Video

Datenbanken

Kernziele:

- Entwicklung einer relationalen Datenbank (von Anforderungsanalyse über Realisierung zur Nutzung)
- Steuermechanismen von Datenbanken

Überblick (evtl. Obermenge)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

1. Grundbegriffe Datenbanken
2. Grundlagen der Entity-Relationship-Modellierung
3. Tabellenableitung
4. Normalformen
5. SQL: Erstellen von Tabellen
6. SQL: Einfache Anfragen
7. SQL: Komplexere Anfragen
8. SQL: Gruppierung und Analyse von NULL-Werten
9. JDBC
10. Effiziente Datenverwaltung
11. Programmierung in der Datenbank
12. Testen von DB-Software
13. Transaktionen
14. Views und Datenbankverwaltung
15. NoSQL mit MongoDB
16. Wiederholung

Zentrale Ziele

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Wozu gibt es Datenbanken
- Wie kann man Anforderungen systematisch modellieren
- Wie kommt man zu qualitativ hochwertigen Tabellen
- Wie formuliert man strukturierte Anfragen
- Wie nutzt man Datenbanken aus anderer Software

- Vorgehen: Von den Anforderungen über die Umsetzung mit Tabellen hin zur vertieften Nutzung und Integration in andere Software

Ich

- Stephan Kleuker, geboren 1967, verheiratet, 2 Kinder
- seit 1.9.09 an der HS, Professur für Software-Entwicklung
- vorher 4 Jahre FH Wiesbaden
- davor 3 Jahre an der privaten FH Nordakademie in Elmshorn
- davor 4 ½ Jahre tätig als Systemanalytiker und Systemberater in Wilhelmshaven

- s.kleuker@hs-osnabrueck.de, Zoom-Termine kurzfristig per E-Mail vereinbar

Ablauf

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- 2h Vorlesung + 2h Praktikum = 5 CP d. h. etwa 150 Arbeitsstunden
- Praktikum :
  - Anwesenheit = (Übungsblatt vorliegen + Lösungsversuche zum vorherigen Aufgabenblatt)
  - ca. 11 Übungsblätter mit Punkten ($\Sigma \geq 100$), zwei bis drei Studis als Team (gemeinsam planen, getrennt lösen, dann besprechen)
  - Praktikumsteil mit 85 oder mehr Punkten bestanden
- Prüfung: Projektbericht

- Folienveranstaltungen sind schnell, bremsen Sie mit Fragen
- von Studierenden wird hoher Anteil an Eigenarbeit erwartet

Verhaltenscodex

- Anwesenheit: Rechner sind zu Beginn der Veranstaltung aus
- Handys sind aus
- Wir sind pünktlich
- Es redet nur eine Person zur Zeit

- Sie haben die Folien zur Kommentierung in der Vorlesung vorliegen, zwei Tage vor VL abends mit Aufgaben im Netz, Aufgabenzettel liegen in der Übung vor (Ihre Aufgabe), auch `http://www.edvsz.hs-osnabrueck.de/skleuker/index.html`

- Probleme sofort melden
- Wer aussteigt, teilt mit, warum

Voraussetzungen und Synergien

- Sie können gut programmieren
- Sie können Java: Klassen, Methoden, Collections, Ausnahmen
- Sie wissen was eine Menge und Relation ist

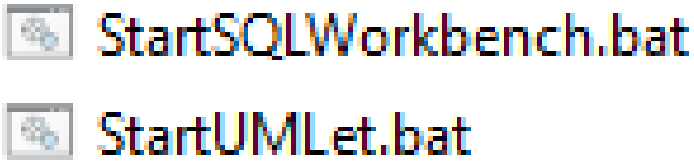
- Einige Gemeinsamkeiten mit OOAD: Modellierung und Umsetzung
- DBs nutzen Algorithmen aus A&D

- Gibt DB-Vertiefung (Prof. Dr. Tapken)
- In SW-Architektur wird Persistenz wieder aufgegriffen

Eingesetzte Technologien

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Windows 10, 64 Bit; gibt SEU zum Herunterladen, enthält alle notwendigen Werkzeuge, ist zu nutzen!
- Start jeweils über bat-Dateien



Details:

- Java, 64 bit , genauer Azul Zulu mit OpenJFX https://www.azul.com/downloads/zulu/
- Apache Derby https://db.apache.org/derby/
- Eclipse für Java https://www.eclipse.org/
- SQLWorkbench/J https://www.sql-workbench.eu/
- UMLet (optional für ER-Diagramme) https://www.umlet.com/

Literatur 1/2

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Zur Vorlesung extrem gut passend ☺

  [KL] S. Kleuker, Grundkurs Datenbankentwicklung, Springer Vieweg, 5. Auflage, 2024 (als PDF über Bibliotheks-Link verfügbar)

