**Praktikum Datenbanken / DB2**
**Woche 2: Modellierung und ausführlichere DB2-Grundlagen**

Betreuer: Gudrun Fischer, Tobias Tuttas, Camille Pieume
Raum: LF 230
Bearbeitung: 15.-18 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`

# Vorgehen und Wochenziele

Der Entwurf einer Datenbank beschreibt den Prozess der Umsetzung einer Mini-Welt in ein Datenbankschema, das in der Lage ist, die gewünschten Daten dieser Welt mit ihren Eigenschaften und Beziehungen darzustellen.

Der Entwurf, an den sich dann die Implementierung anschließt, besteht im Wesentlichen aus diesen Schritten:

- Modellierung
- Umsetzung in ein Relationenschema
- Normalisierung

In dieser Praktikumswoche wollen wir zwei Dinge erreichen:

- Die Semesteraufgabe wird eingeführt, und wir wollen die entsprechende Mini-Welt in UML und E/R-Diagrammen modellieren. Aufgrund dieser Modelle werden wir dann in der nächsten Woche die Tabellenstruktur für die Semesteraufgabe entwerfen.
- Jede Gruppe soll in ihrer eigenen DB2-Instanz eine Datenbank für die Semesteraufgabe einrichten, damit wir dort in der kommenden Woche (nach den weiteren Entwurfsschritten) die ersten Tabellen einrichten können.

# Entwurfsschritt 1: Modellierung

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

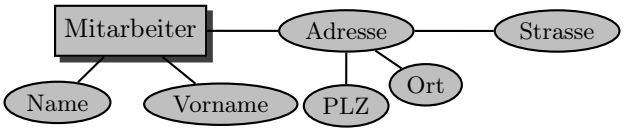
## Modellierung mit E/R-Diagrammen

Die Modellierung dient der systematischen Darstellung einer Abstraktion der abzubildenden Miniwelt. Er dient auch zur Kommunikation mit Nichtfachleuten, die in der Regel nicht das Denken in Relationen oder Attributen gewohnt sind. Als eine Möglichkeit zur Modellierung einer Miniwelt ist das Entity-Relationship-Modell weit verbreitet. Zur Darstellung dieser Modelle werden E-R-Diagramme benutzt.

### Entitäten und Attribute

Ein Objekt der realen Welt wird als **Entität** modelliert. Eine Entität ist normalerweise eine Person, ein Prozess oder ein Gegenstand der realen Welt, z.B. ein Mitarbeiter, eine Lieferung, ein Inventargegenstand oder ein Schriftstück. Gleichartige Entitäten bilden einen **Entitätstyp**, z.B. alle Wissenschaftlichen Angestellten, alle Autoteile oder alle Arbeitsverträge. In einem E-R-Diagramm wird ein Entitätstyp durch ein Rechteck dargestellt, das den Namen des Entitätstypen umschließt.

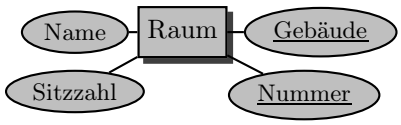

Eine Entität (bzw. ein Entitätstyp) wird durch eine Menge von **Attributen** beschrieben. In einem E-R-Diagramm umschließt eine Ellipse den Namen des Attributes und eine Linie verbindet es mit dem zugehörigen Entitätstyp. Ein Attribut von Mitarbeiter ist z.B. der Name. Man spricht von einem **zusammengesetzten Attribut**, wenn ihm wiederum verschiedene Attribute zugeordnet sind. In einem E-R-Diagramm werden zusammengesetzte Attribute genau wie atomare Attribute dargestellt.



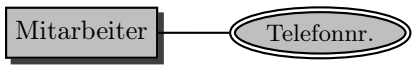

Attributkombinationen, die zur eindeutigen Identifikation einer Entität dienen, werden **Schlüssel** genannt. Beim Entitätstyp Mitarbeiter könnte z.B. (Name, Vorname, Adresse) oder (Mitarbeiternummer) als Schlüssel dienen. Ein Schlüssel identifiziert über die Attributwerte eine Entität eindeutig, insbesondere können keine zwei Entitäten in allen ihren Schlüsselattributen übereinstimmen.

