Cheat Sheet SAP HANA SQLScript

Die perfekte Ergänzung zu den Schulungen von Brandeis Consulting

Dieser Spickzettel enthält SAP HANA SQLScript Syntax, Beispiele und Beschreibungen, passend zu den Inhalten der **SQLScript Grundlagenschulung** bei Brandeis Consulting. Diese Übersicht zeigt nur die wichtigsten Aspekte. Für Details bitte den Links zur Referenz folgen. Wenn mehrere identische Konzepte (z.B. identische SQL-Funktionen) existieren, wird hier meist nur eines davon beschrieben. Im Text sind nur dann Links, wenn dahinter relevante Zusatzinfos sind. Beispiele beziehen sich immer auf das **englische Demo-Datenmodell** für das Buch.

Notation im Code

- **`[EckigeKlammern]`** – Optionale Elemente
- **`UPPERCASE`** – Schlüsselwörter und SQL-Funktionen
- **`lowercase`** – Bezeichner
- **`<SpitzeKlammer>`** – beschreibende Elemente. Abkürzungen sind im Text erklärt
- **`{Wiederholungen}...`** sind optional und werden in geschweiften Klammern mit 3 Punkten geschrieben.

Unter der Linie sind Referenzen, z.B.

- **Buch S.** – Seitenzahl in der 2.Ausgabe von ***SQLScript für SAP HANA***

Bezeichner

Bezeichner von Tabellen oder Spalten sind Casesensitiv. Das fällt bei den Bezeichnern in **einfacher Notation** ohne Gänsefüsschen aber nicht auf, da diese immer implizit in GROßBUCHSTABEN konvertiert werden. Es gibt hier keinen Unterschied zwischen den Namen **`SpalteX`**, **`SpAlTeX`** und **`spaltex`**. Intern werden alle drei als **`SPALTEX`** interpretiert. Es sind nur ASCII Buchstabe, Ziffern und die Zeichen #_ erlaubt.

Namen in **spezieller Notation** werden in Gänsefüßchen eingeschlossen. Damit werden Sie exakt so interpretiert, wie sie dastehen. Die drei Namen **`"SpalteX"`**, **`"SpAlTeX"`** und **`"spaltex"`** sind unterschiedlich. **Alle Unicode**-Zeichen sind erlaubt.

Buch S. 61

Kommentare

Zeilenendkommentare werden mit zwei Bindestrichen **`--`** eingeleitet und gehen bis zum Zeilenumbruch.

Blockkommentare beginnen mit **`/*`** und enden mit **`*/`**. Sie können innerhalb einer Zeile oder auch über mehrere Zeilen gehen.

ABAP Kommentare mit * an der ersten Position funktionieren nur im AMDP und sollten daher nicht verwendet werden.

Buch S. 59

Skalare Ausdrücke

Liefern genau einen elementaren Wert, z.B. ein Datum, eine Zeichenkette oder eine Zahl.

- Feldnamen
- Operatorausdrücke
- skalare Variablen oder Parameter
- **Literale**
- **CASE-Ausdrücke**
- **skalare Unterabfragen in Klammern**
- **SQL-Funktionen** und **UDF-Funktionen**

Buch S. 69

Literale

Literale sind konstante Werte in SQLScript Code.

Bezeichnung	Format	Beispiel
Einfache Zeichenketten	In Hochkomma	‚Peter'
Unicode Zeichenketten	In Hochkomma, mit einem N als Präfix	N'Jörg'
Ganzzahlen	Ziffernfolge	123
Dezimalzahlen	Ziffernfolge mit Dezimalpunkt	123.456
Gleitkomma-zahlen	Mantisse und Exponent, getrennt durch ein E	1,23E+04
Datum	Präfix DATE	DATE'2017-11-10'
Uhrzeit	Präfix TIME	TIME'15:42:04.123'
Zeitstempel	Präfix TIMESTAMP	TIMESTAMP'2011-12-31 23:59:59'

Buch S. 61

SQL-Funktionen

Die HANA Datenbank hält hunderte von SQL-Funktionen für die Berechnung von Daten vor. Die wichtigsten sind auf dieser Seite bei den jeweiligen Datentypen zu finden:

