Datenbank-Programmierung

Kapitel 10: Rekursive Anfragen

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Sommersemester 2023

http://www.informatik.uni-halle.de/~brass/dbp23/

Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- Einige Anwendungen rekursiver Anfragen nennen.
- `WITH RECURSIVE` in SQL-Anfragen nutzen.
- Einschränkungen der relationalen Algebra erläutern.

  Insbesondere die Tatsache kennen, dass die transitive Hülle nicht in relationaler Algebra formuliert werden kann. Sie sollten auch „Transitive Hülle Anfragen“ in konkreten Anwendungen erkennen können.
- Die Beziehung zu deduktiven Datenbanken und zur Sprache Datalog kurz erläutern.

Inhalt

Baumartig strukturierte Daten: Beispiel (1)

- Die klassische Beispiel-Datenbank von Oracle enthält eine Tabelle `EMP` mit Daten von Angestellten („employees“):
  - `EMPNO`: Angestellen-Nummer (Primärschlüssel)
  - `ENAME`: Angestellen-Name
  - `JOB`: Berufsbezeichnung
  - `MGR`: Angestellten-Nummer des direkten Vorgesetzten

    „Manager“. Dies ist ein Fremdschlüssel auf die Tabelle selbst.

    Nullwerte sind erlaubt: Der Präsident der Firma hat keinen Vorgesetzten.
  - `HIREDATE`: Datum der Anstellung
  - `SAL`: Gehalt
  - `COMM`: Provision (nur für Verkäufer, d.h. `JOB='SALESMAN'`)
  - `DEPTNO`: Abteilungs-Nummer (Fremdschlüssel zu `DEPT`)

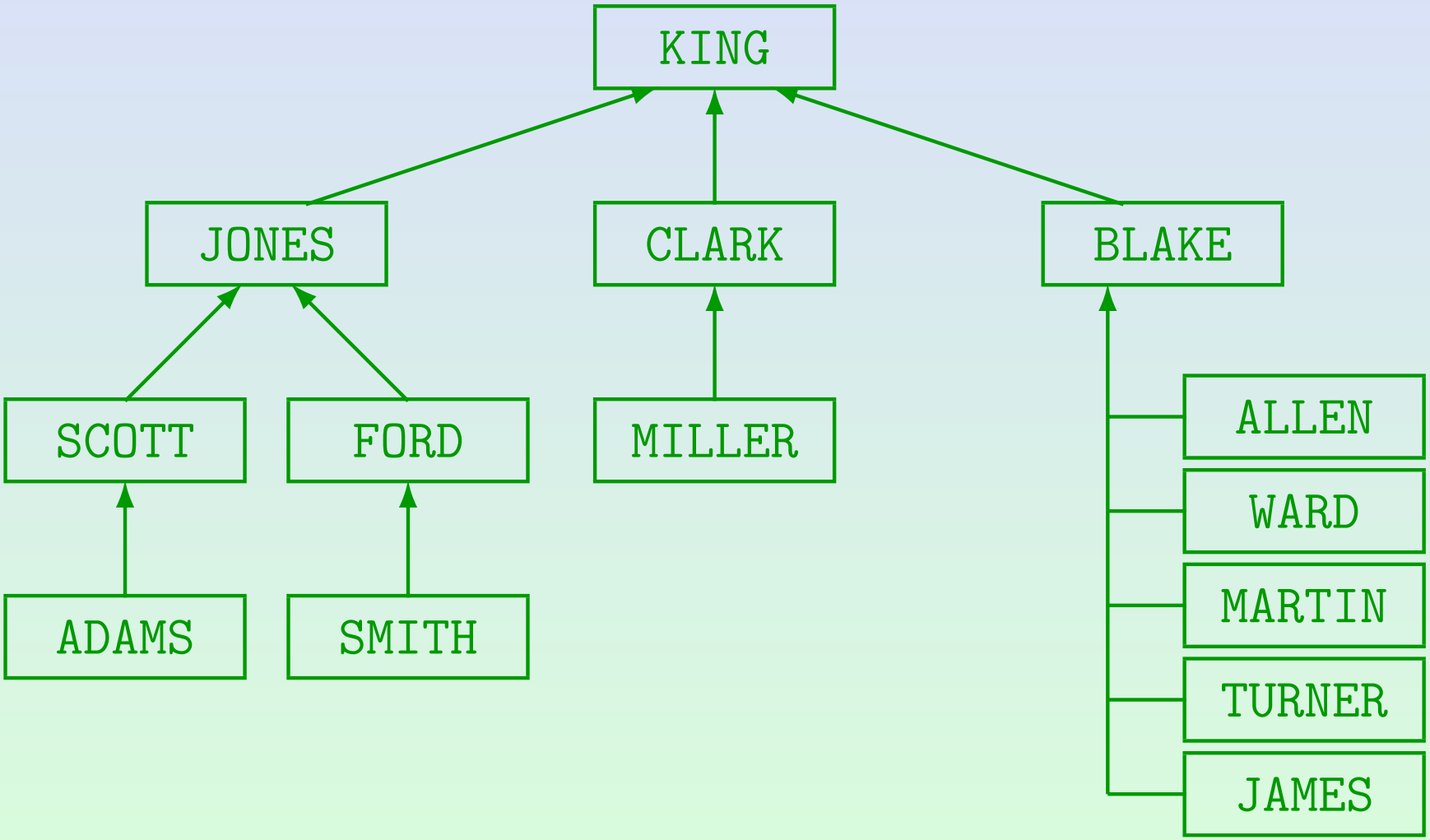
Baumartig strukturierte Daten: Beispiel (2)

