Praktikum Datenbanken / DB2
Woche 3: Tabellenentwurf und -erstellung

Betreuer: Gudrun Fischer, Tobias Tuttas, Camille Pieume
Raum: LF 230
Bearbeitung: 22.-24 Mai 2006

Datum	
Team (Account)	
Vorbereitung	
Präsenz	

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

- die Tabellenstruktur für die Semesteraufgabe entwerfen,
- die eigene Datenbank-Manager-Instanz konfigurieren und
- eine erste Tabelle in der neuen 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.

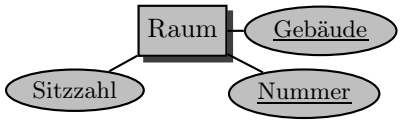
Relationenschemata werden wie in **Kemper/Eick** beschrieben spezifiziert:

SchemaName : { [$\underline{\text{Attribut}_1\text{: Typ}_1}$, $\text{Attribut}_2\text{: Typ}_2$, ...] }

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]}

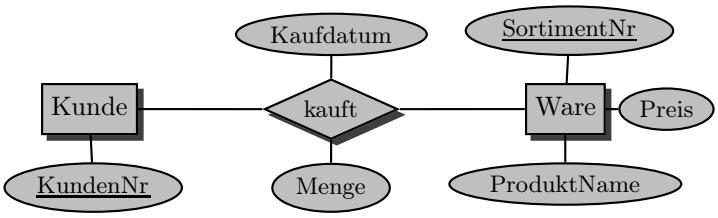
Zusammengesetzte Attribute werden entweder aufgelöst, d.h. die Attribute des zusammengesetzten Attributes werden zu Attributen der Umsetzung des Entitätstyps, oder sie werden als eigene Relation aufgefasst. Diese neue Relation enthält alle Attribute des zusammengesetzten Attributes und alle Primärschlüsselattribute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

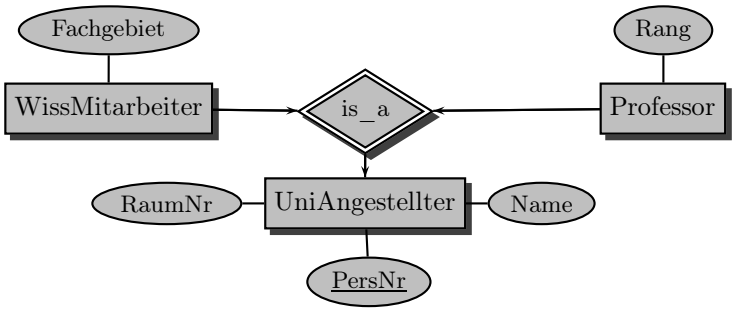
Ein schwacher Entitätstyp wird als eine eigene Relation abgebildet, die zusätzlich zu den eigenen Attributen auch den Primärschlüssel des Vater-Entitätstype als Fremdschlüssel enthält. Dieser bildet zusammen mit den ausgezeichneten Teilschlüsseln des schwachen Entitätstyp den Primärschlüssel der Relation.

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. Oft kann man nun noch Relationen zusammenführen. Allgemein gilt: Relationen mit dem gleichen Schlüssel kann man zusammenfassen (aber auch nur diese).

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 musstet Ihr, um Eure eigene Datenbank in Eurer eigenen Instanz anlegen zu können, zunächst erstmal Eure Instanz auf dem Server `salz.is.informatik.uni-duisburg.de` starten. Dazu hattet Ihr Euch per `ssh` auf dem Server angemeldet und dann den Befehl

`db2start` bzw. `(db2) start database manager`

verwendet.

In Zukunft bitten wir Euch, Eure Instanz auch wieder zu stoppen, sobald Ihr mit den Präsenzaufgaben fertig seid. 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

