



Jörg Böke

# Schnelleinstieg in SQLScript für SAP HANA®

- ▶ Einführung in SAP HANA SQLScript
- ▶ SQLScript in ABAP Core Data Services (CDS) für Reportingzwecke
- ▶ Anwendungsbeispiele von SQLScript in Prozeduren, Funktionen und BW/4HANA
- ▶ Tipps und Tricks für den performanten Einsatz von SQLScript

# Inhaltsverzeichnis

<b>Vorwort</b>	<b>7</b>
<b>1 Einführung in SAP HANA SQL und SQLScript</b>	<b>11</b>
1.1 Geschichte von SQL	11
1.2 Definition von SQL	12
1.3 Definition von SQLScript	12
1.4 SAP ABAP oder SQLScript	16
1.5 SQL/SQLScript – Basiswissen	18
1.6 SQL-Objekttypen	21
1.7 HANA-Schema	23
1.8 HANA-Unload-Priorität	24
1.9 HANA Delta Merge	26
1.10 Namenskonventionen SQLScript	26
1.11 Erste Tabellen	38
1.12 Erste Daten	42
1.13 Testdatentool der SAP	48
1.14 SQLScript-Kommandos	49
<b>2 Anwendungsbeispiele in HANA</b>	<b>97</b>
2.1 HANA-SQLScript-Funktionen	97
2.2 HANA-SQLScript-Prozeduren (Procedures)	110
2.3 HANA Calculation Views und Table Functions	125
2.4 Core Data Services in HANA und ABAP	143
<b>3 Debugging von SQLScript</b>	<b>165</b>
3.1 Anlegen einer Demoprozedur für einen Debug-Test	165
3.2 Aufruf und Einstellung des Debugging	168
3.3 Debug-Prozess in Eclipse	170

<b>4</b>	<b>Monitoring</b>	<b>173</b>
4.1	System Views und Tabellen in SAP HANA	173
<b>5</b>	<b>Performanceaspekte von SQLScript</b>	<b>181</b>
5.1	Analyse von SQLScript mithilfe von PlanViz	182
5.2	Parallel Execution	183
5.3	Abfragen einschränken	187
5.4	Vermeidung von Schleifen	191
5.5	Dynamisches SQL	195
<b>6</b>	<b>Fazit</b>	<b>196</b>
<b>A</b>	<b>Der Autor</b>	<b>198</b>
<b>B</b>	<b>Index</b>	<b>199</b>
<b>C</b>	<b>Disclaimer</b>	<b>201</b>

## 2 Anwendungsbeispiele in HANA

**In diesem Kapitel werde ich auf die Verwendung der Funktionalitäten von SQLScript in den verschiedenen Bereichen eines SAP-Systems eingehen.**

Die Nutzung von Funktionen und Prozeduren ermöglicht es Ihnen, Ihren Code lesbar zu halten, indem Sie die verschiedenen Aufgaben, die ausgeführt werden sollen, nicht in einer einzigen Prozedur mit beispielsweise 500 Zeilen Script erstellen, sondern diese in mehrere Prozeduren und Funktionen aufteilen.

Eine Übersicht der HANA-SQLScript-Referenz erhalten Sie unter:

*<https://help.sap.com/viewer/4fe29514fd584807ac9f2a04f6754767/2.0.05/en-US/20ff532c751910148657c32fe3431a9f.html>*

Im Folgenden werden Sie zusätzlich zu den Funktionen, die die SAP mit Auslieferung der HANA DB bereitstellt, eigene Funktionen und Prozeduren anlegen. Diese Erweiterung des SQLScript-Standards ermöglicht Ihnen, die Funktionalität immer weiter auszubauen und Funktionen genau auf Ihre individuellen Bedürfnisse anzupassen.

### 2.1 HANA-SQLScript-Funktionen

Funktionen (Functions) dienen in HANA SQLScript der einmaligen Definition einer immer wiederkehrenden Funktionalität, um diese dann an anderer Stelle ebenfalls zu nutzen. Falls später eine Anpassung am Funktionscode notwendig wird, muss nur eine zentrale Funktion anstatt einer großen Anzahl von Stellen geändert werden. Dadurch kann der Code wiederverwendbar und lesbar gehalten werden. Die SAP liefert in SQLScript eine Vielzahl an Funktionen in Abhängigkeit von der jeweiligen HANA-DB-Version aus.

Nachfolgend liste ich einige der *SAP HANA Functions* auf, die Sie später noch nutzen werden bzw. schon in Abschnitt 1.14.1 kennengelernt haben.

