Teil VIII

Fortgeschrittene Konzepte in SQL

Fortgeschrittene Konzepte in SQL

1. Rekursion in SQL
2. Fortgeschrittenes SQL
3. Prozedurale SQL-Erweiterungen: SQL/PSM

Lernziele für heute ...

- Verständnis der Formulierung und Auswertung rekursiver Anfragen
- Beispiele komplexer SQL-Anfragen
- Prozedurale SQL-Erweiterungen: SQL/PSM

Rekursion in SQL

Benannte Anfragen

- Anfrageausdruck, der in der Anfrage mehrfach referenziert werden kann

```
with anfrage-name [(spalten-liste) ] as
( anfrage-ausdruck )
```

- Anfrage ohne `with`

```
select *
from WEINE
where Jahrgang >= (
      select avg(Jahrgang) from WEINE) - 2
   and Jahrgang <= (
      select avg(Jahrgang) from WEINE) + 2
```

Benannte Anfragen /2

- Anfrage mit `with`

```
with ALTER(Durchschnitt) as (
   select avg(Jahrgang) from WEINE)
select *
from WEINE, ALTER
where Jahrgang >= Durchschnitt - 2
and Jahrgang <= Durchschnitt + 2
```

Rekursive Anfragen

- Anwendung: **Bill of Material**-Anfragen, Berechnung der **transitiven Hülle** (Flugverbindungen etc.)
- Beispiel:

BUSLINIE

Abfahrt	Ankunft	Distanz
Nuriootpa	Penrice	7
Nuriootpa	Tanunda	7
Tanunda	Seppeltsfield	9
Tanunda	Bethany	4
Bethany	Lyndoch	14

Rekursive Anfrage: Busfahrt mit max. 2x Umsteigen


```
select Abfahrt, Ankunft from BUSLINIE
where Abfahrt = 'Nuriootpa'
      union
select B1.Abfahrt, B2.Ankunft
from BUSLINIE B1, BUSLINIE B2
where B1.Abfahrt = 'Nuriootpa'
   and B1.Ankunft = B2.Abfahrt
      union
select B1.Abfahrt, B3.Ankunft
from BUSLINIE B1, BUSLINIE B2, BUSLINIE B3
where B1.Abfahrt = 'Nuriootpa'
   and B1.Ankunft = B2.Abfahrt
   and B2.Ankunft = B3.Abfahrt
```

Rekursion in SQL:2003

- Formulierung über erweiterte `with recursive`-Anfrage
- Notation

```
with recursive rekursive-tabelle as (
    anfrage-ausdruck -- rekursiver Teil
)
[traversierungsklausel] [zyklusklausel]
anfrage-ausdruck -- nicht rekursiver Teil
```

- nicht rekursiver Teil: Anfrage auf Rekursionstabelle

Rekursion in SQL:2003 /2

- rekursiver Teil:

```
-- Initialisierung
select ...
from tabelle where ...
-- Rekursionsschritt
union all
select ...
from tabelle, rekursionstabelle
where rekursionsbedingung
```

Rekursion in SQL:2003: Beispiel

```
with recursive TOUR(Abfahrt, Ankunft) as (
   select Abfahrt, Ankunft
   from BUSLINIE
   where Abfahrt = 'Nuriootpa'
      union all
   select T.Abfahrt, B.Ankunft
   from TOUR T, BUSLINIE B
   where T.Ankunft = B.Abfahrt)
select distinct * from TOUR
```

Schrittweiser Aufbau der Rekursionstabelle TOUR

Initialisierung

Abfahrt	Ankunft
Nuriootpa	Penrice
Nuriootpa	Tanunda

Schritt 1

Abfahrt	Ankunft
Nuriootpa	Penrice
Nuriootpa	Tanunda
Nuriootpa	Seppeltsfield
Nuriootpa	Bethany

Schritt 2

Abfahrt	Ankunft
Nuriootpa	Penrice
Nuriootpa	Tanunda
Nuriootpa	Seppeltsfield
Nuriootpa	Bethany
Nuriootpa	Lyndoch

