Sructred Query Language

Michael Dienert

11. November 2010

Inhaltsverzeichnis

1 Ein kurzer Versionsüberblick

- vor ca. 35 Jahren entwickelten die IBM-Mitarbeiter R.F. Boyce und D.D Chamberlain die Sprache SEQUEL. SEQUEL kannte noch keine JOINS, d.h. man konnte nur mit einer Tabelle arbeiten. SEQUEL wurde bald in SQL umbenannt.
- 1987 wird SQL zum ANSI-Standard [1] Diese Version wird SQL-1 genannt.
- 1987 wird SQL-1 zum ISO-Standard. 1989 wird dieser nochmals geaendert. Offizielle ISO-Norm: **ISO 9075:1989**.
- 1992 wird der Sprachumfang von SQL erheblich erweitert und SQL wird als SQL-2 auch SQL-92 genannt, normiert. Diese Norm heisst **ISO 9075:1992**.
- weitere Normierungen finden 1999 (SQL-3), 2003, 2006 und 2008 statt

[1] American National Standards Institute

2 SQL-1 mit einigen Erweiterungen aus SQL-92

Für die Abschlussprüfung in IT hat man sich auf den SQL-92-Standard geeinigt. Da man die Prüfung aber bereits mit den Befehlen des SQL-1-Standards, der um ein kleines bisschen SQL-92 erweitert wird, sehr gut bestehen kann, beschränken wir uns zunächst auf diesen Teil

SQL-1 besteht aus zwei Teilen:

DDL - Data Definition Language : die DDL enthält die Befehle:

CREATE : mit CREATE kann man Datenbanken, Tabellen, logische Tabellen (views) und Indizes anlegen.

DROP : mit DROP erreicht man das genaue Gegenteil von CREATE: Datenbanken, Tabellen, Views und Indizes werden gelöscht.

ALTER : mit ALTER kann man Tabellen und Indizes *ändern*. Datenbanken und Views lassen sich nach dem Erstellen **nicht** mehr ändern.

DML - Data Manipulation Language : mit der DML wird direkt mit den Daten in der Datenbank gearbeitet. Folgende Befehle zählen zur DML:

INSERT : Einfügen von Datensätzen (Zeilen) in eine Tabelle

UPDATE : Ändern aller oder bestimmter Datensätze einer Tabellen**spalte**

DELETE : Löschen bestimmter Datensätze einer Tabelle

SELECT : Obwohl man mit SELECT keine Daten manipuliert sondern Datenbank**abfragen** erstellt, zählt SELECT zu DML.
SELECT ist der mit Abstand wichtigste und leider auch komplexeste SQL-Befehl. Um mit Datenbanken arbeiten zu können, muss man die Syntax von SELECT sicher beherrschen.

3 Eine Sprache zur Beschreibung anderer Sprachen

Wie jede Computersprache wird auch SQL mit Hilfe der Backus-Naur-Form beschrieben. Dabei gelten folgende, einfache Regeln: [2] :

Schlüsselwörter alle reservierten Schlüsselwörter werden mit Grossbuchstaben geschrieben: SELECT, INSERT, DROP, CREATE ...

Trennung von Alternativen der senkrechte Strich (das Pipe-Symbol) trennt Alternativen: `DISTINCT | ALL`

Optionen wenn etwas optional ist, steht es in eckigen Klammern: `SELECT [ALL]`

Wiederholungen alles was in geschweiften Klammern steht kann beliebig (0,1,viele) oft wiederholt werden: `FROM table {, table}`

[2] die Backus-Naur-Form wird in den RFCs des IETF in leicht abgewandelter Form verwendet

4 Die drei allerwichtigsten SQL-Befehle

Für den Anfang beschränken wir uns auf drei SQL1-Befehle:

- **CREATE TABLE**
- **INSERT INTO ... VALUES**
- **SELECT ... FROM ... WHERE**

4.1 Tabellen erstellen

In der BNF sieht die Syntax des CREATE-TABLE-Befehls so aus, wobei man beachten muss, dass SQL1 noch keine Primär- und Fremdschlüssel kennt:

```
create-table ::= CREATE TABLE tabellenName
                   ( spaltenDefinition {,  spaltenDefinition} )

spaltenDefinition ::= spaltenName datenTyp [NOT NULL]
datenTyp ::= CHARACTER [ (laenge) ]                                  |
             DECIMAL [ (anzahlZiffern [, anzahlNachkommastellen]) ] |
             INTEGER                                                 |
             DOUBLE PRECISION
```

In dieser Syntaxdefinition wurden einige SQL-1 Datentypen weggelassen (NUMERIC, SMALLINT, FLOAT, REAL).

Der Ausdruck `anzahlZiffern` gibt die **Gesamtanzahl** der Ziffern einer Zahl an (also Vor- plus Nachkommastellen)

`DOUBLE PRECISION` ist eine Fliesskommazahl. Die maximale Ziffernzahl hängt vom RDBMS ab. Bei postgresql sind dies 15 Ziffern.

Lässt man bei `CHARACTER` die Längenangabe weg, wird ein einzelnes Zeichen definiert. `CHARACTER` ist also dasselbe wie `CHARACTER(1)`

Natürlich ist diese kleine Zahl an Datentypen bei modernen Datenbanken nicht ausreichend und die aktuellen RDBMS stellen eine Vielzahl weiterer Datentypen zur Verfügung. Wichtig sind hierbei besonders die Datentypen `TEXT` (kein SQL-Standard!) und `CHARACTER VARYING` (`VARCHAR`).

