Praktikum Datenbanken / DB2
Woche 6: SQL als Anfragesprache

Raum: LF 230 Bearbeitung: 30. Mai - 1. Juni 2005

Datum	
Gruppe	
Vorbereitung	
Präsenz	

Aktuelle Informationen unter:
`http://www.is.informatik.uni-duisburg.de/courses/dbp_ss05/`

Anfragen mit SQL

Subselect

In DB2-SQL gibt es drei Formen von Anfragen: Subselects, Fullselects und das volle SELECT-Statement. Bei jeder Anfrage (Query) muß man für jede verwendete Tabelle oder Sicht zumindest das SELECT-Recht besitzen.

Ein Subselect besteht aus sieben Klauseln, wobei nur die SELECT- und FROM-Klauseln zwingend sind. Die Reihenfolge ist festgelegt:

db2s1e81.pdf, S. 553ff

```
SELECT ...
FROM ...
WHERE ...
GROUP BY ...
HAVING ...
ORDER BY ...
FETCH FIRST ...
```

Zur vollen Spezifikation der Syntax sei zum Beispiel auf die DB2-Dokumentation verwiesen.

SELECT

Im SELECT-Teil wird angegeben, wie das Ergebnis aussehen soll, d.h. welche Spalten die Ergebnisrelation haben soll, und gegebenenfalls wie diese benannt werden. Die einzelnen Spalten des Ergebnisses sind durch Kommata zu trennen, die Nachstellung eines nicht qualifizierten Spaltennamen benennt die Spalte der Ergebnisrelation (um). Durch `*` oder `Tabelle.*` werden alle Spalten (der benannten Tabelle) ins Ergebnis übernommen.

db2s1e81.pdf, S. 555

Select mit Umbenennung einer Ergebnisspalte:

```
SELECT name, code AS 'Landeskennzeichen'
FROM land
```

SELECT DISTINCT nimmt Duplikateliminierung vor. Es ist möglich, die Werte der Ergebnisspalten im Select-Teil durch einen *Ausdruck* (siehe letzte Woche) berechnen zu lassen. Die Datentypen der Ergebnisspalten entsprechen im Wesentlichen den Datentypen der Ursprungsspalten, bzw. dem Resultatdatentyp eines Ausdrucks.

FROM

Der FROM-Teil spezifiziert die zur Anfrage herangezogenen Relationen (Tabellen), dabei kann man durch Nachstellung eines eindeutigen Bezeichners (mit dem optionalen Schlüsselwort `AS`) Tabellen temporär einen neuen Namen geben (Stichwort: Tupelvariablen). Werden mehrere Tabellen in einer Anfrage verwendet, so sind diese im FROM-Teil durch Kommata getrennt anzugeben. Die Anfrage arbeitet dann auf dem kartesischen Produkt dieser Tabellen.

db2s1e81.pdf, S. 560

Beispiel für Tupelvariablen zur eindeutigen Bezeichnung von Tabellen:

```
SELECT *
FROM stadt AS s1,
     stadt AS s2
```

Es ist möglich, im FROM-Teil als Relation eine Subquery (ein Fullselect) anzugeben. In dem Fall ist es zwingend, dieser Relation einen Namen zu geben.

Statt einer einzelnen Tabelle kann man eine "joined table"-Klausel als Zwischenrelation spezifizieren, die Ergebnis eines JOIN ist:

```
tab1 JOIN                          tab2 ON condition
tab1 INNER JOIN                    tab2 ON condition
tab1 {LEFT,RIGHT,FULL} JOIN        tab2 ON condition
tab2 {LEFT,RIGHT,FULL} OUTER JOIN tab2 ON condition
```