Unter allen Schlüsseln wird ein Schlüssel ausgezeichnet, dieser heißt **Primärschlüssel**.

In einem E-R-Diagramm werden die Namen aller Attribute, die zum Primärschlüssel gehören, unterstrichen dargestellt.




**Mehrwertige Attribute** sind Attribute, die als Wert eine Liste oder eine Menge annehmen können. In einem E-R-Diagramm werden mehrwertige Attribute durch den Attributnamen in einem doppelten Oval dargestellt.




**Beziehungen (Relationships)**

Die Objekte der realen Welt stehen zueinander in Beziehung. Durch **Beziehungen** werden im E-R-Modell Verknüpfungen, Zusammenhänge und Interaktionen zwischen Entitäten, bzw. den Entitätstypen, modelliert. Beispiele sind etwa ist-Teil-von, arbeitet-für, usw.

Beziehungen derselben Art werden zu sogenannten **Beziehungstypen** zusammengefaßt. In einem E-R-Diagramm werden Beziehungstypen durch eine Raute dargestellt, die den Namen des Beziehunstyps enthält und mit allen Entitätstypen verbunden ist, die an dieser Beziehung teilnehmen. Ein Entitätstyp kann mehrfach an einer Beziehung teilnehmen. Es ist auch möglich, dass ein Entitätstyp in Beziehung mit sich selbst steht (das nennt man dann *rekursive* Beziehung).

Verbindet ein Beziehungstyp nur zwei Entitätstypen jeweils einmal, so spricht man von einem binären Beziehungstyp, ebenso kennt man ternäre Beziehungen, usw.


Beziehungen können wie Entitätstypen Attribute besitzen. Diese werden im E-R-Diagramm genauso dargestellt wie bei Entitätstypen. Da Beziehungstypen nur Zusammenhänge und Beziehungen modellieren, können sie keine identifizierenden Schlüssel besitzen. Im E-R-Diagramm werde keine Schlüssel für Beziehungstypen ausgezeichnet.

Bei Beziehungstypen ist es sinnvoll, sich schon in der Modellierungsphase klarzumachen, wie oft die verschiedenen Entitäten an einer Beziehung teilnehmen können. Diese Überlegungen führen zum Konzept der **Funktionalitäten**:

Nimmt jede Entität eines Typs genau n-mal an einer Beziehung teil, so wird im E-R-Diagramm die Verbindungslinie mit dieser Zahl markiert. Nimmt jede Entität mindestens p-mal und höchstens k-mal an einer Beziehung teil, so wird in der (*min*,*max*)-Notation im E-R-Diagramm die Verbindungslinie mit (p,k) markiert. Das Zeichen *N* als Maximum zeigt normalerweise an, dass eine Entität dieses Typs beliebig oft an dieser Beziehung teilnehmen kann.


Im E-R-Modell ist es nicht möglich, dass Beziehungen selbst wieder an Beziehungen teilnehmen. Das kann aber manchmal nötig und sinnvoll sein. Hierzu fasst man einen Beziehungstyp mit den an ihm beteiligten Entitätstypen zu einem aggregierten Entitätstyp zusammen. Dieser kann wiederum an Beziehungen teilnehmen. Im E-R-Diagramm wird ein aggregierter Entitätstyp dadurch dargestellt, dass die ihn definierende Entitätstypen- und Beziehungstypensymbole in ein Rechteck eingeschlossen werden.

### Generalisierung

Um die Entitätstypen besser zu strukturieren, kann man im Entwurf **Generalisierungen** einführen. Dieses objektorientierte Konzept abstrahiert Entitätstypen und bildet Obertypen. Die Entitätstypen, die zu einem Obertyp generalisiert wurden, heißen Kategorien oder Untertypen des Obertyps.

