Praktikum Datenbanken / DB2
Woche 3: Tabellenentwurf und -erstellung

Betreuer: Sascha Kriewel, Tobias Tuttas
Raum: LF 230
Bearbeitung: 21.-25 Mai 2007

Datum	
Team (Account)	
Vorbereitung	
Präsenz	

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

Wochenziele

In der vergangenen Woche haben wir die Mini-Welt für die Semesteraufgabe in einem Entity-Relationship-Diagramm und in einem UML-Klassendiagramm modelliert.

Außerdem haben wir für jeden Praktikumsaccount *in dessen DB2-Instanz* eine eigene, neue Datenbank für die Semesteraufgabe angelegt.

In dieser Praktikumswoche wollen wir nun

- den Entwurf an neue bzw. detailliertere Anwendervorgaben anpassen,
- die Tabellenstruktur für die Semesteraufgabe entwerfen,
- Konsistenzbedingungen definieren und
- die Tabellenstruktur in der Datenbank anlegen.

Entwurfsschritt 2: Relationenschema

Als zweiter Schritt beim Entwurf einer Datenbank steht nach der Modellierung die Umsetzung des Modells in ein Relationenschema an. Ziel soll ein Schema sein, das möglichst direkt in einer konkreten relationalen Datenbank implementiert werden kann.

Die Informationen in diesem Abschnitt sollten schon aus der Vorlesung bekannt sein.

Im Allgemeinen gibt es bei dieser Umsetzung mehrere geeignete Schemavarianten. Eine erste Annäherung für die Umsetzung kann wie folgt aussehen:

- ein Entitätentyp wird in eine Relation mit den gleichen Attributen wie der Entitätentyp überführt
- eine Beziehung wird in eine Relation überführt, deren Attribute die Schlüssel der von ihr verbundenen Relationen sind

Normalerweise gibt es allerdings noch einige andere Dinge zu beachten. Im Folgenden werden ein paar einfache Faustregeln dazu angegeben. Dabei ist zu bedenken, dass in Hinblick auf eine konkrete Anwendung es manchmal sinnvoll sein kann, eine theoretisch nicht optimale Umsetzung zu wählen.

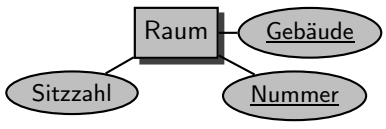
Relationenschemata werden wie in **Kemper/Eick** beschrieben spezifiziert:

SchemaName : { [$\underline{\text{Attribut}_1}$: Typ_1, Attribut_2: Typ_2, ...] }

Dabei wird der Primärschlüssel einer Relation durch Unterstreichung gekennzeichnet. Attribute innerhalb einer Relation müssen eindeutig benannt sein.

Entitätentypen

Ein Entitätentyp wird normalerweise als Relation abgebildet, deren Attribute die Attribute dieses Typs sind. Diese Relation hat zunächst einmal keinerlei Verbindung zu den Beziehungen, an denen der Entitätentyp teilnahm. Mehrwertige Attribute können in ein mengen- oder listenwertiges Attribut abgebildet werden. Diese werden dann bei der Normalisierung behandelt.



Räume : {[Nummer: integer, Gebäude: string, Sitzzahl: integer]}

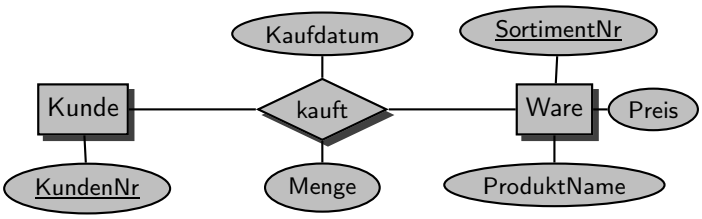
Zusammengesetzte Attribute werden entweder aufgelöst, d.h. die Attribute des zusammengesetzten Attributes werden zu Attributen der Umsetzung des Entitätstyps, oder sie werden als eigene Relation aufgefasst. Diese neue Relation enthält alle Attribute des zusammengesetzten Attributes und alle Primärschlüsselattribute der Ursprungsrelation. Diese Fremdschlüssel bilden den Primärschlüssel der neuen Relation.

