Stored Procedures

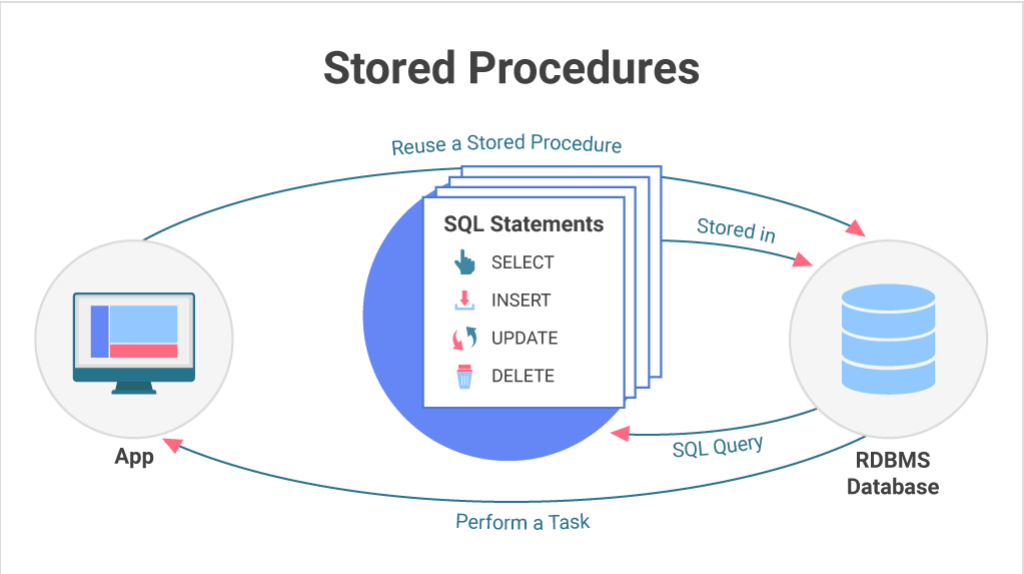
For
Fatma Al-Mamari

By
Samir, Is'haq, Mohammed Al-Khusaibi

TABLE OF CONTENTS

ABOUT

Stored Procedures



Stored Procedures are precompiled SQL statements stored within a database, designed to execute as a single unit. They serve as a powerful DBMS feature, enabling developers to encapsulate SQL queries and business logic. By accepting input parameters and returning output, stored procedures act as reusable modules that can be called by users, applications, or other procedures for efficient database operations.

TOO COMPLICATED I KNOW ! LETS SIMPLIFY IT THEN .. 😊

Simplified Explanation:

Imagine you're at a fancy restaurant. Instead of ordering from the menu each time, there's a big button labeled "EXEC". Once you press it, the automated chef in the kitchen receives parameters like:

@Meal = 'Burger', @Drink = 'Cola'

And starts working quickly:

- 🍔 Prepares the burger
- 🥤 Pours the cola
- 🍽️ Places them on a plate
- 🚶‍♂️ The waiter brings it to you, ready, the same way every time!

This is exactly what a Stored Procedure does:

- You provide parameters (ingredients)
- Press EXEC
- You get the same result, quickly, without repeating the details

👨‍🍳 Chef = SQL Server
🍔 Recipe = Stored Procedure
🧾 Parameters = Defined by you
🎯 Output = Ready-made query result in seconds

📌 شرح مبسّط لمفهوم Stored Procedure

تخيل إنك داخل مطعم فخم، وكل مرة تطلب نفس الطلب:

- بدال ما تعيد الكلام كل مرة، في زر كبير مكتوب عليه EXEC 🧿

بمجرد ما تضغطه:

🍔 الشيف الآلي (SQL Server) يستقبل مدخلاتك:

@Meal = 'Burger', @Drink = 'Cola'

ويبدأ يشتغل بسرعة:

- 🍔 يسوّي البرجر
- 🥤 يصب الكولا
- 🍽️ يحطها في صحن
- 🚻 يجيك النادل يقدّمها لك بنفس الطريقة كل مرة!