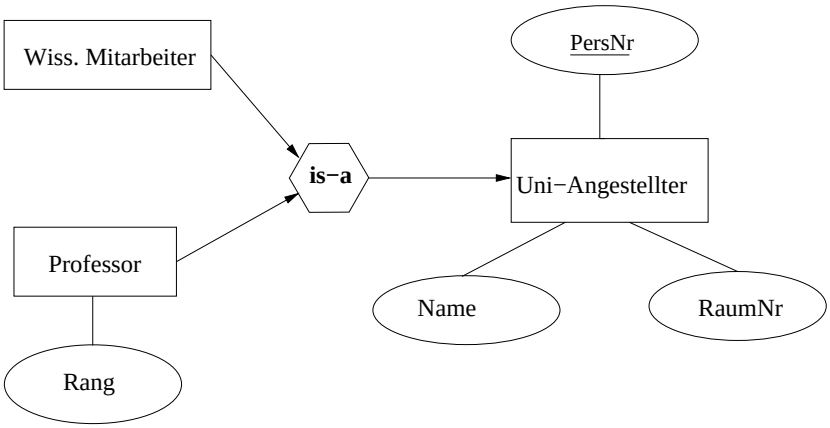
Ein Obertyp faßt allen Kategorien gemeinsame Eigenschaften zusammen. Diese werden vom Untertyp geerbt; zusätzlich kann ein Untertyp Eigenschaften haben, die nur für diesen charakterisierend sind. Die Entitätsmenge des Untertyps ist eine Teilmenge der Entitätsmenge des Obertyps, wobei üblicherweise zwei Fälle besonders interessant sind:

- *disjunkte* Spezialisierung: die Schnittmenge von je zwei Untertypen ist leer
- *vollständige* Spezialisierung: Obertyp ergibt sich als Vereinigung der Untertypen

In einem E-R-Diagramm modelliert man die Generalisierungsbeziehung als Beziehung **is-a** durch ein spezielles Symbol: ein Sechseck statt einer Raute.

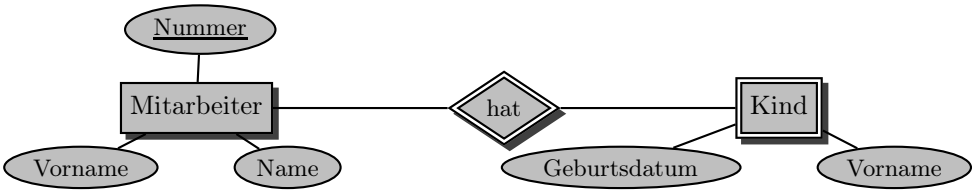
### Schwache Entitätstypen

Wenn der Fall auftritt, dass ein Entitätstyp keinen eigenen Schlüsselkandidaten besitzt, spricht man von **schwachen Entitätstypen**. Sie nehmen an einer




Beziehung mit einem anderen Entitätstyp teil, dem Eltern-Entitätstyp. Jedes Objekt des schwachen Entitätstyp nimmt mindestens an einer Beziehung zum Eltern-Entitätstyp teil. Das Objekt wird über diese Beziehung identifiziert. Der Primärschlüssel des Eltern-Entitätstyps wird auch für den schwachen Entitätstyp verwendet (eventuell durch einen Teilschlüssel erweitert).

In der Regel sind die Objekte des schwachen Entitätstyps von den Objekten des Eltern-Entitätstyp abhängig, d.h. ohne diese sind sie nicht von Interesse bzw. besitzen keine eigene Existenz. Im E-R-Diagramm werden schwache Entitätstypen von normalen Entitätstypen dadurch unterschieden, dass ihr Name durch ein doppeltes Rechteck eingeschlossen ist. Die Teilschlüssel werden durch eine gestrichelte Unterstreichung ausgezeichnet.



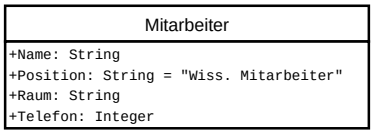

## UML

Die *Unified Modelling Language* UML ist eine Standardmodellierungssprache für den objekt–orientierten Entwurf von Software, und findet auch beim Datenbankentwurf ihre Verwendung. Zentrales Objekt von UML ist die Klasse, mit der gleichartige Objekte hinsichtlich ihrer Struktur und ihres Verhaltens modelliert werden. Klassen entsprechen in etwa den Entitätstypen des E-R-Modells. Assoziationen zwischen Klassen entsprechen Beziehungstypen.