Die Definition der verschiedenen JOIN-Arten ist wie in der Vorlesung **Datenbanken** behandelt, bzw. wie in der begleitenden Literatur beschrieben. Ohne Angabe eines JOIN-Operators wird der INNER JOIN angenommen. Bei mehreren JOINs kommt es auf die Reihenfolge an, daher ist zu beachten, dass die JOINs in der Regel von links nach rechts berechnet werden, wobei die Position der JOIN-Bedingung eine Rolle spielt.

```
SELECT l.name, k.name
FROM land AS l,
     kontinent AS k
```

bildet alle möglichen Kombinationen (also das kartesische Produkt) der Tupel aus `Land` und `Kontinent` und liefert aus der kombinierten Relation den Namen der Länder und Kontinente zurück. Eine solche Kombination ohne JOIN-Bedingung ist in den seltesten Fällen gewünscht.

```
SELECT l.name, k.kontinent
FROM land AS l
  JOIN
     landmasse AS k
  ON l.code = k.land
```

bildet nur die Kombinationen der Tupel aus `Land` und `Lamdmasse`, bei denen der Ländercode aus `Land` mit dem Land in `Landmasse` übereinstimmt, also genau die Kombinationen von Land und Kontinent, bei denen das Land zumindest teilweise auf dem jeweiligen Kontinent liegt.

Das folgende Statement bewirkt das gleiche wie das letzte Beispiel:

```
SELECT l.name, k.kontinent
FROM land AS l,
     landmasse AS k
WHERE l.code = k.land
```

WHERE

Über die WHERE-Klausel werden alle Tupel ausgewählt, für welche die angegebene Suchbedingung zu *true* evaluiert. Für die Suchbedingung muß gelten:

db2s1e81.pdf, S. 568

- die benutzten Spaltennamen müssen eindeutig eine Spalte der im From-Teil angegebenen Relationen oder des Ergebnisses eines JOINS aus dem From-Teil sein
- eine Aggregatfunktion darf nur verwendet werden, wenn die WHERE-Klausel in einer Subquery einer HAVING-Klausel verwendet wird

```
SELECT *
FROM organisation
WHERE substr(abkuerzung,1,2) = 'UN'
```

liefert alle Tupel der Relation `Organisation`, bei denen die Abkürzung mit 'UN' beginnt (Unterorganisationen der UN). Die gleiche Bedingung könnte auch in der folgenden Form geschrieben werden:

```
SELECT *
FROM organisation
WHERE abkuerzung like 'UN%'
```

GROUP BY

Über die GROUP BY-Klausel können die Tupel des bisherigen Ergebnisses gruppiert werden, dabei ist zumindest ein Gruppierungsausdruck anzugeben. Dieser darf keine Subqueries enthalten und alle Tupel, für welche der Gruppierungsausdruck denselben Wert ergibt, gehören zu einer Gruppe. Für jede Gruppe wird ein Ergebnistupel berechnet. Meist benutzt man GROUP BY zusammen mit Aggregratfunktionen, und die Gruppen legen fest, auf welchen Bereich die Funktion angewandt wird.

db2s1e81.pdf, S. 569

Wird in einem Subselect GROUP BY oder HAVING verwendet und treten in den Ausdrücken der SELECT-Klausel Spaltennamen auf, so müssen diese in einer Aggregatfunktion vorkommen, von einem Ausdruck der GROUP BY-Klausel abgeleitet sein oder aus einer umgebenden Anfrage stammen.

HAVING

Über die HAVING-Klausel kann man Bedingungen an die gebildeten Gruppen stellen. Es werden die Gruppen ausgewählt, für welche die Suchbedingung zu *true* evaluiert. Wurde in der Anfrage keine GROUP BY-Klausel angegeben, wird das ganze bisherige Ergebnis als eine Gruppe behandelt.

db2s1e81.pdf, S. 576

Es dürfen in dieser Suchbedingung nur Spaltennamen verwendet werden, die aus einer äußeren Anfrage stammen, innerhalb der Aggregratfunktion verwendet werden, oder die in einem Gruppierungsausdruck vorkommen.

