


Part 00

hacker rank sql Advanced



LinkedIn Article About different between Views, function, Stored Procedure



Mahmoud Akl
Software Engineer | Backend Developer(ASP.NET)
10th of Ramadan, Sharkia

+ Experience

Network smarter with Premium
Try for EGP0

Profile viewers 65
Post impressions 334

View

عبارة عن استعلام SELECT محفوظ، يتعامل معاه كأنه جدول افتراضي. مش بيخزن بيانات، لكن بيعرضها من جداول ثانية. مفيد في تبسيط الاستعلامات المعقدة أو إخفاء أعمدة معينة. مش بيقبل Parameters.

```
CREATE VIEW vw_Student
AS
select * from Student
```

Function (UDF)

يبرجع قيمة واحدة (Scalar) أو جدول (Table-valued). لازم يبرجع نتيجة، ويمكن تستخدمه جوه SELECT أو WHERE. بيقبل Parameters. مفهوش عمليات تعديل بيانات (يعني READ-ONLY).

```
CREATE FUNCTION fn_GetStudentScores (@StudentID INT)
RETURNS TABLE
AS
RETURN
SELECT ExamID, Score
FROM StudentQuestionExam
WHERE StudentID = @StudentID
;
```

Stored Procedure

أكثر مرونة. ممكن يعمل كل أنواع العمليات: SELECT, INSERT, UPDATE, DELETE. بيقبل Parameters، ويمكن يبرجع نتائج أو يطبع رسائل. فيه Error Handling (TRY...CATCH). مناسب جدًا للعمليات الكبيرة أو المتكررة.

```
CREATE PROCEDURE sp_ReplaceEmployeeInProject
,OldEmpID INT@
,NewEmpID INT@
,ProjectID INT@
AS
BEGIN
UPDATE Works_for
SET ESSn = @NewEmpID
```

Part 02 (General)

1. What's the difference between full, differential and transactional back up

Type	Description	Size	Backup Time
FULL Backup	A complete backup of the entire database .	Largest	Longest
DIFFERENTIAL Backup	Backs up only the changes made since the last FULL backup .	Medium	Faster than Full
TRANSACTION LOG Backup	Backs up all transaction log records since the last backup (Full or Log).	Smallest	Fastest

- ❖ Full: للحماية الكاملة.
- ❖ Differential: لتقليل الوقت والحجم بين الـ Fulls.
- ❖ Transaction Log: لنقطة زمنية معينة DB لتسجيل كل الخطوات بدقة — يساعد على استرجاع.

2. What is permission and What's the difference between grant and deny and used on what level

Permission:

A right assigned to a user or role that allows them to perform actions (like SELECT, INSERT, EXECUTE, etc.).

Difference:

Command	Meaning
GRANT	Allows a user to perform an action.
DENY	Explicitly blocks a permission, even if GRANT was given elsewhere.

⚠ DENY overrides GRANT.

Permission Levels:

- **Server Level** → e.g., CREATE DATABASE
- **Database Level** → e.g., SELECT, UPDATE on a whole DB
- **Object Level** → e.g., permissions on a Table, View, or Stored Procedure

3. What's sql profiler and when using it

SQL Server Profiler:

A monitoring and troubleshooting tool that tracks all events in SQL Server, such as:

- Executed queries
- Performance bottlenecks
- Blocking and deadlocks
- Login/logout activity
- Application query behavior

When to use it?

- Performance tuning (slow queries)
- Detecting problematic stored procedures or queries

- Auditing and debugging application behavior
- Capturing and analyzing SQL activity

4. What is trigger and why use it and on what level and what makes it different from normal Stord procedure

Trigger:

A database object that **automatically executes** when a specific event occurs on a table or view (like `INSERT`, `UPDATE`, or `DELETE`).

Why use Triggers?

- Enforce **business rules** automatically
- Audit data changes
- Prevent undesired operations
- Maintain **data integrity**

On what level is a trigger used?

- **Table-level only** — tied directly to actions on a table (or sometimes a view).

Aspect	Trigger	Stored Procedure
Execution	Automatically (on <code>INSERT/UPDATE/DELETE</code>)	Manually (called via <code>EXEC</code> or from code)
Scope	Table/View only	Can be used for any logical operations
Use Case	Enforce rules, audit, prevent operations	Encapsulate reusable logic

Self-Study

1)What is the Procedure Cache?

The **Procedure Cache** is a memory area (in RAM) where SQL Server stores **execution plans** for queries and stored procedures that have been run before.

Purpose:

- **Improves performance** by reusing execution plans.
- Avoids the need to re-parse, re-compile, and re-optimize queries every time they run

Cold Cache

What is a Cold Cache?

A **Cold Cache** means:

- The procedure cache is **empty or cleared**.
- SQL Server **has no stored execution plans**.
- Every query runs as if it's being seen **for the first time** — causing extra overhead.