Beziehungstypen

Ein Beziehungstyp wird üblicherweise als eine eigene Relation abgebildet. Als Attribute für diese Relation werden herangezogen:

- die Attribute des Beziehungstyps
- die Primärschlüsselattribute der teilnehmenden Entitätstypen als Fremdschlüssel

Der Primärschlüssel der Relation enthält die Fremdschlüssel und eventuell zusätzliche Attribute.



kauft : {[KundenNr: integer, SortimentNr: integer,
Kaufdatum: date, Menge: integer]}

Nimmt ein Entitätentyp mehrfach an einer Beziehung teil, so müssen seine Schlüsselattribute entsprechend oft in die Relation übernommen werden, die dieser Beziehung entspricht. Dabei muss man die Attribute umbenennen, um Namenskonflikte zu vermeiden.

Schwache Entitätstypen

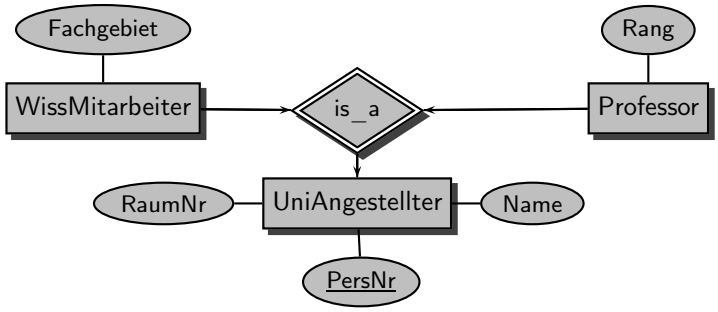
Ein schwacher Entitätstyp wird als eine eigene Relation abgebildet, die zusätzlich zu den eigenen Attributen auch den Primärschlüssel des Vater-Entitätstype als Fremdschlüssel enthält. Dieser bildet zusammen mit den ausgezeichneten Teilschlüsseln des schwachen Entitätstyp den Primärschlüssel der Relation.

Die Beziehung zwischen einem schwachen Entitätstypen und dem Vater-Entitätstyp muss im Allgemeinen überhaupt nicht nachgebildet werden.

Generalisierung

Generalisierung wird durch das relationale Modell nicht direkt unterstützt. Das relationale Modell verfügt über keine Vererbung. Man kann aber versuchen, Generalisierung mit den existierenden Mitteln nachzumodellieren. Dafür gibt es unterschiedliche Strategien, wobei sich die folgende am engsten an die E-R-Modellierung hält:

- Für jeden Entitätstypen E in der Generalisierungshierarchie gibt es eine Relation mit den Schlüsselattributen des Obertypen und den Attributen von E



UniAngestellte : {[PersNr, Name, RaumNr]}

Professoren : {[PersNr, Rang]}

WissMitarbeiter : {[PersNr, Fachgebiet]}

Eine Alternative wäre die Überführung des abgeleiteten Entitätstypen E in eine Relation, die als Attribute alle Attribute des Obertypen sowie die Attribute von E besitzt. In dieser Variante würden in der Relation des Obertyps nur Instanzen gespeichert, die zu keinem der Subtypen gehören.

UniAngestellte : {[PersNr, Name, RaumNr]}

Professoren : {[PersNr, Name, RaumNr, Rang]}

WissMitarbeiter : {[PersNr, Name, RaumNr, Fachgebiet]}

Zusammenführen von Relationen

Die direkte Überführung wie oben beschrieben liefert oft nicht die bestmöglichen Relationen. Häufig kann man etwa noch Relationen zusammenführen. Allgemein gilt: Relationen mit dem gleichen Schlüssel (aber auch nur diese) lassen sich unter bestimmten Voraussetzungen zusammenfassen. Dabei kann es sinnvoll sein, auf eine Zusammenfassung zu verzichten, wenn diese zu zahlreichen NULL-Werten führen würde.