EMP					
EMPNO	ENAME	JOB	MGR	SAL	DEPTNO
7369	SMITH	CLERK	7902	800	20
7499	ALLEN	SALESMAN	7698	1600	30
7521	WARD	SALESMAN	7698	1250	30
7566	JONES	MANAGER	7839	2975	20
7654	MARTIN	SALESMAN	7698	1250	30
7698	BLAKE	MANAGER	7839	2850	30
7782	CLARK	MANAGER	7839	2450	10
7788	SCOTT	ANALYST	7566	3000	20
7839	KING	PRESIDENT		5000	10
7844	TURNER	SALESMAN	7698	1500	30
7876	ADAMS	CLERK	7788	1100	20
7900	JAMES	CLERK	7698	950	30
7902	FORD	ANALYST	7566	3000	20
7934	MILLER	CLERK	7782	1300	10

Baumartig strukturierte Daten: Beispiel (3)

- Z.B. sind „SCOTT“ und „FORD“ Untergebene von „JONES“.
  - „SCOTT“ ist selbst Vorgesetzter von „ADAMS“.
  - „FORD“ ist Vorgesetzter von „SMITH“.
- „JONES“ ist Untergebener von „KING“, dem Präsidenten der Firma.
- „KING“ hat keinen Vorgesetzten (Nullwert in der Spalte MGR).
- Man kann die Hierarchie darstellen als gerichteten Graphen mit
  - den Angestellten als Knoten,
  - einer Kante von X nach Y genau dann, wenn Y direkter Vorgesetzter von X ist.
- Dieser Graph ist ein Baum mit KING als Wurzel.

Baumartig strukturierte Daten: Beispiel (4)



Baumartig strukturierte Daten: Beispiel (5)

- Angenommen, man sucht alle Angestellen, die JONES direkt oder indirekt untergeben sind.
- Direkt ist kein Problem (Selbstverbund):

```
SELECT E.EMPNO, E.ENAME
FROM   EMP JONES, EMP E
WHERE  JONES.ENAME = 'JONES'
AND    E.MGR = JONES.EMPNO
```

- Die nächste Stufe wird etwas mühsamer:

```
SELECT E2.EMPNO, E2.ENAME
FROM   EMP JONES, EMP E1, EMP E2
WHERE  JONES.ENAME = 'JONES'
AND    E1.MGR = JONES.EMPNO
AND    E2.MGR = E1.EMPNO
```

Baumartig strukturierte Daten: Beispiel (6)

- Angestellte bis zu zwei Stufen unter JONES bekommt man durch Vereinigung der beiden vorangegangenen Ergebnisse:

```
SELECT E.EMPNO, E.ENAME
FROM   EMP JONES, EMP E
WHERE  JONES.ENAME = 'JONES'
AND    E.MGR = JONES.EMPNO
UNION ALL
SELECT E2.EMPNO, E2.ENAME
FROM   EMP JONES, EMP E1, EMP E2
WHERE  JONES.ENAME = 'JONES'
AND    E1.MGR = JONES.EMPNO
AND    E2.MGR = E1.EMPNO
```

Baumartig strukturierte Daten: Beispiel (7)

- Zufällig gibt es im gegebenen Datenbank-Zustand keine Angestellten drei und mehr Ebenen unter `JONES`.
- Damit liefert die Anfrage in dem Beispiel-Zustand das gewünschte Ergebnis.
- Wenn die Anfrage aber nach zukünftigen Einstellungen noch funktionieren soll, muss man weitere Teile mit immer längeren Joins hinzufügen.