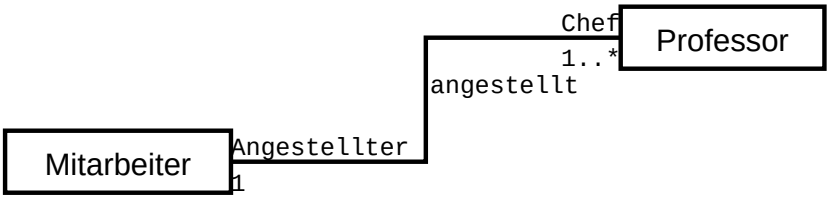
Ebenso können Generalisierung und Aggregation mit der UML modelliert werden. Die Angabe von Multiplizitäten entspricht den Funktionalitäten im ER-Modell. Zu Details der UML wird auf das Material der Vorlesung verwiesen, das aus Platzgründen hier nicht wiederholt wird.

**Notation**

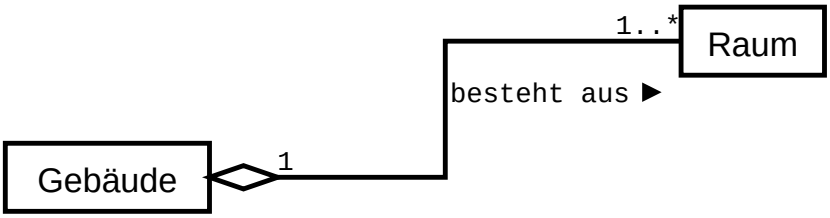
Man stellt in UML-Klassendiagrammen eine Klasse durch ein Rechteck dar, das am oberen Rand den Namen der Klasse und darunter die Liste der Attribute und Operationen der Klasse enthält. Klassenname, Attribute und Operationen werden jeweils durch eine horizontale Linie voneinander getrennt. Dabei stehen alle Klassennamen im Singular und beginnen mit einem Großbuchstaben. Attribute können näher beschrieben sein, etwa durch Attributtyp oder Initialwerte (in der Form `Name: Typ = Initialwert`), besitzen aber mindestens einen Namen.




Assoziationsbeziehungen werden durch eine Verbindungslinie zwischen zwei Klassensymbolen dargestellt. Diese Linie wird mit dem Namen der Beziehung versehen. An den Enden der Verbindungslinie kann die Multiplizität der Beziehung angegeben werden. Sie wird als einzelne Zahl oder als Wertebereich auf jeder Seite der Assoziation notiert (z.B. `1` oder `0..*`). Zusätzlich kann an den Enden der Verbindungslinie auch eine Rolle angegeben werden.




Eine Aggregation wird wie eine Assoziation als Linie zwischen zwei Klassen dargestellt, und zusätzlich auf einer Seite der Linie mit einer Raute versehen. Die Raute steht auf der Seite des Aggregats (des Ganzen) und symbolisiert das Behälterobjekt, in dem die Teile gesammelt sind. Auch hier können Multiplizitäten angegeben werden.



# Ausführlichere Grundlagen zu DB2

Im Praktikum wird als Datenbank-Managementsystem (DBMS) das kommerzielle System *IBM DB2 Universal Database* in der Version 8.1 verwendet. Dieses läuft auf dem Rechner `salz`, auf den übrigen Maschinen des Rechnerpools LF230 laufen *Administration Clients* für IBM DB2.

Das Kommandozeilenwerkzeug (CLP) wurde bereits in der vergangenen Woche iengeführt. Zur Erinnerung – es wird zwischen drei Benutzungsarten unterschieden:

- Aufruf aus der Kommandozeile mit vorangestelltem `db2`
- Aufruf mit einer Batchdatei als `db2 -tvf filename.sql`
- interaktiver Aufruf, eingeleitet mit `db2 -t`

Zeilen, die mit −− beginnen, werden als Kommentar betrachtet und ignoriert. Mit `? {Befehlsname}` kann man Hilfen zu Befehlsnamen, Warnungs- oder Fehlercodes bekommen.

