Praktikum Datenbanken / DB2
Woche 4: SQL als DDL (Fortsetzung)

Betreuer: Gudrun Fischer, Tobias Tuttas, Camille Pieume
Raum: LF 230
Bearbeitung: 29. Mai - 1. Juni 2006

Datum	
Team (Account)	
Vorbereitung	
Präsenz	

Aktuelle Informationen, Ansprechpartner und Material unter:
`http://www.is.informatik.uni-duisburg.de/courses/dbp_ss06/index.html`

Allgemeine Hinweise:

Das Druckproblem aus der ersten Woche wurde behoben. Ihr solltet nun auch im Datenbanken-Pool Unterlagen ausdrucken können.

Wochenziele

In der letzten Woche hattet Ihr Eure erste Tabelle in Eurer eigenen Datenbank angelegt. In dieser Woche wollen wir nun

- den Entwurf an neue bzw. detailliertere Anwendervorgaben anpassen
- Konsistenzbedingungen definieren
- die vollständige Tabellenstruktur in der Datenbank anlegen

Hinweise: Vorgehen bei den Präsenzaufgaben

Bitte arbeitet nur dann, wenn es unbedingt nötig ist, auf dem Server `salz`. Das heißt für den Praxisteil:

1) Loggt Euch auf einem der normalen Datenbank-Pool-Rechner ein.

   Eine Liste von Rechnernamen findet Ihr im Materiel „Erste Schritte ubter Linux" aus der ersten Woche.

2) Startet Eure Datenbank-Manager-Instanz auf `salz` *und verlasst den Rechner* `salz` *anschließend wieder.*

   Beispiel:

```
dbp0638@curry:~$ ssh salz
Enter passphrase for key '/home/dbp0638/.ssh/id_dsa':
Linux salz 2.4.25-is #1 Wed Mar 3 17:34:58 CET 2004 i686 GNU/Linux

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

Last login: Tue May 23 17:00:12 2006 from curry.is.informatik.uni-duisburg.de
dbp0638@salz:~$ db2start
SQL1063N  DB2START processing was successful.
dbp0638@salz:~$ exit
logout
Connection to salz closed.
dbp0638@curry:~$
```

3) Verbindet Euch mit Eurer Datenbank.

   Beispiel:

```
dbp0638@curry:~$ db2 connect to video38

   Database Connection Information

 Database server        = DB2/LINUX 8.1.2
 SQL authorization ID   = DBP0638
 Local database alias   = VIDEO38

dbp0638@curry:~$
```

4) Löst Eure Präsenzaufgaben.

5) Meldet Euch von Eurer Datenbank wieder ab.

   Beispiel:

```
dbp0638@curry:~$ db2 terminate
```

```
DB20000I  The TERMINATE command completed successfully.
dbp0638@curry:~$
```

6) Stoppt Eure Datenbank-Manager-Instanz auf `salz` und verlasst den Rechner `salz` anschließend wieder.

   Beispiel:

```
dbp0638@curry:~$ ssh salz
Enter passphrase for key '/home/dbp0638/.ssh/id_dsa':
Linux salz 2.4.25-is #1 Wed Mar 3 17:34:58 CET 2004 i686 GNU/Linux

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

Last login: Wed May 24 12:54:44 2006 from curry.is.informatik.uni-duisburg.de
dbp0638@salz:~$ db2stop
SQL1064N  DB2STOP processing was successful.
dbp0638@salz:~$ exit
logout
Connection to salz closed.
dbp0638@curry:~$
```

7) Loggt Euch auch vom lokalen Rechner aus.

SQL als DDL (Fortsetzung)

Tabellenerstellung: Schlüssel und Constraints

Die im Folgenden beschrieben Konsistenzbedingungen können direkt beim Erzeugen einer Tabelle als Teil des `CREATE TABLE`-Befehls mit angegeben werden.

