Praktikum Datenbanken / DB2
Wochen 5-6: SQL als Anfragesprache

Betreuer: Sascha Kriewel, Tobias Tuttas
Raum: LF 230
Bearbeitung:
1. Woche: 30. Mai, 1. und 4. Juni 2007
2. Woche: 6., 8. unf 11. Juni 2007

Datum	
Team (Account)	
Vorbereitung	
Präsenz	

Aktuelle Informationen, Ansprechpartner und Material unter:
`http://www.is.inf.uni-due.de/courses/dbp_ss07/index.html`

Hinweis:

Dieses Praktikumsmaterial ist auf zwei Wochen ausgelegt. Die Besprechung der mit (V) gekennzeichneten Vorbereitungsaufgaben findet in der ersten Woche statt. Die Abnahme der Präsenzaufgaben erfolgt in Woche 2.

Wochenziel

Nachdem Ihr in der letzten Woche Eure Datenbanken mit Daten gefüllt habt, machen wir uns in dieser Woche mit SQL als Anfragesprache vertraut.

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 drehjahr, titel AS 'Filmtitel'
FROM film
```

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 anwender AS a1,
     anwender AS a2
```

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 DISTINCT s.name,s.vorname,f.titel,f.drehjahr
FROM schauspieler AS s,
     spieltinfilm AS f
```

bildet alle möglichen Kombinationen (also das kartesische Produkt) der Tupel aus `Schauspieler` und `SpieltInFilm` 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 s.name,s.vorname,f.titel,f.drehjahr
FROM schauspieler AS s
  JOIN
     spieltinfilm AS f
  ON s.id = f.schauspielerid
```

bildet nur die Kombinationen der Tupel aus `Schauspieler` und `SpieltInFilm`, bei denen die ID aus `Schauspieler` mit der SchauspielerID in `SpieltInFilm` übereinstimmt, also genau die Kombinationen von `Schauspieler` und `SpieltInFilm`, bei denen der Schauspieler wirklich in dem betreffenden Film mitspielt.

Das folgende Statement bewirkt das gleiche wie das letzte Beispiel:

```
SELECT s.name,s.vorname,f.titel,f.drehjahr
FROM schauspieler AS s,
     spieltinfilm AS f
WHERE s.id = f.schauspielerid
```

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 anwender
WHERE substr(nick,1,2) = 'Op'
```

liefert alle Tupel der Relation `Anwender`, bei denen der Nick mit 'Op' beginnt. Die gleiche Bedingung könnte auch in der folgenden Form geschrieben werden:

```
SELECT *
FROM anwender
WHERE nick like 'Op%'
```

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.

Die folgende Anfrage listet alle Filmdaten nach Jahren und Titeln geordnet auf, mit den neuesten Filmen zuerst:

```
SELECT *
FROM film
ORDER BY drehjahr DESC,titel ASC
```

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

Die ersten 5 Serienfolgen, bei denen der Folgentitel nicht leer (Null) und auch nicht die leere Zeichenkette ist:

```
SELECT serientitel,drehjahr,staffelnr,folgennr,folgenTitel
FROM serienfolge
WHERE folgentitel IS NOT NULL
  AND folgentitel<>''
ORDER BY drehjahr,serientitel,staffelnr,folgennr DESC
FETCH FIRST 5 ROWS ONLY
```

Aggregatfunktionen

db2s1e81.pdf, S. 269ff

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.

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

Beispiel:

Welches ist das letzte Jahr, für welches Filme in der Datenbank vorliegen?

```
SELECT max(drehjahr)
FROM film
```

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

Anzahl Folgen pro „Buffy"-Staffel:

```
SELECT serientitel,drehjahr,staffelnr,COUNT(folgennr) AS anzahl
  FROM serienfolge
  WHERE serientitel like 'Buffy%'
  GROUP BY serientitel,drehjahr,staffelnr
```

Längster Filmtitel:

```
SELECT titel
FROM film
WHERE length(titel) >= ALL (SELECT DISTINCT titel FROM film)
```

Wieviele Folgen haben die Staffeln von „Buffy" durchschnittlich? (Mit einer Genauigkeit von 2 Stellen hinter dem Komma.)