  Die Länge der Gesamtanfrage nach obigem Muster wächst quadratisch in der Maximalanzahl n der Hierarchiestufen, die behandelt werden können. Man könnte das durch Definition von Hilfstabellen mit `WITH` vermeiden.
- Zu jeder Anfrage, die man so basteln kann, kann man aber einen Datenbank-Zustand angeben mit einer noch tieferen Hierarchie, für den die Anfrage nicht mehr funktioniert.

Baumartig strukturierte Daten: Beispiel (8)

- Die Lösung ist eine rekursiv definierte Sicht (in der Definition von SUPERVISOR wird diese Sicht schon verwendet):

```
WITH RECURSIVE
    SUPERVISOR(EMPNO, MGR) AS
        (SELECT EMPNO, MGR FROM EMP
         UNION
         SELECT E.EMPNO, S.MGR
         FROM   EMP E, SUPERVISOR S
         WHERE  E.MGR = S.EMPNO)

SELECT E.ENAME
FROM   EMP E, SUPERVISOR S, EMP JONES
WHERE  JONES.ENAME = 'JONES'
AND    E.EMPNO = S.EMPNO
AND    S.MGR = JONES.EMPNO
```

Auswertung rekursiver Anfragen (1)

- Zuerst wird eine Hilfstabelle SUPERVISOR mit dem Ergebnis des nicht-rekursiven Teils initialisiert:

```
SELECT EMPNO, MGR FROM EMP
```

- Dann wird der rekursive Teil wiederholt mit dem jeweils aktuellen Stand von SUPERVISOR ausgewertet:

```
SELECT E.EMPNO, S.MGR
FROM   EMP E, SUPERVISOR S
WHERE  E.MGR = S.EMPNO
```

- Wurden dabei neue Ergebnistupel hergeleitet (die nicht bereits in SUPERVISOR enthalten waren), so werden sie in SUPERVISOR eingefügt, und die Auswertung wird wiederholt.
- Die Auswertung endet, wenn sich keine neuen Tupel ergeben.

Auswertung rekursiver Anfragen (2)

- Natürlich besteht die Möglichkeit, dass die Auswertung nicht terminiert, wenn immer neue Tupel hergeleitet werden.
- Falls (wie hier) nur Datenwerte neu kombiniert werden, die bereits in den Datenbank-Tabellen vorkommen, kann das nicht passieren.

  Das sind ja endlich viele Werte. Daraus können nur endlich viele Tupel einer Tabelle mit fester Spaltenzahl erstellt werden.

- Wenn man dagegen neue Werte berechnet (z.B. die Anzahl Hierarchie-Ebenen zwischen den Angestellten), wäre das möglich.

  Solange die Vorgesetzten-Relation keine Zyklen enthält, ist die Terminierung auch garantiert: Dann ist der berechnete Wert durch die Anzahl Tabellenzeilen beschränkt.

Auswertung rekursiver Anfragen (3)

- Es ist auch möglich, `UNION ALL` statt `UNION` in der rekursiven Sichtdefinition zu verwenden.
- Dann entfällt die Prüfung, ob die hergeleiteten Tupel neu sind.

  Allerdings werden neue Tupel nur im jeweils nächsten Iterationsschritt eingesetzt, sonst wären Duplikate ja sicher. Das funktioniert nur bei linearer Rekursion (s.u.).
- Das ist effizienter, hat aber ein größeres Risiko der Nicht-Terminierung („Endlosschleife“).
- Im Beispiel funktioniert auch `UNION ALL`, weil die Vorgesetzten-Beziehung azyklisch ist.

