Kapitel 2: Einstieg in SQL

- SQL (Structured Query Language) ist die in der Praxis am weitesten verbreitete Datenbanksprache für relationale Datenbanken.
- Die Historie von SQL geht zurück bis 1974, die Anfangszeit der Entwicklung relationaler Datenbanken.
- Alles begann mit SEQUEL, der *Structured English Query Language*.
- Der Sprachumfang von SQL ist einer permanenten Weiterentwicklung und Standardisierung unterworfen. Derzeit relevant sind der Stand von 1992, 1999, 2003 und 2008 entsprechend bezeichnet mit SQL-92, SQL:1999, SQL:2003 und SQL:2008.

Ein *Anfrageausdruck* in SQL besteht aus einer `SELECT`-Klausel, gefolgt von einer `FROM`-Klausel, gefolgt von einer `WHERE`-Klausel.

SFW-Ausdruck

```
SELECT A_1,...,A_n   (...Attribute der Ergebnisrelation)
   FROM R_1,...,R_m  (...benötigte Relationen)
   WHERE F           (...Auswahlbedingung)
```

2.1 Beispiels-Datenbank

Mondial-Datenbank Teil 1

Land

LName	LCode	HStadt	Fläche
Austria	A	Vienna	84
Egypt	ET	Cairo	1001
France	F	Paris	547
Germany	D	Berlin	357
Italy	I	Rome	301
Russia	RU	Moscow	17075
Switzerland	CH	Bern	41
Turkey	TR	Ankara	779

Provinz

PName	LCode	Fläche
Baden	D	15
Bavaria	D	70,5
Berlin	D	0,9
Ile de France	F	12
Franken	D	null
Lazio	I	17

Stadt

SName	LCode	PName	Einwohner	LGrad	BGrad
Berlin	D	Berlin	3472	13,2	52,45
Freiburg	D	Baden	198	7,51	47,59
Karlsruhe	D	Baden	277	8,24	49,03
Munich	D	Bavaria	1244	11,56	48,15
Nuremberg	D	Franken	495	11,04	49,27
Paris	F	Ile de France	2125	2,48	48,81
Rome	I	Lazio	2546	12,6	41,8

Mondial-Datenbank Teil 2

Lage

LCode	Kontinent	Prozent
D	Europe	100
F	Europe	100
TR	Asia	68
TR	Europe	32
ET	Africa	90
ET	Asia	10
RU	Asia	80
RU	Europe	20

Mitglied

LCode	Organisation	Art
A	EU	member
D	EU	member
D	WEU	member
ET	UN	member
I	EU	member
I	NAM	guest
TR	UN	member
TR	CERN	observer

2.2 Einfache Anfragen

. . . . . Anfragen über einer Relation: *gesamter Inhalt*.

Gib den vollständigen Inhalt der Tabelle `Stadt`.

```
SELECT * FROM Stadt
```

Stadt

SName	LCode	PName	Einwohner	LGrad	BGrad
Berlin	D	Berlin	3472	13,2	52,45
Freiburg	D	Baden	198	7,51	47,59
Karlsruhe	D	Baden	277	8,24	49,03
Munich	D	Bavaria	1244	11,56	48,15
Nuremberg	D	Franken	495	11,04	49,27
Paris	F	Ile de France	2125	2,48	48,81
Rome	I	Lazio	2546	12,6	41,8

. . . . . Anfragen über einer Relation: *einzelne Spalten*.

In welchen Provinzen liegen die Städte der Tabelle `Stadt`?

```
SELECT PName FROM Stadt
```

PName
Berlin
Baden
Baden
Bavaria
Franken
Ile de France
Lazio

In welchen *unterschiedlichen* Provinzen liegen die Städte der Tabelle `Stadt`?

```
SELECT DISTINCT PName FROM Stadt
```

PName
Berlin
Baden
Bavaria
Franken
Ile de France
Lazio

. . . . . Anfragen über einer Relation: *einzelne Zeilen*.