```
SELECT serientitel,drehjahr,DEC(AVG(FLOAT(anzahl)),4,2) AS avgfolgen FROM
  (SELECT serientitel,drehjahr,staffelnr,COUNT(folgennr) AS anzahl
     FROM serienfolge
     WHERE serientitel like 'Buffy%'
     GROUP BY serientitel,drehjahr,staffelnr
  ) as tmp
  GROUP BY serientitel,drehjahr
```

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

Fullselect

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).

db2s1e81.pdf, S. 579

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 Schauspieler aus der Serie „Buffy the Vampire Slayer" und dem Film „Buffy the Vampire Slayer" von 1992:

```
  SELECT schauspielerid
  FROM spieltInFolge
  WHERE serientitel='Buffy the Vampire Slayer'
UNION
  SELECT schauspielerid
  FROM spieltInFilm
  WHERE titel='Buffy the Vampire Slayer' and drehjahr=1992
```

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 Schauspieler, die sowohl in der Serie von 1997, als auch dem Film „Buffy the Vampire Slayer" von 1992 mitgespielt haben:

```
  SELECT schauspielerid
  FROM spieltInFolge
  WHERE serientitel='Buffy the Vampire Slayer'
INTERSECT
  SELECT schauspielerid
  FROM spieltInFilm
  WHERE titel='Buffy the Vampire Slayer' and drehjahr=1992
```

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 Schauspieler aus der ersten „Buffy"-Staffel, die nicht (!) auch in der siebten Staffel mitgespielt haben:

```
  SELECT schauspielerid
  FROM spieltInFolge
  WHERE serientitel='Buffy the Vampire Slayer'
    AND staffelnr=1
EXCEPT
  SELECT schauspielerid
  FROM dbprak.spieltInFolge
  WHERE serientitel='Buffy the Vampire Slayer'
    AND staffelnr=7
```

Views

Ein wichtiges Datenbankkonzept sind Sichten oder Views. Diese können in SQL mit

db2s2e81.pdf, S. 470ff

dem CREATE VIEW-Statement erzeugt werden.

Eine Sicht, die nur die Nicks, Wohnorte und Mailadressen der Anwender angibt, Passwörter und Realnamen dagegen verschweigt:

```
CREATE VIEW Nicks AS
  SELECT nick,wohnort,email
  FROM anwender
```

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 film (
   titel VARCHAR(200) NOT NULL,
   drehjahr INTEGER NOT NULL
)

CREATE TABLE serienfolge (
   serientitel VARCHAR(200) NOT NULL,
   drehjahr INTEGER NOT NULL
   folgentitel VARCHAR(200),
   staffelnr INTEGER NOT NULL,
   folgennr INTEGER NOT NULL
)

CREATE VIEW produktionen AS
    SELECT titel,drehjahr FROM film
  UNION
    SELECT serientitel AS titel,drehjahr FROM serienfolge
```

Ebenso wie in diesem Beispiel der Obertyp als Sicht realisiert wurde, lassen sich auch Untertypen als Sichten eines Obertyps 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:

- `FLOAT(anzahl)`

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

- `DEC(durchschnittAlsFloat,4,2)`

  stellt den Wert von `durchschnittAlsFloat` als Dezimalzahl mit insgesamt 4 Stellen, davon 2 Stellen hinter dem Komma, dar

- `SUBSTR(nick, 1, 2)`

  liefert die ersten 2 Zeichen der Werte von `nick`

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

Sequenzen generieren

Wenn Ihr Daten aus fremden Datenbanken importieren wollt, stellt sich möglicherweise die Frage, wie laufende IDs beim Import generiert werden können. Was zu tun ist, wenn in der Zieltabelle `IDENTITY`-Spalten definiert sind, wurde auf dem letzten Blatt behandelt.