Ein binärer Beziehungstyp R zwischen Entitätentypen E und F, an dem mindestens einer der beiden beteiligten Entitätstypen (sagen wir E) mit Funktionalität 1 teilnimmt, kann mit der E entsprechenden Relation zusammengeführt werden. Dazu führt man eine Relation mit folgenden Attributen ein:

- die Attribute von E
- die Primärschlüsselattribute von F
- die Attribute der Beziehung R

Überlegt selbst, warum das vorteilhafter ist, als die Überführung der Beziehung R in eine eigene Relation. Überlegt auch, was es für Gegenargumente gibt.

Entwurfsschritt 3: Normalformen

Normalformen stellen eine theoretische Basis dar, um relationale Datenbankschemata bewerten zu können. Insbesondere sollen durch Normalisierung Redundanzen und implizite Darstellung von Information verhindert werden. Durch Einhalten der Normalformen bei der Modellierung können bei der späteren Benutzung Änderungs-, Einfüge- oder Löschungsanomalien verhindert werden.

Die Informationen in diesem Abschnitt sollten schon aus der Vorlesung bekannt sein.

Erste Normalform

Eine Relation ist in *Erster Normalform (1NF)*, wenn alle Attribute nur atomare Werte annehmen dürfen. Zusammengesetzte oder mengenwertige Attribute sind unter der 1NF nicht zulässig. In klassischen relationalen DBMS lassen sich Relationen, die nicht in 1NF sind, nicht speichern. Die Relation

Prüfung: {Matrikel, Name, Fachbereich, FBNr, {Prüfungsnr, Prfnote}}

ist nicht in 1NF, da es ein zusammengesetzes und mengenwertiges Attribut {Prüfungsnr, Prfnote} gibt.

Nachteile einer Relation, die nicht in 1NF ist, sind unter anderem fehlende Symmetrie, Speicherung redundanter Daten und Aktualisierungsanomalien. Bei der Überführung in 1NF entstehende neue Redundanzen werden im Laufe der weiteren Normalisierung beseitigt.

Vorgehen:

- Listen- bzw. mengenwertige Attribute werden durch ihre Elemente ersetzt. Auf Tupelebene wird das durch Ausmultiplikation der Liste mit dem restlichen Tupel erreicht.
- Zusammengesetzte Attribute werden durch die Teilattribute ersetzt.
- Alternativ können auch neue Relationen gebildet werden.

Dies liefert die Relation

Prüfung: {Matrikel, Name, Fachbereich, FBNr, Prüfungsnr, Prfnote}

in der für jede Prüfung mit Prüfungsnote ein eigenes Tupel, d.h. eine eigene Zeile in der Tabelle existiert.

Zweite Normalform

Eine Relation ist in *Zweiter Normalform (2NF)*, wenn jedes Nichtschlüsselattribut von jedem Schlüssel voll funktional abhängig ist. Relationen, die diese Normalform verletzen, enthalten Information über zwei Entitätstypen, was zu Änderungsanomalien führt. Bei der Überführung in 2NF werden die verschiedenen Entitätstypen separiert.

Die folgende Relation mit den funktionalen Abhängigkeiten *Matrikel* $\rightarrow$ *Name*, *Matrikel* $\rightarrow$ *Fachbereich*, *Matrikel* $\rightarrow$ *FBNr*, *Fachbereich* $\rightarrow$ *FBNr* und *Matrikel, Prüfungsnr* $\rightarrow$ *Prfnote* ist nicht in 2NF, da Name, Fachbereich und FBNr nur von einem Teil des Schlüssels abhängig sind.

Prüfung: {Matrikel, Name, Fachbereich, FBNr, Prüfungsnr, Prfnote}

Vorgehen:

- Ermittlung der funktionalen Abhängigkeiten zwischen Schlüssel- und Nichtschlüsselattributen wie in der Vorlesung gezeigt
- Eliminierung partieller funktionaler Abhängigkeiten, durch Überführung der abhängigen Attribute zusammen mit den Schlüsselattributen in eine eigene Relation und Entfernung der abhängigen Attribute aus der ursprünglichen Relation.