Stadt

SName	LCode	PName	Einwohner	LGrad	BGrad
Berlin	D	Berlin	3472	13,2	52,45
Freiburg	D	Baden	198	7,51	47,59
Karlsruhe	D	Baden	277	8,24	49,03
Munich	D	Bavaria	1244	11,56	48,15
Nuremberg	D	Franken	495	11,04	49,27
Paris	F	Ile de France	2125	2,48	48,81
Rome	I	Lazio	2546	12,6	41,8

Wie heißen die Städte, die mehr als 1 Mio. Einwohner haben?

```
SELECT * FROM Stadt
   WHERE Einwohner > 1000
```

Stadt

SName	LCode	PName	Einwohner	LGrad	BGrad
Berlin	D	Berlin	3472	13,2	52,45
Munich	D	Bavaria	1244	11,56	48,15
Paris	F	Ile de France	2125	2,48	48,81
Rome	I	Lazio	2546	12,6	41,8

. . . . . Anfragen über einer Relation: *geänderte Spaltenbezeichnungen und neue Spalten*.

Stadt

SName	LCode	PName	Einwohner	LGrad	BGrad
Berlin	D	Berlin	3472	13,2	52,45
Freiburg	D	Baden	198	7,51	47,59
Karlsruhe	D	Baden	277	8,24	49,03
Munich	D	Bavaria	1244	11,56	48,15
Nuremberg	D	Franken	495	11,04	49,27
Paris	F	Ile de France	2125	2,48	48,81
Rome	I	Lazio	2546	12,6	41,8

Kennzeichne die Tatsache, dass eine Stadt mehr als 1 Mio. Einwohner hat, durch den Wert `'Großstadt'` einer neuen Spalte mit Namen `StadtKategorie`; die Spalte `SName` soll die Bezeichnung `Stadt` erhalten.

```
SELECT SName AS Stadt, 'Großstadt' AS StadtKategorie FROM Stadt
   WHERE Einwohner > 1000
```

Stadt	StadtKategorie
Berlin	Großstadt
Munich	Großstadt
Paris	Großstadt
Rome	Großstadt

. . . . . Anfragen über einer Relation: *Pattern Matching* (a).

Land

LName	LCode	HStadt	Fläche
Austria	A	Vienna	84
Egypt	ET	Cairo	1001
France	F	Paris	547
Germany	D	Berlin	357
Italy	I	Rome	301
Russia	RU	Moscow	17075
Switzerland	CH	Bern	41
Turkey	TR	Ankara	779

Erstelle eine Namensliste der Länder, deren Namen mit 'G' anfängt oder mit 'y' aufhört?

```
SELECT LName FROM Land WHERE LName LIKE 'G%' OR LName LIKE '%y'
```

LName
Germany
Italy
Turkey

. . . . . Anfragen über einer Relation: *Pattern Matching* (b).

Land

LName	LCode	HStadt	Fläche
Austria	A	Vienna	84
Egypt	ET	Cairo	1001
France	F	Paris	547
Germany	D	Berlin	357
Italy	I	Rome	301
Russia	RU	Moscow	17075
Switzerland	CH	Bern	41
Turkey	TR	Ankara	779

Erstelle eine Namensliste der Länder, deren dritter Buchstabe des Namen 'y' ist?

```
SELECT LName FROM Land WHERE LName LIKE '__y%'
```

LName
Egypt

. . . . . Anfragen über mehreren Relationen.