- **Zeit-Datentypen**
- **Zeichenketten**
- **Numerische Datentypen**

Buch S. 167ff

`CASE`-Ausdrücke

Ein CASE-Ausdruck liefert einen skalaren Wert zurück. Es gibt zwei Varianten von CASE-Ausdrücken.
Der **einfache `CASE`-Ausdruck** vergleicht einen Ausdruck mit mehreren anderen Ausdrücken auf Gleicheit:

```
SELECT id,
       CASE status
         WHEN 1 THEN 'Neu'
         WHEN 2 THEN 'In Bearbeitung'
         ELSE 'Nicht relevant'
       END AS status_text
  FROM tasks;
```

Der **komplexe `CASE`-Ausdruck** wertet N unabhängige **Prädikate** aus. Das erste, das zu **`TRUE`** ausgewertet wird, liefert das Ergebnis:

```
SELECT CASE
         WHEN status NOT IN (1, 2, 3)
           THEN 'Open'
         WHEN due_date < current_date
           THEN 'Critical'
         ELSE 'Others' END AS statustext
  FROM tasks;
```

Wenn kein wahrer Vergleich bzw. Prädikat gefunden wurde, wird entweder der Wert aus der **`ELSE`**-Klausel oder **`NULL`** zurückgegeben.

Buch S. 119

Tabellenausdrücke

Ein Ausdruck, der eine Tabelle zurückgibt. Das können z.B. sein:

- Namen von DB-Tabellen oder Views
- Mit **`JOIN`** verbundene Tabellenausdrücke
- **Tabellenvariablen**
- **Tabellen-Unterabfragen**
- Mit Mengenoperatoren verbundene **SELECT Abfragen**

Buch S. 59

Brandeis Consulting

Schulungen und Beratung vom Autor des Buches SQLScript für SAP HANA. Bei Fragen zu SQLScript und Schulungen einfach eine Mail an info@brandeis.de schicken.

SELECT

Die **`SELECT`**-Anweisung definiert eine Tabelle. Die Spalten werden mit der **Feldliste** erzeugt, während die zugehörigen Zeilen sich aus den anderen Klauseln ergeben.

Syntax:

```
SELECT [TOP number]
  [DISTINCT] Feldliste
  FROM Tabellenausdruck
  [WHERE Prädikat]
  [GROUP BY Ausdrucksliste]
  [Mengenoperation]
  [ORDER BY Ausdrucksliste]
  [LIMIT AnzahlZeilen [OFFSET ZeilenÜberspringen]]
```

Die **FROM-Klausel** beschreibt die Quelle der Daten

```
FROM Tabellenausdruck [[AS] Alias]
```

Gegebenenfalls kommt hier noch eine Verbidung von weiteren Tabellenausdrücken mit einem **`JOIN`** hinzu. Die **`WHERE`-Klausel** filtert die Daten. Nur Datensätze, für die ein **Prädikat** zu **`TRUE`** ausgewertet wird, kommen mit in die Ergebnismenge der Abfrage.

Buch S. 114

Feldliste

Definition der Spalten der Abfrage.

```
Skalarer_Ausdruck1 [[AS] Aliasname1 ] [ { ,
Skalarer_Ausdruck2 [[AS] Aliasname2] } ... ]
```

Mehrere Spalten werden durch Komma getrennt. Alle Spalten der Quellen werden mit * addressiert.

```
SELECT *,
       LEFT(coarea, 2) AS country,
       amount * 1,16   AS net_amount,
       CASE left(coarea,2)
         WHEN 'DE' THEN '1'
         WHEN 'FR' THEN '2'
         ELSE '9'
       END AS ccode,
  FROM Tabellenausdruck
```

Buch S. 116

JOINS

Mit einem **`JOIN`** wird ein Verbund aus mehren Tabellen hergestellt. Die **`ON`**-Bedingung definiert, welche Zeilen aus den beteiligten Tabellen gemeinsam in einer Zeile des Abfrageergebnis stehen.

```
SELECT ...
  FROM Tabellenausdruck1
  Jointyp JOIN Tabellenausdruck2
  ON JoinPrädikat;
```

Der **`CROSS JOIN`** ist der einzige Jointyp ohne **`ON`**-Bedingung. Er bildet das kartesische Produkt zweier Tabellen.