💡 طيب، هذا إيش يعني برمجياً؟

- 👨‍🍳 الشيف = SQL Server
- 📋 الوصفة = Stored Procedure (وصفة محفوظة مسبقًا)
- 🧾 الباراميترات = مدخلاتك (Meal, Drink)
- 🎯 الناتج = استعلام جاهز ينفذ بسرعة وكفاءة

✅ النتيجة:

- ما تحتاج تكرر الكود كل مرة
- الإخراج دائمًا ثابت وسريع
- يوفّر وقتك ومجهودك

Why Use Stored Procedures?

Stored procedures offer several key benefits in database management

---

1- Improved performance

- Precompiled execution reduces parsing and optimization overhead.
- Minimizes network traffic by bunding multiple SQL operations into single call.

2- Reusability And Modularity

- Encapsulate complex logic into a single callable unit.
- Avoid code duplication by centralizing business rules in database.

3- secure

- limit direct table access by granting permissions to the procedure instead of tables.

Limit stored procedures

1- Hard to debug

- fewer tools for testing and fixing errors compared to application code.

2- Version control issues

- Difficult to track changes and sync with application code

3- Maintenance problems

- Business logic split between app code and procedures makes updates harder

Types of SQL Stored Procedures

SQL stored procedures are categorized into different types based on their use case and functionality understanding these categories can help developers choose the right of procedure for specific scenario.

1- System Stored Procedures

- These are predefined stored procedures provided by the SQL server for performing administrative tasks such as database management, troubleshooting or system configuration.
- **Examples:**
  - **SP_help:** viewing database object information.
  - **SP_rename:** for renaming database object.

2- User Defined Stored procedures (UDPs)

- These are custom stored procedures created by the user to perform specific operations.
- UDPs can be tailored to business's needs such as calculation totals, processing orders.
- **Example**
  - Creating a procedure that calculates the total sales for a particular product category.

3- Extended Stored Procedures

- These allow for the execution of external functions which might be implemented in other languages such C, C++. Extended procedures provide a bridge between SQL.

4- CLR Stored procedures

- These are stored procedures written in .NET languages (like C#) and executed within SQL Server. CLR stored procedures are useful when advanced functionality is needed that isn't easily achievable with T-SQL alone, such as complex string

SQL Stored Procedure Syntax

The syntax you've provided is for creating a stored procedure in SQL. Here's a breakdown of the components:

```
CREATE PROCEDURE procedure_name
(parameter1  data_type,  parameter2  data_type,
...)
AS
BEGIN
  -- SQL statements to be executed
END
```

1- CREATE PROCEDURE:

- The command to define a new stored procedure

2-procedure_name:

A- Listed in parentheses.

B- Each has a name and data type.

C- Can be input or output parameters (depending on the SQL dialect).

4-AS:

- Keyword that separates the header from the body.

5-BEGIN...END:

- The block that contains the procedure's SQL statements

🎬 Project Scenario: AnimeRed – Anime Catalog and Rating System

Scenario Description:

With the growing popularity of anime worldwide, many fans seek a platform where they can discover new series, track what they've watched, and share ratings or reviews. AnimeRed is designed to serve as an internal management system and public-facing platform for anime enthusiasts. It maintains a structured anime catalog and allows users to rate and explore content based on genres, popularity, and community feedback.

Objective:

A To store anime data such as title, genre, episode count, and average rating.

B To allow administrators to manage anime records efficiently via stored procedures.

C To allow users to interact with data — rate anime, view top-rated shows, etc.

D To provide backend reports using SQL loops and conditions (e.g., anime above a certain rating).

The steps below show how to build anime app databases by applying stored procedures.