Rekursion: Beispiel /2

- arithmetische Operationen im Rekursionsschritt

```
with recursive TOUR(Abfahrt, Ankunft, Strecke) as (
   select Abfahrt, Ankunft, Distanz as Strecke
   from BUSLINIE
   where Abfahrt = 'Nuriootpa'
      union all
   select T.Abfahrt, B.Ankunft,
      Strecke + Distanz as Strecke
   from TOUR T, BUSLINIE B
   where T.Ankunft = B.Abfahrt)
select distinct * from TOUR
```

Sicherheit rekursiver Anfragen

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

- Sicherheit (= Endlichkeit der Berechnung) ist wichtige Anforderung an Anfragesprache
- Problem: Zyklen bei Rekursion

```
insert into BUSLINIE (Abfahrt, Ankunft, Distanz)
   values ('Lyndoch', 'Tanunda', 12)
```

- Behandlung in SQL
  - Begrenzung der Rekursionstiefe
  - Zyklenerkennung

Sicherheit rekursiver Anfragen /2

- Einschränkung der Rekursionstiefe

```
with recursive TOUR(Abfahrt, Ankunft, Umsteigen) as (
   select Abfahrt, Ankunft, 0
   from BUSLINIE
   where Abfahrt = 'Nuriootpa'
      union all
   select T.Abfahrt, B.Ankunft, Umsteigen + 1
   from TOUR T, BUSLINIE B
   where T.Ankunft = B.Abfahrt and Umsteigen < 2)
select distinct * from TOUR
```

Sicherheit durch Zyklenerkennung

- Zyklusklausel
  - beim Erkennen von Duplikaten im Berechnungspfad von *attrib*: `Zyklus = '*'` (Pseudospalte vom Typ `char(1)`)
  - Sicherstellen der Endlichkeit des Ergebnisses „von Hand“

```
cycle attrib set marke to '*' default '-'
```

Sicherheit durch Zyklenerkennung

```
with recursive TOUR(Abfahrt, Ankunft, Weg) as (
   select Abfahrt, Ankunft,
      Abfahrt || '-' || Ankunft as Weg
   from BUSLINIE where Abfahrt = 'Nuriootpa'
      union all
   select T.Abfahrt, B.Ankunft,
      Weg || '-' || B.Ankunft as Weg
   from TOUR T, BUSLINIE B
where T.Ankunft = B.Abfahrt)
cycle Ankunft set Zyklus to '*' default '-'
select Weg, Zyklus from TOUR
```

Sicherheit durch Zyklenerkennung /2

Weg	Zyklus
Nuriootpa-Penrice	-
Nuriootpa-Tanunda	-
Nuriootpa-Tanunda-Seppeltsfield	-
Nuriootpa-Tanunda-Bethany	-
Nuriootpa-Tanunda-Bethany-Lyndoch	-
Nuriootpa-Tanunda-Bethany-Lyndoch-Tanunda	*

SQL-Versionen

- Geschichte
  - SEQUEL (1974, IBM Research Labs San Jose)
  - SEQUEL2 (1976, IBM Research Labs San Jose)
  - SQL (1982, IBM)
  - ANSI-SQL (SQL-86; 1986)
  - ISO-SQL (SQL-89; 1989; drei Sprachen Level 1, Level 2, + IEF)
  - (ANSI / ISO) SQL2 (als SQL-92 verabschiedet)
  - (ANSI / ISO) SQL3 (als SQL:1999 verabschiedet)
  - (ANSI / ISO) SQL:2003 . . . aktuell SQL:2011
- trotz Standardisierung: teilweise Inkompatibilitäten zwischen Systemen der einzelnen Hersteller

Fortgeschrittenes SQL

Fortgeschrittenes SQL

- SQL ist weitaus mächtiger als SFW
- Unterstützung von komplexen Transformationen, Manipulationen und Analysen auch extrem großer Datenbestände
- Turing-Vollständigkeit durch prozedurale Erweiterungen (SQL/PSM, PL/SQL, Transact-SQL, . . .)
- nachfolgend: erweiterte SQL-Konstrukte mit Beispielen als Muster für Problemlösungen