Im Ergebnis des **`INNER JOIN`** sind nur Zeilen, die jeweils auf der anderen Seite einen Partner gefunden haben.

Bei den **`OUTER JOIN`** wird jeweils die andere Seite mit **`NULL`** aufgefüllt, falls kein Partner gefunden wurde.

Buch S. 134

Unterabfragen

Skalare Unterabfragen liefern genau eine Zeile und eine Spalte, z.B. in Feldlisten oder zum Vergleich. Bei mehr als einer Ergebniszeile gibt es einen Laufzeitfehler.

```
SELECT assignee,
       id,
       due_date,
      (SELECT lastname
         FROM users AS u
        WHERE o.assignee = u.id)
       AS assignee_name
  FROM tasks AS o
 WHERE  o.due_date =
          (SELECT MAX(due_date)
             FROM tasks AS i
            WHERE o.assignee = i.assignee )
```

Spalten-Unterabfragen liefern mehrere Werte in genau einer Spalte. Sie werden für den Mengenvergleich mit IN verwendet.

```
SELECT *
FROM tasks
WHERE
   status IN (SELECT id
                FROM status
               WHERE is_final = true)
```

Tabellen-Unterabfragen liefern eine Tabelle, die als **Tabellenausdruck** in der FROM-Klausel verwendet werden kann. Tabellen-Unterabfragen können elegant durch **Tabellenvariablen** ersetzt werden.

```
SELECT category,
       COUNT(*)
  FROM ( SELECT
           CASE
             WHEN status IN (1, 2)
             AND due_date < current_date
                THEN 'Critical'
             WHEN status IN (1, 2)
                THEN 'Open'
             ELSE 'OK' END AS category
         FROM tasks)
 GROUP BY category;
```

Buch S.159ff

Prädikate

Prädikate sind logische Ausdrücke, die entweder den Wert **`TRUE`**, **`FALSE`** oder **`UNKNOWN`** annehmen. In **`WHERE`**-Klauseln, **`ON`**-Klauseln oder bei der Auswertung von Bedingungen in **`CASE`**-Ausdrücken ist immer nur relevant, ob ein Prädikat zu **`TRUE`** ausgewertet wird oder nicht.

Prädikate können mit den logischen Operatoren **`NOT`**, **`AND`** und **`OR`** kombiniert werden. Klammern erhöhen hier die Lesbarkeit enorm!

Die wichtigsten Prädikate sind:

- Vergleiche:
  `<Ausdruck1> <Operator> <Ausdruck2>`
- **`IS NULL`** – Das einzige Prädikat, das **`NULL`** Werte ermitteln kann
- **`EXISTS`** – Wird zu **`TRUE`** ausgewertet, wenn die Unterabfrage mindestens eine Zeile liefert.
- Mengenvergleiche mit **IN**, ANY, SOME oder ALL
- LIKE und LIKE_REGEXPR – Suche nach Mustern, auch mit Regulären Ausdrücken möglich.

Buch S. 71

EXISTS-Prädikat

Der EXISTS-Quantor prüft, ob eine Unterabfrage ein Ergebnis liefert oder nicht.

```
SELECT DISTINCT assignee
  FROM tasks AS t
 WHERE NOT EXISTS (
       SELECT id
       FROM projects AS p
       WHERE p.project_manager= t.assignee );
```

Buch S.146

Tabellenvariablen

Tabellenvariablen werden meistens durch Zuweisung deklariert und mit Daten gefüllt. Da sie mit **vorangestelltem Doppelpunkt** einen **Tabellenausdruck** darstellen, kann man auf Tabellenvariablen mit einem **SELECT**-Abfrage genau so zugreifen, wie auf eine DB-Tabelle. Man kann sich eine Tabellenvariable auch als View vorstellen, was auch Ihre Rolle bei der Ausführung recht gut beschreibt.

```
DO BEGIN
  lt_tmp = SELECT id,
                  title,
                  assignee
             FROM :tasks;
  SELECT *
    FROM :lt_tmp;
END;
```