Dieses Verfahren liefert die Relationen:

Student: {Matrikel, Name, Fachbereich, FBNr}

Prüfung: {Matrikel, Prüfungsnr, Prfnote}

Dritte Normalform

Eine Relation ist in *Dritter Normalform (3NF)*, wenn jedes Nichtschlüsselattribut von keinem Schlüssel transitiv abhängig ist. In der Relation Student liegt die transitive Abhängigkeit *Matrikel* $\rightarrow$ *Fachbereich* $\rightarrow$ *FBNr* vor, sie ist daher nicht in 3NF.

Vorgehen:

- Ermittlung der funktionalen und transitiven Abhängigkeiten
- Für jede transitive funktionale Abhängigkeit $A \rightarrow B \rightarrow C$ werden alle von B funktional abhängigen Attribute entfernt und zusammen mit B in eine eigene Relation überführt.

Dieses Verfahren liefert die Relationen:

Student: {Matrikel, Name, Fachbereich}

Fachbereich: {Fachbereich, FBNr}

DB2: Der Datenbank-Manager

Eine Datenbank-Manager-Instanz verwaltet die Systemressourcen für die Datenbanken, die zu einem bestimmten Systemaccount gehören. In der letzten Woche mustet Ihr, um Eure eigene Datenbank in Eurer eigenen Instanz anlegen zu können, zunächst erstmal Eure Instanz auf dem Server `salz.is.inf.uni-due.de` starten.

Da laufende Datenbank-Manager Ressourcen verbrauchen, sollten diese nach Beendigung der Arbeit mit der Datenbank wieder gestoppt werden. Dazu gibt es die Befehle

`db2stop` bzw. `(db2) stop database manager`

die ebenfalls auf dem Rechner `salz` ausgeführt werden müssen.

In DB2 kann man auf verschiedenen Zugriffsebenen Konfigurationswerte einstellen, die das Standardverhalten einer Instanz oder auch einer Datenbank regeln.

Während eine Datenbank-Manager-Instanz läuft, kann man sich deren Konfiguration wie folgt anzeigen lassen:

`(db2) get dbm cfg` bzw. `(db2) get database manager configuration`

Einen Konfigurationswert ändert man für eine (laufende) Instanz wie folgt:

`(db2) update dbm cfg using` *config-keyword value*

Dabei steht *config-keyword* für eine Eigenschaft der Konfiguration (z.B. `AUTHENTICATION`), und *value* für den neuen Wert.

Gibt man es nicht explizit anders an, so muss in den meisten Fällen die Datenbank-Manager-Instanz anschließend neu gestartet werden, damit die Konfigurationsänderung aktiv wird.

DB2: SQL als DDL

Eine relationale Datenbank speichert Daten als eine Menge von zweidimensionalen Tabellen. Jede Spalte der Tabelle repräsentiert ein Attribut des Relationenschemas, jede Zeile entspricht einer spezifischen Instanz dieses Relationenschemas.

In relationalen Datenbanksystemen wie DB2 hat SQL (Structured Query Language) mehrere Rollen:

- Datendefinition – *Data Definition Language, DDL*
  Definition von Tabellenstrukturen: Erstellen, Anpassen und Löschen von Tabellen mitsamt ihren Attributen, Datentypen und Konsistenzbedingungen
- Datenmanipulation – *Data Manipulation Language, DML*
  Manipulation von Tabelleninhalten: Einfügen, Löschen und Ändern von Datensätzen
- Anfragesprache – *Query Language*
  Selektieren von Datensätzen nach bestimmten Auswahlkriterien