ORDER BY

Über die ORDER BY-Klausel wird angegeben, wie die Ergebnisrelation sortiert werden soll, d.h. in welcher Reihenfolge die Ergebnistupel ausgegeben werden sollen. Es können mehrere Sortierschlüssel angegeben werden, dann wird zuerst gemäß dem ersten Schlüssel sortiert, dann für in diesem Schlüssel übereinstimmende Tupel nach dem zweiten Schlüssel, usw.

db2s1e81.pdf, S. 576

Über die Angaben von ASC bzw. DESC wird festgelegt, ob die Sortierung aufsteigend oder absteigend erfolgt. Der Standard ist ASC, wobei der NULL-Wert als größer als alle anderen Werte angenommen wird.

Als Sortierschlüssel ist entweder ein Spaltenname der Ergebnisrelation (alternativ kann auch ein Integerwert benutzt werden, um die Spalte anhand ihrer Position auszuwählen), oder einer der in der FROM-Klausel verwendeten Relaionen oder ein Ausdrück erlaubt. Wurde im SELECT-Teil mit DISTINCT gearbeitet, so muß der Sortierschlüssel exakt einem Ausdruck der SELECT-Liste entsprechen. Ist das Subselect gruppiert, so können nur Audrücke der SELECT-Liste, Gruppierungsausdrücke und Ausdrücke, die eine Aggregatfunktion, Konstante oder Hostvariable enthalten verwendet werden.

```
SELECT *
FROM politik
ORDER BY unabhaengigkeit DESC
```

FETCH FIRST

Die FETCH FIRST-Klausel erlaubt es, die Ergebnisrelation auf eine bestimmte Zahl von Tupeln einzuschränken. Das ist in jedem Fall für das Testen von Anfragen zu empfehlen, weil unter Umständen nicht alle Tupel berechnet werden müssen und so die Performance steigt.

db2s1e81.pdf, S. 579

Geogr. Lage der 10 größten Städte, bei denen die Angaben vorhanden sind (auf 1 Dezimalstelle genau):

```
SELECT name, dec(laengengrad,4,1) as laenge,
             dec(breitengrad,4,1) as breite
FROM stadt
WHERE breitengrad IS NOT NULL
  AND laengengrad IS NOT NULL
ORDER BY bevoelkerung DESC
FETCH FIRST 10 ROWS ONLY
```

Aggregatfunktionen

Aggregatfunktionen werden auf eine Menge von Werten angewendet, die von einem Ausdruck abgeleitet sind. Die verwendeten Ausdrücke können sich auf Attribute, nicht aber auf skalarwertige Anfragen oder andere Aggregatfunktionen beziehen. Der Bereich, auf den eine Aggregatfunktion angewendet wird, ist eine Gruppe oder eine ganze Relation. Ist das Ergebnis der bisherigen Bearbeitung eine leere Menge und wurde GROUP BY verwendet, so werden die Aggregatfunktionen nicht berechnet, und es wird eine leere Menge zurückgegeben. Wurde GROUP BY verwendet, werden die Aggregatfunktionen auf die leere Menge angewendet.

db2s1e81.pdf, S. 269ff

Wurde in einem Subselect kein GROUP BY und kein HAVING verwendet, dürfen in der SELECT-Klausel Aggregatfunktionen nur auftreten, wenn sie alle Attributnamen umfassen.

Die folgenden Aggregratfunktionen nehmen jeweils genau ein Argument. NULL-Werte werden gegebenenfalls vor der Berechnung eliminiert. Auf die leere Menge angesetzt, liefern sie NULL zurück. Über das zusätzliche Schlüsselwort DISTINCT kann man erreichen, dass vor der Berechnung Duplikateliminierung durchgeführt wird.

AVG: Durchschnitt der Werte

MAX: Maximum der Werte

MIN: Minimum der Werte

STDDEV: Standardabweichung der Werte

SUM: Summe der Werte

VARIANCE: Varianz der Werte

```
SELECT max(bevoelkerung)
FROM stadt

SELECT avg(gebiet)
FROM land

SELECT sum(bsp)
FROM wirtschaft
```

Die COUNT-Funktion wird ausgewertet zur Anzahl der Werte, zu denen der Argumentausdruck evaluiert. COUNT(*) liefert die Anzahl der Tupel im jeweiligen Bereich zurück, Gruppierungen werden berücksichtigt. Im Ausdruck werden NULL-Werte eliminiert. Das Ergebnis ist dann jeweils die Anzahl der (bei Verwendung von DISTINCT paarweies verschiedener) Werte im jeweiligen Bereich.