- Sehr gelungene Bücher mit tieferen Einblicken

  A. Kemper, A. Eickler, Datenbanksysteme, Oldenbourg

  [EN] R. Elmasri, S. B. Navathe, Grundlagen von Datenbanksystemen, Pearson/Addison Wesley

  M. Schubert, Datenbanken: Theorie, Entwurf und Programmierung relationaler Datenbanken, Vieweg+Teubner

  G. Matthiessen, M. Unterstein, Relationale Datenbanken und SQL, Addison-Wesley,

  (jeweils aktuelle Auflage)

Literatur 2/2

- Eigentlich für Nicht-Informatiker, deshalb aber auch für DB-Einsteiger geeignet

  R. Steiner, Grundkurs Relationale Datenbanken, Vieweg+Teubner
- Nicht als Einstieg, aber als preiswertes Nachschlagewerk

  G. Kuhlmann, F. Müllmerstadt, SQL, Rowohlt Tb.

Warum Datenbanken?

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Warum haben wir überhaupt ''Datenbanken''?

- Dateien und Dateisysteme sind doch gut genug?
- Oder? (Stand 1965)

Beispiel Arbeitsprozesse: In einem kleinen Unternehmen findet die Datenverwaltung ohne Software statt. Das Unternehmen wickelt Bestellungen ab, die mehrere Artikel umfassen können und in schriftlicher Form vorliegen. Um schnell reagieren zu können, werden auch nicht vollständige Bestellungen versandt und den bestellenden Personen mitgeteilt, dass die weiteren Artikel nicht lieferbar sind. Die bestellende Person müsste bei Bedarf diese Artikel wieder neu bestellen. Zur Analyse der Abläufe möchte man eine Übersicht haben, wie viele Bestellungen eine bestellende Person gemacht hat und wie hoch die bisherige Bestellsumme war. Weiterhin soll bekannt sein, wie häufig ein Artikel bestellt wurde und wie häufig dieser Artikel nicht vorrätig war, obwohl eine Bestellung vorlag.

Skizzieren sie den Arbeitsablauf ab einer eingehenden Bestellung ohne SW mit seinen möglichen Alternativen, so dass alle Informationen erfasst werden können.

Was ist eine Datenbank (informell)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

Unter einer *Datenbank* wird eine Sammlung von Daten verstanden. Eine Datenbank entspricht einem elektronischen Aktenschrank, auf dem der nutzende Person eine Reihe von Operationen ausführen kann. Die nutzende Person hat die Möglichkeit, neue „Dateien“ (Datenschemata) anzulegen, Datensätze hinzuzufügen, zu ändern oder zu löschen und Datensätze herauszusuchen.

Forderung 1: Garantie von Persistenz

Forderung 2: Anlegen von Datenschema (Tabellen)

