

Einführung in Datenbanken

Kapitel 3: SQL: Geschichte und Lexikalische Syntax

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2018/19

<http://www.informatik.uni-halle.de/~brass/db18/>

Geschichte

- SEQUEL, eine frühere Version von SQL, wurde von Chamberlin, Boyce et al. bei IBM Research, San Jose (1974) entwickelt.

SEQUEL steht für "Structured English Query Language" ("Anfragesprache in strukturiertem Englisch"). Manche Leute Sprechen SQL noch heute auf diese Art aus. Der Name wurde aus rechtlichen Gründen geändert (SEQUEL war bereits eine eingetragene Marke). Codd war auch in San Jose, als er das relationale Modell erfand.

- SQL war die Sprache des System/R (1976/77).

System/R war ein sehr einflussreicher Forschungs-Prototyp.

- Ersten SQL-unterstützenden kommerziellen Systeme waren Oracle (1979) und IBM SQL/DS (1981).

Standards (4)

- Erweiterungen 2008 u.a.:
 - TRUNCATE
 - ORDER BY auch in Unteranfragen (für **FETCH FIRST**).
 - INSTEAD OF Trigger.
- Erweiterungen 2011 u.a.:
 - Zeitbezogene Daten (PERIOD FOR)
 - Erweiterungen der Window Functions.
- Erweiterungen 2016 u.a.:
 - JSON-Unterstützung
 - Row Pattern Matching.

Standards (5)

- Aktuelle Version des Standards ist **SQL:2016**.

ISO/IEC 9075-1:2016	Framework (SQL/Framework)
ISO/IEC 9075-2:2016	Foundation (SQL/Foundation)
ISO/IEC 9075-3:2016	Call-Level Interface (SQL/CLI)
ISO/IEC 9075-4:2016	Persistent Stored Modules (SQL/PSM)
ISO/IEC 9075-9:2016	Management of External Data (SQL/MED)
ISO/IEC 9075-10:2016	Object Language Bindings (SQL/OLB)
ISO/IEC 9075-11:2016	Information and Definition Schemas (SQL/Schemata)
ISO/IEC 9075-13:2016	SQL Routines and Types Using the Java TM Programming Language (SQL/JRT)
ISO/IEC 9075-14:2016	XML-Related Specifications (SQL/XML)
Teil 1, 2, 11: minimale Anforderungen, andere Teile: Erweiterungen.	

Die Lücken in den Nummern der Bände sind nur historisch zu verstehen.

Es gibt einen verwandten Standard ISO/IEC 13249:

“SQL multimedia and application packages”.

- Teil 2 definiert den Kern der Sprache und hat 1707 Seiten.

Syntax-Formalismus

- In dieser Vorlesung wird die Syntax von SQL-Anfragen mit "Syntax-Graphen" definiert.

Beispiel auf nächster Folie. Alternative zu kontextfreien Grammatiken (definiert dieselbe Klasse von Sprachen). In Anhang A genauer.

- Um eine Zeichenfolge syntaktisch richtig zu erstellen, muss man einen Pfad vom Start bis zum Ziel durch den Graphen verfolgen.

Wörter in Ovalen werden direkt in die Ausgabe geschrieben, Kästen sind "Aufrufe" anderer Graphen: An diesem Punkt muss man einen Pfad durch den Graphen finden, dessen Name in dem Kasten steht. Anschließend kehrt man zu der Kante zurück, die den Kasten verlässt. Das Oracle-SQL-Referenz-Handbuch enthält auch Syntax-Graphen, aber dort ist die Bedeutung von Ovalen und Kästen umgekehrt.

Basis-Anfrage-Syntax (2)

- Jede SQL-Anfrage muss die Schlüsselwörter **SELECT** und **FROM** enthalten.

Oracle stellt eine Relation "DUAL" zur Verfügung, die nur eine Zeile hat. Sie kann benutzt werden, wenn nur eine Berechnung ohne Zugriff auf die DB durchgeführt wird: "**SELECT TO_CHAR(SQRT(2)) FROM DUAL**" berechnet $\sqrt{2}$.

- In PostgreSQL, SQL Server, Access und MySQL kann die FROM-Klausel weggelassen werden, z.B. **SELECT 1+1**.

In Oracle, DB2 und dem SQL-92-Standard ist dies ein Syntax-Fehler.