Unterstützung rekursiver Anfragen

- Rekursive Sichten (`WITH RECURSIVE`) gibt es im SQL-Standard seit SQL-99.
- Nach der Wikipedia und „modern SQL“ werden rekursive CTEs u.a. in folgenden DBMS unterstützt:
  - PostgreSQL (seit Version 8.4, 2009)
  - MariaDB (seit Version 10.2, 2016)
  - MySQL (seit Version 8.0.1, 2018)
  - SQLite (seit Version 3.8.3, 2014)
  - Oracle (seit Oracle 11g Release 2, 2009)

    Oracle hatte schon viel länger eine spezielle Syntax zur Verarbeitung hierarchischer Daten mit der `CONNECT BY` Klausel.
  - Microsoft SQL Server (seit SQL Server 2008 R2, 2010)
  - DB2 (Vorreiter in rekursiven Anfragen, 1998?)

Anwendungen der Rekursion

- Rekursion ist wichtig zur Verarbeitungen von Hierarchien (wie im Beispiel) oder allgemeiner graph-artig strukturierter Daten.

  Graph-Datenbanksysteme wie etwa Neo4j sind eine aktuelle Entwicklung.

- Z.B. CAD: Das Produkt besteht aus Baugruppen, die selbst aus kleineren Baugruppen und elementaren Teilen bestehen.

  „Bill of Materials Problem“: Berechnung der Kosten für jede Baugruppe und das Endprodukt, wenn Stücklisten der Baugruppen und Preise für die elementaren (zugekauften) Teile gegeben sind.

- Abhängigkeiten, z.B. Aufrufstruktur von Methoden in einem Java-Programm, bilden einen Graphen.

- Extraktion einer unbekannten Anzahl von Stücken aus einer Zeichenkette (strukturierte Strings: schlechter DB-Entwurf).

Inhalt

Grenzen der relationalen Algebra (1)

- Sei R ein Relationsname mit Schema $(A\colon D, B\colon D)$ und sei der Wertebereich $\mathcal{I}_{\mathcal{D}}[D]$ des Datentyps D unendlich.
- Die transitive Hülle von $\mathcal{I}[R]$ ist die Menge aller $(d, e) \in \mathcal{I}_{\mathcal{D}}[D] \times \mathcal{I}_{\mathcal{D}}[D]$, so dass es ein $n \in \mathbb{N}$ $(n \geq 1)$ und $d_0, \ldots, d_n \in \mathcal{I}_{\mathcal{D}}[D]$ gibt mit $d = d_0$, $e = d_n$ und $(d_{i-1}, d_i) \in \mathcal{I}[R]$ für $i = 1, \ldots, n$.
- Z.B. könnte R die Relation "`ELTERNTEIL`" sein, dann besteht die transitive Hülle aus allen Vorfahr-Beziehungen (Eltern, Großeltern, Urgroßeltern, ...).
- Im Beispiel ist direkte Vorgesetzten-Beziehung `(EMPNO, MGR)` in der Datenbank gegeben, und die transitive Hülle davon ist genau die gesuchte Beziehung zu direkten und indirekten Vorgesetzten.

Grenzen der relationalen Algebra (2)

Satz:

- Es gibt keinen RA-Ausdruck Q, so dass $\mathcal{I}[Q]$ die transitive Hülle von $\mathcal{I}[R]$ ist (für alle DB-Zustände $\mathcal{I}$).

Intuitive Erklärung:

- Zur Berechnung der Vorfahren braucht man einen zusätzlichen Verbund für jede weitere Generation.
- Daher kann man keine Anfrage schreiben, die bei beliebigen DB-Zuständen funktioniert.

  Natürlich kann man eine Anfrage schreiben, die z.B. bis zu den Ururgroßeltern funktioniert. Aber dann funktioniert sie nicht korrekt, falls die Datenbank $(\text{Ur})^3$großeltern enthält.

- Ein Beweis würde eine Doppelstunde brauchen.

Grenzen der relationalen Algebra (3)

- Das impliziert natürlich, dass die relationale Algebra nicht berechnungsuniversell ist:
  - Nicht jede Funktion von DB-Zuständen auf Relationen (Antwortmengen), die man mit einem Java-Programm berechnen könnte, kann auch in relationaler Algebra formuliert werden.
  - Man kann dies auch nicht fordern, da man garantieren möchte, dass die Anfrageauswertung terminiert.