Forderung 3: Einfügen, Löschen, Ändern von Daten

Forderung 4: Strukturiertes Lesen von Daten

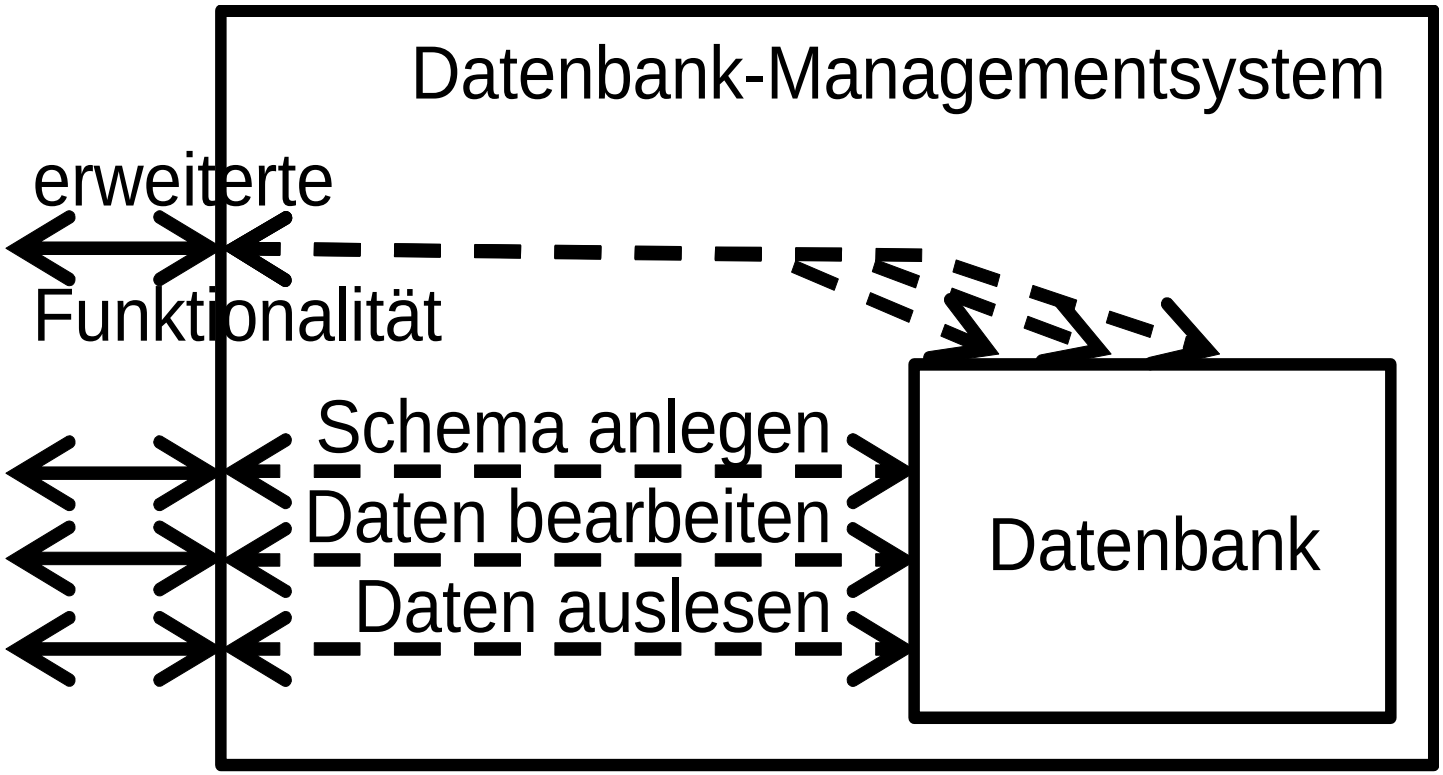
Was ist ein Datenbank-Management-System

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Ein *Datenbank-Management-System (DBMS)* umfasst die Gesamtheit an Programmen, die zum Aufbau, zur Nutzung und zur Verwaltung von *Datenbanken* notwendig ist.
- Das DBMS ermöglicht verschiedenen Nutzungsgruppen einen einfachen Zugang zu den gespeicherten Datenbeständen.

Kürzel	Begriff	Erläuterung
DB	**Datenbank**	**Strukturierter, von DBMS verwalteter Datenbestand**
DBMS	**Datenbank-Management-System**	**SW zur Verwaltung von Datenbanken**
DBS	**Datenbanksystem**	**DBMS plus Datenbank(en)**

DBS = DBMS kapselt DB [KL]



Aufgaben eines DBMS (1/3)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Operationen
  - Speichern, Suchen, Ändern, Löschen
- Integration
  - einheitliche Verwaltung aller Daten, z. B. enthalten alle Bestellungen die gleichen Daten
  - Möglichkeit zur redundanzfreien Datenhaltung, z. B. jede Bestellung wird durch eine eindeutige Nummer identifiziert
- Konsistenzüberwachung (Integritätssicherung)
  - Garantie der Korrektheit bei Datenbankänderungen
  - z.B. abhängige Daten werden mit verändert

Aufgaben eines DBMS (2/3)

- Benutzungssichten
  - Unterschiedliche Anwendungen benötigen unterschiedliche Sichten
  - z.B. nur bestimmte Teildaten
  - z.B. bestimmte Übersichten
- Zugriffskontrolle
  - welcher nutzende Person darf auf welche Daten in welcher Form zugreifen
- Katalog
  - Verwaltung der Information welche Informationen in der DB vorhanden sind
  - z.B. Aufbau von Tabellen
  - z.B. Randbedingungen, die durch Daten eingehalten werden müssen

Aufgaben eines DBMS (3/3)

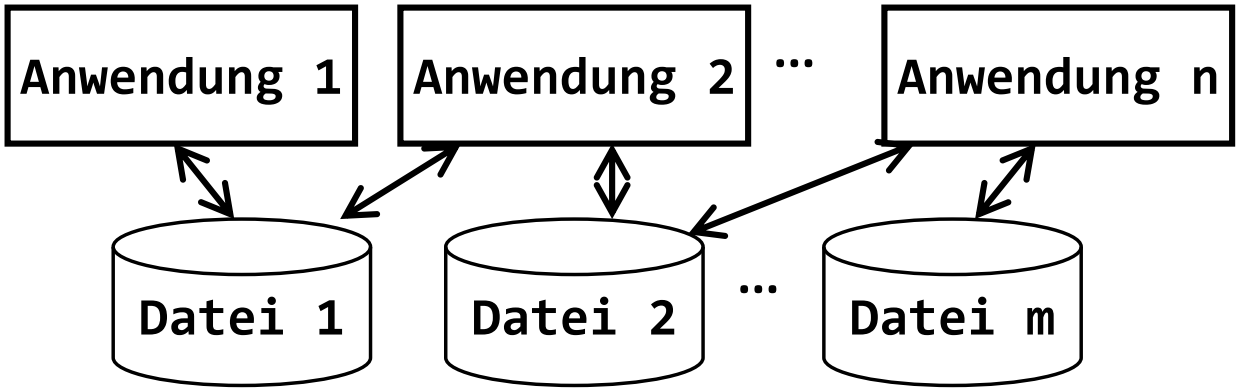
HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Transaktionen
  - Zusammenfassung von Datenbankänderungen zu einer Aktion, deren Effekt bei Erfolg permanent in der DB gespeichert werden soll
- Synchronisation
  - konkurrierende Transaktionen mehrerer Benutzungen müssen synchronisiert werden, um gegenseitige Beeinflussungen zu vermeiden
- Datensicherung
  - Ermöglichung der Systemwiederherstellung z.B. nach einem Systemabsturz