Buch S. 112

Anonyme Blöcke

Ein Anonymer Block ist eine Prozedur, die nicht unter einem Namen in der DB abgespeichert wird. Statt dessen wird der komplette Code von der Anwendung bzw. der Konsole an die DB übergeben.

```
DO BEGIN
  SourceCode
END;
```

Buch S. 81 ff.

UDF-Funktionen

User Defined Functions (UDF) sind Unterprogramme, die einen Ausdruck darstellen, entweder

- **Skalare Ausdrücke** oder
- **Tabellenausdrücke**

Sie werden häufig per WebIDE, AMDP oder über das HANA XS Repository angelegt. Per SQL geht das so:

```
CREATE FUNCTION FunctionName
  [(ParameterList)]
  RETURNS ParameterDefinition
  [LANGUAGE SQLSCRIPT]
  [SQL SECURITY {DEFINER|INVOKER} ]
  [DEFAULT SCHEMA DefaultSchema]
  [DETERMINISTIC]
AS BEGIN
  SourceCode
END
```

Buch S. 91

Aggregation

Bei der Aggregation wird die Anzahl der Zeilen reduziert. Die Ausdrucksliste in der **`GROUP BY`** Klausel legt die Granularität des Abfrageergebnis fest, da für jede vorhandene Kombination eine Zeile gebildet wird.

Spalten, die nicht in der **`GROUP BY`** Klausel vorkommen, müssen mit einer Aggregatfunktion zusammengefasst werden, z.B. mit **`MIN()`** oder **`SUM()`**. Typische Abfrage:

```
SELECT assignee,
       status,
       SUM(effort)
  FROM tasks
  GROUP BY assignee,
           status;
```

Buch S.123

Prozeduren

sind Unterprogramme in SQLScript. Sie werden häufig über eine Entwicklungsumgebung wie z.B. WebIDE, Eclipse-HANA Repository oder via AMDP angelegt. Direktes Erzeugen per SQL ist aber auch möglich:

```
CREATE [OR REPLACE] PROCEDURE ProcedureName
      [(ParameterList)]
      [LANGUAGE {SQLSCRIPT|RLANG} ]
      [SQL SECURITY {DEFINER|INVOKER} ]
      [DEFAULT SCHEMA DefaultSchema]
      [READS SQL DATA]
      [WITH ENCRYPTION]
AS
BEGIN [SEQUENTIAL EXECUTION]
    SourceCode
END
```

Buch S. 82

`NULL`

Eigentlich ist **`NULL`** kein Wert, sondern ein Symbol, das für die Abwesenheit eines Wertes steht. Ein Vergleich mit **`NULL`** ergibt immer **`UNKNOWN`**. Jede Berechnung mit **`NULL`** ergibt wiederum **`NULL`.**

Nur mit dem **`IS NULL`** Prädikat kann auf **`NULL`** in einer Spalte gefiltert werden. Um es in Ausdrücken abzufangen, gibt es die beiden **SQL-Funktionen**:

- **`IFNULL( Ausdruck, Fallback )`**
- **`COALESCE( <Ausdruck1> { , <Ausdruck2> } ...)`**

Typische Ursachen für NULL-Werte

- **`OUTER JOINs`**
- **`CASE`** ohne **`ELSE`** Zweig
- **`NULL`** in DB-Tabellen
- Die **`NULLIF()`** SQL-Funktion

Buch S.73

Die Tabelle DUMMY

Diese Tabelle ist nicht änderbar und sie enthält exakt eine Spalte mit dem Namen DUMMY und eine Zeile die den Wert X hält. Die Tabelle ist nützlich für den Test von skalaren Ausdrücken:

```
SELECT Ausdruck FROM DUMMY;
```