Pivotierung

Problem: Zeilen in Spalten bzw. umgekehrt umwandeln

LDATA

Quartal	Jahr	Umsatz
1	2020	12
2	2020	10
3	2020	11
4	2020	9
1	2021	13
2	2021	12

$\Longleftrightarrow$

RDATA

Jahr	Q1	Q2	Q3	Q4
2020	12	10	11	9
2021	13	12	$\perp$	$\perp$

Pivotierung /2

Lösung (Zeilen in Spalten): Aggregatfilter

```
select Jahr, sum(Umsatz) filter (where Quartal=1) Q1,
   sum(Umsatz) filter (where Quartal=2) Q2,
   sum(Umsatz) filter (where Quartal=3) Q3,
   sum(Umsatz) filter (where Quartal=4) Q4
from LDATA
group by Jahr
```

- Warum `sum` und `group by`?
- `filter` nicht in allen Systemen verfügbar $\rightsquigarrow$ Alternative über `case`

Pivotierung /3

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

Lösung (Spalten in Zeilen): `union all`

```
select 1 as Quartal, Jahr, Q1 as Umsatz from RDATA
   union all
select 2 as Quartal, Jahr, Q2 as Umsatz from RDATA
   union all
select 3 as Quartal, Jahr, Q3 as Umsatz from RDATA
   union all
select 4 as Quartal, Jahr, Q4 as Umsatz from RDATA
```

- Warum `union all` statt `union`?

Telefonverzeichnis

Problem: Pivotierung mit fehlenden Werten → Ausgabe der Telefonnummern zu einer Person in einer Zeile

`Personal`

PNr	Vorname	Name
101	Bernd	K.
102	Simone	S.
103	Franz	M.

`Kontakt`

PNr	KTyp	Nummer
101	Mobil	0175...
101	Arbeit	5556
102	Mobil	0165...
103	Arbeit	5557

- zweifacher Verbund mit `Kontakt`?

Telefonverzeichnis

Lösung: linker äußerer Verbund

```
select Vorname, Name, K.Nummer as Mobil, K2.Nummer as Arbeit
from (Personal P left outer join Kontakt K
      on P.PNr = K.PNr and K.KTyp = 'Mobil')
   left outer join Kontakt K2
      on P.PNr = K2.PNr and K.KTyp = 'Arbeit'
```