Land

LName	LCode	HStadt	Fläche
Austria	A	Vienna	84
Egypt	ET	Cairo	1001
France	F	Paris	547
Germany	D	Berlin	357
Italy	I	Rome	301
Russia	RU	Moscow	17075
Switzerland	CH	Bern	41
Turkey	TR	Ankara	779

Stadt

SName	LCode	PName	Einwohner	LGrad	BGrad
Berlin	D	Berlin	3472	13,2	52,45
Freiburg	D	Baden	198	7,51	47,59
Karlsruhe	D	Baden	277	8,24	49,03
Munich	D	Bavaria	1244	11,56	48,15
Nuremberg	D	Franken	495	11,04	49,27
Paris	F	Ile de France	2125	2,48	48,81
Rome	I	Lazio	2546	12,6	41,8

Gib zu jedem Land die zugehörigen Städte an.

```
SELECT DISTINCT Land.LName AS Land, Stadt.SName AS Stadt
    FROM Land, Stadt
    WHERE Land.LCode = Stadt.LCode
```

Land	Stadt
Germany	Berlin
Germany	Freiburg
Germany	Karlsruhe
Germany	Munich
Germany	Nuremberg
France	Paris
Italy	Rome

. . . . . Anfragen über mehreren Relationen - was passiert, wenn man die Auswahlbedingung entfernt?

```
SELECT DISTINCT Land.LName AS Land, Stadt.SName AS Stadt
   FROM Land, Stadt
```

?

... man berechnet das kartesische Produkt der angegebenen Relationen!

. . . . . Anfragen über mehreren Relationen mit Auswahlbedingung: Semantik

Land

LName	LCode	HStadt	Fläche
Austria	A	Vienna	84
Egypt	ET	Cairo	1001
France	F	Paris	547
Germany	D	Berlin	357
Italy	I	Rome	301
Russia	RU	Moscow	17075
Switzerland	CH	Bern	41
Turkey	TR	Ankara	779

Stadt

SName	LCode	PName	Einwohner	LGrad	BGrad
Berlin	D	Berlin	3472	13,2	52,45
Freiburg	D	Baden	198	7,51	47,59
Karlsruhe	D	Baden	277	8,24	49,03
Munich	D	Bavaria	1244	11,56	48,15
Nuremberg	D	Franken	495	11,04	49,27
Paris	F	Ile de France	2125	2,48	48,81
Rome	I	Lazio	2546	12,6	41,8

$\Longrightarrow$

Land	Stadt
Germany	Berlin
Germany	Freiburg
Germany	Karlsruhe
Germany	Munich
Germany	Nuremberg
France	Paris
Italy	Rome

Gib zu jedem Land die zugehörigen Städte an.

```
SELECT DISTINCT Land.LName AS Land, Stadt.SName AS Stadt
   FROM Land, Stadt
   WHERE Land.LCode = Stadt.LCode
```

intuitive deklarative Semantik

Das Ergebnis besteht aus denjenigen Tupeln des kartesische Produktes `Land` $\times$ `Stadt`, die die Bedingung der `WHERE`-Klausel erfüllen, wobei nur die Werte der Attribute der `SELECT`-Klausel angegeben werden.

Algorithmus (nested-loop-Semantik)

FOR each Tupel t_1 in Relation *Land* DO
  FOR each Tupel t_2 in Relation *Stadt* DO
    IF Die WHERE-Klausel ist erfüllt nach Ersetzen der Attributnamen
      Land.*LCode*, *Stadt*.*LCode* durch die entsprechenden Werte der gerade betrachteten Tupel t_1, t_2,
    THEN Bilde ein Antwort-Tupel aus den Werten der in der
      SELECT-Klausel angegebenen Attributen *Land*.*LName*, *Stadt*.*SName* dieser Tupel t_1, t_2.

... Verwende optionale Korrelationsnamen.

```
SELECT DISTINCT S.SName, L.LName
   FROM Stadt S, Land L
   WHERE S.LCode = L.LCode
```

. . . . . Anfragen über mehreren Relationen mit Auswahlbedingung.