Das Ergebnis von COUNT kann nie NULL sein.

Beispiele

Bevölkerung pro Kontinent (Ergebnis als BIGINT):

```
SELECT kontinent, bigint( SUM(bevoelkerung*prozentanteil) )
FROM land JOIN landmasse
   ON code=land
GROUP BY kontinent
```

Kleinstes Land:

```
SELECT name
FROM land
WHERE gebiet <= ALL (SELECT gebiet FROM land)
```

Weitere Beispiele zu Subselects, Joins und Gruppierung finden sich in der Dokumentation zu DB2, bzw. können dem Begleitmaterial entnommen werden. Vorlesung entr

Fullselect

db2s1e81.pdf, S. 579

Ein Fullselect besteht aus zwei oder mehr Teilselects, die über Mengenoperatoren verknüpft werden. Dabei müssen die Ergebnistupel der Teilselects jeweils die gleiche Stelligkeit besitzen, und die Datentypen an den entsprechenden Positionen müssen kompatibel sein. Die verfügbaren Mengenoperatoren in SQL sind UNION, UNION ALL, EXCEPT, EXCEPT ALL, INTERSECT, INTERSECT ALL. Zwei Tupel werden im Folgenden als gleich betrachtet, wenn die jeweils korrespondierenden Komponenten gleich sind (wobei zwei NULL-Werte als gleich betrachtet werden).

Werden mehr als zwei Verknüpfungsoperatoren verwendet, so werden zuerst die geklammerten Fullselects berechnet, dann werden alle INTERSECT-Operationen ausgeführt, und zuletzt werden die Operationen von links nach rechts ausgewertet.

UNION

UNION ist die Mengenvereinigung, dabei findet Duplikateliminierung statt. Bei Verwendung von UNION ALL werden bei der Vereinigung keine Duplikate eliminiert.

Alle Länder und Provinzen:

```
  SELECT name
  FROM land
UNION
  SELECT name
  FROM provinz
```

INSERSECT

INTERSECT ist der Mengendurchschnitt, d.h. das Ergebnis wird durch alle Tupel gebildet, die in beiden Relationen liegen. Bei der Verwendung von INTERSECT ALL enthält das Ergebnis auch Duplikate, im Falle von INTERSECT nicht.

Alle Länder, die auch Provinzen sind:

```
  SELECT name
  FROM land
INTERSECT
  SELECT name
  FROM provinz
```

EXCEPT

Bei EXCEPT ALL wird das Ergebnis dadurch bestimmt, dass man – unter Beachtung von Duplikaten – aus der linksstehenden Relation alle Tupel entfernt, die auch in der rechten Relation vorkommen. Bei Verwendung von EXCEPT wird das Ergebnis durch alle Tupel der linken Relation gebildet, die **nicht** in der rechten Relation vorkommen. Dann enthält das Ergebnis auch keine Duplikate.

Alle Länder, die nicht(!) auch Provinzen sind:

```
  SELECT name
  FROM land
EXCEPT
  SELECT name
  FROM provinz
```

Views

Ein wichtiges Datenbankkonzept sind Sichten oder Views. Diese können in SQL mit dem CREATE VIEW-Statement erzeugt werden:

db2s2e81.pdf, S. 470ff

```
CREATE VIEW EUCountry AS
  SELECT land.*, mitglied.art
  FROM land JOIN mitglied ON
     land.code = mitglied.land
  WHERE organisation = 'EU'
```