Top-k

Problem: Beschränkung der Ergebnismenge auf k Elemente, z.B. nach Sortierung

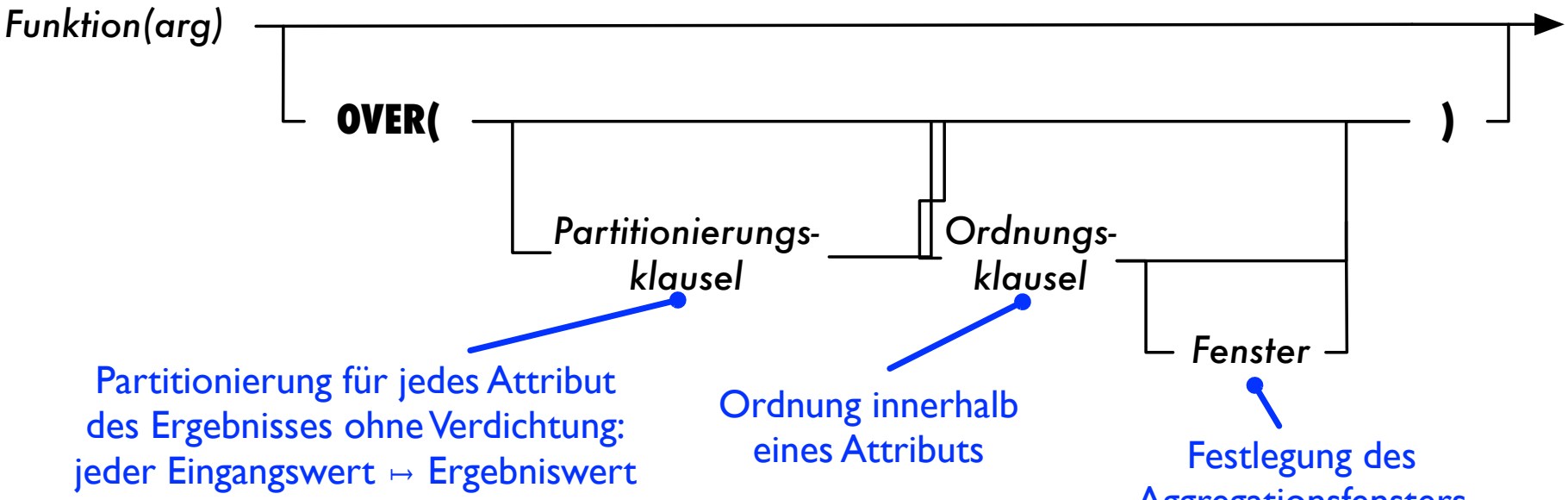
RACE

Name	Distanz	Zeit
Klaus	HM	1:20
Bernd	HM	1:40
Tanja	M	3:58
Franz	SM	5:42
Martina	M	3:05
Heike	HM	1:34
Corinna	SM	5:53
Jens	M	2:51
Herbert	SM	6:07

Lösungen:

- proprietäre Erweiterungen wie z.B. `limit`
- Window-Funktionen in SQL + `rank()`

Window-Funktionen: Syntax



Window-Funktionen: Prinzip

- Anzahl der Tupel, die in ein Ergebnistupel eingehen entspricht Position des Tupels bzgl. gegebener Ordnung
- Eingangstupel t_i, Ergebnistupel s_i

$$
\begin{array}{lcll}
t_1 & \longrightarrow & \texttt{sum}(\{t_1\}) & \longrightarrow & s_1 \\
t_2 & \longrightarrow & \texttt{sum}(\{t_1, t_2\}) & \longrightarrow & s_2 \\
t_3 & \longrightarrow & \texttt{sum}(\{t_1, t_2, t_3\}) & \longrightarrow & s_3 \\
 & & \ldots & &
\end{array}
$$

- Schrittweise Vergrößerung des Analyse**fensters**

Window-Funktionen: Anwendung

Was liefert ...

```
select count(*) over()
from RACE
```

count
9
9
9
...

und ...

```
select count(*) over(order by Zeit)
from RACE
```

count
1
2
3
...
9

Ranking-Funktionen

- `rank()`: liefert Rang eines Tupels bzgl. vorgegebener Ordnung innerhalb der Partition
  - Bei Duplikaten gleicher Rang (mit Lücken)
- `dense_rank()`: wie `rank()`, jedoch ohne Lücken

```
select Name, Zeit, rank() over (order by Zeit)
from RACE
where Distanz = 'M'
```

Name	Zeit	rank
Jens	2:51	1
Martina	3:05	2
Tanja	3:58	3

Top-k mit Ranking-Funktion

```
select * from (
   select Name, Zeit, rank() over (order by Zeit) Rang
   from RACE
   where Distanz = 'M') T
where Rang <= 2
```

- Schachtelung erforderlich, weil Projektion **nach** Selektion ausgeführt wird: `Rang` in der `where`-Klausel der inneren Anfrage nicht verfügbar
- Top-2: `Rang <= 2`

Top-k mit Ranking-Funktion /2

- Top-2 pro Distanz durch Gruppierung **in** der Window-Funktion über `partition by`

```
select * from (
   select Name, Distanz, Zeit,
      rank() over (partition by Distanz order by Zeit) Rang
   from RACE) T
where Rang <= 2
```