SQL-Syntax in der Vorlesung

- SQL:2016 ist zu groß, um es hier vollständig zu behandeln. Ein großer Teil des Standards ist auch noch nicht in derzeitigen DBMS implementiert.
- SQL/89 (~ Einstiegs-Level von SQL-92) wird vollständig behandelt und ein Teil von SQL:2016, der in fast allen DBMS implementiert ist.

Manchmal werden Details der SQL-Syntax spezieller Systeme erklärt, meist im Kleingedruckten. Dies ist irrelevant für Klausuren. Sie sollen einen Eindruck von der Portabilität der Konstrukte geben (und helfen, wenn man mit diesem DBMS arbeiten muss). In Klausuren werden nur Punkte abgezogen, wenn man extrem nicht-portable Konstrukte verwendet (z.B. Anfragen, die nur in MySQL laufen).

- In der Vorlesung “Datenbank-Programmierung” werden weitere fortgeschrittenen SQL-Konstrukte behandelt.

Lexikalische Syntax

- Die lexikalische Syntax einer Sprache definiert, wie Wortsymbole (“Token”) aus einzelnen Zeichen zusammengesetzt werden.
- Z.B. definiert sie die genaue Syntax von
 - Bezeichnern (Namen für z.B. Tabellen, Spalten),
 - Literalen (Datentyp-Konstanten, z.B. Zahlen),
 - Schlüsselwörtern, Operatoren, Trennzeichen.
- Anschließend wird die Syntax von Anfragen u.s.w. basierend auf diesen Wortsymbolen definiert.

D.h. eine Anfrage wird dann als Folge von Token definiert, nicht als Folge von Zeichen. Diese Zweiteilung bewirkt z.B., dass man auf der höheren Syntaxebene nicht mehr über eingestreuten Leerplatz nachdenken muss.

Leerzeichen und Kommentare

Leerplatz ist zwischen Wortsymbolen (Token) erlaubt:

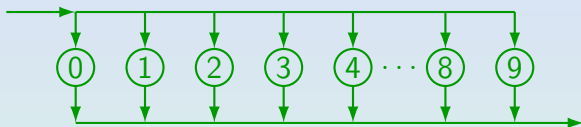
- Leerzeichen (meist auch Tabulator-Zeichen)
- Zeilenumbrüche
- Kommentare:
 - Von “--” bis <Zeilenende>
Unterstützt in SQL-92, PostgreSQL, Oracle, SQL Server, IBM DB2, MySQL. MySQL benötigt ein Leerzeichen nach “--”, SQL-92 nicht. Access unterstützt diesen Kommentar nicht und auch nicht /* ...*/.
 - Von “/*” bis “*/”
Nur in PostgreSQL, Oracle, SQL Server und MySQL unterstützt: weniger portabel.

Formatfreie Sprache

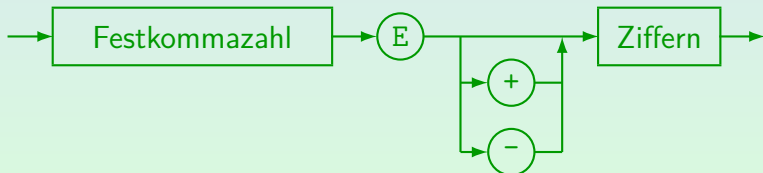
- Obige Regel (beliebigere Leerplatz zwischen Token) bedeutet, dass SQL eine formatfreie Sprache wie z.B. Java ist:
 - Es ist nicht nötig, dass “**SELECT**”, “**FROM**”, “**WHERE**” am Anfang neuer Zeilen stehen. Man kann auch die ganze Anfrage in eine Zeile schreiben.
 - Man kann z.B. komplexe Bedingungen auf mehrere Zeilen verteilen und Einrückungen verwenden, um die Struktur deutlich zu machen.

Zahlen (3)

- Ziffer:



- Fließkommazahl (“Approximate Numeric Literal”):



Zeichenketten (1)

- Ein Zeichenketten-Literal ist eine Folge von Zeichen, eingeschlossen in Hochkommas, z.B.
 - `'abc'`
 - `'Dies ist ein String.'`

- Hochkommas in Zeichenketten müssen verdoppelt werden, z.B. `'John''s book'`.

Der tatsächliche Wert der Zeichenkette ist `John's book` (mit einfachem Hochkomma). Das Verdoppeln ist nur eine Art, es einzugeben. Beachten Sie, dass die aus Java bekannten Escape-Sequenzen wie `\'` in SQL nicht vorgesehen sind (und z.B. in PostgreSQL auch tatsächlich nicht funktionieren).