Ein View liefert eine bestimmte Ansicht auf die Datenbank. Man kann Sichten für verschiedene Zwecke einsetzen. Im Datenschutz kann man durch Sichten etwa für bestimmte Nutzergruppen sensible Daten über eine Sicht ausschließen. Man kann Anfragen vereinfachen, wenn man Sichten als eine Art Makro betrachtet, das an geeigneter Stelle in ein SELECT-Statement übernommen werden kann. Schließlich lassen sich mit Sichten auch Generalisierungsmodelle nachempfinden:

```
CREATE TABLE person (
   name      varchar(30) not null,
   geboren   date not null,
   gestorben date
)

CREATE TABLE schauspieler_daten (
   name      varchar(30) not null,
   alias     varchar(30)
)

CREATE VIEW schauspieler AS
   SELECT *
   FROM person JOIN schauspieler_daten s
     ON person.name = s.name
```

Ebenso wie in diesem Beispiel der Untertyp als Sicht realisiert wurde, läßt sich auch der Obertyp als Sicht der Untertypen realisieren.

Sichten können wie Tabellen benutzt werden. Sichten haben aber das Problem, dass sie oft nicht updatefähig sind. Eine Sicht ist veränderbar (d.h. erlaubt UPDATE), wenn sie weder Aggregratfunktoinen, noch Anweisungen wie DISTINCT, GROUP BY oder HAVING enthält, in der SELECT-Klausel nur eindeutige Spaltennamen stehen und ein Schlüssel der Basisrelation enthalten ist, und sie nur genau eine Tabelle verwendet, die ebenfalls veränderbar ist.

Zur Wiederholung

Hier noch einmal die Liste der wichtigsten Funktionen, die von DB2 unterstützt werden.

Beispiele:

- `BIGINT(bevoelkerung)`

  wandelt die Werte des Attributes `bevoelkerung` vor der Weiterverarbeitung in einen anderen Datentyp um

- `YEAR(gruendungsdatum)`

liefert den Jahresanteil des Datumsattributes `gruendungsdatum` als Integer

- `SUBSTR(name, 0, 3)`

  liefert die ersten 3 Zeichen der Werte von `name`

Typkonvertierung: BIGINT, BLOB, CHAR, CLOB, DATE, DBCLOB, DECIMAL, DREF, DOUBLE, FLOAT, GRAPHIC, INTEGER, LONG, LONG_VARCHAR, LONG_VARGRAPHIC, REAL, SMALLINT, TIME, TIMESTAMP, VARCHAR, VARGRAPHIC

Mathematik: ABS, ACOS, ASIN, ATAN, CEIL, COS, COT, DEGREES, EXP, FLOOR, LN, LOG, LOG10, MOD, POWER, RADIANS, RAND, ROUND, SIGN, SIN, SQRT, TAN, TRUNC

Stringmanipulation: ASCII, CHR, CONCAT, DIFFERENCE, DIGITS, HEX, INSERT, LCASE, LEFT, LENGTH, LOCATE, LTRIM, POSSTR, REPEAT, REPLACE, RIGHT, RTRIM, SOUNDEX, SPACE, SUBSTR, UCASE, TRANSLATE

Datumsmanipulation: DAY, DAYNAME, DAYOFWEEK, DAYOFYEAR, DAYS, HOUR, JULIAN_DAY, MICROSECOND, MIDNIGHT_SECONDS, MINUTE, MONTH, MONTHNAME, QUARTER, SECOND, TIMESTAMP_ISO, TIMESTAMPDIFF, WEEK, YEAR

System: EVENT_MON_STATE, NODENUMBER, PARTITION, RAISE_ERROR, TABLE_NAME, TABLE_SCHEMA, TYPE_ID, TYPE_NAME, TYPE_SCHEMA

Sonstige: COALESCE, GENERATE_UNIQUE, NULLIF, VALUE

Übersicht über die neuen Befehle

select ...	Anfrage an die Datenbank
create view *name*	erstelle eine Sicht auf eine oder mehrere Tabelle(n)

Aufgaben

Zur Vorbereitung:

(a) Das Beispiel der 'Vereinigten Staaten von Europa' soll nun weiterentwickelt werden. Schreibe SQl–Statements, um die folgenden Veränderungen am Datenbestand vorzunehmen.