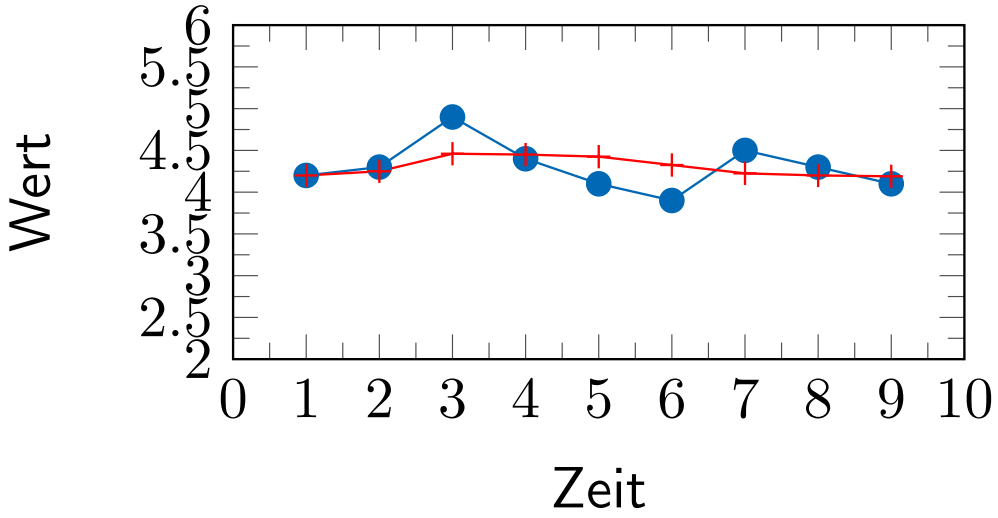
Name	Distanz	Zeit	Rang
Klaus	HM	1:20	1
Heike	HM	1:34	2
Jens	M	2:51	1
Martina	M	3:05	2
Franz	SM	5:42	1
Corinna	SM	5:53	2

Gleitender Durchschnitt

Problem: Glättung von Daten- bzw. Zeitreihen $\rightsquigarrow$ Berechnung des Mittelwertes über ein Fenster (=Ausschnitt) der Datenwerte

SERIES

Zeit	Wert
0:01	4.2
0:02	4.3
0:03	4.9
0:04	4.4
0:05	4.1
0:06	3.9
0:07	4.5
0:08	4.3
0:09	4.1



Lösung: Window-Funktion mit Aggregationsfenster

Aggregationsfenster

explizite Angabe des Fensters

- `rows`: Anzahl der Tupel; `range`: Anzahl der wertmäßig verschiedenen Tupel

ausgehend von definierten Startpunkt bis zum aktuellen Tupel

- `unbounded preceding`: erstes Tupel der jeweiligen Partition; `n preceding`: n-ter Vorgänger relativ zur aktuellen Position; `current row`: aktuelles Tupel (nur mit `range` und Duplikaten sinnvoll)

Angabe der unteren und oberen Schranken

```
BETWEEN untereGrenze AND obereGrenze
```

Spezifikation der Grenzen

- `unbounded preceding`, `unbounded following`, `n preceding`, `n following`, `current row`

obereGrenze muss höhere Position als *untereGrenze* spezifizieren

Gleitender Durchschnitt /2

Glättung der Datenreihe über Fenster von 3 Werten

```
select Zeit,
       avg(Wert) over(order by Zeit asc rows 3 preceding)
from SERIES
```

Mengengleichheit

Problem: Test auf Gleichheit von zwei Mengen
Beispiel: Welche zwei Personen haben exakt die gleichen Hobbies?

Name	Hobby
Kevin	Schach
Kevin	Musik
Corinna	Parties
Martin	Handball
Martin	Musik
Katja	Handball
Katja	Musik

Mengengleichheit

Lösung: 7 verschiedene Varianten in Celko's SQL Puzzles; hier: Mengentheorie

- $A = B \leftrightarrow A \subset B \wedge B \subset A$
- in SQL für $\subset$ besser: $\nexists e \in A : e \notin B$ und demzufolge $A \subset B$
- über `not exists` bzw. `except`