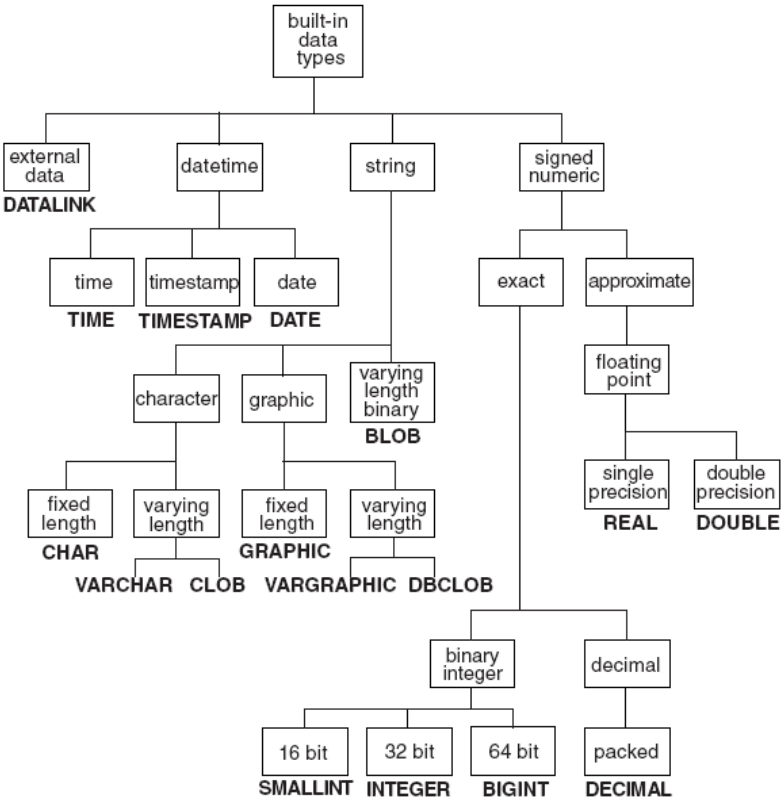
- Zugriffskontrolle – `Data Control Language, DCL`
  Diese Rolle haben wir in der vergangenen Woche schon in Form der Rechtevergabe mittels `GRANT` kennengelernt.

In dieser Praktikumswoche beschäftigen wir uns mit SQL als Datendefinitionssprache (DDL).

Datentypen

db2s1e80.pdf, S. 92ff

DB2 kennt die in der nachstehenden Grafik abgebildeten Datentypen. Weitere Datentypen können vom Benutzer definiert werden (mehr dazu zu einem späteren Zeitpunkt).



`SMALLINT`	kleine Ganzzahl
`INTEGER`	Ganzzahl
`BIGINT`	große Ganzzahl
`REAL`	Fließkommazahl mit einfacher Genauigkeit
`DOUBLE`	Fließkommazahl mit doppelter Genauigkeit
`DECIMAL(`p`,`s`)`	Dezimalbruch der Länge p mit s Nachkommastellen
`CHAR(`n`)`	String der Länge n (max. 254)
`VARCHAR(`n`)`	String variabler Länge mit maximal n Zeichen (max. 32672)
`BLOB(`sF`)`	Binary Large Object (z.B. BLOB(2 M) für einen 2MB großen BLOB
`DATE`	Datum
`TIME`	Time
`TIMESTAMP`	Zeitstempel
`DATALINK`	Verweis auf eine Datei außerhalb der Datenbank

Zu jedem Datentyp gehört ein NULL-Wert. Dieser ist von allen anderen Werten des Datentyps zu unterscheiden, da er nicht einen Wert an sich darstellt, sondern das Fehlen eines Wertes anzeigt. Der Wert 0 im Gehaltsfeld eines Angestellten könnte z.B. bedeuten, dass der Angestellte ehrenamtlich tätig ist, während der NULL-Wert bedeuten könnte, dass das Gehalt nicht bekannt ist. IBM DB2 bietet Prädikate an, die es z.B. in Anfragen erlauben, zu prüfen, ob ein NULL-Wert vorliegt oder eine andere Ausprägung des Datentyps.

Defaultwerte sind Werte, die vom DBMS eingetragen werden, wenn für das betreffende Attribut kein Wert angegeben wird. Sie bieten die Möglichkeit, Attribute automatisch mit Daten zu versehen, z.B. einer bestimmten Konstante, dem Namen des Benutzers (USER), der aktuellen Uhrzeit (CURRENT TIME), dem aktuellen Datum (CURRENT DATE) oder automatisch generierten Werten (GENERATE). Wird kein spezieller Defaultwert angegeben, so wird der NULL-Wert als Defaultwert verwendet.