4.2 Datensätze in eine Tabelle eintragen

Die Syntax der INSERT-Anweisung:

```
insert-anweisung ::= INSERT INTO tabellenName
                     [ ( spaltenReferenz {, spaltenReferenz} ) ]
                     VALUES ( eintrag {, eintrag} )

spaltenReferenz ::= [tabellenName.]spaltenName
eintrag ::= wert | NULL
```

Lässt man die optionale Liste mit Spalten-Referenzen weg, muss man Werte für alle Attribute und in der Reihenfolge, wie sie in der CREATE-TABLE-Anweisung aufgeführt sind, angeben. Hat man eine Liste mit Spalten-Referenzen, muss man die Werte in der Reihenfolge dieser Liste übergeben.

Dies ist noch nicht die vollständige INSERT-Anweisung, es fehlt noch eine optionale SELECT-spezifikation am Schluss. Dazu muss man aber erst die Syntax der SElECT-Anweisung definieren, was im nächsten Kapitel passiert.

4.3 Datensätze auswählen

Das SELECT-Kommando ist leider sehr komplex, fangen wir also langsam an:

```
select-anweisung ::= SELECT [ALL | DISTINCT] spaltenausdruck
                                              tabellenausdruck
```

DISTINCT - DISTINCT unterdückt doppelte Zeilen im Ergebnis

ALL - ALL ist die Grundeinstellung: alle selektierten Zeilen tauchen im Ergebnis auf, auch wenn sie mehrfach vorkommen

spaltenausdruck - Der Spaltenausdruck bestimmt, welche Attribute = Spalten in die Ergebnistabelle aufgenommen werden.

tabellenausdruck - Der Tabellenausdruck wählt die *Zeilen* aus den Eingangstabellen aus

4.3.1 Spaltenausdruck bei SELECT

Jetzt wird es gleich deutlich komplizierter (obwohl diese Syntaxbeschreibung etwas vereinfacht wurde):

```
spalten-ausdruck ::= rechenAusdruck {, rechenAusdruck} | *

rechenAusdruck ::= [+|-] summand {+|- summand}
summand ::= faktor {*|/ faktor}
faktor ::= konstante | spaltenReferenz | funktion | ( rechenAusdruck )
funktion ::= COUNT(*) | aggregatFunktion
aggregatFunktion ::= AVG | MAX | MIN | SUM | COUNT ( rechenAusdruck )
spaltenReferenz ::= [tabelle | aliasName . ] spaltenName
```

Vereinfacht gesagt ist der Spaltenausdruck also eine Liste von *arithmetischen Ausdrücken* (Berechnungen). In diesen arithmetischen Ausdrücken dürfen beliebig Konstanten, Funktionen und Spalten mit den vier Grundrechenarten (+,-,*,/) verknüpft werden.

Beachten muss man noch die Doppelbedeutung des Zeichens '*': Als mathematischer Operator steht es natürlich für die Multiplikation. Der '*' im Spalten-Ausdruck wählt *alle* Spalten der angegebenen Tabellen aus. COUNT(*) zählt die Ergebniszeilen.

4.3.2 Tabellenausdruck bei SELECT

Fangen wir gleich mit der Syntaxbeschreibung an, Erklärungen folgen später:

```
tabellenAusdruck ::= fromAusdruck
                     [whereAusdruck]
                     [groupByAusdruck]
                     [havingAusdruck]

fromAusdruck ::= FROM tabelle [aliasName] { , tabelle [aliasName] }
groupByAusdruck ::= GROUP BY spaltenReferenz { , spaltenReferenz }
havingAusdruck ::= HAVING suchbedingung
```

FROM legt fest, welche Eingangstabellen verwendet werden sollen. Eingangstabellen können normale Tabellen oder logische Tabellen (Views) sein.

WHERE legt Bedingungen fest, mit der die Zeilen des kartesischen Produkts der Eingangstabellen (sämtliche Kombinationen der *Zeilen* der Eingangstabellen. Siehe 4.4) selektiert werden.

GROUP BY fasst die Zeilen des Produkts der Eingangstabellen auf Basis gleicher Spaltenwerte zusammen.

HAVING wählt aus den mit GROUP BY erzeugten Gruppen aufgrund einer Bedingung aus

aliasName mit tabelle aliasName wird ein anderer Name fuer eine Tabelle erzeugt. Das wird oft benutzt, wenn der Tabellenname abgekuerzt werden soll. Bei bestimmten SELECT-Abfragen sind Alias-Namen zwingend, z.b. wenn man eine Tabelle mit sich selbst verknüpfen möchte.

4.4 Das Produkt zweier oder mehrerer Tabellen

Beim *kartesischen Produkt* von A und B wird jede Zeile von A mit jeder Zeile von B verknüpft, es kommt also *jede* Kombination der Zeilen der Einzeltabellen vor:

Tabelle A:

attribut
a
b
c
d
e

Tabelle B:

attribut
1
2
3

Tabelle A $\times$ Tabelle B:

A.attribut	B.attribut
a	1
a	2
a	3
b	1
b	2
b	3
c	1
c	2
c	3
d	1
d	2
d	3
e	1
e	2
e	3