Causes of Cold Cache:

- SQL Server restart
- Manual cache clearing using:
`DBCC FREEPROCCACHE;`
- Memory pressure forcing SQL Server to evict old plans

Example:

```
SELECT * FROM Students WHERE StudentID = 5;
```

First time (Cold Cache):

- SQL Server:
 - Parses the query
 - Compiles it
 - Generates an execution plan
 - Executes it
- This takes longer than usual.

Second time (Warm Cache):

- SQL Server finds the execution plan in cache and **reuses it**.
- Result: **Much faster** execution.

Why test on Cold Cache?

When testing **performance** of queries or stored procedures, developers sometimes **clear the cache** to see how a query performs from scratch.

```
DBCC FREEPROCCACHE;
```

This simulates a **cold cache** scenario.

2) Number indicates SP behavior" — What does it mean?

This refers to a situation where the **value (number)** passed to a **stored procedure** affects how SQL Server **builds and executes the query plan** for that procedure.

This is mainly due to a concept in SQL Server called **Parameter Sniffing**.

What is Parameter Sniffing?

When a stored procedure is executed for the first time, SQL Server uses the parameter value(s) passed in that first call to **generate an execution plan**. That execution plan is then **cached and reused** for subsequent executions.

If the first parameter value is **unusual or not representative**, the cached execution plan may not be optimal for other values, which can cause **performance issues**.

Example:

Suppose you have this stored procedure:

```
CREATE PROCEDURE GetStudentByID
    @StudentID INT
AS
BEGIN
    SELECT * FROM Students WHERE StudentID = @StudentID
END
```

Now, if you run:

```
EXEC GetStudentByID 1
```

SQL Server builds a plan assuming you're fetching a single row.

But if you run:

```
EXEC GetStudentByID 99999
```

And that value returns a large number of rows or has different performance characteristics, the original plan built for value 1 might perform poorly.

3) Parameter في Stored Procedure أولاً: يعني إيه

Parameter : يعني متغير (variable) بتبعتله قيمة من بره الـ procedure علشان يشتغل بيها

Input Parameter

ده النوع العادي، اللي بنبعته للـ Procedure علشان يستخدمه جوا الاستعلام

```
CREATE PROCEDURE GetStudent
    @StudentID INT
AS
BEGIN
    SELECT * FROM Students WHERE StudentID = @StudentID
END
لما تنفذ Input Parameter. هو @StudentID هنا
EXEC GetStudent 1
```

هيوصل القيمة (1) للـ procedure ويجب بيانات الطالب.

Output Parameter

ده بيستخدم علشان يرجعك قيمة من جوا الـ **procedure** ، زي مثلاً اسم الطالب أو مجموع درجاته.
مثال:

```
CREATE PROCEDURE GetStudentName
    @StudentID INT,
    @StudentName NVARCHAR(100) OUTPUT
AS
BEGIN
    SELECT @StudentName = FirstName FROM Students WHERE StudentID = @StudentID
END
```

```
DECLARE @Name
NVARCHAR(100)

EXEC GetStudentName 1, @Name OUTPUT
PRINT @Name
```

Merits of Using Dynamic Query in a Stored Procedure?

Advantage	Description
✓ High flexibility	You can change the SQL logic at runtime.
✓ Dynamic WHERE conditions	Easily add filters based on input.
✓ Support for complex logic	You can build very flexible and conditional queries.
✓ Good for reporting systems	Users can select columns/filters dynamically.

Demerits of Using Dynamic Query in a Stored Procedure?

Disadvantage	Description
SQL Injection risk	If values are concatenated directly, it can be dangerous.
Harder to read and maintain	Strings are harder to debug than regular SQL.
No IntelliSense or validation	SQL Server Management Studio (SSMS) can't validate the query inside the string.
No plan reuse / caching	Execution plan may not be cached properly, affecting performance.
Potentially slower	Especially when the query is very dynamic and executed frequently.

```
DECLARE @sql NVARCHAR(MAX)
DECLARE @TrackName NVARCHAR(100) = 'Full Stack'

SET @sql = 'SELECT * FROM Students WHERE Track = @TName'

EXEC sp_executesql @sql, N'@TName NVARCHAR(100)', @TName = @TrackName
```

Passing Parameters in Stored Procedures?

```
CREATE PROCEDURE GetOrders
    @OrderID INT,
    @CustomerName NVARCHAR(100)
AS
BEGIN
    SELECT * FROM Orders
    WHERE OrderID = @OrderID AND CustomerName = @CustomerName
END
```

EXEC GetOrders @OrderID = 101, @CustomerName = 'Ali'

Using Default Values for Parameters?

```
CREATE PROCEDURE GetOrdersWithDefaults
    @OrderID INT = NULL,
    @CustomerName NVARCHAR(100) = 'Guest'
AS
BEGIN
    SELECT * FROM Orders
    WHERE
        (@OrderID IS NULL OR OrderID = @OrderID) AND
        CustomerName = @CustomerName
END
```

EXEC GetOrdersWithDefaults @OrderID = 105, @CustomerName = 'Ahmed'

EXEC GetOrdersWithDefaults