Oder für die Konstruktion von festen Tabellenvariablen:

```
lt_year = SELECT '2020' AS year FROM dummy
          UNION ALL
          SELECT '2021' AS year FROM dummy;
```

Buch S. 76

UNION ALL und Mengenoperatoren

SELECT-Abfragen mit einer kompatiblen Spaltenstruktur können mit den folgenden Operatoren verknüpft werden:

- **`UNION ALL`** – Die Vereinigung zweier Tabellen.
- **`UNION`** – dito, ist aber langsamer, weil es Duplikate eliminiert
- **`INTERSECT`** – Bildet die Schnittmenge, was sich alternativ auch mit einem **`INNER JOIN`** realisieren lässt.
- **`EXCEPT`** bzw. **`MINUS`** – Ist eine Mengensubtraktion, die man alternativ auch mit dem **`EXISTS`-Prädikat** implementieren kann.

Diese Operatoren, ausser **`UNION ALL`**, betrachten die Zeilen als Elemente einer Menge. Die Elemente sind identisch, wenn alle Felder identisch sind, d.h. es gibt hier keine „Schlüsselfelder". Die angegebenen Alternativen bieten sich an, wenn die Operationen nicht auf allen Spalten durchgeführt werden sollen.

Buch S. 157

Zeit-Datentypen

Für die **Zeitpunkte (ZP)** können die folgenden Datentypen verwendet werden.

Datentyp	Standardformat
DATE	**`'YYYY-MM-DD'`**
TIME	**`'HH24-MI-SS'`**
SECONDDATE	**`'YYYY-MM-DD HH24-MI-SS'`**
TIMESTAMP	**`'YYYY-MM-DD HH24-MI-SS.FF7'`**

SQL-Funktionen

SQL-Funktion	Beschreibung
`CURRENT_<DT>`	Lokalzeit für den Datentyp (DT)
`CURRENT_UTC<DT>`	dito mit koordinierter Weltzeit
`ADD_<ZE>S(<ZP>, <Abstand>)`	addiert zum Zeitpunkt (ZP) den Abstand (+/-) in Zeiteinheit (ZE)
`<ZE>S_BETWEEN( <ZP1>, <ZP2>)`	Abstand der Zeitpunkte in der Zeiteinheit.
`<ZK>(<ZP>)`	Zeitkomponente (ZK) als INT
`ISOWEEK(<ZP>)`	**KW ISO**, z.B. 2021-W12
`WEEK(<ZP>)`	**KW US**, Num.
`WEEKDAY(<ZP>)`	Numerisch: Mo=0, So=6
`QUARTER(<ZP>, [<Offset>])`	Quartal, ggf. abw. Geschäftsjahr
`LOCALTOUTC(<ZP>, <Zeitzone>])`	Lokalzeit nach UTC
`UTCTOLOCAL(<ZP>, <Zeitzone>])`	UTC nach Lokalzeit

Die **Zeiteinheit** (ZE) in den Funktionen ist entweder **`SECOND`**, **`DAY`**, **`MONTH`** oder **`YEAR`**.

Buch S. 191

Konvertierung zwischen Zeit und Zeichenketten

`TO_VARCHAR( <ZP>, [<Format>])`
Konvertierung des Zeitpunkts (ZP) in eine Zeichenkette.

`TO_<ZDT>( <ZK>, [<Format>])`
Konvertierung Zeichenkette (ZK) in den Zeit-Datentyp (ZDT).

Symbole für die Formatierung

Einheit	Symbol	Beschreibung
Jahr	YYYY	Jahr, 4-Stellig
	YY	Jahr, 2-Stellig
Quartal	Q	Numerisch
Monat	MM	Numerisch, 2-Stellig
	MONTH	Name in EN
	MON	Abkürzung in EN
	RM	Römsiche Schreibweise
Woche	W	W. im Monat
	WW	W. im Jahr, nicht ISO!!!
Tag	D	Numerisch
	DD	Numerisch, 2-Stellig
	DAY	Name in EN
	DY	Abkürzung in EN
Stunde	HH12	12h Zeit ohne AM/PM
	HH24	Stunden (0-23)
	AM | PM	Vor- oder Nachmittag
Minute	MI	Numerisch, 2-Stellig
Sekunde	SS	Numerisch, 2-Stellig
	SSSSS	Sek. nach Mitternacht
	FF [1..7]	NK-Stellen der Sek.