    Für die Relationenalgebra gilt, dass Anfragen immer in endlicher Zeit vollständig ausgewertet werden können. Für allgemeine Programme ist dagegen unentscheidbar, ob sie terminieren („Halteproblem“). Jede berechnungsuniverselle Sprache muss notwendigerweise zulassen, dass Anfragen/Programme formuliert werden können, deren Ausführung nicht endet.

Grenzen der relationalen Algebra (4)

- Alle RA-Ausdrücke können in einer Zeit berechnet werden, die polynomiell in der Größe der DB ist.
- Somit können auch sehr komplexe Funktionen nicht in relationaler Algebra formuliert werden.

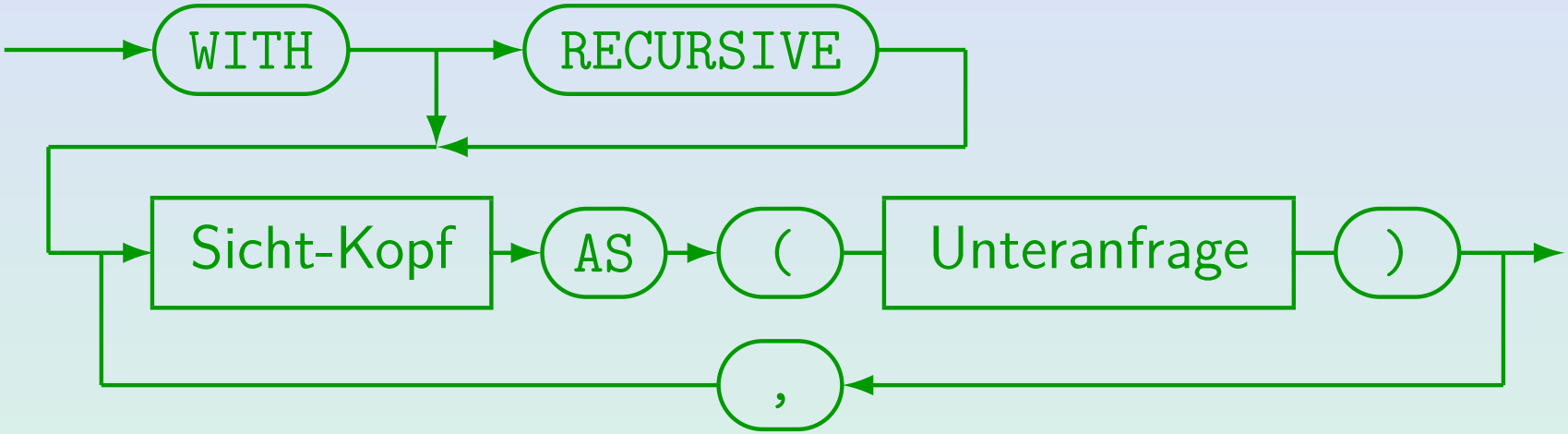
  Wenn Sie z.B. einen Weg finden sollten, das Travelling Salesman Problem in relationaler Algebra zu formulieren, haben Sie das berühmte P=NP Problem gelöst. Da dies sehr unwahrscheinlich ist, sollten Sie es gar nicht versuchen, sondern ein Java-Programm schreiben.
- Aber man kann nicht alle Probleme von polynomialer Komplexität in RA formulieren (z.B. die transitive Hülle).

  Mit einem Fixpunktoperator und einer linearen Ordnung auf den Domains ist dies möglich ($\rightarrow$ Deduktive DB).