Die Dokumentation zu DB2 steht online und in Form der englischsprachigen Benutzerhandbücher als PDFs im Verzeichnis `/home/dbprak/doc` zur Verfügung. Diese Handbücher sind sehr umfangreich und sollten nur zum Betrachten am Bildschirm herangezogen und nicht ausgedruckt werden. Die Referenzen in diesem Arbeitsblatt beziehen sich auf diese Handbücher. Die gleichen Informationen sind jedoch auch in der Online-Dokumentation zu finden. Besonders wichtig sind hier die „SQL Reference" und die „Command Reference".

*db2s1e80.pdf*
*db2s2e80.pdf*
*db2n0e80.pdf*

## Struktur der DB2-Installation

Innerhalb einer DB2-Installation existieren die folgenden wichtigen Datenbanksystem-Objekte:

- Instanzen
- Datenbanken
- Schemata
- Tabellen
- Views

Tabellen und Views werden in den folgenden Sitzungen behandelt.

### Instanz

Eine Instanz (auch: Datenbankmanager) ist ein Objekt, das Daten verwaltet. Es kontrolliert, was mit den Daten gemacht werden kann, und verwaltet die ihm zugewiesenen Systemressourcen. Jede Instanz ist eine vollständige Arbeitsumgebung, die ihre eigenen, für andere Instanzen unzugänglichen Datenbanken

enthält. Alle Datenbankpartitionen einer Instanz teilen sich das gleiche Systemverzeichnis. Eine Instanz hat eine eigene Rechteverwaltung, die unabhängig von der anderer Instanzen ist.

Jede Kleingruppe im Praktikum besitzt eine eigene Instanz auf `salz` unter dem Namen ihres Accounts (z.B. `dbp0601`). Auf dieser besitzt die Gruppe das Administratorrecht, und kann Datenbanken anlegen, ändern und löschen.

Die Konfiguration der eigenen Instanz kann man mit dem Befehl `db2 get dbm cfg` oder `db2 get database manager configuration` abfragen. Im interaktiven oder Batchmodus muß man natürlich das einleitende `db2` weglassen. Im Folgenden werden DB2-Befehle durch (db2) gekennzeichnet.

Um sich mit einer Instanz auf einem entfernten Rechner zu verbinden, muß man diese zunächst als einen lokalen Knoten katalogisieren. Wer dies in der Einführungsveranstaltung bereits getan hat, kann diesen Schritt überspringen. Der Befehl zum Katalogisieren einer externen Instanz über TCP/IP lautet für das Praktikum:

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

### Datenbank

Eine relationale Datenbank ist eine Sammlung von Tabellen. Eine Tabelle besteht aus einer festen Zahl von Spalten, die den Attribute des Relationenschemas entsprechen, sowie einer beliebigen Zahl an Reihen. Zu jeder Datenbank gehören Systemtabellen, welche die logische und physikalische Struktur der Daten beschreiben, eine Konfigurationsdatei mit den Parametern der Datenbank und ein Transaktionslog.

Eine neue Datenbank wird mit dem Befehl (db2) `create database {name}` erstellt. Dies erzeugt eine Datenbank mit der Standardkonfiguration. Gelöscht werden kann die Datenbank wieder mit dem Befehl (db2) `drop database {name}`.

Der Befehl (db2) `connect to {name}` stellt eine Verbindung zu einer Datenbank her, die mit (db2) `terminate` wieder beendet wird.

Um sich von einem entfernten Rechner mit einer Datenbank zu verbinden muß diese zunächst an einem lokalen Knoten katalogisiert werden. Angenommen es existiert bereits ein lokaler Knoten *salz*, dann lautet der Befehl zum Katalogisieren einer entfernten Datenbank *sample* auf dem mit *salz* verbundenen Server wie folgt:

(db2) `catalog database sample as samplexx at node salz`

Dabei ist der *samplexx* der Aliasname, unter dem man sich mit der Datenbank lokal verbinden kann.