Eine andere Möglichkeit ist das Anlegen einer benannten Sequenz mit dem Befehl `CREATE SEQUENDE`. Das folgende Beispiel generiert eine bei 1 beginnende Integer-Reihe, die in jedem Schritt um eins erhöht wird:

```
CREATE SEQUENCE myids
  AS INTEGER
  START WITH 1 INCREMENT BY 1
```

Nun kann man zum Beispiel beim Einstellen von Daten in eine Tabelle diese generierten IDs verwenden:

```
INSERT INTO film (id, titel, drehjahr)
   SELECT NEXT VALUE FOR myids, q.titel, q.drehjehr
   FROM quelltabelle q
```

Man setzt die Sequenz zurück durch:

```
ALTER SEQUENCE myids RESTART
```

Übersicht über die neuen Befehle

select ...	Anfrage an die Datenbank
create view *name*	erstelle eine Sicht auf eine oder mehrere Tabelle(n)
create sequence *name* as integer	erstelle eine Sequenz mit Auto-Inkrement
alter sequence *name* restart	setzt eine Sequenz wieder auf den Startwert zurück

Aufgaben

In der vergangenen Woche habt Ihr Daten zu Filmen und Serienfolgen aus der Beispieldatenbank `FILME` exportiert und in Eure eigene Datenbank integriert. Die Beispieldatenbank `FILME` enthält außerdem noch die Relationen `Schauspieler`, `spieltInFolge` und `spieltInFilm`. Diese Daten dürft Ihr in Eure Datenbank übernehmen, *müsst* dies aber *nicht* (optional).

Vorbereitung (Hausaufgaben)

V1: Anfragen verstehen

Analysiert die folgenden Anfragen an die Beispieldatenbank `FILME`:

Wie sehen die Ergebnistupel aus, d.h. welchen Datentyp und welchen Inhalt haben sie voraussichtlich? Wieviele Tupel wird das Ergebnis enthalten (0,1,mehrere)? Was bewirken die Funktionsaufrufe?

Schlagt Schlüsselwörter und Funktionen in der Dokumentation nach, soweit nötig.

(a)
```
SELECT COUNT(folgennr)
  FROM dbprak.serienfolge
  WHERE serientitel='Babylon 5'
```

(b)
```
SELECT MIN(staffelnr),MAX(staffelnr)
  FROM dbprak.spieltinfolge
  WHERE serientitel='Babylon 5'
    AND rolle LIKE '%Marcus Cole%'
```

(c)
```
SELECT DISTINCT serientitel,
    DATE(CONCAT(RTRIM(CHAR(drehjahr)),'-01-01')) AS test
  FROM dbprak.serienfolge
  WHERE serientitel LIKE 'Buffy%'
```

(d)
```
SELECT name,vorname
  FROM dbprak.schauspieler s
    JOIN
    (SELECT s1.schauspielerid
     FROM dbprak.spieltinfolge s1
     WHERE s1.serientitel='Babylon 5'
       AND s1.staffelnr=1
     GROUP BY s1.schauspielerid
     HAVING COUNT(DISTINCT s1.folgennr) =
       (SELECT COUNT(DISTINCT s2.folgennr)
        FROM dbprak.spieltinfolge s2
        WHERE s2.serientitel='Babylon 5'
       )
    ) tmp ON (s.id=tmp.schauspielerid)
```

V2: Anfrage korrigieren

Die folgende Anfrage soll beantworten, welche Rollen Andreas Katsulas in Serien und Filmen gespielt hat. Warum würde diese Anfrage nicht das erwartete Ergebnis liefern? Was müsste man ändern, damit sie wie vorgesehen funktioniert?

```
  SELECT serientitel AS titel,drehjahr
  FROM dbprak.spieltInFolge, dbprak.schauspieler
  WHERE name='Katsulas' and vorname='Andreas'
UNION
  SELECT titel,drehjahr
  FROM dbprak.spieltInFilm, dbprak.schauspieler
  WHERE name='Katsulas' and vorname='Andreas'
```

V3: Eigene Anfragen vorbereiten

Überlegt Euch, wie Ihr im Präsenzteil die dort geforderten Anfragen umsetzen wollt. Schlagt die Syntax nach und informiert Euch über die benötigten Funktionen.