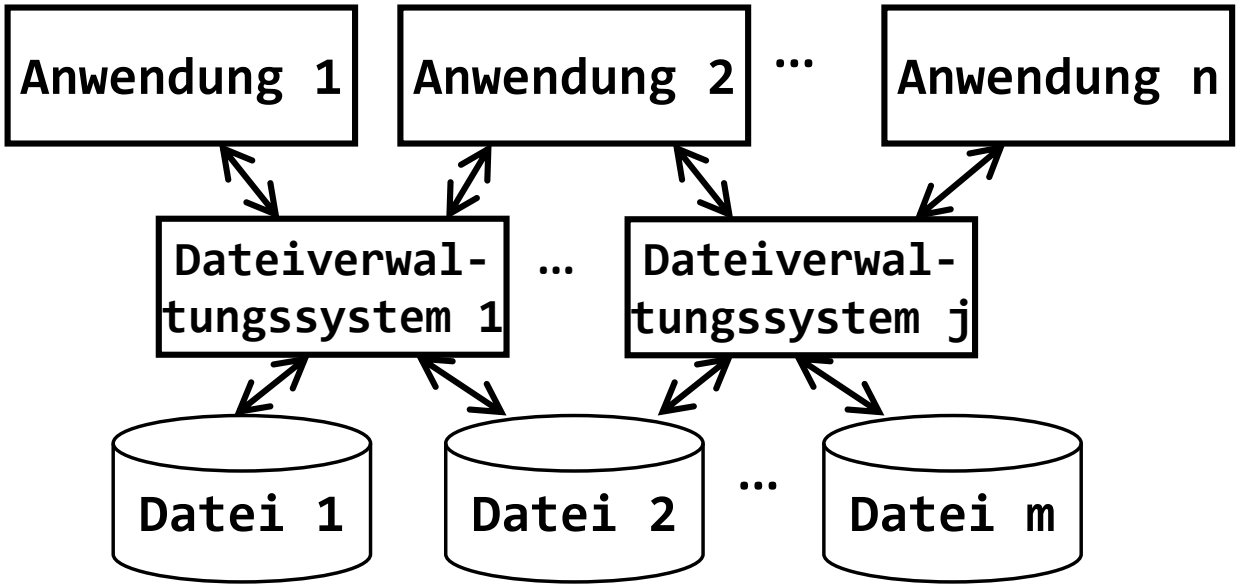
Entwicklung von Datenbanken (1/2)

