

## SQL-Lab – Aufgabenblatt 4 – Three degrees of Heath Ledger

### Szenario

Damit *NET.FIRST* (siehe [voriges Aufgabenblatt](#)) ein großer kommerzieller Erfolg wird und um die Benutzerzahlen zu steigern, möchte euer Boss die Attraktivität des Angebots erhöhen, indem angemeldeten Benutzern verschiedene „APPS“ zur Verfügung gestellt werden. Zum Glück hat er genug Vertrauen, um die Aufgabe nicht an eine Drittfirma auszulagern, sondern *EUCH* damit zu betrauen. Bevor es ans Programmieren geht, sollt ihr euch aber zunächst Gedanken über mögliche Anwendungen machen.



Euer Boss konsultiert einen *Web2.0-Experten*, der prompt mit einer Idee aufwarten kann:

**Implementiert die *six degrees of Kevin Bacon* – ohne *Kevin Bacon*, aber mit *Heath Ledger* und nur für 2008!**

Euer Boss ist überzeugt – also seid *IHR* es auch. Am Anfang des Projekts gibt euch der Experte noch ein wenig Hilfestellung durch einen detailliert ausgearbeiteten 10-Punkte-Plan, der vor dem Programmieren bearbeitet werden muss!

### 10-Punkte-Plan

Die folgenden 10 Aufgaben sollen in diesem Aufgabenblatt bearbeitet werden!

**Aufgabe 1:** Um *relational* zu sein, muss ein DBMS nach Edgar F. Codd (dem Erfinder des Relationalen Modells) zwölf Regeln genügen. Regel vier gibt vor, dass die Datenbank *selbstbeschreibend* sein muss – also spezielle Systemtabellen haben muss, die ihre eigene Struktur beschreiben. Diese Systemtabellen werden auch als *database catalog* bezeichnet. Ein Überblick über DB2s *database catalog* kann unter folgendem Link gefunden werden:

<http://pic.dhe.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=%2Fcom.ibm.db2.luw.sql.ref.doc%2Fdoc%2Fr0011297.html>

- Gib mit einer *SELECT* Anfrage auf den *database catalog* alle *IMDB* Schema **Tabellen** aus.
- Stell eine *SELECT* Anfrage, die die Struktur der *IMDB.MOVIES* Tabelle ausgibt. Gib nur die Attribute aus, die in einem *CREATE* benötigt werden würden.
- Kopiere manuell die *IMDB.MOVIES* Tabelle als *tv\_movies\_tmp* in **dein Schema** (ohne die Daten). Nutze die Ergebnisse aus **b.** und verwende **nicht** *CREATE TABLE LIKE*.

**Aufgabe 2:** Eine andere Möglichkeit, die Struktur einer Tabelle zu kopieren, ist es, das CREATE TABLE LIKE Statement zu benutzen. Kopiere damit die *IMDB.ACTORS* Tabelle als *tv\_actors\_tmp* (nur die Struktur, nicht die Daten) in dein eigenes Schema!

**Aufgabe 3:** Keines der Attribute der Tabellen im IMDB Schema hat einen NOT NULL Constraint. Sie sollen in dieser Aufgabe nachträglich hinzugefügt werden!

- a. Finde (evtl. über den *database catalog*) heraus, welche der Attribute der *IMDB.MOVIES* und *IMDB.ACTORS* Tabellen überhaupt keine NULL-Werte enthalten.
- b. Füge in den *tv\_movies\_tmp* und *tv\_actors\_tmp* Tabellen in deinem Schema **nachträglich** NOT NULL Constraints zu den Attributen hinzu, die in der IMDB keine NULL VALUES enthalten. Orientiere dich dabei an den Ergebnissen aus a..

**Aufgabe 4:** Kopiere die Struktur der *tv\_movies\_tmp* und *tv\_actors\_tmp* Tabellen erneut (mit einer beliebigen Technik) in zwei neue Tabellen. Nenne Sie *tv\_movies* und *tv\_actors*.

**Aufgabe 5:** Füge nachträglich in *tv\_movies* und *tv\_actors* passende PRIMARY KEYS und FOREIGN KEYS ein.

*Tipps:* Es ist wichtig, dass alle Attribute für PRIMARY KEYS vorher bereits NOT NULL sind!