Neben den Symbolen können auch noch Trennzeichen verwendet werden:

```
SELECT TO_VARCHAR(CURRENT_TIME,
                  'HH24.MI.SS')
       TO_DATE('DEC-29-20',
               'MON-DD-YY')
FROM DUMMY;
```

Buch S. 193

Zeichenketten

Im Gegensatz zu ABAP werden Leerzeichen am Ende nicht automatisch entfernt! Mit dem Operator **`||`** werden zwei Zeichenketten verkettet.

Datentypen

Datentyp	Beschreibung	Max. Länge
NVARCHAR(N)	Unicode Zeichenkette	5000
VARCHAR(N)	ASCII Zeichenkette	5000
ALPHANUM(N)	Alphakonvertiert	127
CLOB, NCLOB	Große Zeichenketten	2GB

SQL-Funktionen

SQL-Funktion	Beschreibung
`LENGTH(<ZK>)`	Länge
`ABAP_LOWER(<ZK>)`	Konvertierung der Zeichenkette in Klein- / Großbuchstaben
`ABAP_UPPER(<ZK>)`	
`LEFT(<ZK>, <Länge>)`	Linker/Rechter Teil der Zeichenkette mit der Länge
`RIGHT(<ZK>, <Länge>)`	
`SUBSTR(<ZK>, <Pos>, <Länge>)`	Teil der Zeichenkette
`SUBSTR_BEFORE(<ZK1>, <ZK2>)`	Teil der ZK1 vor/nach ZK2
`SUBSTR_AFTER(<ZK1>, <ZK2>)`	
`LOCATE(<ZK1>, <ZK2>)`	Position von ZK2 in ZK1
`REPLACE(<ZK1>, <ZK2>, <ZK3>)`	Ersetzt ZK2 in ZK1 durch ZK3
`LPAD(<ZK>, <Länge> [<Muster>])`	Auffüllen von Links/Rechts mit Muster bis zur Länge
`RPAD(<ZK>, <Länge> [<Muster>])`	
`ABAP_ALPHANUM(<ZK>, <Länge>)`	Alphakonvertierung
`LTRIM(<ZK>[, <ZM>])`	Entfernen der Zeichenmenge (ZM) von Links/Rechts
`RTRIM(<ZK>[, <ZM>])`	

Dazu gibt es einige der Funktionen gibt es auch in einer Variante für die Verwendung mit Regulären Ausdrücken:

- LOCATE_REGEXPR( )
- OCCURRENCES_REGEXPR( )
- REPLACE_REGEXPR(**`<Muster> IN <ZK> WITH <Ersatz>`**)
- SUBSTRING_REGEXPR(**`<Muster> IN <ZK> [GROUP <Gruppe>]`**)

Buch S. 82

Numerische Datentypen

Im Gegensatz zu ABAP wird bei einer Zuweisung nicht automatisch kaufmännisch gerundet! Wenn das Zielformat nicht passt, wird abgeschnitten. 😢

Datentyp	Beschreibung
INT	Ganzzahl
DEC(p,s)	Festkommazahl
DEC	Dez. Gleitkommazahl
REAL	Bin. Gleitkommazahl

SQL-Funktionen

SQL-Funktion	Beschreibung
`ROUND(<Z>, <NK>)`	Kaufmännisches Runden
`NDIV0(<Z>, <N>)`	Division Z/N, NULL falls N=0
`RAND()`	Zufallszahl zw. 0 und 1
`ABS(<Z>)`	Absolutwert
`SIGN(<Z>)`	Vorzeichen 1 oder -1
`MOD(<Z>, <N>)`	Divisionsrest Z/N
`CEIL(<Z>)`	Auf/Abrunden auf INT
`FLOOR(<Z>)`	

Und noch viele andere mehr in der SAP Doku…

Buch S.207