- Natürlich ist auch die leere Zeichenkette erlaubt: `''`.

In Oracle werden die leere Zeichenkette und der Nullwert identifiziert. Das entspricht nicht dem Standard.

Zeichenketten (3)

- SQL-92 erlaubt das Splitten von Zeichenketten (jedes Segment eingeschlossen in '...') zwischen Zeilen.
PostgreSQL und MySQL unterstützen diese Syntax, Oracle, SQL Server und Access nicht. Zeichenketten können aber mit dem Konkatenations-Operator (|| in PostgreSQL und Oracle, + in SQL Server und Access) kombiniert werden.
- SQL-92 und alle sechs DBMS erlauben Zeilenumbrüche in Zeichenketten-Konstanten.
D.h., das Hochkomma kann man auf einer folgenden Zeile schließen.
- Access und MySQL erlauben auch in Anführungszeichen " eingeschlossene String-Literale. Nicht konform zum Standard!
Z.B. bei PostgreSQL und Oracle ist dies ein Syntaxfehler.
Microsoft SQL Server hat die Option "SET QUOTED_IDENTIFIER ON" (inzwischen standardmäßig gesetzt), die das standard-konforme Verhalten liefert.

Andere Konstanten (1)

- Es gibt mehr Datentypen als nur Zahlen und Zeichenketten, z.B. (s. Kapitel 10):
 - Zeichenketten mit nationalem Zeichensatz
 - Datum, Zeit, Zeitstempel, Datum-/Zeit-Intervall
 - Bit-Strings, binäre Daten
 - Large Objects (Dateien als Tabelleneintrag)
- Die Syntax der Konstanten dieser Datentypen ist allgemein sehr systemabhängig.

Oft gibt es keine Konstanten dieser Typen, aber es gibt eine automatische Typ-Konvertierung ("coercion") von Strings.

Andere Konstanten (2)

- Z.B. werden Datumswerte wie folgt geschrieben:
 - Oracle: `'31-OCT-02'` (US), `'31.10.2002'` (D).

Das Default-Format (Teil der nationalen Sprach-Einstellungen) wird automatisch konvertiert, ansonsten:
`TO_DATE('31.10.2002', 'DD.MM.YYYY')`.
 - SQL-92-Syntax: `DATE '2002-10-31'`.
 - PostgreSQL und MySQL verstehen diese Syntax (auch ohne `"DATE"`), zusätzlich noch weitere Varianten.
 - DB2: `'2002-10-31'`, `'10/31/2002'`, `'31.10.2002'`.
 - SQL Server: z.B. `'20021031'`, `'10/31/2002'`, `'October 31, 2002'` (abhängig von Sprache).
 - Access: `#10/31/2002#` (US), `#31.10.2002#` (D).

Bezeichner (2)

- Bezeichner können bis 18 Zeichen haben (mind.).

System	Länge	Erstes Zeichen	Andere Zeichen
SQL-86	≤ 18	A-Z	A-Z,0-9
SQL-92	≤ 128	A-Z,a-z	A-Z,a-z,0-9,_
PostgreSQL	≤ 63	A-Z,a-z,_	A-Z,a-z,0-9,_,\$
Oracle	≤ 30	A-Z,a-z	A-Z,a-z,0-9,_,#,\$
MySQL	≤ 64	A-Z,a-z,0-9,_,\$	A-Z,a-z,0-9,_,\$
SQL Server	≤ 128	A-Z,a-z,_,(@,#)	A-Z,a-z,0-9,_,@,#,\$
IBM DB2	≤ 18 (8)	A-Z,a-z	A-Z,a-z,0-9,_
Access	≤ 64	A-Z,a-z	A-Z,a-z,0-9,_

In MySQL müssen Bezeichner einen Buchstaben enthalten, ggf. aber nicht vorn.

- Müssen verschieden von reservierten Wörtern sein.

Es gibt viele reservierte Wörter, siehe unten. Einbettungen in Programmiersprachen (PL/SQL, Visual Basic) fügen noch mehr hinzu.

Bezeichner (3)

- Es ist möglich, nationale Zeichen zu verwenden.

Das ist implementierungsabhängig. Z.B. wählt man in Oracle bei der Installation einen DB-Zeichensatz. Alphanumerische Zeichen von diesem Zeichensatz können in Bezeichnern verwendet werden.

- Bezeichner/Schlüsselwörter nicht case-sensitiv.