Primärschlüssel

Mit PRIMARY KEY wird für die Tabelle ein Primärschlüssel festgelegt. Dieser besteht aus den angegebenen Attributen. Die Attribute im Primärschlüssel dürfen keine Nullwerte zulassen. Soll der Schlüssel nur aus einem Attribut bestehen, dann genügt es, das Schlüsselwort hinter dem jeweiligen Attribut anzugeben:

```
id SMALLINT NOT NULL PRIMARY KEY,
```

Sekundärschlüssel

Durch das Schlüsselwort UNIQUE wird ein Sekundärschlüssel festgelegt. Auch für diesen wird automatisch ein Index erstellt und es gelten die gleichen Einschränkungen wie für Primärschlüssel. Attribute eines Sekundärschlüssels dürfen nicht schon Teil eines anderen Schlüssels sein. Beispiel:

```
jobnr INTEGER NOT NULL UNIQUE,
```

Fremdschlüssel

Fremdschlüssel werden benutzt, um zu erzwingen, dass ein oder mehrere Attribute einer abhängigen Tabelle nur Werte annehmen dürfen, die auch als Werte eines Schlüssels einer anderen Tabelle auftreten. Dieses Konzept wird auch *Referentielle Integrität* genannt. Der Benutzer muss das REFERENCES- oder ein übergeordnetes Recht auf der referenzierten Tabelle besitzen. Die Tabelle, auf die verwiesen wird, muss existieren und die referenzierten Attribute müssen dort als Schlüssel definiert sein.

Fremdschlüssel werden entweder als *Constraint* (s.u.) oder mit dem Schlüsselwort REFERENCES angegeben. In diesem Beispiel wird durch das Attribut `jobnr` die Spalte `internal_number` der Tabelle `jobs` referenziert:

```
jobnr INTEGER REFERENCES jobs(internal_number)
```

Checks

Bei der Definition eines Tabellenattributes kann zusätzlich zum Datentyp eine Einschränkung der Werte angegeben werden. Dies wird mit dem Schlüsselwort CHECK eingeleitet:

```
department SMALLINT CHECK (department BETWEEN 1 AND 5),
```

```
job CHAR(10) CHECK (job in ('Professor','WiMi','NichtWiMi')),
```

Constraints

Checks und *Fremdschlüssel* können auch als benannte *Constraints* mit dem Schlüsselwort CONSTRAINT angegeben werden:

```
CONSTRAINT jobref FOREIGN KEY jobnr REFERENCES jobs (int_no)
```

```
CONSTRAINT yearsal CHECK (YEAR(hiredate) > 1996 OR SALARY > 1500)
```

Beispiel

Ein Beispiel für das Erzeugen einer Tabelle mit mehreren Konsistenzbedingungen wäre:

```
CREATE TABLE employee (
   id           SMALLINT NOT NULL,
   name         VARCHAR(50),
   department   SMALLINT CHECK (department BETWEEN 1 AND 5),
   job          CHAR(10) CHECK (job in ('Prof','WiMi','NiWiMi')),
   hiredate     DATE,
   salary       DECIMAL(7,2),

   PRIMARY KEY (ID),
   CONSTRAINT yearsal CHECK (YEAR(hiredate) > 1996
                             OR SALARY > 1500)
)
```

Index

Ein Index ist ein Datenbankobjekt, das den Zugriff über bestimmte Attribute einer Tabelle beschleunigen soll und dies in den allermeisten Fällen auch tut. Man kann auch (UNIQUE)-Indexe verwenden, um die Einhaltung von Schlüsselbedingungen sicherzustellen.

Werden bei der Tabellendefinition Primärschlüssel- (PRIMARY KEY) oder Sekundärschlüsselbedingungen (UNIQUE) angegeben, so legt DB2 selbständig Indexe an, um die Einhaltung dieser sicherzustellen. Ein einmal angelegter Index wird dann automatisch bei Änderungen an den Inhalten der entsprechenden Tabelle gepflegt. Dies ist natürlich mit einem erhöhten Verwaltungsaufwand verbunden.