Video

1. Klassisch ohne spezielle Verwaltung

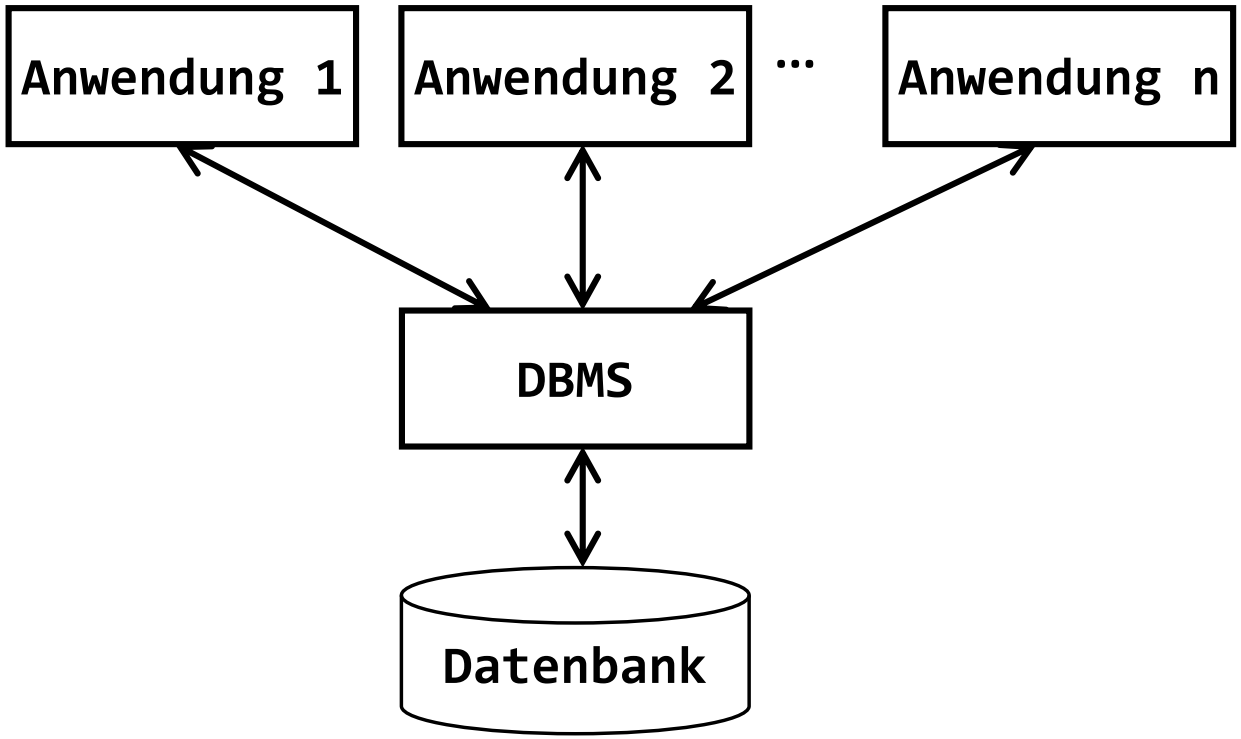


2. Nutzung von Dateiverwaltungssoftware für Dateien



Entwicklung von Datenbanken (2/2)

3. Einführung eines Datenbank-Management-Systems (DBMS)



Anmerkung: oft echtes Client-Server-System, d. h. DBMS läuft auf anderem Rechner als Anwendungen

Einsatz von DBMS

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Wie werden Datenbankmanagementsysteme verwendet?

- Betriebliche Anwendungen
- Web-Anwendungen
- mobile Programme
- Spezialprogramme
- ...
- als Teil eines *Informationssystems*

- heute typischerweise Client/Server-Architekturen
- in gleicher Form relevant für Cloud-basierte Systeme, Docker-Images, virtuelle Maschinen, Micro-Architekturen

Beispiele für DBMS

• IBM	DB2 UDB (relational, OO, XML)
• Oracle	Oracle (relational, OO, XML)
	Berkeley DB (auch XML-Variante)
• Microsoft	SQL-Server 20xx (MSSQLServer 2012)
	Access / Visual FoxPro
• Sybase	Sybase
• Informix / IBM	Informix
• MySQL /MariaDB	Oracle / freier Fork
• SAP	MaxDB (SAP-DB, Adabas)
• PostgreSQL	PostgreSQL
• **Apache**	**Apache Derby (früher auch JavaDB)**
• SQLite	SQLite, in mobile Betriebssysteme integriert
• Firebird	Firebird, freier Ableger von InterBase
• Lotus	Lotus Domino Server / Lotus Notes
• …	

Datenbankarten

- letzte Folie „relationale Datenbanken“ (vereinfacht: beliebige Verknüpfung einfacher Tabellen)
- ist der deutlich am weitesten verbreitete Anteil
- gibt andere Arten von Datenbanken für spezielle Aufgaben
  - hierarchisch, netzwerkartig (historisch interessant)
  - objektorientiert (einfache Verknüpfung mit OO)
  - dokumentenorientiert (Fokus auf zusammenhängende Daten, typisch No[t only]SQL-Datenbanken)
  - XML-basiert, verteilt, …

- oft nicht Frage „welche DB“ sondern „welche Kombination von DBs“
- Hinweis: Vertiefung Prof. Dr. Heiko Tapken

Überblick DB-Entwurf

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

Grob sind die Einzelschritte:

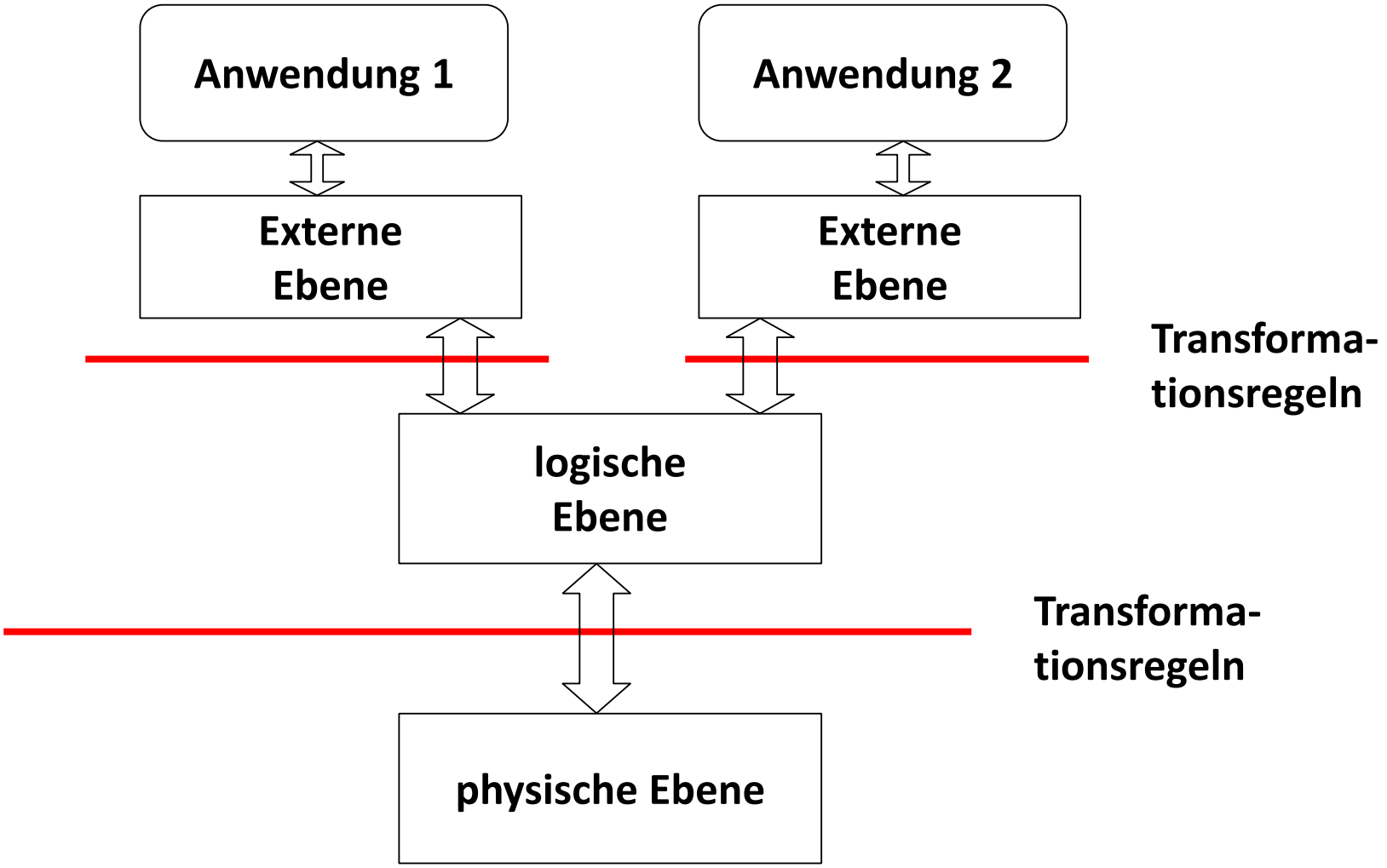
- die logische Datenmodellierung
- die physische Datenmodellierung
- der Aufbau einer Datenbank, sowie
- der Betrieb (Administration, Konfiguration) derselben

Nächste Schritte

- Auswahl, Kauf, Installation, Konfiguration eines DBMS
- Anlegen der Datenbank, Einspielen der Daten, ...
- Administration
- Leistungsoptimierung
- Sicherheitsaspekte
- Anwendungsentwicklung
- Backup, Replikation, Clustering, Recovery, ...

- Aufgabe: Überlegen Sie Kriterien, die bei der Auswahl eines DBMS eine Rolle spielen

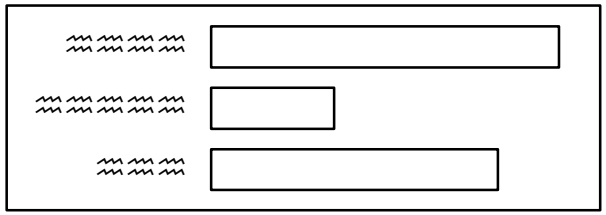
ANSI/SPARC-Modell



Überblick über Ebenen [KL]

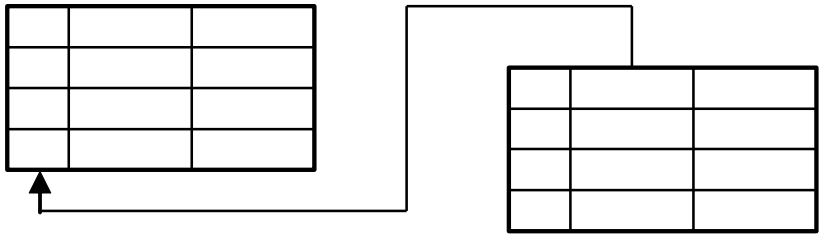
externe Ebene

Sichten (Ein- und Ausgabemasken)



logische Ebene

Tabellen und Abhängigkeiten



physische Ebene

Konfiguration / DB-Einrichtung

```
<table name="Artikel">
  <size> 5G <\size>
<\table>
```