Das scheint das zu sein, was der SQL-92-Standard sagt (das Buch von Date/Darwen über den Standard stellt es klar so dar). In PostgreSQL sind alle Bezeichner nicht case-sensitiv (werden in Kleinbuchstaben konvertiert). Oracle konvertiert alle Zeichen außerhalb von Hochkommas in Großbuchstaben. In SQL Server kann Case-Sensitivität bei der Installation gewählt werden. In MySQL hängt die Case-Sensitivität der Tabellennamen von der Case-Sensitivität von Dateinamen im zugrundeliegenden Betriebssystem ab (Tabellen als Dateien gespeichert). Innerhalb einer Anfrage muss man in MySQL konsistent bleiben. Schlüsselwörter und Spaltennamen sind in MySQL jedoch nie case-sensitiv.

Bezeichner (4)

- Betrachten Sie z.B. die Anfrage

```
SELECT X.VORNAME, X.NACHNAME  
FROM   STUDENTEN X
```

- Folgende Anfrage ist vollkommen äquivalent:

```
select X . Vorname ,  
       x.NachName  
From Studenten X
```

(Dies zeigt auch die Formatfreiheit von SQL.)

Achtung: Zeichenketten-Vergleiche sind normalerweise case-sensitive:

```
select x.name from studenten x where x.name = 'lisa'
```

wird keine Antwort liefern (obwohl es 'Lisa' gibt).

Reservierte Wörter - SQL (1)

¹ = Oracle 8.0

² = SQL-92

³ = SQL Server 7

— A —

ABSOLUTE²

ACCESS¹

ACTION²

ADD^{1,2,3}

ALL^{1,2,3}

ALLOCATE²

ALTER^{1,2,3}

AND^{1,2,3}

ANY^{1,2,3}

ARE²

AS^{1,2,3}

ASC^{1,2,3}

ASSERTION²

AT²

AUTHORIZATION^{2,3}

AUDIT¹

AVG^{2,3}

— B —

BACKUP³

BEGIN^{2,3}

BETWEEN^{1,2,3}

BIT²

BIT_LENGTH²

BOTH²

BREAK³

BROWSE³

BULK³

BY^{1,2,3}

— C —

CASCADE^{2,3}

CASCADE²

CASE^{2,3}

CATALOG²

CHAR^{1,2}

CHARACTER²

CHAR_LENGTH²

CHARACTER_LENGTH²

CHECK^{1,2,3}

CHECKPOINT³

CLOSE^{2,3}

CLUSTER¹

CLUSTERED³

COALESCE^{2,3}

COLLATE²

COLLATION²

COLUMN^{1,3}

COMMENT¹

COMMIT^{2,3}

COMMITTED³

COMPRESS¹

COMPUTE³

Reservierte Wörter - SQL (2)

CONFIRM ³	CURRENT ^{1,2,3}	DECLARE ^{2,3}	DOMAIN ²
CONNECT ^{1,2}	CURRENT_DATE ^{2,3}	DEFAULT ^{1,2,3}	DOUBLE ^{2,3}
CONNECTION ²	CURRENT_TIME ^{2,3}	DEFERRABLE ²	DROP ^{1,2,3}
CONSTRAINT ^{2,3}	CURRENT_TIMESTAMP ^{2,3}	DEFERRED ²	DUMMY ³
CONSTRAINTS ²	CURRENT_USER ^{2,3}	DELETE ^{1,2,3}	DUMP ³
CONTAINS ³	CURSOR ^{2,3}	DENY ³	— E —
CONTAINSTABLE ³	— D —	DESC ^{1,2}	ELSE ^{1,2,3}
CONTINUE ^{2,3}	DATABASE ³	DESCRIBE ²	END ^{2,3}
CONTROLROW ³	DATE ^{1,2}	DESCRIPTOR ²	END-EXEC ²
CONVERT ^{2,3}	DAY ²	DIAGNOSTICS ²	ERRLVL ³
CORRESPONDING ²	DBCC ³	DISCONNECT ²	ERROREXIT ³
COUNT ^{2,3}	DEALLOCATE ^{2,3}	DISK ³	ESCAPE ^{2,3}
CREATE ^{1,2,3}	DEC ²	DISTINCT ^{1,2,3}	EXCEPT ^{2,3}
CROSS ^{2,3}	DECIMAL ^{1,2}	DISTRIBUTED ³	EXCEPTION ²

Reservierte Wörter - SQL (3)

