**SQL Assignment 4**

1. Explain different types of views. Demonstrate with suitable examples.

Answer :

A **View** in SQL as a logical subset of data from one or more tables. Views are used to restrict data access. A View contains no data of its own but its like window through which data from tables can be viewed or changed.

There are two types of View :

CREATE database study;

use study;

create table employee (

emp\_id int ,

emp\_name varchar(50),

dep\_id int,

qualification varchar(20));

select \* from employee;

insert into employee values(

1,'Rahul',11,'B.E.');

insert into employee values(2,'Ravi',11,'B.A.'),(3,'Ram',11,'B.Com'),(4,'Sham',12,'C.A.'),(5,'Sam',12,'C.A.');

create table departments(

dep\_id int,

dep\_name varchar(50));

insert into departments values(

11,'Tech'),(12,'Sales');

Simple view

Contains only one single base table or is created from only one table.

Ex.

CREATE VIEW employee\_dept1\_view AS

SELECT \* FROM employee

WHERE dep\_id = 11;

2. Complex View:

Contains more than one base tables or is created from more than one tables.

CREATE VIEW employee\_details AS

SELECT e.emp\_name,

e.dep\_id,

e.emp\_id,

e.qualification,

d.dep\_name

FROM

employee as e JOIN departments as d

ON e.dep\_id = d.dep\_id;

1. What is the difference between function and stored procedure? Write syntax for creating functions and stored procedures.

Answer : Both stored procedures and functions are database objects which contain a set of SQL statements to complete a task.

## Stored Procedures

Stored Procedures are pre-compiled objects which are compiled for the first time and its compiled format is saved, which executes (compiled code) whenever it is called.

## Functions

A function is compiled and executed every time whenever it is called. A function must return a value and cannot modify the data received as parameters.

## Basic Differences between Stored Procedure and Function in SQL Server

* 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. What is an index in SQL? What are the different types of indexes in SQL?

Answer :

Indexes are used to speed-up query process in SQL Server, resulting in high performance. They are similar to textbook indexes. In textbooks, if you need to go to a particular chapter, you go to the index, find the page number of the chapter and go directly to that page. Without indexes, the process of finding your desired chapter would have been very slow.

The same applies to indexes in databases. Without indexes, a DBMS has to go through all the records in the table in order to retrieve the desired results. This process is called table-scanning and is extremely slow. On the other hand, if you create indexes, the database goes to that index first and then retrieves the corresponding table records directly.

There are two types of Indexes in SQL Server:

1. Clustered Index :

A clustered index defines the order in which data is physically stored in a table. Table data can be sorted in only way, therefore, there can be only one clustered index per table. In SQL Server, the primary key constraint automatically creates a clustered index on that particular column.

1. Non-Clustered Index :

A non-clustered index doesn’t sort the physical data inside the table. In fact, a non-clustered index is stored at one place and table data is stored in another place. This is similar to a textbook where the book content is located in one place and the index is located in another. This allows for more than one non-clustered index per table.

1. Showcase an example of exception handling in SQL stored procedure.

Answer :

The TRY CATCH construct allows you to gracefully handle exceptions in SQL Server. To use the TRY CATCH construct, you first place a group of Transact-SQL statements that could cause an exception in a BEGIN TRY...END TRY block as follows:

BEGIN TRY

-- statements that may cause exceptions

END TRY

BEGIN CATCH

-- statements that handle exception

END CATCH

eg.:

CREATE PROC usp\_delete\_person(

@person\_id INT

) AS

BEGIN

BEGIN TRY

BEGIN TRANSACTION;

-- delete the person

DELETE FROM sales.persons

WHERE person\_id = @person\_id;

-- if DELETE succeeds, commit the transaction

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

-- report exception

EXEC usp\_report\_error;

-- Test if the transaction is uncommittable.

IF (XACT\_STATE()) = -1

BEGIN

PRINT N'The transaction is in an uncommittable state.' +

'Rolling back transaction.'

ROLLBACK TRANSACTION;

END;

-- Test if the transaction is committable.

IF (XACT\_STATE()) = 1

BEGIN

PRINT N'The transaction is committable.' +

'Committing transaction.'

COMMIT TRANSACTION;

END;

END CATCH

END;

GO

1. Create a SQL function to split strings into rows on a given character?

Input String: Stephen;peter;berry;Olivier;caroline;

| Stephen |
| --- |
| Peter |
| Berry |
| Oliver |
| Caroline |

CREATE FUNCTION SplitString

(

@Input NVARCHAR(MAX),

@Character CHAR(1)

)

RETURNS @Output TABLE (

Item NVARCHAR(1000)

)

AS

BEGIN

DECLARE @StartIndex INT, @EndIndex INT

SET @StartIndex = 1

IF SUBSTRING(@Input, LEN(@Input) - 1, LEN(@Input)) <> @Character

BEGIN

SET @Input = @Input + @Character

END

WHILE CHARINDEX(@Character, @Input) > 0

BEGIN

SET @EndIndex = CHARINDEX(@Character, @Input)

INSERT INTO @Output(Item)

SELECT SUBSTRING(@Input, @StartIndex, @EndIndex - 1)

SET @Input = SUBSTRING(@Input, @EndIndex + 1, LEN(@Input))

END

RETURN

END

GO

1. What is a temporary and a variable table? Write suitable syntax to create temporary tables and variable tables.

Answer :

**Temporary tables are visible in the created routine and also in the child routines.** **Whereas, Table variables are only visible in the created routine**. Temporary tables are allowed CREATE INDEXes whereas, Table variables aren't allowed CREATE INDEX instead they can have index by using Primary Key or Unique Constraint .

Temporary table ; CREATE TABLE #EmpDetails (id INT, name VARCHAR(25))

Variable Table : DECLARE @LOCAL\_TABLEVARIABLE TABLE

(column\_1 DATATYPE,

column\_2 DATATYPE,

column\_N DATATYPE

)