Näheres zu Indexen und den zur Realisierung verwendeten Datenstrukturen in der Vorlesung **Datenbanken** oder in der begleitenden Literatur (siehe Blatt der ersten Woche).

Das Anlegen eines Index geht mit dem CREATE INDEX-Statement, das Löschen entsprechend über DROP INDEX. Man benötigt dazu mindestens das

db2s2e81.pdf, S. 268

INDEX- oder das CONTROL-Recht für die Tabelle, sowie das CREATEIN-Recht im angegebenen Schema. Der Benutzer, der einen Index anlegt, erhält auf diesem das CONTROL-Recht.

Über das Schlüsselwort UNIQUE kann man bei der Erstellung eines Index angeben, dass keine Tupel in allen Werten der angegebenen Attribute übereinstimmen dürfen. Sollten zum Zeitpunkt der Indexgenerierung Tupel in der Tabelle dagegen verstoßen, so wird eine Fehlermeldung ausgegeben und der Index nicht angelegt.

Es ist nicht möglich, zwei gleiche Indexe anzulegen. Dabei darf ein Index 16 Attribute umfassen und die Attribute dürfen zusammen nicht länger als 1024 Bytes sein. Die Ordnung ASC bzw. DESC gibt an, ob die Indexeinträge in aufsteigender bzw. absteigender Reihenfolge angelegt werden. Nur wenn UNIQUE angegeben wurde, können über die INCLUDE-Klausel weitere Attribute in den Index aufgenommen werden, für die aber keine Schlüsselbedingung gelten soll.

Beispiel:

```
CREATE INDEX adressen
   ON anwender (adresse)
```

Indexe werden nicht explizit aufgerufen, sondern vom Datenbank-Managementsystem implizit bei der Bearbeitung von Anfragen herangezogen.

Verändern von Tabellendefinitionen

Zum Ändern einer Tabellendefinition sind das ALTER- oder CONTROL-Recht an der zu ändernden Tabelle, das ALTERIN-Recht für das Schema der entsprechenden Tabelle oder ein übergeordnetes Recht nötig. Um eine Fremdschlüsselbeziehung einzuführen oder zu entfernen benötigt man für die referenzierte Tabelle zusätzlich noch das REFERENCES-, das CONTROL-, das DBADM- oder das SYSADM-Recht. Entsprechend muß beim Löschen einer Schlüsselbedingung für alle die zugehörigen Attribute referenzierenden Tabellen die anfangs genannten Rechte bestehen.

Geändert wird eine Tabellendefinition durch ein ALTER TABLE-Statement: *db2s2e80.pdf, Seite 41ff*

```
ALTER TABLE example.employee
    ALTER name SET DATA TYPE VARCHAR (100)
    ADD hasemail CHAR(1)
    DROP CONSTRAINT yearsal
```

Das Beispiel demonstriert die drei weesentlichen Änderungsmöglichkeiten in einem ALTER TABLE-Statement für die (hypothetische) Tabelle EMPLOYEE im Schema EXAMPLE:

- eine Attributdefinition wird geändert

  Über die ALTER-Klausel können VARCHAR Attribute vergrößert (aber nicht verkleinert) werden, außerdem kann durch diese Klausel der Ausdruck zur Erzeugung von automatisch generierten Attributwerten verändert werden.

- ein neues Attribut wird hinzugefügt

  Bei der Verwendung der ADD-Klausel gilt das gleiche wie beim Erstellen einer neuen Tabelle. Wird eine neue Schlüsselbedingung hinzugefügt, und existiert noch kein Index hierfür, so wird ein neuer Index angelegt. Die ADD COLUMN-Klausel wird stets zuerst ausgeführt.