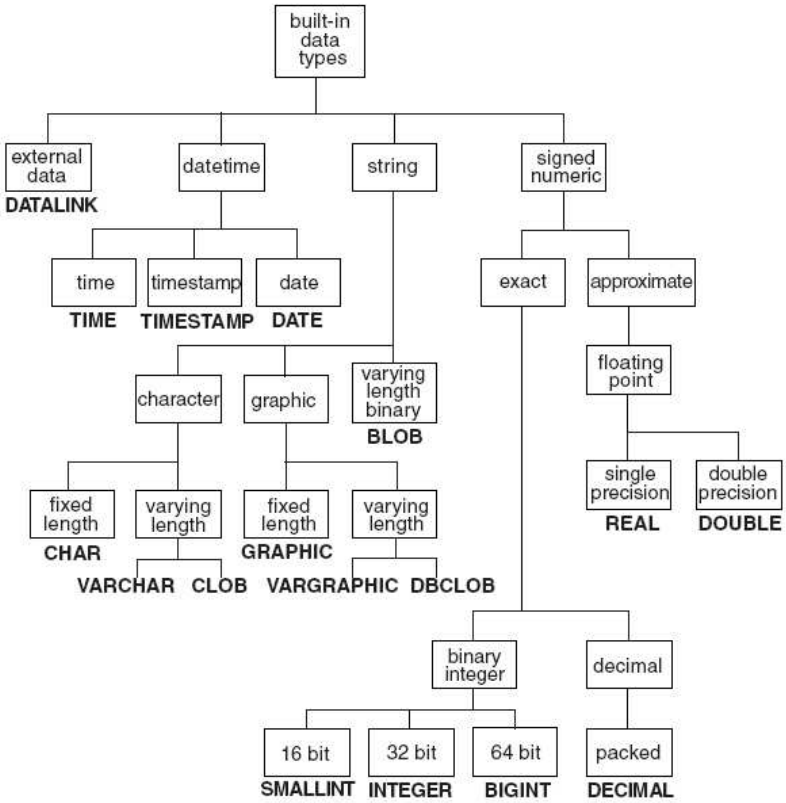
- Datenmanipulation – *Data Manipulation Language, DML*
  Manipulation von Tabelleninhalten: Einfügen, Löschen und Ändern von Datensätzen
- Anfragesprache – *Query Language*
  Selektieren von Datensätzen nach bestimmten Auswahlkriterien
- Zugriffskontrolle – `Data Control Language, DCL`
  Diese Rolle 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 ei-

nes 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,
...
```

Mit Schlüsseln und Konsistenzbedingungen werden wir uns in der nächsten Woche weiter befassen.

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

Aufgaben

Vorbereitung (Hausaufgaben zur Semesteraufgabe)

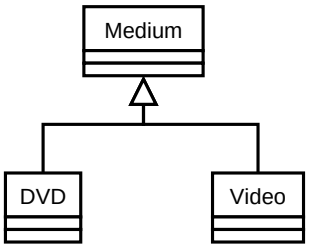
V1: Plausibilitätstest

Stellt Euch folgende Situationen vor und überprüft Euer Modell von letzter 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.informatik.uni-duisburg.de". Sein Passwort wird vom System generiert und lautet „rumpelstilzchen".
- 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" die erste Staffel auf 3 DVDs. Adam Boyce-Codd leiht sich von Eva am 01.06.2006 die erste dieser Buffy-DVDs 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.

Passt Eure beiden Modelle entsprechend an.

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



V2: Relationenschema

Nehmt die E/R-Modellierung der Mini-Welt der „dezentralen Videothek" aus der vergangenen Woche 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.

V3: Dritte Normalform

Bringt Euer relationales Schema in die 3. Normalform.

Präsenz

P1: Datenbank-Manager starten und konfigurieren

(a) Loggt Euch per `ssh` auf dem Rechner `salz` ein und startet Eure Datenbank-Manager-Instanz.

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?

(c) Ändert die Authentifizierungsmethode auf den Wert `CLIENT`.

Befehl:

(d) Stoppt Eure Datenbank-Manager-Instanz und startet sie neu, damit die geänderte Konfiguration aktiv wird.

Befehle:

(e) Verlasst `salz` mit dem Befehl `exit`.

P2: Troubleshooting

In der vergangenen Woche haben wir u.a. die Datenbanksystem-Objekte Instanz und Datenbank kennengelernt. In der Präsenzphase hattet Ihr in Eurer eigenen *Instanz* (`dbp06XX`) auf dem *Rechner* `salz` eine eigene Datenbank angelegt.

Dabei traten folgende Probleme auf:

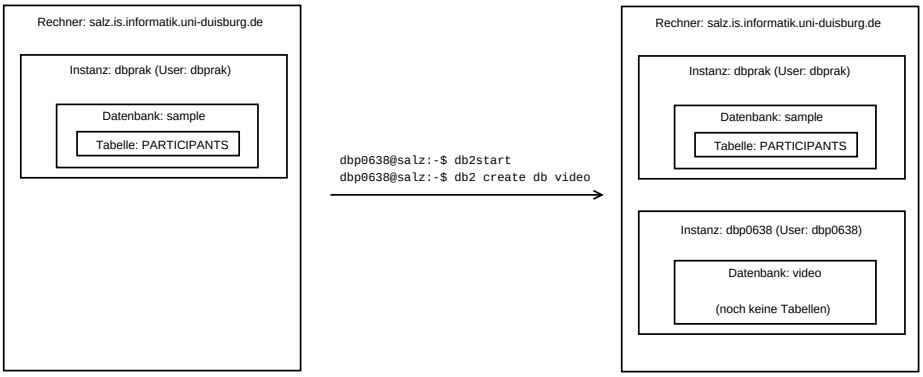
- Der ursprünglich vorgesehene Datenbankname `videothek` war zu lang. Daher habt Ihr die neue Datenbank (wahrscheinlich) `video` genannt.
- Bei dem Versuch, die Datenbank anzulegen, kam zunächst eine Fehlermeldung, dass kein *Datenbank-Manager* läuft. Ihr habt daher auf dem Rechner `salz` mit dem Kommando `db2start` Euren eigenen Datenbank-Manager gestartet.
- Falls Ihr nachher versucht habt, Euch zu Eurer neuen Datenbank zu verbinden, traten höchstwahrscheinlich wieder Fehler auf, auch wenn ihr sie wie gewohnt katalogisiert hattet.

Was war passiert? Betrachtet die folgende Abbildung mit einem Beispiel für Account `dbp0638`:

Auf dem *Rechner* `salz` läuft die ganze Zeit die *Instanz* `dbprak`. Innerhalb dieser Instanz ist die *Datenbank* `sample` definiert, auf die ihr in der ersten Woche zugegriffen habt.

Um Eure eigene Datenbank in Eurer eigenen Instanz zu erstellen, musstet Ihr zuerst *Eure* Instanz starten. Das geschah mit dem Befehl `db2start`, und zwar auf dem Rechner `salz`. Dann konntet Ihr die Datenbank `video` anlegen.

Um auf eine Datenbank *zuzugreifen*, müssen jedoch sowohl deren *Knoten* (d.h. die Kombination aus *Instanz* und *Rechner*), als auch die Datenbank selbst ka-



talogisiert sein. Wenn man nun die Datenbank `video` auf dem Knoten `salz` katalogisiert, so wie er in der ersten Woche definiert war, dann sucht DB2 die Datenbank `video` in der Instanz `dbprak` und findet sie nicht.

Behebt dieses Problem, indem Ihr folgende Schritte ausführt:

(a) Räumt Euren Datenbank-Katalog auf, so dass nur noch die Beispieldatenbank aus Woche 1 und die lokale Datenbank `VIDEO` aufgeführt sind.

Beispielsweise sollte der Datenbank-Katalog für den Account `dbp0638` so aussehen:

```
 System Database Directory

 Number of entries in the directory = 2

Database 1 entry:

 Database alias                       = MYSAMPLE
 Database name                        = SAMPLE
 Node name                            = SALZ
 Database release level               = a.00
 Comment                              =
 Directory entry type                 = Remote
 Catalog database partition number    = -1

Database 2 entry:

 Database alias                       = VIDEO
 Database name                        = VIDEO
 Local database directory             = /export/a/home/dbp0638
 Database release level               = a.00
 Comment                              =
 Directory entry type                 = Indirect
 Catalog database partition number    = 0
```

(b) Katalogisiert den *zusätzlichen Knoten* `salzXX` (wobei `XX` für Eure Accountnummer steht). Verwendet dabei die *Instanz* `dbp06XX` und den *Port* `500XX`.

Zur Erinnerung – so hattet Ihr die *Instanz* `dbprak` auf dem *Rechner* `salz` auf *Port* 50050 als *Knoten* `salz` katalogisiert:

```
db2 catalog tcpip node salz
remote salz.is.informatik.uni-duisburg.de server 50050
remote_instance dbprak system salz ostype linux
```

Nach diesem Schritt sollte der Knoten-Katalog für den Account `dbp0638` zum Beispiel so aussehen:

```
 Node Directory

 Number of entries in the directory = 2

Node 1 entry:

 Node name                      = SALZ
 Comment                        =
 Directory entry type           = LOCAL
 Protocol                       = TCPIP
 Hostname                       = salz.is.informatik.uni-duisburg.de
 Service name                   = 50050

Node 2 entry:

 Node name                      = SALZ38
 Comment                        =
 Directory entry type           = LOCAL
 Protocol                       = TCPIP
 Hostname                       = salz.is.informatik.uni-duisburg.de
 Service name                   = 50038
```

(c) Katalogisiert nun die *Datenbank* `video` unter dem *Alias* `videoXX` auf dem *Knoten* `salzXX`.

Nach diesem Schritt sollte der Datenbank-Katalog für den Account `dbp0638` zum Beispiel so aussehen:

```
 System Database Directory

 Number of entries in the directory = 3

Database 1 entry:

 Database alias                       = MYSAMPLE
 Database name                        = SAMPLE
 Node name                            = SALZ
 Database release level               = a.00
 Comment                              =
 Directory entry type                 = Remote
 Catalog database partition number    = -1

Database 2 entry:
```

```
 Database alias                       = VIDEO38
 Database name                        = VIDEO
 Node name                            = SALZ38
 Database release level               = a.00
 Comment                              =
 Directory entry type                 = Remote
 Catalog database partition number    = -1

Database 3 entry:

 Database alias                       = VIDEO
 Database name                        = VIDEO
 Local database directory             = /export/a/home/dbp0638
 Database release level               = a.00
 Comment                              =
 Directory entry type                 = Indirect
 Catalog database partition number    = 0
```

P3: Erste Tabelle anlegen und anschauen

Verbindet Euch nun mit Eurer eigenen Datenbank und legt als erste Tabelle die Tabelle für die Anwenderdaten (Name, Vorname, ...) an. Lasst Euch danach die Tabellenstruktur anzeigen und beendet dann die Verbindung.

Befehle:

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.