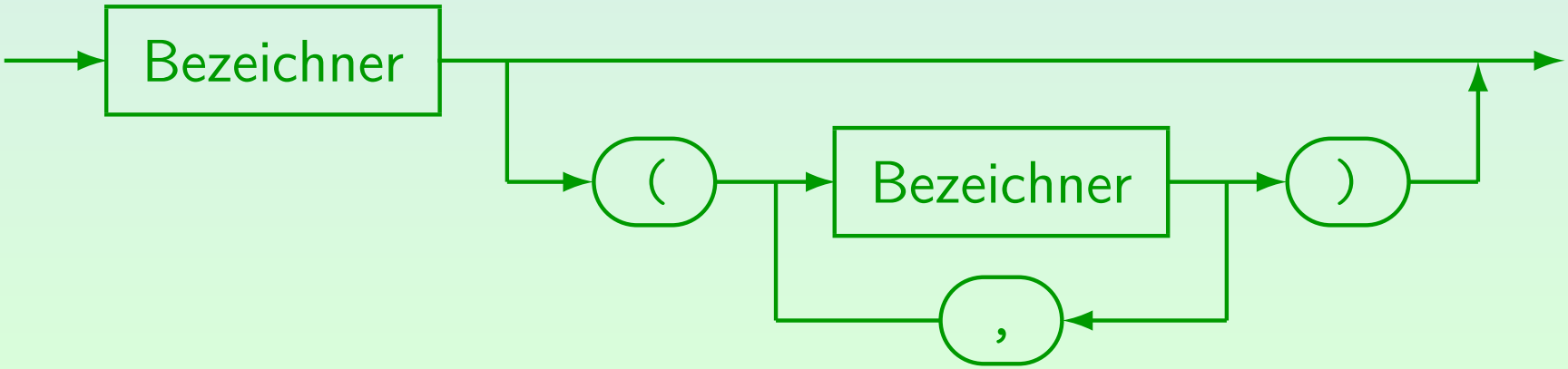
Inhalt

Syntax der WITH-Klausel (1)

Lokale Sicht-Definition (vor SELECT-Anfrage):



Sicht-Kopf:



Syntax der WITH-Klausel (2)

- In der WITH-Klausel kann man mehrere Sichten definieren (nur für die eine Anfrage, deswegen „lokal“).

  Im Standard heißt es „query name“, nicht „lokale Sicht“.

  Bei Microsoft SQL Server heißt es „common_table_expression“ (CTE).

- Man schreibt „RECURSIVE“ nur ein Mal (direkt nach WITH), nicht einzeln vor den betroffenen Sichten.

  Bei Microsoft SQL Server und DB2 muss man „RECURSIVE“ weglassen, selbst wenn die Definition rekursiv ist.

- Ohne RECURSIVE kann jede Sicht nur in den nachfolgend definierten Sichten verwendet werden.

- Mit RECURSIVE ist der Gültigkeitsbereich jeder Sicht die ganze WITH-Klausel: Man kann also auch in einer früheren Sicht eine erst später definierte Sicht verwenden.

Syntax der WITH-Klausel (3)

- Z.B. ist Folgendes möglich (erste Sicht verwendet zweite):

```
WITH RECURSIVE
    RESEARCH_EMP(EMPNO, ENAME) AS
        (SELECT EMPNO, ENAME
         FROM   EMP E, RESEARCH_DEPT D
         WHERE  E.DEPTNO = D.DEPTNO),
    RESEARCH_DEPT(DEPTNO) AS
        (SELECT DEPTNO
         FROM   DEPT
         WHERE  DNAME = 'RESEARCH')

SELECT ENAME
FROM   RESEARCH_EMP
```

Die Definition ist nicht rekursiv, aber ohne das Schlüsselwort „RECURSIVE“ wären solche Vorwärtsreferenzen ausgeschlossen (ohnehin schlechter Stil).

Nur lineare Rekursion (1)

- Logisch könnte man die transitive Hülle auch so bilden:

```
WITH RECURSIVE
    SUPERVISOR(EMPNO, MGR) AS
        (SELECT EMPNO, MGR FROM EMP
         UNION
         SELECT A.EMPNO, B.MGR
         FROM   SUPERVISOR A, SUPERVISOR B
         WHERE  A.MGR = B.EMPNO)

SELECT E.ENAME
FROM   EMP E, SUPERVISOR S, EMP JONES
WHERE  JONES.ENAME = 'JONES'
AND    E.EMPNO = S.EMPNO
AND    S.MGR = JONES.EMPNO
```

Nur lineare Rekursion (2)

- Die obige Anfrage (mit zwei Tupelvariablen über der rekursiven Sicht) gibt folgende Fehlermeldung:

```
ERROR:  recursive reference to query "supervisor"
        must not appear more than once
LINE 6: FROM SUPERVISOR A, SUPERVISOR B
```

- Wenn es nur einen rekursiven Aufruf in der `FROM`-Klausel gibt, spricht man von „linearer Rekursion“.
- Diese ist besonders einfach auszuwerten, weil man in jedem Iterationsschritt nur die im letzten Schritt neu hergeleiteten Tupel einsetzen muss.