### Schemata

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 Objekt in der Datenbank wird durch einen *qualifizierten* Namen identifiziert. Dieser besteht aus dem Namen des Schemas, in dem das Objekt liegt, gefolgt von einem Punkt und dem Objektnamen: DBPRAK.STAFF würde beispielsweise das Objekt STAFF im Schema DBPRAK bezeichnen.

Wird nur ein Objektname angegeben, so qualifiziert DB2 diesen standardmässig durch den Benutzernamen. Man kann diesen Standardwert auch ändern:

(db2) `SET SCHEMA dbprak`

### Tabellen

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 Schemas. Daten in diesen Tabellen werden mit Hilfe der Structured Query Language (SQL) manipuliert, einer standardisierten Sprache zur Definition und Manipulation von Daten in relationalen Datenbanken.

Tabellenerstellung, Constraints und Datentypen werden in der nächsten Sitzung behandelt. Um sich in einer existierenden Datenbank die vorhandenen Tabellen anzeigen zu lassen, benutzt man den Befehl

(db2) `list tables for schema {name}`

Mit Hilfe von (db2) `describe table {schemaname}.{tablename}` kann man Einzelheiten über den Aufbau einer existierenden Tabelle herausfinden.

## Benutzerverwaltung

Das Authentifizierungskonzept von DB2 sieht keine gesonderten Benutzernamen vor, sondern baut auf das Benutzerkonzept des Betriebssystems auf. Jede Kleingruppe meldet sich unter dem Namen ihres Accounts bei der Datenbank an, zusätzlich gehört jede Kleingruppe zur Benutzergruppe `students`.

In einer DB2-Instanz gibt es drei Stufen der Zugriffsberechtigung. Als Instanzbesitzer hat jede Gruppe in ihrer Instanz das SYSADM-Recht.

**SYSADM:** Zugriff auf alle Daten und Ressourcen der Instanz

**SYSCTRL:** Verwalten der Instanz, aber kein direkter Zugriff auf Daten

**SYSMAINT:** niedrigste Berechtigungsstufe; kann z.B. Backups erstellen

Um Berechtigungen in der Datenbank, mit der man gerade verbunden ist, zu setzen oder zu verändern, bedient man sich des GRANT-Statements. Umgekehrt wird das REVOKE-Statement benutzt, um Rechte auf einer Datenbank zu entziehen. `GET AUTHORIZATIONS` zeigt die aktuellen Berechtigungen an. Beispiele:

*db2s2e80.pdf, S. 569/642*

```
(db2) GRANT dbadm ON DATABASE TO USER dbpw0301
(db2) GRANT connect, createtab ON DATABASE TO GROUP students
(db2) GRANT connect ON DATABASE TO public
(db2) REVOKE connect ON DATABASE TO public
```

Ein Benutzer oder eine Gruppe mit dem DBADM-Recht besitzt automatische auch alle anderen Rechte auf dieser Datenbank, insbesondere auch das Recht IMPLICIT_SCHEMA. Das Recht DBADM kann nicht an PUBLIC vergeben werden. Nur ein Benutzer mit SYSADM-Recht kann das DBADM-Recht an andere Benutzer oder Gruppen verleihen oder entziehen. Wird einem Benutzer ein Recht entzogen, bedeutet das nicht notwendig, dass dieser das Recht nun nicht mehr besitzt, da Rechte auch durch Zugehörigkeit zu Gruppen bestehen können.

Wer das Recht DBADM oder SYSADM besitzt kann Berechtigungen für ein Schema ändern. Dabei kann einem Benutzer auch das Recht gegeben werden, die erhaltenen Rechte an Dritte weiterzugeben. Der Befehl zum Vergeben bzw. Entziehen von Rechten auf Schemata ist GRANT/REVOKE ... ON SCHEMA.