- ▶ UCASE (identisch mit UPPER-Funktion): Der Befehl dient dazu, alles in Großbuchstaben umzusetzen.
- ▶ UPPER: Der Befehl dient dazu, alles in Großbuchstaben umzusetzen.
- ▶ RIGHT: Dieser Befehl gibt ab dem rechten String-Ende nach links eine Anzahl von Zeichen aus.
- ▶ LEFT: Dieser Befehl gibt vom linken String-Anfang nach rechts eine Anzahl von Zeichen aus.

Für Funktionen besteht in HANA 1.0 keine Möglichkeit, rekursiv eine Hierarchie der Tabelleninhalte (z. B. Kunden) aufzubauen; ab der Version HANA 2.0 kann dies durch SAP umgesetzt werden. Es lohnt sich daher, die Funktionsmöglichkeiten der jeweiligen Versionen von HANA SQLScript kennenzulernen, um das eigene Wissen auf dem aktuellen Stand zu halten.

In den folgenden Beispielen möchte ich Ihnen zeigen, wie Sie eigene Funktionen erstellen und nutzen können. Um SQLScript übersichtlich zu halten, ist es sehr sinnvoll, Teilaufgaben in entsprechende Funktionen auszulagern.

## 2.1.1 SQLScript-Funktionsdeklaration

SQLScript-Funktionen bestehen aus einem Kopf, der definiert, in welchem Schema die Funktion erzeugt werden soll, dem Namen der Funktion sowie den Input-Parametern.

Bei Funktionen wird der Rückgabewert über RETURNS übergeben. Dieser kann je nach Deklaration eine einzelne Variable oder Tabelleninhalt sein.

Im nächsten Teil, der standardmäßig deklariert werden sollte (siehe Listing 2.1), ist die Sprache (in diesem Fall `SQLScript`) definiert sowie, mit welcher Berechtigung `SQL SECURITY INVOKER` (Developer, Invoker – also Entwickler oder zum Zeitpunkt der Ausführung gültige Anwender/ User) die Funktion ausgeführt wird.

Im weiteren Verlauf kann mit dem Code `READS SQL DATA AS` (Prozedur darf nur lesenden Zugriff auf die DB haben) oder `AS` (Funktion darf auch Daten verändern) definiert werden, ob eine Prozedur intern lesend oder schreibend arbeitet.

Ab der Zeile `BEGIN` folgt das von Ihnen manuell erstellte Coding wie der Aufruf von Funktionen, `SELECT` Statements etc. Mit `END` wird die Funktion anschließend beendet.

Jeglicher Kommentar sollte mit zwei Minuszeichen `--` beginnen, damit dieser nicht als `SQLScript` interpretiert werden kann.

```
--Funktion Rumpfprogramm
CREATE FUNCTION MEIN_SCHEMA.GET_MY_NAME (
    IN ip_MEIN_INPUT NVARCHAR(50) ,
)
    RETURNS op_MEINOUTPUT NVARCHAR(256)
    LANGUAGE sqlscript
    SQL SECURITY INVOKER
    READ SQL DATA AS - oder nur AS

BEGIN
--Hier steht Ihr Coding

END;
```

*Listing 2.1: Beispiel für den Funktionskopf/Deklaration*

## 2.1.2 Nested Functions

Eine *Nested Function* ist eine Funktion, die intern eine andere Funktion aufruft. Wie bereits in der Einleitung beschrieben, kann dies dazu

dienen, SQLScript lesbarer zu halten, da z. B. anstelle von 100 Zeilen nur zwei bis drei Zeilen benötigt werden.

Anhand der Berechnung der Anzahl von Tagen zwischen zwei Daten wird dies im folgenden Beispiel veranschaulicht (siehe Listing 2.2).

Als Erstes müssen Sie die Funktion erstellen, die Sie später in den Routinen benutzen wollen.

```
--Nested function für Aufruf in Prozedur
CREATE FUNCTION get_days(
  IN first_d date, IN second_d date )

  RETURNS num_days bigint
  AS
  BEGIN
    num_days = days_between (:first_d,:second_d );
  END;
```

*Listing 2.2: Beispiel für den Aufruf von Nested Functions*

Wie zu erkennen ist, nutzt diese Funktion wiederum eine SAP-Funktion (DAYS\_BETWEEN), wodurch auch eine mehrstufige Schachtelung ermöglicht wird.