(*i*) Zunächst sollen die alten Mitgliedsstaaten der 'EU' zu Provinzen der 'Vereinigten Staaten von Europa werden.

(*ii*) Die Größe (Fläche) und Bevölkerungszahl (Einwohner) des neuen Staates ergibt sich als Summe der Einzelstaaten.

(*iii*) Die wirtschaftlichen Daten des neuen Staates sollen durch Mittelung der Einzeldaten entstehen.

(b) Schreibe zum Aufwärmen SQL–Anfragen für die folgenden Problemstellungen:

(*i*) Wie hoch ist die Bevölkerungsdichte von Europa?

(*ii*) Welche Staaten sind sowohl Mitglied der EU als auch der NATO? Löse dieses Problem einmal als Fullselect und einmal mit Tupelvariablen.

(*iii*) Welche NATO-Mitglieder sind nicht in der EU? Löse das Problem einmal als Fullselect und einmal mit Quantor.

(c) In der Beispieldatenbank ist die Relation *Grenze* nicht symmetrisch gespeichert, d.h. für Deutschland findet man die Nachbarländer nur über folgende Anfrage:

```
select * from grenze where land1='D' or land2='D'

LAND1 LAND2 LAENGE
----- ----- -----------
A     D             784
CZ    D             646
D     F             451
CH    D             334
PL    D             456
B     D             167
L     D             138
NL    D             577
DK    D              68
```

Sinnvoll wäre eine symmetrische Relation. Verwirkliche dies mittels eines Views, der die symmetrische Hülle der Tabelle enthält.

Präsenzaufgaben:

(a) Führt die Anfragen aus Vorbereitungsaufgabe (b) auf Eurer Datenbank aus. Haltet die ersten drei Zeilen (wenn nicht anders in der Aufgabe spezifiziert) des Ergebnisses fest. Ist das Ergebnis jeweils wie erwartet?

(b) Erstellt den View `SymGrenze` und benutzt diesen für die weiteren Aufgaben. Falls Eure Grenz–Relation bereits symmetrisch ist, überlegt Euch einen anderen sinnvollen View und verwirklicht diesen. Haltet das SQL–Statement zu diesem View fest.

(c) Löst die folgenden Probleme mit Hilfe von SQL und Eurer eigenen Datenbank. Haltet sowohl das benutzte Statement als auch die ersten drei Zeilen des Ergebnisses fest.

Tip: Wenn man zwei Ganzzahlen (Integer) durcheinander teilt, so erhält man als Ergebnis eine Ganzzahl, daher sollten die Operanden für die Berechnung von Prozentanteilen zunächst in einen anderen Datentyp gewandelt werden.

Tip: Beim Aufsummieren oder Durchschnittbilden von großen Ganzzahlen kann es leicht zum Überlauf kommen. Daher gegebenfalls vorher in einen anderen Datentyp wandeln.

(*i*) Bestimme für alle Länder die Gesamtlänge ihrer Grenzen.

(*ii*) Eine Liste der Organisationen mit der Gesamtfläche aller Mitgliedsländer (die Art der Mitgliedschaft soll keine Rolle spielen). Bestimme auch noch die Gesamtbevölkerung.

(*iii*) Zu jedem Land in Afrika soll der höchste Berg (wenn in der Datenbank vorhanden) ermittelt werden. Die Ausgabe soll der Höhe nach sortiert werden.

(*iv*) Berechne den prozentualen Anteil der Anhänger jeder Religion an der Weltbevölkerung.

(*v*) Die in der Datenbank aufgeführten Städte seien alle existierenden Städte. Berechne den Anteil der städtischen Bevölkerung an der Gesamtbevölkerung für die Länder Europas.

(*vi*) *Bonus:* Bestimme alle Organisationen, die auf jedem Kontinent mindestens ein Mitgliedsland besitzen (die Art der Mitgliedsschaft soll keine Rolle spielen).