Land

LName	LCode	HStadt	Fläche
Austria	A	Vienna	84
Egypt	ET	Cairo	1001
France	F	Paris	547
Germany	D	Berlin	357
Italy	I	Rome	301
Russia	RU	Moscow	17075
Switzerland	CH	Bern	41
Turkey	TR	Ankara	779

Stadt

SName	LCode	PName	Einwohner	LGrad	BGrad
Berlin	D	Berlin	3472	13,2	52,45
Freiburg	D	Baden	198	7,51	47,59
Karlsruhe	D	Baden	277	8,24	49,03
Munich	D	Bavaria	1244	11,56	48,15
Nuremberg	D	Franken	495	11,04	49,27
Paris	F	Ile de France	2125	2,48	48,81
Rome	I	Lazio	2546	12,6	41,8

Gib zu jedem Land die zugehörigen Städte mit mehr als 1 Mio Einwohner an.

```
SELECT DISTINCT L.LName AS Land, S.SName AS Stadt
   FROM Land L, Stadt S
   WHERE L.LCode = S.LCode AND S.Einwohner > 1000
```

Land	Stadt
Germany	Berlin
Germany	Munich
France	Paris
Italy	Rome

. . . . . Anfragen mehrmals über dieselbe Relation.

Lage

LCode	Kontinent	Prozent
D	Europe	100
F	Europe	100
TR	Asia	68
TR	Europe	32
ET	Africa	90
ET	Asia	10
RU	Asia	80
RU	Europe	20

Bestimme alle Paare von Ländern, die im selben Kontinent liegen.

```
SELECT DISTINCT L1.LCode AS Land1, L2.LCode AS Land2
   FROM Lage L1, Lage L2
   WHERE L1.Kontinent = L2.Kontinent
   AND L1.LCode < L2.LCode
```

(1) FROM Lage L1, Lage L2

Lage L1 (8 Tupel)

LCode	Kontinent	Prozent
D	Europe	100
F	Europe	100
TR	Asia	68
TR	Europe	32
ET	Africa	90
ET	Asia	10
RU	Asia	80
RU	Europe	20

$\times$

Lage L2 (8 Tupel)

LCode	Kontinent	Prozent
D	Europe	100
F	Europe	100
TR	Asia	68
TR	Europe	32
ET	Africa	90
ET	Asia	10
RU	Asia	80
RU	Europe	20

$=$

$L_1 \times L_2$ (64 Tupel)

L1.LCode	L1.Kontinent	L1.Prozent	L2.LCode	L2.Kontinent	L2.Prozent
D	Europe	100	D	Europe	100
D	Europe	100	F	Europe	100
...	...	...	...	...	...
D	Europe	100	RU	Asia	80
...	...	...	...	...	...
RU	Asia	20	D	Europe	100
...	...	...	...	...	...
RU	Europe	20	RU	Europe	20

(2)
```
FROM Lage L1, Lage L2
WHERE L1.Kontinent = L2.Kontinent
```

$L_1 \times L_2$ (**26** Tupel)

L1.LCode	L1.Kontinent	L1.Prozent	L2.LCode	L2.Kontinent	L2.Prozent
D	Europe	100	D	Europe	100
D	Europe	100	F	Europe	100
...	...	...	...	...	...
D	Europe	100	RU	Europe	80
...	...	...	...	...	...
RU	Europe	20	D	Europe	100
...	...	...	...	...	...
RU	Europe	20	RU	Europe	20

(3)
```
FROM Lage L1, Lage L2
WHERE L1.Kontinent = L2.Kontinent
AND L1.LCode < L2.LCode
```

(**9** Tupel)

L1.LCode	L1.Kontinent	L1.Prozent	L2.LCode	L2.Kontinent	L2.Prozent
D	Europe	100	F	Europe	100
ET	Asia	10	TR	Asia	68
RU	Asia	80	TR	Asia	68
D	Europe	100	TR	Europe	32
F	Europe	100	TR	Europe	32
RU	Europe	20	TR	Europe	32
ET	Asia	10	RU	Asia	80
D	Europe	100	RU	Europe	20
F	Europe	100	RU	Europe	20

(4)
```
SELECT DISTINCT L1.LCode AS Land1,
                L2.LCode AS Land2
FROM Lage L1, Lage L2
WHERE L1.Kontinent = L2.Kontinent
AND L1.LCode < L2.LCode
```