Schreibt Euch die Anfragen für den Präsenzteil schon zuhause auf.

V4: Sichten vorbereiten

Bereitet den SQL-Code für Präsenzaufgabe P3 vor.

Präsenz

P0: Testdaten eintragen bzw. importieren

Vervollständigt die Datenbasis durch Import der wichtigsten Daten aus der Beispieldatenbanken `FILME`. Tragt auch noch ein paar weitere Nutzer mit ihren Daten ein.

P1: Testdaten abfragen

Formuliert folgende Anfragen an Eure *eigene* Datenbank. Schreibt jeweils den SQL-Befehl und die ersten 3 Zeilen des Ergebnisses auf:

(a) Welche Daten sind zu den Nicks „Adam" und „Ophelia" bekannt?

(b) In welchen Sprachen ist der Film „Pirates of the Caribbean: The Curse of the Black Pearl" vorhanden?

(c) Welche Medien sind derzeit ausgeliehen (und noch nicht zurückgegeben)?

(d) Wer besitzt die meisten Serienfolgen oder Filme? (Hier geht es um die Gesamtzahl pro Anwender, d.h. Serienfolgen + Filme, und dabei dann um den Anwender, der die größte Anzahl erreicht.)

P2: Anfragen an die Beispieldatenbank

Formuliert folgende Anfragen an die Beispieldatenbank `FILME`. Schreibt jeweils den SQL-Befehl auf:

(a) Ermittle 10 verschiedene Schauspieler (Name,Vorname,Rolle), die in der Serie „Babylon 5" mitgewirkt haben.

(b) Welche Schauspieler haben in der Serie „Friends" schon einmal sich selbst gespielt? (Hinweis: Im Feld `Rolle` steht dann „Himself" oder „Herself".)

(c) In welchen Jahren sind Filme zu „Star Wars" entstanden, in denen nicht auch ein „Lord of the Rings"-Film produziert wurde?

P3: View definieren

Erstellt in Eurer *eigenen* Datenbank eine Sicht (View) `Serie` für Serien. Die Ergebnistupel sollen Titel und Drehjahr der Serien enthalten, und jede Serie soll nur einmal genannt werden.

P4: Weitere Anfragen an die eigene Datenbank

(a) Welche Anwender (Nick,Name,Vorname) haben ein Telefon?

(b) Wie lauten die Folgentitel der „Buffy"-Folgen, die Eva Musterfrau besitzt?

(c) Wieviele Medien besitzen die einzelnen Anwender jeweils?

(d) Wie lange wurden Medien durchschnittlich ausgeliehen?

(e) Welches sind die 10 beliebtesten Filme oder Serienfolgen (d.h. welche wurden am häufigsten ausgeliehen)?

P5: Weitere Anfragen an die `FILME`-Datenbank

(a) In welchen Filmen tritt Woody Allen auf?

(b) Welche Schauspieler haben in mehr als 500 Filmen mitgespielt? Ermittle von diesen die 10 Schauspieler mit den meisten Filmen und gib auch die Anzahl der Filme an.

(c) In wieviel *Prozent* der Folgen von „Babylon 5" spielt Andreas Katsulas mit?

(d) Welche(r) Film(e) haben die meisten Schauspieler?

(e) Wieviele Schauspieler hat ein Film durchschnittlich?

P6: Verwendung von Sichten

Ein Kunde möchte, dass Ihr eine Sicht `EigeneMedien` erstellt, die für einen bestimmten Anwender nur die Leihobjekte anzeigt, die ihm gehören, mit ihrem aktuellen Ausleihstand (verliehen oder verfügbar).

Zum Schluss ...
stoppt bitte Eure Datenbank-Manager-Instanz auf `salz`**,**
loggt Euch von `salz` **und vom lokalen Rechner aus,**
aber schaltet den Rechner nicht ab.