TIPP 2: Es können auch mehrere Attribute einen PRIMARY KEY bilden!

**Aufgabe 6:** Jetzt sollen *tv\_movies* und *tv\_actors* mit Daten aus den entsprechenden IMDB Tabellen gefüllt werden. Dabei sollen nur TV movies aus dem Jahr 2008 und alle ihre *actors* und *actresses* eingetragen werden.

*Tipps:* Hier sind korrekt gesetzte PRIMARY KEYS wichtig!

**Aufgabe 7:** Nachdem die *tv\_movies* und *tv\_actors* Tabellen erstellt wurden, kann nun die Grundlage für die *three degrees of Heath Ledger* gelegt werden. Dazu soll ein *materialized view* erstellt werden. Views wurden bereits in der [ersten RDBI Vorlesung](#) angesprochen. Sie bieten im Grunde eine andere Sicht auf die Daten in der Datenbank. Während eine normale view nicht viel mehr als ein benanntes SELECT Statement ist, ist eine *materialized view* ein SELECT Statement, das ausgewertet und physikalisch auf der Festplatte abgelegt wurde. Es gibt zwei Arten von *materialized views*: Die erste Art wird neu berechnet (und wieder physikalisch gespeichert), falls sich Daten in den Tabellen, auf denen es basiert, ändern. Die andere Art wird bei solchen Datenänderungen **nicht** neu berechnet und ist fortan in einem inkonsistenten Zustand. DB2 bezeichnet *materialized views* alternativ auch als *materialized query tables* (<http://www.ibm.com/developerworks/data/library/techarticle/dm-0509melnky/>).

Erstelle eine *materialized view* namens *actor\_co\_occurrences*, die auf *tv\_movies* und *tv\_actors* basiert und die für jeden *actor* alle anderen *actors* enthält, mit denen er 2008 zusammen gespielt hat und den dazugehörigen *movie*. Sie soll **nicht** neu berechnet werden!

Sie soll folgende Signatur haben: `actor_co_occurrences(actor1, actor2, movie)`

**Aufgabe 8:** Um den Zugriff auf die erstellte *materialized view actor\_co\_occurrences* zu beschleunigen, sollen jetzt passende Indexe auf ihre Attribute angelegt werden. Dazu soll das CREATE INDEX Statement benutzt werden, das für DB2 unter folgendem Link erklärt wird: <http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp?topic=/com.ibm.db2.udb.doc/admin/r0000919.htm>

Leg jeweils für *actor1*, *actor2* und *movie* aus der *actor\_co\_occurrences* Tabelle einen eigenen Index an!

**Aufgabe 9:** Nach der vielen Vorarbeit kann jetzt endlich eine Anfrage formuliert werden, die einen ersten Eindruck von den *three degrees of Heath Ledger* gibt (das in der Einführung angekündigte Programm wird erst im nächsten Aufgabenblatt entwickelt). Ziel dieser Anfrage ist folgendes:

Finde alle Schauspieler, die eine *Heath Ledger Zahl* (angelehnt an die *Kevin Bacon Zahl*) von maximal 3 haben. Gib ihre Namen und ihre *Heath Ledger Zahl* aus. Das benutzte SELECT Statement soll **nicht** rekursiv sein!

**Aufgabe 10:** In der [neunten RDBI Vorlesung](#) wurden bereits *rekursive SELECT Statements* vorgestellt. Sie werden auch in DB2 (mit Einschränkungen) unterstützt. In dieser Aufgabe soll ein solches **rekursives** SELECT Statement geschrieben werden, das die Anfrage aus Aufgabe 9 löst.



#### Liste der Abgaben für Aufgabenblatt 4

- Für **alle** zehn **Aufgaben** müssen **alle SQL Statements**, die zum Lösen der jeweiligen Aufgabe benutzt wurden, abgegeben werden.
- Jedes Statement soll **gut lesbar formatiert** sein.
- Die abgegebenen Statements sollen in der angegebenen Reihenfolge **fehlerfrei ausführbar** und euer Lösungsweg damit für euren Hiwi **reproduzierbar** sein.
- Alle Statements sollen sowohl **lesbar ausgedruckt** auf **DINA4** eingereicht werden, als auch als **Textfile** (.txt) per **Email** an euren Hiwis gesendet werden.