- ein benannter CONSTRAINT wird aus der Tabelle gelöscht

  Man kann keine Attribute löschen und keine Constrants oder Checks, die direkt und ohne Bezeichner bei einer Attributdefinition angegeben wurden.

Übersicht über die neuen Befehle (Woche 3/4: SQL als DDL)

create schema *name*	erstelle ein neues Schema (einen Namensraum für Datenbankobjekte)
drop schema *name* restrict	lösche ein existierendes Schema
create table *name*	erstelle eine neue Tabelle
drop table *name*	lösche eine existierende Tabelle
create index *name* on *table*	erzeuge einen neuen Index auf einer Tabelle
describe table *name* show detail	zeige Details zu einer existierende Tabelle an
describe indexes for table *name*	zeige die Indexe auf der Tabelle *name* an
alter table *name*	ändere die Definiation einer existierenden Tabelle

Aufgaben

Vorbereitung (Hausaufgaben zur Semesteraufgabe)

Gegeben seien die folgenden (größtenteils schon vom letzten Mal bekannten) Testfakten:

1) Ein neuer Anwender trägt sich in das System ein. Er heißt Adam Boyce-Codd, wohnt in Duisburg und hat die Telefonnummern 0203 1234567 und 0173 7654321. Seine E-Mail-Adresse lautet „dbprak@is.informatik.uni-duisburg.de". Sein Passwort lautet „rumpelstilzchen".
   Eva Musterfrau wohnt in Oberhausen, hat kein Telefon, ist aber unter der E-Mail-Adresse „eva@nowhere-in-the.net" zu erreichen. Ihr Passwort lautet „rotkaeppchen".

2) Adam Boyce-Codd hat den Film „Herr der Ringe" aus dem Jahr 2003 als „Special Extended Edition" auf 6 DVDs. Er will die DVDs aber nur zusammen verleihen.

3) Eva Musterfrau hat von der Serie „Buffy" aus dem Jahr 1997 die erste Staffel auf 3 DVDs. Adam Boyce-Codd leiht sich von Eva am 01.06.2006 die erste dieser Buffy-DVDs (mit den Folgen 1 bis 4) und gibt ihr die DVD am 03.06.2006 zurück.

4) Eva Musterfrau hat den Film „Romeo und Julia" aus dem Jahr 1996 (IMDB-Link: `http://www.imdb.com/title/tt0117509/`) auf VHS (Videokassette) im englischen Originalton. Adam Boyce-Codd besitzt eine DVD mit demselben Film in den Sprachen Englisch, Deutsch und Niederländisch.

Nun stellt Euch vor:

Nach dem Entwurf in Woche 3 sind mehrere Detailfragen aufgekommen, die mit den Kunden (den zukünftigen Anwendern der „dezentralen Videothek") geklärt werden mussten.

Ein Gespräch mit den Kunden ergab folgende neue bzw. detailliertere Anforderungen:

- Ein Anwender soll eine interne, eindeutige ID haben, die vom System generiert wird. Außerdem soll er aber auch einen selbstgewählten, eindeutigen Nicknamen haben können (damit er sich die ID nicht merken muss).
- Anwender sollen beliebig viele Telefonnummern und genau eine E-Mail-Adresse haben können. Telefonnummern dürfen aus Ziffern und den Zeichen ‚-' und ‚/' bestehen. E-Mail-Adressen müssen genau ein ‚@' enthalten.
- Ein *Film* ist durch Titel und Drehjahr eindeutig identifiziert, eine *Serie* ebenfalls. Zu Serien gibt es jedoch noch *Staffeln*, die durch Serientitel, Drehjahr der Serie und Staffelnummer eindeutig identifiziert sind. „Drehjahr" ist dabei etwas allgemeiner als das Jahr zu verstehen, in dem der erste Teil der Serie herauskam. Die Anwender möchten nachher auch nach einzelnen *Folgen* einer Serie suchen können, die entsprechend durch Serientitel, Drehjahr, Staffelnummer und Folgennummer identifiziert sind.