(8 Tupel)

Land1	Land2
D	F
ET	TR
RU	TR
D	TR
F	TR
ET	RU
D	RU
F	RU

Auswertung einfacher SQL-Anfragen

SFW-Ausdruck

```
SELECT A_1,...,A_n   (...Attribute der Ergebnisrelation)
   FROM R_1,...,R_m  (...benötigte Relationen)
   WHERE F           (...Auswahlbedingung)
```

Algorithmus (nested-loop-Semantik)

FOR each Tupel t_1 in Relation R_1 DO
  FOR each Tupel t_2 in Relation R_2 DO
    $\vdots$
      FOR each Tupel t_m in Relation R_m DO
        IF Die WHERE-Klausel ist erfüllt nach Ersetzen der Attributnamen in F durch die entsprechenden Werte der gerade betrachteten Tupel $t_1, \ldots, t_m$.
        THEN Bilde ein Antwort-Tupel aus den Werten der in der SELECT-Klausel angegebenen Attributen $A_1, \ldots, A_n$ bezüglich der gerade betrachteten Tupel $t_1, \ldots, t_m$.

Verbund (engl. join)

Anfragen mit mehreren Relationen in der FROM-Klausel sind sogenannte *Verbund*-Anfragen (engl. join-queries).

Gib zu jedem Land die zugehörigen Städte an.

```
SELECT DISTINCT S.SName, L.LName
   FROM Stadt S, Land L
   WHERE S.LCode = L.LCode
```

.... explizit als Verbund:

```
SELECT DISTINCT S.SName, L.LName
   FROM Stadt S JOIN Land L
   ON S.LCode = L.LCode
```

.... wird Gleichheit über Attributen mit identischen Bezeichnern gefordert redet man von einem *natürlichen Verbund* (engl. natural join) und schreibt kürzer:

```
SELECT DISTINCT S.SName, L.LName
   FROM Stadt S NATURAL JOIN Land L
```

.... Spezialfall *kartesisches Produkt*:

```
SELECT DISTINCT S.SName, L.LName
   FROM Stadt S CROSS JOIN Land L
```

Sortierung.

Sortiere die Zeilen der Tabelle `Stadt` aufsteigend nach `LCode` und für gemeinsame Werte zu LCode absteigend nach dem Breitengrad.

```
SELECT * FROM Stadt
   ORDER BY LCode ASC, BGrad DESC
```

2.3 Basis-Datentypen und Built-in Functions

Basis-Datentypen

SQL bietet eine Fülle von unterschiedlichen Datentypen an, mittels derer die Wertebereiche der Spalten einer Tabelle festgelegt werden können.

- `INTEGER`, `SMALLINT`
- `NUMERIC`, `DECIMAL`. Angabe Anzahl Ziffern insgesamt und Anzahl Kommastellen.
- `REAL`, `DOUBLE PRECISION`, `FLOAT`
- `CHARACTER`, `CHARACTER VARYING`, `CHARACTER LARGE OBJECT`
- `BIT`, `BIT VARYING`, `BINARY LARGE OBJECT`
- `BOOLEAN`
- `DATE`, `TIME`, `TIMESTAMP`, `INTERVALL`, ...

Built-in Functions

Zur Manipulation der Werte der einzelnen Datentypen können spezielle Funktionen verwendet werden. Während der SQL-Standard hier sehr sparsam ist, bieten die einzelnen Hersteller von Datenbanksystemen einen großen Vorrat an. Die folgenden Beispiele basieren auf *Oracle Database SQL Reference 10g*. Es handelt sich um eine Auswahl sowohl der Funktionstypen als auch der jeweiligen Funktionen des jeweiligen Typs.