  Beim Join zweier rekursiver Aufrufe müsste man dagegen auch alte mit neuen Tupeln kombinieren (siehe „seminaive Auswertung“ bei deduktiven DBen).

Vorgegebene UNION-Struktur (1)

- Die folgende Anfrage (mit dem rekursiven Teil vorne) ist bei PostgreSQL nicht erlaubt:

```
WITH RECURSIVE
    SUPERVISOR(EMPNO, MGR) AS
        (SELECT E.EMPNO, S.MGR
         FROM   EMP E, SUPERVISOR S
         WHERE  E.MGR = S.EMPNO)
         UNION
         SELECT EMPNO, MGR FROM EMP)

SELECT E.ENAME
FROM   EMP E, SUPERVISOR S, EMP JONES
WHERE  JONES.ENAME = 'JONES'
AND    E.EMPNO = S.EMPNO
AND    S.MGR = JONES.EMPNO
```

Vorgegebene UNION-Struktur (2)

- Normal ist `UNION` kommutativ, aber für rekursive Anfragen darf nur ein ganz bestimmtes Muster verwendet werden.
- Bei PostgreSQL ist nur ein einziger rekursiver Aufruf erlaubt, und dieser muss sich im letzten Teil des `UNION` befinden.
- Bei der Anfrage von der vorigen Folie bekommt man diese Fehlermeldung:

```
ERROR:  recursive reference to query "supervisor"
        must not appear within
        its non-recursive term
LINE 4: FROM EMP E, SUPERVISOR S
```

- Bei MariaDB (10.3.27) geht es dagegen.

Inhalt

Zahlen bis 100

- Die folgende Anfrage generiert die Zahlen bis 100:

```
WITH RECURSIVE
    ZAHL(N) AS
        (SELECT 1 AS N
         UNION ALL
         SELECT VORGAENGER.N + 1 AS N
         FROM   ZAHL VORGAENGER
         WHERE VORGAENGER.N < 100)

SELECT N
FROM   ZAHL
ORDER BY N
```

Fakultätsfunktion

- Diese Anfrage generiert eine Tabelle mit Paaren $(n, n!)$ für $n \leq 10$ wobei $n! = 1 * 2 * \cdots * n$:

```
WITH RECURSIVE
    FAKULTAET(N, F) AS
        (SELECT 1 AS N, 1 AS F
         UNION ALL
         SELECT VORIG.N + 1 AS N,
                VORIG.F * (VORIG.N + 1) AS F
         FROM   FAKULTAET VORIG
         WHERE VORIG.N < 10)

SELECT N, F
FROM   FAKULTAET
ORDER  BY N
```

Eine Funktion mit k Argumenten entspricht einer Relation mit $k + 1$ Spalten.

Inhalt

Datalog (1)

- Datalog ist eine Anfrage- und Programmiersprache für Datenbanken, die im wesentlichen eine vereinfachte und „besonders saubere“ Teilmenge der Sprache „Prolog“ ist.

  Prolog („Programming in Logic“) enthält einige Konstrukte wie den berüchtigten „Cut“, die keine logische Semantik haben, sondern nur operational verstanden werden können. Für die praktische Programmierung ist es öfters nötig, die Auswertungsreihenfolge zu kennen, und z.B. Elemente einer Konjunktion passend zu ordnen. In Datalog ist das alles kein Thema, dafür ist viel Forschung nötig, um es neben der Abfragesprache (wie SQL) auch zu einer vollwertigen Programmiersprache werden zu lassen.

- Datalog basiert auf dem Bereichskalkül (mit Variablen über Wertebereichen wie Zahlen und Zeichenketten).
- SQL basiert dagegen auf dem Tupelkalkül (mit Variablen über Tabellenzeilen/Tupeln).

Datalog (2)

- Im Bereichskalkül werden die Tabellen als Prädikate aufgefasst, wobei jede Spalte einem Argument entspricht.

  Die Spalten werden dann üblicherweise über ihre Position identifiziert (nicht über Namen). Bei wenigen Spalten kürzer, bei vielen unübersichtlich.

- Z.B. Abfrage der Relation `EMP`:

```
emp(EMPNO, ENAME, JOB, MGR, HIREDATE, SAL,
    COMM, DEPTNO)
```

  Es werden die Werte aller Variablen, die die Bedingung erfüllen, ausgegeben.