Teilschritt: verschiedene Hobbies, z.B. für Kevin und Martina vs. Katja und Martin

```
select h3.Hobby from Hobbies as h3
where 'Kevin' = h3.Name
   except
select h4.Hobby from Hobbies as h4
where 'Martina' = h4.Name)
```

Mengengleichheit: Lösung

```
select distinct h1.Name, h2.Name
from Hobbies as h1, Hobbies as h2
where h1.Name < h2.Name -- nicht die gleiche Person
   and not exists ( -- e in h1 aber nicht in h2
      select h3.Hobby from Hobbies as h3
      where h1.Name = h3.Name
         except
      select h4.Hobby from Hobbies as h4
      where h2.Name = h4.Name)
   and not exists ( -- e in h2 aber nicht in h1
      select h5.Hobby from Hobbies as h5
      where h2.Name = h5.Name
         except
      select h6.Hobby from Hobbies as h6
      where h1.Name = h6.Name)
```

Anschlussbus

Problem: Finde die nächste Busverbindung an einer Haltestelle

Buslinie	Abfahrt	Ankunft
10	8:00	9:20
11	9:10	10:35
12	9:10	11:00
13	9:55	10:45

z.B. für aktuelle Zeit (`current_time`) oder gegebene Zeit (`time '8:00'`)

Anschlussbus

Lösung: Finde minimale Abfahrtszeit, die nach der gewünschten Zeit liegt

```
select * from Fahrplan f1
where f1.Abfahrt = (
   select min(f2.Abfahrt) from Fahrplan f2
   where f2.Abfahrt >= time '8:05')
```

mit Berechnung der Wartezeit:

```
select Buslinie, f1.Abfahrt - time '8:55'
...
```

Mode – Häufigster Wert

Problem: Bestimmung des häufigsten Wertes in einer Spalte

- Aggregatfunktion `avg` aber kein `mode`?

Name	Studiengang
Kevin	Informatik
Heike	BWL
Corinna	Mathematik
Martina	BWL
Ronny	BWL
Katharina	Informatik

Mode – Häufigster Wert

Lösung: Bestimmung der Anzahl des häufigsten Wertes + `having count(*)` bezogen auf diesen Wert

```
select Studiengang, count(*)
from Students
group by Studiengang
having count(*) >= all (
   select count(*)
   from Students group by Studiengang)
```

Sudoku

Problem: Lösen von Sudoku

- 81 Felder (9x9) mit Ziffern 1 . . . 9
- Auffüllen mit Ziffern, so dass **in jeder der je neun Zeilen, Spalten und Blöcke jede Ziffer von 1 bis 9 nur einmal auftritt**

	3							
			1	9	5			
		8					6	
8				6				
4			8					1
				2				
	6					2	8	
			4	1	9			5
							7	

Sudoku

Lösung: systematisch alle Felder mit Werten von 1 bis 9 auffüllen und prüfen
Quellen: `http://technology.amis.nl/blog/6404/oracle-rdbms-11gr2-solving-a-sudoku-using-recursive-subquery-fa`
`https://wiki.postgresql.org/wiki/Sudoku_puzzle`

Hilfsfunktionen (PostgreSQL)

- `generate_series(s,e)`: erzeugt eine Folge von Werten von `s` bis `e`
- `substr(s, start, len)`: liefert einen Substring der Länge `len` aus `s` beginnend bei `start`
- `repeat(s, n)`: erzeugt einen String durch `n`-fache Wiederholung von `s`
- `position(sub in s)`: liefert die Position von `sub` in `s`

Sudoku

Einsetzen aller Ziffern von 1 bis 9 an einem bestimmten Feld (hier: 3)

```
select substr(s.sud, 1, 3 - 1) || z || substr(s.sud, 3 + 1),
   position(' ' in repeat('x', 3) || substr(s.sud, 3 + 1))
from (select '53 7 ' || '6 195 ' || ' 98 6 ' ||
   '8 6 3' || '4 8 3 1' || '7 2 6' || ' 6 28 ' ||
   ' 419 5' || ' 8 79'::text as sud) s,
   (select gs::text as z from generate_series(1,9) gs) z
```