EXCLUSIVE¹
EXEC^{2,3}
EXECUTE^{2,3}
EXISTS^{1,2,3}
EXIT³
EXTERNAL²
EXTRACT²
— **F** —
FALSE²
FETCH^{2,3}
FILE^{1,3}
FILLFACTOR³
FIRST²
FLOAT^{1,2}

FLOPPY³
FOR^{1,2,3}
FOREIGN^{2,3}
FOUND²
FREETEXT³
FREETEXTTABLE³
FROM^{1,2,3}
FULL^{2,3}
— **G** —
GET²
GLOBAL²
GO²
GOTO^{2,3}
GRANT^{1,2,3}

GROUP^{1,2,3}
— **H** —
HAVING^{1,2,3}
HOLDLOCK³
HOUR²
— **I** —
IDENTITY^{2,3}
IDENTITY_INSERT³
IDENTITYCOL³
IDENTIFIED¹
IF³
IMMEDIATE^{1,2}
IN^{1,2,3}
INCREMENT¹

INDEX^{1,3}
INDICATOR²
INITIAL¹
INITIALLY²
INNER^{2,3}
INPUT²
INSENSITIVE²
INSERT^{1,2,3}
INT²
INTEGER^{1,2}
INTERSECT^{1,2,3}
INTERVAL²
INTO^{1,2,3}
IS^{1,2,3}

Reservierte Wörter - SQL (4)

ISOLATION^{2,3}— **J** —JOIN^{2,3}— **K** —KEY^{2,3}KILL³— **L** —LANGUAGE²LAST²LEADING²LEFT^{2,3}LEVEL^{1,2,3}LIKE^{1,2,3}LINENO³LOAD³LOCAL²LOCK¹LONG¹LOWER²— **M** —MATCH²MAX^{2,3}MAXEXTENTS¹MIN^{2,3}MINUS¹MINUTE²MIRROREXIT³MODE¹MODIFY¹MODULE²MONTH²— **N** —NAMES²NATIONAL^{2,3}NATURAL²NCHAR²NETWORK¹NEXT²NO²NOAUDIT¹NOCHECK³NOCOMPRESS¹NONCLUSTERED³NOT^{1,2,3}NOWAIT¹NULL^{1,2,3}NULLIF^{2,3}NUMBER¹NUMERIC²— **O** —OCTET_LENGTH²OF^{1,2,3}OFF³OFFLINE¹OFFSETS³ON^{1,2,3}

Reservierte Wörter - SQL (5)

ONCE ³	— P —	PRIOR ^{1,2}	RELATIVE ²
ONLINE ¹	PARTIAL ²	PRIVILEGES ^{1,2,3}	RENAME ¹
ONLY ^{2,3}	PCTFREE ¹	PROC ³	REPEATABLE ³
OPEN ^{2,3}	PERCENT ³	PROCEDURE ^{2,3}	REPLICATION ³
OPENDATASOURCE ³	PERM ³	PROCESSEXIT ³	RESOURCE ¹
OPENQUERY ³	PERMANENT ³	PUBLIC ^{1,2,3}	RESTORE ³
OPENROWSET ³	PIPE ³	— R —	RESTRICT ^{2,3}
OPTION ^{1,2,3}	PLAN ³	RAISERROR ³	RETURN ³
OR ^{1,2,3}	POSITION ²	RAW ¹	REVOKE ^{1,2,3}
ORDER ^{1,2,3}	PRECISION ^{2,3}	READ ^{2,3}	RIGHT ^{2,3}
OUTER ^{2,3}	PREPARE ^{2,3}	READTEXT ³	ROLLBACK ^{2,3}
OUTPUT ²	PRESERVE ²	REAL ²	ROW ¹
OVER ³	PRIMARY ^{2,3}	RECONFIGURE ³	ROWCOUNT ³
OVERLAPS ²	PRINT ³	REFERENCES ^{2,3}	ROWGUIDCOL ³

Reservierte Wörter - SQL (6)