- Numeric:
  ABS, ACOS, ASIN, ATAN, ATAN2, BITAND, CEIL, COS, COSH, EXP, FLOOR, LN, LOG, MOD, POWER, REMAINDER, ROUND, SIGN, SIN, SINH, SQRT, TAN, TANH, TRUNC
- Character:
  CHR, CONCAT, INITCAP, LOWER, LPAD, LTRIM, REGEXP_REPLACE, REGEXP_SUBSTR, REPLACE, RPAD, RTRIM, SUBSTR, TRANSLATE, TREAT, TRIM, UPPER

  ASCII, INSTR, LENGTH, REGEXP_INSTR
- Aggregate:
  AVG, COLLECT, CORR, COUNT, FIRST, LAST, MAX, MEDIAN, MIN, RANK, REGRESSION, STATS_BINOMIAL_TEST, STDDEV, SUM, VARIANCE

CAST und CASE

- CAST erlaubt eine *explizite* Typkonversionen zwischen unterschiedlichen Typen;
- CASE beschränkt die Konversionen im Wesentlichen auf Wertumwandlungen innerhalb eines Typs.

Erstelle eine Tabelle mit Spalten LName und LCode, die zu jedem Land den Namen auf die ersten zwei Zeichen reduziert und anstatt 'D' den Wert 'BRD' von LCode verwendet. Beide Werte sollen konkateniert werden wobei getrennt durch '>'.

```
SELECT CONCAT(CONCAT(CAST(LName AS VARCHAR(2)), '>'),
   CASE LCode
      WHEN 'D' THEN 'BRD'
      ELSE LCode
   END)
   FROM Land
```

2.4 empfohlene Lektüre

History Repeats Itself: Sensible and NonsenSQL Aspects of the NoSQL Hoopla

C. Mohan
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120, USA
+1 408 927 1733
cmohan@us.ibm.com

ABSTRACT

In this paper, I describe some of the recent developments in the database management area, in particular the NoSQL phenomenon and the hoopla associated with it. The goal of the paper is not to do an exhaustive survey of NoSQL systems. The aim is to do a broad brush analysis of what these developments mean - the good and the bad aspects! Based on my more than three decades of database systems work in the research and product arenas, I will outline what are many of the pitfalls to avoid since there is currently a mad rush to develop and adopt a plethora of NoSQL systems in a segment of the IT population, including the research community. In rushing to develop these systems to overcome some of the shortcomings of the relational systems, many good principles of the latter, which go beyond the relational model and the SQL language, have been left by the wayside. Now many of the features that were initially discarded as unnecessary in the NoSQL systems are being brought in, but unfortunately in ad hoc ways. Hopefully, the lessons learnt over three decades with relational and other systems would not go to waste and we wouldn't let history repeat itself with respect to simple minded approaches leading to enormous pain later on for developers as well as users of the NoSQL systems!

Caveat: What I express in this paper are my personal opinions and they do not necessarily reflect the opinions of my employer.

1. INTRODUCTION

Over the last few years, many new types of database management systems (DBMSs) have emerged which are being labeled as NoSQL systems. These systems have different design points compared to the relational DBMSs (RDBMSs), like DB2 and Oracle, which have now existed for about 3 decades with SQL as their query language. The NoSQL wave was initially triggered not by the traditional software product companies but by the non-traditional, namely the Web 2.0, companies like Amazon, Facebook, Google and Yahoo. Some of the NoSQL systems which have emerged from such companies are BigTable, Cassandra [14], Dynamo and HBase. Different types of NoSQL systems have emerged. Some of the types are: key-value stores, document DBMSs, graph DBMSs and column-oriented stores. There are older non-relational systems that are also being classified these days as NoSQL systems: object-oriented (OODBMSs) and XML DBMSs.

Of late, a good amount of the momentum for the NoSQL developments is also being provided by a number of smaller software companies which are fuelled by venture funding [22] and/or the open-source movement. Some of the systems emerging from such efforts are Aerospike (formerly Citrusleaf), Couchbase, MongoDB and Riak. Oracle has also enhanced its previously

1

[1] In: *Proceedings Conference EDBT/ICDT, March 18-22, 2013*. Findet man im Web.