to some internal database format.
* To run other non-database operations coded in user-defined functions.

**BEFORE DELETE triggers**

Run before a delete. Checks values (a raises an error, if necessary).

**AFTER triggers**

Run after an update, insert, or delete. You can use triggers that run after an update or insert in several ways:

* To update data in other tables. This capability is useful for maintaining relationships between data or in keeping audit trail information.
* To check against other data in the table or in other tables. This capability is useful to ensure data integrity when referential integrity constraints aren't appropriate, or when table check constraints limit checking to the current table only.
* To run non-database operations coded in user-defined functions. This capability is useful when issuing alerts or to update information outside the database.

**INSTEAD OF triggers**

Describe how to perform insert, update, and delete operations against views that are too complex to support these operations natively. They allow applications to use a view as the sole interface for all SQL operations (insert, delete, update and select).

1. What is the difference between Trigger and Stored Procedure?

A: A trigger is a special type of stored procedure in database that automatically invokes/runs/fires when an event occurs in the database server. A trigger uses the special table to keep a copy of the row which we have just inserted, deleted or modified.

# Write queries for following scenarios (1,2,3,7,8 not required)

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.
   1. A new region called “Middle Earth”;
   2. A new territory called “Gondor”, belongs to region “Middle Earth”;
   3. A new employee “Aragorn King” who's territory is “Gondor”.
2. --Change territory “Gondor” to “Arnor”.
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.
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\_Liu

AS

SELECT P.ProductID, ProductName, SUM(OD.Quantity) AS TotalQuantity

FROM Products P RIGHT JOIN [Order Details] OD ON P.ProductID = OD.ProductID

GROUP BY P.ProductID, ProductName

1. 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\_Liu

@p\_id INT,

@total\_Quan INT OUTPUT

AS

BEGIN

SELECT @total\_Quan = SUM(quantity)

FROM [Order Details]

WHERE ProductID = @p\_id

GROUP BY ProductID

END

1. 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\_Liu

@p\_name VARCHAR(20),

@city VARCHAR(20) OUTPUT,

@total\_quan int OUT

AS

BEGIN

select top 5 @city = City, @total\_quan = SUM(Quantity)

from Customers c join Orders o on

o.CustomerID=c.CustomerID join

[Order Details] od on od.OrderID=o.OrderID

join Products p on od.ProductID = p.ProductID

where p.ProductName = @p\_name

group by city

order by SUM(Quantity) desc

END

1. --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”.
2. --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.
3. 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.

--create two tables begin

CREATE TABLE people\_Liu(Id INT PRIMARY KEY, Name varchar(20), City INT) --FOREIGN KEY REFERENCES city\_Liu(ID))

INSERT INTO people\_liu

Values (1, 'Aaron Rodgers', 2), (2, 'Russell Wilson', 1), (3, 'Jody Nelson', 2)

--SELECT \* FROM people\_Liu

CREATE TABLE city\_Liu(Id INT PRIMARY KEY, City varchar(20) NOT NULL)

INSERT INTO city\_Liu

VALUES (1, 'SEATTLE'), (2, 'GREEN BAY')

--SELECT \* FROM city\_Liu

--create two tables end

--modify tables begin

UPDATE city\_Liu

SET CITY = 'Madison'

WHERE City = 'Seattle'

SELECT \* FROM city\_Liu

SELECT \* FROM people\_Liu

--modify tables end

--create view begin

CREATE VIEW Packers\_Qixin\_Liu

AS

SELECT P.Id, P.Name

FROM people\_Liu P JOIN city\_Liu C ON P.City = C.Id

WHERE C.City = 'Green Bay'

SELECT \* FROM Packers\_Qixin\_Liu

--create view end

--dropping tables and view

DROP TABLE people\_Liu

DROP TABLE city\_Liu

DROP VIEW Packers\_Qixin\_Liu

1. 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\_Liu

AS

begin

SELECT \*

INTO birthday\_employees\_Liu

FROM Employees e

WHERE MONTH(e.BirthDate) = 2

END

EXEC sp\_birthday\_employees\_Liu

select \* from birthday\_employees\_Liu

Graphical user interface, text, application, email

Description automatically generated

1. Create a stored procedure named “sp\_your\_last\_name\_1” that returns all cities 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\_Liu\_1

AS

BEGIN

SELECT C2.City

FROM CUSTOMERS C2

WHERE C2.CUSTOMERID IN (SELECT C1.CustomerID

FROM CUSTOMERS C1 JOIN Orders O ON C1.CustomerID = O.CustomerID JOIN [Order Details] OD ON O.OrderID = OD.OrderID

GROUP BY C1.City, C1.CustomerID

HAVING COUNT(OD.ProductID) <= 1)

GROUP BY C2.CustomerID, C2.City

HAVING COUNT(CUSTOMERID) >= 2

END

CREATE PROC sp\_Liu\_2

AS

BEGIN

SELECT City

FROM Customers

GROUP BY City

HAVING COUNT(CUSTOMERID) >=2

INTERSECT

SELECT C3.CITY

FROM CUSTOMERS C3 JOIN Orders O1 ON C3.CustomerID = O1.CustomerID JOIN [Order Details] OD1 ON O1.OrderID = OD1.OrderID

GROUP BY C3.City, C3.CustomerID

HAVING COUNT(OD1.ProductID) <= 1

END

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

A:

Using the (EXCEPT) function for the two tables, if the result is empty that means they are equal.

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.

A:

SELECT [FIRST NAME] + ‘ ‘ + [LAST NAME] + ‘ ‘ + [MIDDLE NAME] + ‘.’

FROM TABLE\_NAME

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.

A:

SELECT MAX(MARKS)

FROM TABLE\_NAME

WHERE SEX = ‘F’

16.

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

How do you out put this?

A:

SELECT STUDENT, MARKS, SEX

FROM TABLE\_NAME

ORDER BY SEX, MARKS, STUDENT