### 2.1.3 Verwendung von User Defined Functions (UDF)

Im Gegensatz zur ABAP-Programmierung, bei der Funktionen nur im Coding genutzt werden können, verwenden Sie in SQLScript SAP-Funktionen oder Ihre eigenen Funktionen (UDF) direkt mit SQL-Kommandos.

Stellen Sie sich vor, Sie möchten aus der Tabelle KUNDE den Namen sowie Vornamen auslesen und beide mit einem bestimmten Trennzeichen (z. B. --) ausgeben. Die gewünschte Ausgabe sollte also wie folgt sein: (Nachname--Vorname) »Böke--Jörg«.

Sie können dies direkt als SQLScript-Abfrage implementieren oder in eine Funktion auslagern, sollte die Anforderung häufiger benötigt werden. Ein kleines Beispiel dazu zeigt Listing 2.3.

In der Funktion, die aus drei Input-Parametern und einem Return-Parameter besteht, wird überprüft, ob innerhalb der IF-Anweisung ein Trennzeichen geliefert wird. Je nach Ergebnis der Anweisung wird dieses mittels PIPE (||)-Funktionalität verkettet und ausgegeben.

```
--Beispiel einer Namens-Funktion
CREATE FUNCTION GET_FULL_NAME (
    IN ip_vorname NVARCHAR(50) ,
    IN ip_trennzeichen NVARCHAR(50) ,
    IN ip_nachname NVARCHAR(50)
)
    RETURNS op_fullname NVARCHAR(256)
LANGUAGE sqlscript
SQL SECURITY INVOKER
AS
BEGIN
    IF :ip_trennzeichen IS NULL THEN
        op_fullname = :ip_nachname || '** ' ||
:ip_vorname;
    ELSE
        op_fullname = :ip_nachname || :ip_
trennzeichen || :ip_vorname ;
    END IF;
END;
```

*Listing 2.3: Beispiel für eine User Defined Function*

Diese Funktion können Sie nun mittels einer SQL-Konsole testen.

```
select GET_FULL_NAME('Jörg','--','Böke') from dummy
```

Bei Übergabe eines Trennzeichens erhalten Sie das in Abbildung 2.1 gezeigte Ergebnis.



# B Index

## A

- ABAP Managed Database  
Procedures 17
- ABDOC\_CDS\_ANNOS 145
- advanced Datastore Object 31
- AMDP 17, 151
- Annotation
  - Field 144
  - Header 144

## B

- BW-Transformationen 72

## C

- Calculation Views 126
- CASE 74
- CAST 74
- CDS-Annotationen 144
- CDS-View 143
- CONCAT 52
- Concat PIPE 53
- Core Data Services 17
- CURRENT\_UTCDATE 46

## D

- DBMS 18
- DDL 38
- Delta Merge 26

## E

- END FOR 66

- ENDWHILE 66
- Exit Handler 115
- Expression 128

## F

- Fehlerbehandlung in Prozedur  
115
- FOR 66
- Funktionen 98

## G

- GROUPING SET 75

## H

- HANA 2.0 87
- HANA Developer Guide 95
- Hierarchien 87

## I

- IF THEN ELSE 69
- Index 67
- Invoker 99

## K

- KBA 25
- Konvertierungsmöglichkeiten 81

## L

- LCASE 55
- LEAST 65
- LEFT 57

LENGTH 74  
LPAD 49

## M

Mainentry  
    CDS 17  
    Schleife 66  
MAP 60  
MAX 82  
MIN 82  
Multi Database Container (MDC)  
    33  
multiple Parameterwerte 138  
MULTIPLE RESULTSETS 80  
Multitenant 33  
MY\_PERIODS\_IN\_BETWEEN  
    123

## N

Nested Function 99

## O

Offset 67  
ORPHAN 91  
OVERVIEW 79  
OVERVIEW PREFIX 79

## P

PlanViz 182  
Prozeduren 110

## R

RANK 83  
rekursive Aufrufe 107  
REPLACE 60

REPLACE\_REGEXPR 64  
RIGHT 57, 74  
RPAD 52

## S

skalare Funktion 107, 130  
SQL  
    dynamisch erstellt 117  
SQLEXCEPTION 119  
STRUCTURED RESULT 76  
SUBSTR\_REGEXPR 61  
Systemlandschaft 113

## T

Table Functions 129, 132  
TVARVC 115

## U

UCASE 55  
UDF 132  
Union 75  
Unload Priority 25  
User Defined Functions (UDF)  
    100

## V

Variablen 127

## W

WHILE 66  
Window Functions 82