*db2s2e80.pdf, S. 583/657*

## Übersicht über die neuen Befehle

| | |
|---|---|
| ? | Hilfe |
| ? *command* | Hilfe zu einem Befehl |
| get dbm configuration | Konfiguration des Datenbankmanagers anzeigen |
| catalog tcpip node *name* | entfernten Datenbankserver als lokalen Knoten *name* katalogisieren |
| list node directory | Verzeichnis aller katalogisierten Knoten anzeigen |
| uncatalog node *name* | katalogisierten Knoten *name* aus dem Verzeichnis entfernen |
| catalog database *name* as *alias* | Datenbank *name* unter lokalem Alias *alias* katalogisieren |
| list database directory | Verzeichnis aller katalogisierter Datenbanken anzeigen |
| uncatalog database *alias* | als *alias* katalogisierte Datenbank aus dem Verzeichnis entfernen |
| create databse *name* | neue Datenbank *name* erstellen |
| drop database *name* | Datenbank *name* löschen |
| connect to *name* | Verbinden mit Datenbank *name* |
| terminate | Verbindung mit Datenbank beenden |
| set schema *name* | das aktuelle Schema auf *name* setzen |
| get authorizations | die Privilegien auf der aktuellen Datenbank anzeigen |
| grant *privilege* on database to public/group *name*/user *name* | allen, einer Gruppe oder einem Benutzer auf der aktuellen Datenbank ein Privileg verleihen |
| revoke *privilege* on database to public/group *name*/user *name* | ein Privileg auf der aktuellen Datenbank wieder zurücknehmen |
| list tables for schema *name* | alle Tabellen für das Schema *name* anzeigen |
| describe table *name* | den Aufbau und die Attribute der Tabelle Name anzeigen |

# Aufgaben

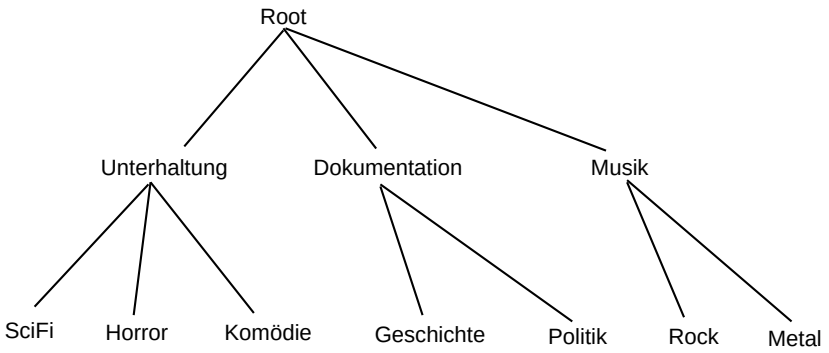
## Vorbereitung (Hausaufgaben zur Semesteraufgabe)

Im Laufe dieses Praktikums soll eine sogenannte „dezentrale Videothek" entwickelt werden. Das ist ein Datenbank-basiertes Programm, mit welchem eine Gruppe von Anwendern ihre privaten DVDs und Videos katalogisieren und untereinander verleihen kann (selbstverständlich kostenlos und im rein privaten Rahmen).

Dazu müssen Anwender und deren Kontaktdaten verwaltet werden, außerdem natürlich Medien und deren Besitz, Verleih und Rückgabe. Ein Medium sei in diesem Kontext ein physischer Datenträger (Videokassette oder DVD), auf dem ein oder mehrere Filme und eventuell Bonusmaterial enthalten sind. Ein Film sei dabei eindeutig durch seinen Titel und sein Drehjahr bestimmt.

Filme, egal ob auf DVD oder Video, sollen außerdem einem hierarchischen Kategoriensystem zugeordnet werden können. Später soll die Datenbank eventuell um weitere Informationen zu Filmen, wie z.B. Schauspieler und Regisseure, erweitert werden können.

Beispiel für eine solche Kategorienhierarchie:




Der Film „Event Horizon" würde in dieser Hierarchie den Kategorien [`Unterhaltung` → `SciFi`] und [`Unterhaltung` → `Horror`] zugeordnet werden.

### V1: Modellierung als E/R-Diagramm

(a) Modelliert eine Mini-Welt für die oben beschriebene Semesteraufgabe. Verwendet dabei die Entitäten `Anwender`, `Medium`, `DVD`, `Video`, `Film` und `Kategorie`.