Eine Serienfolge kann auch noch einen eigenen Titel besitzen, muss aber nicht.

- Ein Medium soll tatsächlich ein physischer Datenträger sein. Wenn eine Produktion (Film oder Serienfolge) über mehrere Datenträger verteilt ist (Beispiel: „Herr der Ringe" auf 6 DVDs), so soll jedes Medium (also im Beispiel: jede der 6 DVDs) einzeln gespeichert werden. Dabei soll pro Medium nicht nur nachvollziehbar sein, welcher Teil einer Produktion darauf gespeichert ist, sondern auch, wieviele Teile es insgesamt gibt.
- Ein Medium kann durchaus mehrere Produktionen enthalten (Beispiel: „Buffy", 1997, Staffel 1, Folgen 1 bis 4 auf einer DVD).
- Wenn zwei Medien eigentlich gleich sind (Beispiel: Adam und Eva besitzen beide die gleiche DVD-Ausgabe von „Herr der Ringe"), dann sollen die Informationen trotzdem doppelt gespeichert werden. Die dadurch entstehende Redundanz nehmen die Anwender in Kauf.

V1: Anwenderwünsche und Testfakten

(a) Passt Eure Modelle aus Woche 2 und Eure Relationen aus Woche 3 so an, dass die neuen Anwenderwünsche berücksichtigt werden.

(b) Prüft, ob sich die oben angegebenen Testfakten 1 bis 4 in Euren Relationen ausdrücken lassen.

Welche Tupel werden beispielsweise in welchen Relationen generiert, um Testfaktum 2 auszudrücken:
Adam Boyce-Codd hat den Film „Herr der Ringe" aus dem Jahr 2003 als „Special Extended Edition" auf 6 DVDs. Er will die DVDs aber nur zusammen verleihen.

Gebt also für die 4 Testfakten die entsprechenden Tupel an.

(c) Gibt es Details in den Testfakten, die Ihr *nicht* ausdrücken könnt? Welche, und warum? Überlegt Euch dazu eine Lösung.

V2: Schlüssel und weitere Konsistenzbedingungen

Notiert zu jeder Eurer Relationen:

- Was ist der Primärschlüssel?
- Gibt es Sekundärschlüssel (also Attribute, deren Werte ebenfalls eindeutig sein müssen und nicht leer sein dürfen)?
- Gibt es Fremdschlüssel?
- Gibt es Attribute, deren Werte weiter eingeschränkt werden müssen?

V3: Constraints in SQL

Formuliert jeweils eine Fremdschlüsselbedingung und einen Wertebereichsprüfung aus Eurem Entwurf als benannten Constraint in SQL.

V4: Index

Gibt es in Euren Relationen Nichtschlüsselattribute (außer der Adresse bei den Anwenderdaten), auf die besonders schnell zugegriffen werden sollte?

Wenn nein: Woran liegt das?

Wenn ja: Entscheidet Euch für ein Beispiel und formuliert die Indexerstellung in SQL.

Präsenz

P1: Konsolidierung des Entwurfs

Setzt Euch mit Eurem Teampartner zusammen und vergleicht Eure neuen Entwürfe. Einigt Euch auf einen gemeinsamen Entwurf für das Relationenschema, passende Schlüssel und Constraints, und haltet diesen gemeinsamen Entwurf beide schriftlich fest.

P2: Tabellen anlegen und/oder anpassen

(a) Passt die Tabelle der Anwenderdaten, die Ihr in Woche 3 angelegt hattet, an das neue Schema an. Verwendet dazu den SQL-Befehl `ALTER TABLE`.

(b) Legt die restlichen Tabellen Eures Schemas in Eurer Datenbank an. Berücksichtigt dabei auch die Schlüssel- und Konsistenzbedingungen aus den Aufgaben V2 und P1.

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.