**Aufruf von Stored Procedures, Views, Table- und Scalar-Funktionen in EF Core**

**1. Was lernen Sie hier**

* die Unterschiede zwischen **Stored Procedures, Views, Table-Valued Functions und Scalar Functions** verstehen,
* wissen, wie sich diese Objekte auf Performance, Sicherheit und Wartbarkeit auswirken,
* den **richtigen Einsatz** im Entity Framework Core erkennen,
* die Objekte in EF Core korrekt aufrufen und Ergebnisse verarbeiten können.

**2. Einführung: SQL-Objekte und deren Rollen**

**a) Stored Procedures (SP)**

* **Definition**: Vorgefertigte SQL-Befehle (SELECT, INSERT, UPDATE, DELETE) im Datenbankserver.
* **Vorteile**:
  + Logik direkt auf dem Server, reduziert Netzwerklast.
  + Wiederverwendbar in verschiedenen Anwendungen.
  + Sicherheit: Berechtigungen können auf SP gesetzt werden, Tabellen müssen nicht direkt zugänglich sein.
* **Nachteile**:
  + Komplexität steigt bei vielen SPs.
  + Debugging schwieriger als bei C#-Code.
* **EF Core**:
  + FromSqlRaw für Abfragen, die Ergebnisse zurückliefern.
  + ExecuteSqlRaw für Prozeduren, die keine Ergebnisse liefern.

**b) Views**

* **Definition**: Virtuelle Tabellen, die aus einer SELECT-Abfrage entstehen.
* **Vorteile**:
  + Vereinfachen komplexe Abfragen.
  + Einheitliche Sicht auf Daten für verschiedene Anwendungen.
  + Können gefilterte, aggregierte oder join-basierte Daten darstellen.
* **Nachteile**:
  + Standardmäßig schreibgeschützt (manche Views können mit INSTEAD OF Triggern updatable sein).
* **EF Core**:
  + Views können wie DbSet<T> eingebunden werden.
  + Entitäten müssen als [Keyless] markiert werden, falls kein Primärschlüssel existiert.

**c) Table-Valued Functions (TVF)**

* **Definition**: Funktionen, die eine Tabelle zurückliefern.
* **Vorteile**:
  + Modularisierung von Abfragen.
  + Wiederverwendbar in verschiedenen SQL-Abfragen.
  + Kombinierbar mit anderen SQL-Objekten (JOINs, WHERE).
* **Nachteile**:
  + Keine Datenänderungen – nur SELECT.
  + Performance hängt von Implementierung ab (Inline TVFs sind schneller als Multi-Statement TVFs).
* **EF Core**:
  + TVFs können über modelBuilder.HasDbFunction() registriert werden.
  + Rückgabe ist ein IQueryable<T> – LINQ kann darauf angewendet werden.

**d) Scalar Functions**

* **Definition**: Funktionen, die genau **einen Wert** zurückgeben (z. B. COUNT, SUM, benutzerdefinierte Berechnungen).
* **Vorteile**:
  + Kapseln Logik für Berechnungen.
  + Können direkt in SELECT-Statements oder WHERE-Klauseln verwendet werden.
* **Nachteile**:
  + Bei großen Tabellen kann die Performance leiden, besonders bei Row-by-Row-Aufrufen.
* **EF Core**:
  + Scalar Functions lassen sich ebenfalls über HasDbFunction registrieren.
  + Können in LINQ-Ausdrücken eingebunden werden.

**3. Vergleich der SQL-Objekte**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Objekttyp | Rückgabe | Schreibbar | EF Core-Aufruf | Typischer Einsatz |
| Stored Procedure | Tabelle / Zeilenanzahl | ggf. | FromSqlRaw / ExecuteSqlRaw | Insert/Update/Delete/ komplexe Logik |
| View | Tabelle | Nein | DbSet<T> | Komplexe Abfragen, aggregierte Sichten |
| Table-Valued Func | Tabelle | Nein | HasDbFunction | Wiederverwendbare SELECT-Logik |
| Scalar Function | Einzelwert | Nein | HasDbFunction | Berechnungen, Zähler, Aggregationen |

**4. Praktische Hinweise für EF Core**

1. **Stored Procedures**:
   * Nutzen Sie FromSqlRaw für Abfragen, ExecuteSqlRaw für Änderungen.
   * Parameter immer absichern (@p0, @p1) → SQL Injection vermeiden.
2. **Views**:
   * [Keyless]-Attribute setzen, wenn kein Primärschlüssel existiert.
   * Keine Änderungen per EF Core möglich, außer Trigger/INSTEAD OF vorhanden.
3. **TVF**:
   * Inline TVFs sind effizienter als Multi-Statement TVFs.
   * Rückgabe in IQueryable<T> → LINQ kann gefiltert, sortiert, gruppiert werden.
4. **Scalar Functions**:
   * Können in LINQ-Ausdrücken wie reguläre C#-Methoden verwendet werden.
   * Nicht jede SQL-Funktion kann direkt in EF Core abgebildet werden – Registrierung in OnModelCreating nötig.

**5. Best Practices**

* Verwenden Sie **Stored Procedures** für Schreiboperationen, komplexe Business-Logik und Sicherheitszwecke.
* Verwenden Sie **Views** und **TVFs** für Abfragen, die oft wiederverwendet werden.
* Verwenden Sie **Scalar Functions** für Berechnungen und Aggregationen, die in Abfragen mehrfach gebraucht werden.
* Immer auf **Performance** achten: Inline TVFs + Indizierung der Tabellen verbessern die Abfragegeschwindigkeit.

**Und die Erstellung per EFC?**

In **Entity Framework Core** (EFC) gibt es für **Tabellen und Spalten** ein klares Modell, aber **Stored Procedures, Views, Funktionen oder Trigger** gehören nicht zum klassischen Entity-Modell.  
Trotzdem kannst du sie über **Migrationen** mit anlegen (und ggf. auch wieder entfernen).

**Vorgehensweise: Migrationen für SPs, Views & Co.**

1. **Migration erstellen**
2. dotnet ef migrations add CreateStoredProceduresAndViews
3. **Migration anpassen**  
   In der generierten Migrationsklasse kannst du im Up- und Down-Bereich eigene SQL-Kommandos einfügen.

Beispiel: Stored Procedure

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.Sql(@"

CREATE PROCEDURE GetKundenMitBestellungen

AS

BEGIN

SELECT k.Id, k.Name, COUNT(b.Id) as AnzahlBestellungen

FROM Kunden k

LEFT JOIN Bestellungen b ON k.Id = b.KundeId

GROUP BY k.Id, k.Name

END

");

}

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.Sql("DROP PROCEDURE GetKundenMitBestellungen");

}

Beispiel: View

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.Sql(@"

CREATE VIEW KundenUebersicht AS

SELECT k.Id, k.Name, COUNT(b.Id) as AnzahlBestellungen

FROM Kunden k

LEFT JOIN Bestellungen b ON k.Id = b.KundeId

GROUP BY k.Id, k.Name

");

}

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.Sql("DROP VIEW KundenUebersicht");

}

Du musst hier **manuell SQL schreiben**. EFC selbst erzeugt keine SPs oder Views automatisch.

1. **Migration anwenden**
2. dotnet ef database update