Externe Ebene

Dies ist die *Benutzungssicht* auf die Daten:

- nutzende Person sieht nur die Daten und Beziehungen, die im zugeordneten externen Modell vom Anwendungsadministrator definiert sind.
- Der logische Inhalt des externen Modells ist vollständig aus dem konzeptionellen Modell ableitbar.
- Im externen Modell können Felder vorhanden sein, die im logischen Modell fehlen (berechnete Felder).
- Typischer Teil der Benutzungssicht: Masken zur Ein-und Ausgabe von Daten

Logische (konzeptionelle) Ebene

Zentraler Inhalt ist das *logische Datenmodell:*

- Beschreibt die Daten der Miniwelt auf logischer Ebene (Datenobjekte, Integritätsregeln, ...).
- Bezugspunkt für alle Anwendungen
- Logische Datenunabhängigkeit
- Anwendungsübergreifendes Datenmodell

- Im relationalen Modell kann man sich Tabellen vorstellen, die die Basisinformationen beinhalten

Physische Ebene (auch interne Ebene)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Definiert die *Speicherstruktur* der Daten:

- Hier wird die *physische* Datenorganisation festgelegt
- Festlegung von internen Datensatztypen, Verkettungsmechanismen, physische Indizierung etc.
- Ist direkt oberhalb der Ressourcenverwaltung durch das Betriebssystem angesiedelt
- Die Güte des internen Schemas hat wesentlichen Einfluss auf die Leistung des Gesamtsystems
- hier hat DB-Administration Möglichkeiten zur Optimierung und Pflege der Datenbank

Wozu ANSI/SPARC-Dreischichtenmodell?

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Dieses ANSI/SPARC-Modell stammt von 1975

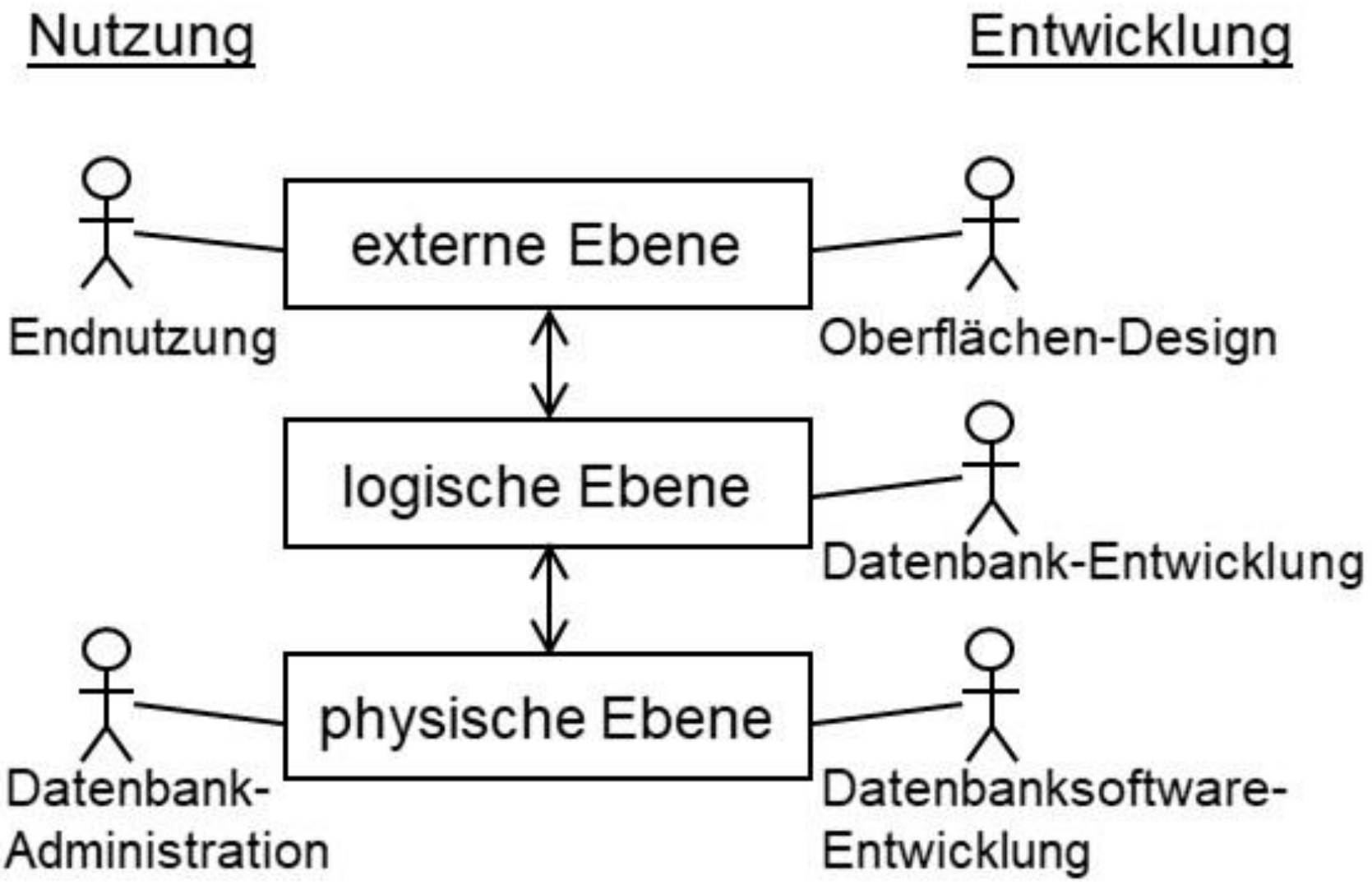
(ANSI/X3/SPARC Study Group on Data Base Management Systems, FDT ACM SIGMOD 7,2 (1975))