Step 1: Start with creating a table

```
CREATE TABLE Anime (
 AnimeID INT PRIMARY KEY IDENTITY(1,1),
 Title NVARCHAR(100),
 Genre NVARCHAR(50),
 Episodes INT,
 Rating FLOAT
);
```

Step 2: Insert sample data to table

```
INSERT INTO Anime (Title, Genre, Episodes, Rating)
VALUES
('Attack on Titan', 'Action', 75, 9.2),
('One Piece', 'Adventure', 1000, 9.0),
('Death Note', 'Thriller', 37, 9.1),
('Naruto', 'Action', 220, 8.3),
('Fullmetal Alchemist: Brotherhood', 'Fantasy', 64, 9.1),
('Demon Slayer', 'Action', 26, 8.7),
('My Hero Academia', 'Superhero', 113, 8.4),
('Jujutsu Kaisen', 'Action', 48, 8.8),
('Spy x Family', 'Comedy', 25, 8.6),
('Chainsaw Man', 'Horror', 12, 8.5);
```

Step 3: Create a stored procedure add new anime

```
CREATE PROCEDURE AddAnime
 @Title NVARCHAR(100),
 @Genre NVARCHAR(50),
 @Episodes INT,
 @Rating FLOAT
AS
BEGIN
 INSERT INTO Anime (Title, Genre, Episodes, Rating)
 VALUES (@Title, @Genre, @Episodes, @Rating);
END;
```

Step 4: Execute it

```
EXEC AddAnime
 @Title = 'Attack on Titan',
 @Genre = 'Action',
 @Episodes = 75,
 @Rating = 9.2;
```

Step 5: Alter the Procedure: Add a Print Statement

```
ALTER PROCEDURE AddAnime
 @Title NVARCHAR(100),
 @Genre NVARCHAR(50),
 @Episodes INT,
 @Rating FLOAT
AS
BEGIN
 INSERT INTO Anime (Title, Genre, Episodes, Rating)
 VALUES (@Title, @Genre, @Episodes, @Rating);

 PRINT 'Anime successfully added!';
END;
```

Step 6: Drop the procedure

```
DROP PROCEDURE AddAnime;
```

Step 7: Use output parameter return newly inserted AnimeID

```
CREATE PROCEDURE AddAnimeWithOutput
 @Title NVARCHAR(100),
 @Genre NVARCHAR(50),
 @Episodes INT,
 @Rating FLOAT,
 @NewAnimeID INT OUTPUT
AS
BEGIN
 INSERT INTO Anime (Title, Genre, Episodes, Rating)
 VALUES (@Title, @Genre, @Episodes, @Rating);

 SET @NewAnimeID = SCOPE_IDENTITY();
END;
```

Step 8: Execute with output

```
DECLARE @ID INT;
EXEC AddAnimeWithOutput
 @Title = 'One Piece',
 @Genre = 'Adventure',
 @Episodes = 1000,
 @Rating = 9.0,
 @NewAnimeID = @ID OUTPUT;
PRINT 'New Anime ID: ' + CAST(@ID AS NVARCHAR);
```

Step 9: Stored procedure with conditions, variables and loop

```
CREATE PROCEDURE PrintTopRatedAnime
 @MinRating FLOAT
AS
BEGIN
 DECLARE @AnimeTitle NVARCHAR(100);
 DECLARE @CurrentID INT = 1;
 DECLARE @MaxID INT;

 SELECT @MaxID = MAX(AnimeID) FROM Anime;
 WHILE @CurrentID <= @MaxID
 BEGIN
 IF EXISTS (
 SELECT 1 FROM Anime
 WHERE AnimeID = @CurrentID AND Rating >= @MinRating
 )
 BEGIN
 SELECT @AnimeTitle = Title
 FROM Anime
 WHERE AnimeID = @CurrentID;

 PRINT 'Top-rated Anime: ' + @AnimeTitle;
 END;
```

```
 SET @CurrentID += 1;

 END

END;
```

Step 10: Execute it

```
EXEC PrintTopRatedAnime @MinRating = 8.5;
```

AnimeRed showcases how SQL Server can be used to build a real-world anime catalog system. It uses stored procedures, input/output parameters, and control structures to manage data efficiently. The project reflects real-life apps like MyAnimeList and demonstrates key database concepts in a practical way, making it ideal for learning and future expansion.