Sudoku: Überprüfen der Lösung

- `s` ist die aktuell betrachtete Lösung
- `ind` ist die Position, an der Ziffer eingesetzt wurde
- `z.z` ist die aktuell eingesetzte Ziffer
- `lp` enthält die zu prüfenden Positionen von 1 bis 9

```
– für ind = 3
select * from generate_series(1, 9) lp
– Zeile: 1, 2, 3, 4, ...
where z.z = substr(s, ((ind - 1) / 9) * 9 + lp, 1)
– Spalte: 3, 12, 21, 30, ...
      or z.z = substr(s, mod(ind - 1, 9) - 8 + lp * 9, 1)
– Block: 1, 2, 3, 10, 11, 12, ...
      + ((ind - 1) / 27) * 27 + lp + ((lp - 1) / 3) * 6, 1))
```

Sudoku: Vollständige Anfrage

```
with recursive x(s, ind) as (
   select sud, position(' ' in sud)
   from (select 'rätselstring'::text as sud) xx,
union all
   select substr(s, 1, ind - 1) || z || substr(s, ind + 1),
      position(' ' in repeat('x', ind) || substr(s, ind + 1))
   from x,
(select gs::text as z from generate_series(1,9) gs) z
where ind > 0 and not exists (
   select null from generate_series(1,9) lp
   where z.z = substr(s, ((ind - 1) / 9) * 9 + lp, 1)
      or z.z = substr(s, mod(ind - 1, 9) - 8 + lp * 9, 1)
      or z.z = substr(s, mod(((ind - 1) / 3), 3) * 3
      + ((ind - 1) / 27) * 27 + lp + ((lp - 1) / 3) * 6, 1))
)
select s from x where ind = 0
```

Sudoku: Verbesserte Ausgabe

- `regexp_replace (txt, pat, repl, flag)`: ersetzt in `txt` einen zum Muster `pat` passenden String durch `repl`
- `regexp_split_to_table (txt, pat)`: teilt den String `txt` anhand des Musters `pat` und liefert eine Relation

```
...
select regexp_replace(regexp_split_to_table(
   regexp_replace(s, '.9(?!$)', '\&-', 'g'), '-'),
   '.3(?!$)', '\&|', 'g')
from x
where position(' ' in s) = 0
```

Prozedurale SQL-Erweiterungen: SQL/PSM

SQL/PSM: Der Standard

- SQL-Standard für prozedurale Erweiterungen
- PSM: Persistent Stored Modules
  - gespeicherte Module aus Prozeduren und Funktionen
  - Einzelroutinen
  - Einbindung externer Routinen (implementiert in C, Java, ...)
  - syntaktische Konstrukte für Schleifen, Bedingungen etc.
  - Basis für Methodenimplementierung für objektrelationale Konzepte

Vorteile gespeicherter Prozeduren

- bewährtes Strukturierungsmittel
- Angabe der Funktionen und Prozeduren erfolgt in der Datenbanksprache selbst; daher nur vom DBMS abhängig
- Optimierung durch DBMS möglich
- Ausführung der Prozeduren erfolgt vollständig unter Kontrolle des DBMS
- zentrale Kontrolle der Prozeduren ermöglicht eine redundanzfreie Darstellung relevanter Aspekte der Anwendungsfunktionalität
- Konzepte und Mechanismen der Rechtevergabe des DBMS können auf Prozeduren erweitert werden
- Prozeduren können in der Integritätssicherung verwendet werden (etwa als Aktionsteil von Triggern)

SQL/PSM: Variablendeklaration

- Variablen vor Gebrauch deklarieren
- Angabe von Bezeichner und Datentyp
- optional mit Initialwert

```
declare Preis float;
declare Name varchar(50);
declare Menge int default 0;
```

SQL/PSM: Ablaufkontrolle

- Zuweisung

```
set var = 42;
```

- Bedingte Verzweigungen

```
if Bedingung then Anweisungen
 [ else Anweisungen ] end if;
```