Schemata (DB2-spezifisch)

Ein Schema ist eine Sammlung von benannten Objekten (Tabellen, Sichten, Trigger, Funktionen,...). Jedes Objekt in der Datenbank liegt in einem Schema. Objektnamen müssen innerhalb eines Schemas eindeutig sein, ein Schema entspricht also in etwa einem Namensraum. Ein neues Schema kann mit Hilfe des Befehls (db2) `create schema {name}` angelegt, und mit (db2) `drop schema {name} restrict` gelöscht werden.

Ein Benutzer, der über die Datenbankberechtigung IMPLICIT_SCHEMA verfügt, kann ein Schema implizit erzeugen, wenn er in einem CREATE-Statement ein noch nicht existierendes Schema verwendet. Dieses neue, von DB2 erzeugte Schema, gehört standardmäßig SYSTEM und jeder Benutzer hat das Recht, in diesem Schema Objekte anzulegen.

Erstellen von Tabellen

Jedes Team kann aufgrund der vorgegebenen Rechte in den von ihm erstellten Datenbanken neue Tabellen anlegen, eventuell mit impliziter Erstellung eines neuen Schemas (siehe oben). Zum Anlegen einer Tabelle benutzt man das CREATE TABLE-Statement. Der Benutzer, der die Tabelle erstellt, erhält automatisch das CONTROL-Recht auf dieser Tabelle.

db2s2e80.pdf, S. 332

Beim Erstellen einer Tabelle wird auch ein Primärschlüssel angegeben. Ein Primärschlüssel besteht aus den angegebenen Attributen, die keine NULL-Werte zulassen dürfen und für die Einschränkungen bezüglich der möglichen Datentypen gelten. DB2 legt automatisch einen Index auf dem Primärschlüssel an. Jede Tabelle kann nur einen Primärschlüssel haben, dieser kann jedoch aus mehreren Attributen bestehen.

```
CREATE TABLE employee (
   id           SMALLINT NOT NULL,
   name         VARCHAR(50),
   department   SMALLINT,
   job          CHAR(10),
   hiredate     DATE,
   salary       DECIMAL(7,2),
   PRIMARY KEY (ID)
)
```

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:

```
CREATE TABLE employee (
   id           SMALLINT NOT NULL PRIMARY KEY,
...
```

Als Shortcut kann man Tabellen auch mit folgendem Statement erstellen:

(db2) `CREATE TABLE name LIKE other_table`

Dies übernimmt die Attribute der Tabelle mit den exakt gleichen Namen und Definitionen aus einer anderen Tabelle oder einem View.

Gelöscht werden Tabellen mit dem DROP-Statement. Dazu benötigt man das CONTROL-Recht für die Tabelle, das DROPIN-Recht im Schema der Tabelle, das DBADM- oder das SYSADM-Recht. Wird eine Tabelle gelöscht, so werden in allen Tabellen, die diese Tabelle referenzieren, die zugehörigen Fremdschlüsselbeziehungen gelöscht. Alle Indexe, die auf der Tabelle bestehen, werden gelöscht.

db2s2e80.pdf, S. 512

Tabellenerstellung: Schlüssel und Constraints

Die im Folgenden beschrieben Konsistenzbedingungen können (und sollten) 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 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.

db2s2e81.pdf, S. 268

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)

V1: Plausibilitätstest und Testfakten

Betrachtet folgende Situationen und überprüft Euer Modell aus der letzten Woche dahingehend. Fehlen noch Entitäten? Fehlen Attribute?

- 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.inf.uni-due.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".
- Adam Boyce-Codd hat den Film „Herr der Ringe" als „Special Extended Edition" auf 6 DVDs. Er will die DVDs aber nur zusammen verleihen.
- 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.
- 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.