ROWID ¹	SET ^{1,2,3}	SUCCESSFUL ¹	TIMEZONE_HOUR ²
ROWNUM ¹	SETUSER ³	SUM ^{2,3}	TIMEZONE_MINUTE ²
ROWS ^{1,2}	SHARE ¹	SYNONYM ¹	TO ^{1,2,3}
RULE ³	SHUTDOWN ³	SYSDATE ¹	TOP ³
— S —	SIZE ^{1,2}	SYSTEM_USER ^{2,3}	TRAILING ²
SAVE ³	SMALLINT ^{1,2}	— T —	TRAN ³
SCHEMA ^{2,3}	SOME ^{2,3}	TABLE ^{1,2,3}	TRANSACTION ^{2,3}
SCROLL ²	SQL ²	TAPE ³	TRANSLATE ²
SECOND ²	SQLCODE ²	TEMP ³	TRANSLATION ²
SECTION ²	SQLERROR ²	TEMPORARY ^{2,3}	TRIGGER ^{1,3}
SELECT ^{1,2,3}	SQLSTATE ²	TEXTSIZE ³	TRIM ²
SERIALIZABLE ³	START ¹	THEN ^{1,2,3}	TRUE ²
SESSION ^{1,2}	STATISTICS ³	TIME ²	TRUNCATE ³
SESSION_USER ^{2,3}	SUBSTRING ²	TIMESTAMP ²	TSEQUAL ³

Reservierte Wörter - SQL (7)

— **U** —

UID¹

UNCOMMITTED³

UNION^{1,2,3}

UNIQUE^{1,2,3}

UNKNOWN²

UPDATE^{1,2,3}

UPDATETEXT³

UPPER²

USAGE²

USE³

USER^{1,2,3}

USING²

— **V** —

VALIDATE¹

VALUE²

VALUES^{1,2,3}

VARCHAR^{1,2}

VARCHAR2¹

VARYING^{2,3}

VIEW^{1,2,3}

— **W** —

WAITFOR³

WHEN^{2,3}

WHENEVER^{1,2}

WHERE^{1,2,3}

WHILE³

WITH^{1,2,3}

WORK^{2,3}

WRITE²

WRITETEXT³

— **Y** —

YEAR²

— **Z** —

ZONE²

Delimited Identifier (2)

- Delimited Identifier sind keine String-Konstanten! Solche haben die Form `'...'`.

SQL Server akzeptiert `'` und `"` für String-Konstanten und nimmt `[...]` für Delimited Identifier. `"SET QUOTED_IDENTIFIER ON"` schaltet auf den SQL-92-Standard um (aber Delimited Identifier sind nicht case-sensitive). Access versteht `[...]` und `'...'` für Delimited Identifier, schließt aber die Zeichen `! . ' [] "` und Leerzeichen am Anfang aus.

- Wenn man z.B. in Oracle schreibt:

```
SELECT * FROM STUDENTEN WHERE VORNAME = "Lisa"
```

Fehler: `"Lisa"` ist ein ungültiger Spaltenname.

Delimited Identifier werden normalerweise nur verwendet, um ausgegebene Spaltennamen umzubenennen (oder wenn Spaltennamen in einer neuen DBMS-Version zu reservierten Wörtern werden).

Lexikalische Fehler

- Anführungszeichen, z.B. **"Lisa"**, für String-Literale verwenden (Delimited Identifier, kein String).

Manche Systeme erlauben "...", aber das verletzt den Standard.

- Hochkommas für Zahlen verwenden, z.B. **'123'**.

Das sollte einen Typfehler geben. Das DBMS könnte jedoch einfach den Typ von einem der Operanden konvertieren. Da < usw. für Strings und Zahlen anders definiert ist, kann dies gefährlich sein und sollte vermieden werden. Z.B. '12' < '3'.

- Reservierte Wörter als Tabellen-, Spalten- oder Tupelvariablenamen verwenden.

Die Fehlermeldung könnte seltsam sein (nicht verständlich). Daher sollte man diese Möglichkeit im Auge behalten.

SQL-Anfragen: Begrenzung

- In Oracle SQL*Plus muss jedes SQL-Statement mit einem Semikolon “;” abgeschlossen werden.

Da SQL-Statements über mehrere Zeilen gehen können, ist dies notwendig, damit SQL*Plus weiß, wann das SQL-Statement beendet ist. Auch wenn SQL in C-Programme eingebettet ist, wird das Semikolon als Begrenzer verwendet.

- Aber eigentlich gehört das Semikolon nicht zum SQL-Statement.

Z.B. ist in dem Anfrage-Analyse-Fenster von MS SQL Server kein Semikolon erforderlich. Es könnte sogar ein Fehler sein, wie im Kommandozeilen-Interface von DB2. Auch wenn SQL-Statements als Strings an Prozeduren übermittelt werden, wie z.B. in ODBC, ist kein Semikolon erforderlich.