- In Prolog und Datalog beginnen
  - Bezeichner (wie `emp`) mit einem Kleinbuchstaben,
  - Variablen (wie `EMPNO`) dagegen mit einem Großbuchstaben.

    Natürlich könnte man die Variablen auch anders nennen, z.B. `X` statt `EMPNO`, aber Spaltennamen als Variablen sind übersichtlicher.

Datalog (3)

- Ein Datalog-Programm besteht aus der Definition abgeleiteter Prädikate (Sichten) über Regeln.
- Z.B. könnte man Prädikate definieren, die den Schlüssel (EMPNO) und jeweils eine Spalte enthalten:

```
ename(EMPNO, ENAME) ←
    emp(EMPNO, ENAME, _, _, _, _, _, _).
mgr(EMPNO, MGR) ←
    emp(EMPNO, _, _, MGR, _, _, _, _).
```

Argumente, die man nicht braucht, kann man mit dem Unterstrich „_“ („anonyme Variable“) ausfüllen (formal ist jedes Auftreten eine neue Variable).

- Die linke Seite der Regel (was definiert wird) heißt auch „Kopf“ der Regel, die rechte Seite entsprechen „Rumpf“.

Datalog (4)

- Wenn man `mgr(EMPNO, MGR)` hat, kann man die transitive Hülle so definieren:

  $$\texttt{supervisor(X, Y)} \leftarrow \texttt{mgr(X, Y).}$$
  $$\texttt{supervisor(X, Z)} \leftarrow \texttt{mgr(X, Y)} \wedge \texttt{supervisor(Y, Z).}$$

  Man kann mehrere Regeln über das gleiche Prädikat definieren.
  Alle können benutzt werden, um zu zeigen, dass ein bestimmtes Tupel in der abgeleiteten Relation enthalten ist. Effektiv bekommt man so die Vereinigung der Anfrageergebnisse der rechten Seiten der Regeln.

- Die Anfrage nach dem Namen der Untergebenen von Jones würde dann so aussehen:

  `ename(J, 'JONES'), supervisor(E, J), ename(E, NAME)`

  Man könnte noch ein Prädikat definieren, um die Angestellten-Nummern `J` und `E` auszublenden.

Datalog (5)

- Mit Datalog wurde im Bereich der „deduktiven Datenbanken“ Rekursion eingeführt, lange bevor SQL sie hatte.

  Wie so häufig wurde Konkurrenz (in diesem Fall den deduktiven Datenbanken) das Wasser abgegraben, indem SQL entsprechend erweitert wurde.

- Wenn man Rekursion in Reinform studieren will (und z.B. über Optimierungstechniken für die Auswertung nachdenken), ist die sehr einfache Syntax von Datalog viel besser geeignet als die sehr komplexe Sprache SQL.

  Band 2 des SQL-2003 Standards hat 1268 Seiten.

- Datalog hat nicht die Einschränkungen von SQL, z.B. ist Datalog keineswegs auf lineare Rekursion eingeschränkt.

- $\rightarrow$ „Logische Programmierung und deduktive Datenbanken“ (Vorlesung im Master)

Literatur/Quellen

- Wikipedia: Hierarchical and recursive queries in SQL [https://en.wikipedia.org/wiki/Hierarchical_and_recursive_queries_in_SQL]
- Markus Winand: „Can I use ... with recursive (top level) in SQL“ (modernSQL) [https://modern-sql.com/caniuse/with_recursive_(top_level)]
- Markus Winand: „Can I use ... with recursive (non-linear) in SQL“ [https://modern-sql.com/caniuse/with_recursive_(non-linear)]
- International Standard ISO/IEC 9075-2 (Second Edition 2003–12–15) Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation), Section 7.13 „query expression“
- PostgreSQL Documentation: WITH Queries (Common Table Expressions) [https://www.postgresql.org/docs/9.2/queries-with.html]
- MariaDB: Recursive Common Table Expressions Overview [https://mariadb.com/kb/en/recursive-common-table-expressions-overview/]
- Ben Lis: Writing Recursive Queries. PostgresConf US 2019 [https://postgresconf.org/system/events/document/000/000/953/writing-recursive-queries-postgresconfus19.pdf]
- Why does a recursive CTE in Transact-SQL require a UNION ALL and not a UNION? [https://stackoverflow.com/questions/47998833/]