Bei der Analyse der Testfakten sind nun mehrere Detailfragen aufgekommen, die mit den Kunden (den zukünftigen Anwendern der „dezentralen Videothek") geklärt werden mussten.

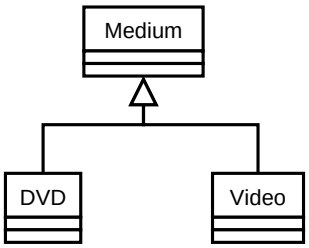
Ein Gespräch mit den Kunden ergab folgende neue bzw. detailliertere Anforderungen:

- 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 *Medium* soll ein physischer Datenträger sein. Auf einem Medium sollen *Produktion* gespeichert sein, die entweder ein Film oder eine Serienfolge ist. 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 für Medium nachvollziehbar sein, welcher Teil einer Produktion darauf gespeichert ist und wieviele Teile es insgesamt gibt.

- Medien werden in *DVD* und *VHS* unterschieden. Zu DVDs sollen Informationen wie z.B. der Regionalcode gespeichert werden - zu VHS-Cassetten der Kopierschutz (z.B. Macrovision).
- Ein *Film* ist durch Titel und Drehjahr eindeutig identifiziert, eine *Serie* durch Titel und Startjahr. Zu Serien gibt es jedoch noch *Staffeln*, die durch Serientitel, Startjahr der Serie und Staffelnummer eindeutig identifiziert sind. Staffeln bestehen aus *Serienfolgen*, die einen eigenen Titel besitzen können, es aber nicht müssen. Die Anwender möchten nachher auch nach einzelnen *Folgen* einer Serie suchen können, die entsprechend durch Serientitel, Drehjahr, Staffelnummer und Folgennummer identifiziert sind.
- Ein Medium kann durchaus mehrere Produktionen enthalten (Beispiel: „Buffy", 1997, Staffel 1, Folgen 1 bis 4 auf einer DVD).
- Da es durchaus mehrere gleiche Medien geben könnte (Beispiel: Adam und Eva besitzen beide die gleiche DVD-Ausgabe von „Herr der Ringe"), sollen Anwender *Ausleihobjekte* besitzen. Ein Ausleihobjekt ist eine konkrete, benutzerspezifische Ausführung des Mediums, zu dem noch weitere Informationen wie Zustand, Kaufpreis und Kaufdatum bekannt sein könnten.

(a) Passt Eure beiden Modelle wenn nötig an die präzisierten Anforderungen an.

Hinweis: In UML verwendet man für Generalisierung statt der Beschriftung „is a" eine besondere Pfeilform (siehe Abbildung).



V2: Relationenschema

(a) Nehmt die verbesserte E/R-Modellierung der Mini-Welt der „dezentralen Videothek" und übertragt sie zunächst möglichst modellierungsgetreu in ein relationales Schema. Verfeinenert das relationale Schema dann soweit möglich und sinnvoll wie beschrieben.

Eventuell werden hier bereits erste Probleme der ursprünglichen Modellierung sichtbar. Haltet diese Beobachtungen fest.

(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 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.

V3: 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?

V4: Dritte Normalform

Bringt Euer relationales Schema in die 3. Normalform.

V5: Constraints in SQL

Formuliert jeweils eine Fremdschlüsselbedingung und einen Wertebereichsprüfung aus Eurem Entwurf als benannten Constraint 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: Datenbank-Manager starten und konfigurieren

(a) Verbindet Euch mit Eurer Datenbank-Manager-Instanz auf `salz`.

Befehl:

(b) Lasst Euch die Konfiguration Eurer Datenbank-Manager-Instanz anzeigen:

Befehl:

Auf welchem Port (TCP/IP-Service) läuft die Instanz?

Wer hat auf dieser Instanz das Recht `SYSADM`?

Nach welcher Methode authentifiziert der Datenbank-Manager Anfragen von Clients?

P3: Tabellen anlegen

(a) Legt die Tabellen Eures Schemas in Eurer Datenbank an. Berücksichtigt dabei auch die nötigen Schlüssel- und Konsistenzbedingungen.

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.