SQL/PSM: Ablaufkontrolle /2

- Schleifen

```
loop Anweisungen end loop;
while Bedingung do
   Anweisungen end while;
repeat Anweisungen
   until Bedingung end repeat;
```

SQL/PSM: Ablaufkontrolle /3

- Schleifen mit Cursor

```
for SchleifenVariable as CursorName cursor for
    CursorDeklaration
do
    Anweisungen
end for;
```

SQL/PSM: Ablaufkontrolle /4

```
declare wliste varchar(500) default ' ';
declare pos integer default 0;

for w as WeinCurs cursor for
   select Name from WEINE where Weingut = 'Helena'
do
   if pos > 0 then
      set wliste = wliste || ',' || w.Name;
   else
      set wliste = w.Name;
   end if;
   set pos = pos + 1;
end for;
```

SQL/PSM: Ausnahmebehandlung

- Auslösen einer Ausnahme (Condition)

```
signal ConditionName;
```

- Deklarieren von Ausnahmen

```
declare fehlendes_weingut condition;
declare ungueltige_region
   condition for sqlstate value '40123';
```

SQL/PSM: Ausnahmebehandlung /2

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

Ausnahmebehandlung

```
begin
   declare exit handler for ConditionName
   begin
      -- Anweisungen zur Ausnahmebehandlung
   end
   -- Anweisungen, die Ausnahmen auslösen können
end
```

SQL/PSM: Funktionen

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

Funktionsdefinition

```
create function geschmack (rz int)
   returns varchar(20)
begin
   return case
      when rz <= 9 then 'Trocken'
      when rz > 9 and rz <= 18 then 'Halbtrocken'
      when rz > 18 and rz <= 45 then 'Lieblich'
      else 'Süß'
   end
end
```

SQL/PSM: Funktionen /2

- Aufruf innerhalb einer Anfrage

```
select Name, Weingut, geschmack(Restzucker)
from WEINE
where Farbe = 'Rot' and
   geschmack(Restzucker) = 'Trocken'
```

- Nutzung außerhalb von Anfragen

```
set wein_geschmack = geschmack (12);
```

SQL/PSM: Prozeduren

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

- Prozedurdefinition

```
create procedure weinliste (in erz varchar(30),
      out wliste varchar(500))
begin
   declare pos integer default 0;

   for w as WeinCurs cursor for
      select Name from WEINE where Weingut = erz
   do
      -- siehe Beispiel von Folie 13-56
   end for;
end; end;
```

SQL/PSM: Prozeduren /2

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

- Nutzung über `call`-Anweisung

```
declare wliste varchar(500);
call weinliste ('Helena', wliste);
```

SQL/PSM: Zugriffscharakteristik

- Eigenschaften von Prozeduren, die Anfrageausführung und -optimierung beeinflussen
  - `deterministic`: Routine liefert für gleiche Parameter gleiche Ergebnisse
  - `no sql`: Routine enthält keine SQL-Anweisungen
  - `contains sql`:Routine enthält SQL-Anweisungen (Standard für SQL-Routinen)
  - `reads sql data`: Routine führt SQL-Anfragen (`select`-Anweisungen) aus
  - `modifies sql data`: Routine, die DML-Anweisungen (`insert`, `update`, `delete`) enthält

PostgreSQL: PL/pgSQL

- prozedurale SQL-Erweiterung in PostgreSQL
- ähnlich zu Oracle's PL/SQL und zu SQL/PSM

Funktionsdefinition

```
create function geschmack (rz int) returns varchar(20) as $$
begin
   return case when rz <= 9 then 'Trocken'
      when rz > 9 and rz <= 18 then 'Halbtrocken'
      when rz > 18 and rz <= 45 then 'Lieblich'
      else 'Süß'
   end;
end;
$$ language plpgsql;
```

Zusammenfassung

- Rekursion in SQL
- komplexe SQL-Anfragen mit erweiterten Sprachmitteln
- Prozedurale SQL-Erweiterungen

Kontrollfragen

- Welchem Zweck dienen rekursive Anfragen in SQL?
- Wie kann SQL prozedural erweitert werden?