(b) Welche Beziehungen gibt es zwischen den genannten Entitäten? Gebt auch die Rollen und Funktionalitäten an.

(c) Zwischen welchen Entitäten gibt es eine *Generalisierungsbeziehung*?

(d) Welche Attribute sind für die jeweiligen Entitäten und Beziehungen unbedingt notwendig? Welche wären vielleicht sinnvoll, aber nicht unbedingt notwendig?

(e) Welche Attribute bzw. Attributkombinationen sind Schlüsselkandidaten?

(f) Gibt es in Eurem Modell schwache Entitäten?

(g) Gibt es in Eurem Modell rekursive Beziehungen?

Hinweis: Für diese Aufgabe sind ein Diagramm und begleitende Notizen ausreichend, es muss nicht jede Teilaufgabe separat gelöst werden. Ihr solltet vor allem in der Lage sein, Eure Lösung in der Praktikumssitzung vorzustellen und zu erklären.

**V2: Modellierung in UML**

(a) Modelliert die Mini-Welt der Semesteraufgabe nun in UML.

(b) Gibt es in Eurem Modell ein Beispiel für *Aggregation*?

(c) Gebt in Eurem UML-Modell die Rollen und Multiplizitäten für die Beziehungen an.

(d) Was ist der Unterschied zwischen *Funktionaliäten* im E/R-Kontext und *Multiplizitäten* im UML-Kontext?

(e) Welche Vor- und Nachteile hat UML gegenüber der Modellierung durch E/R-Diagramme? (Gemeint ist: im Kontext der Semesteraufgabe, zur Modellierung für einen späteren Datenbankentwurf.)

## Präsenz

In der ersten Praktikumswoche habt Ihr bereits erste Erfahrungen mit dem Kommandozeilenwerkzeug (CLP) `db2` gemacht. Ihr habt den Knoten `salz` mit der Instanz `dbprak` katalogisiert und konntet das im *node directory* nachvollziehen. Ebenso habt Ihr die Datenbank `sample` unter dem Alias `mysample` katalogisiert und sie dann im *db directory* wiedergefunden. Vielleicht habt Ihr auch schon auf die Datenbank zugegriffen. Sowohl der Knoten (also die Instanz `dbprak` auf dem Recner `salz`), als auch die Datenbank gehören jedoch dem Account `dbprak`. Ihr konntet nur lesend zugreifen.

In dieser Woche wollen wir nun eine *eigene* Datenbank in Eurer *eigenen* DB2-Instanz anlegen und zum Schluss dem Account `dbprak` Leserechte geben.

### P1: Informationen über eine Datenbank ermitteln

(a) Verschafft Euch wie in der letzten Woche Zugriff auf die Instanz `dbprak` auf dem Rechner `salz` (beides zusammen entspricht dem Knoten, den Ihr letzte Woche katalogisiert habt). Testet, ob Ihr auf die existierende Datenbank `sample` zugreifen könnt.

(b) Versucht, möglichst viel über diese Datenbank herauszufinden. Zu welchem Schema gehört sie? Welche Tabellen existieren, wie sind diese aufgebaut? Welche Berechtigungen habt Ihr auf der Datenbank?

Tabellen:

Schema der Datenbank:

Berechtigungen:

**P2: Eigene Datenbank anlegen**

(a) Um eine eigene, neue Datenbank anzulegen, müsst Ihr Euch mit dem Datenbank-Server verbinden:

```
ssh salz
```

(b) Legt dann eine neue, eigene Datenbank mit dem Namen `videothek` an.

Befehl:

(c) Gebt Eurem Tutor (mit dem Accountnamen `dbprak`) die Berechtigung, sich mit Eurer Datenbank zu verbinden.

Befehl:

(d) Verlasst den Rechner `salz` mit dem Befehl

```
exit
```

**P3: Eigene, entfernte Datenbank lokal katalogisieren**

Katalogisiert nun auf dem lokalen Rechner die neu erstellte Datenbank unter dem Alias `videothekXX` (ersetzt das `XX` durch Eure Gruppennummer).

Befehl(e):