Wozu ist das gut?

- Änderungen an der internen Darstellung können vorgenommen werden, ohne die konzeptionelle Ebene zu berühren
- Ebenso ist es möglich, Teile der konzeptionellen Schicht zu ändern, ohne die Benutzungssichten zu berühren

→ höhere Robustheit gegenüber Änderungen

Nutzung und Entwicklung der Ebenen [KL]



Beispiel: Datenbank Hochschule

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Logische Ebene (z.B. Tabellen [nicht optimal])
  - *Studi (studid: int, name: string, login: string, alter: int)*
  - *Kurs (kursid: int, kname: string, stunden: int)*
  - *Fachbereich (fbid: int, fbname: string, budget: real)*
  - *Lehrt (fbid: int, kursid: int)*
  - *Eingeschrieben (studid: int, kursid: int)*
- Physische (interne) Ebene
  - Speicherung der Relationen als Files: unsortierte Menge von physischen Records
  - Index auf der ersten Spalte von *Studis* und *Kurse* zur Beschleunigung des Datenzugriffs
- Externe Ebene (View)
  - Anfragemaske: Wie viele Studierende haben sich in jedem Kurs eingeschrieben?
  - *Kurs_Info (kursid: int, einschreibanzahl: int)*

2. Grundlagen der Entity-Relationship-Modellierung

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- Was ist ein Modell
- Was sind Entitäten
- Was sind Relationen
- Was sind Attribute
- Spezialfälle von ER-Modellen

Konzeptioneller Entwurf (logische Ebene)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Der konzeptionelle Datenbankentwurf ist dem Modellieren in den Naturwissenschaften und der Technik ähnlich:

Ein Modell wird konstruiert um das Verständnis zu verbessern und um Details zu abstrahieren.

Verständnis: Die Bedeutung der Daten und ihre Beziehungen untereinander als Informationsstrukturen darstellen

Abstraktion: Vernachlässigung der Details individueller Datenwerte

Datenmodellierung

- Objekte der realen Welt, die für die Aufgabenstellung relevant sind, werden mit ihren Beziehungen untereinander in abstrakter Weise beschrieben, d.h. modelliert

- Zentrale Fragen:

  Welche Objekte / Entitäten spielen eine Rolle?

  Welche Eigenschaften / Attribute haben diese Entitäten?

  Wie stehen die Entitäten miteinander in Beziehung (Relation)?

  Welche Eigenschaften haben diese Beziehungen?

- Es stellen sich die Probleme der Anforderungsanalyse beim Versuch, Bestellendinteressen in Datenbankanforderungen umzuformen, wie sie z. B. in der Veranstaltung OOAD diskutiert werden

Grundbegriffe der Entity-Relationship-Modelle

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Entität *entity* (oft auch „Objekt“)
  - individuelles, identifizierbares Exemplar
  - beschrieben durch Eigenschaften

- Entitätsmenge *entity set* (oft auch „Objekttyp“, „Entitätstyp“, vereinfacht ungenau auch „Entität“)
  - Zusammenfassung von Entitäten mit gleichartigen Eigenschaften
  - Name (Substantiv) als Oberbegriff für alle Entitäten der Menge

Grundbegriffe der Entity-Relationship-Modelle

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Attribut *attribut, property*
  - Eigenschaft von allen Entitäten einer Entitätsmenge
  - Name entsprechend fachlicher Bedeutung
  - vorgegebener Wertebereich (auch „Domäne“ *domain*)
  - beschreibende / identifizierende Attribute

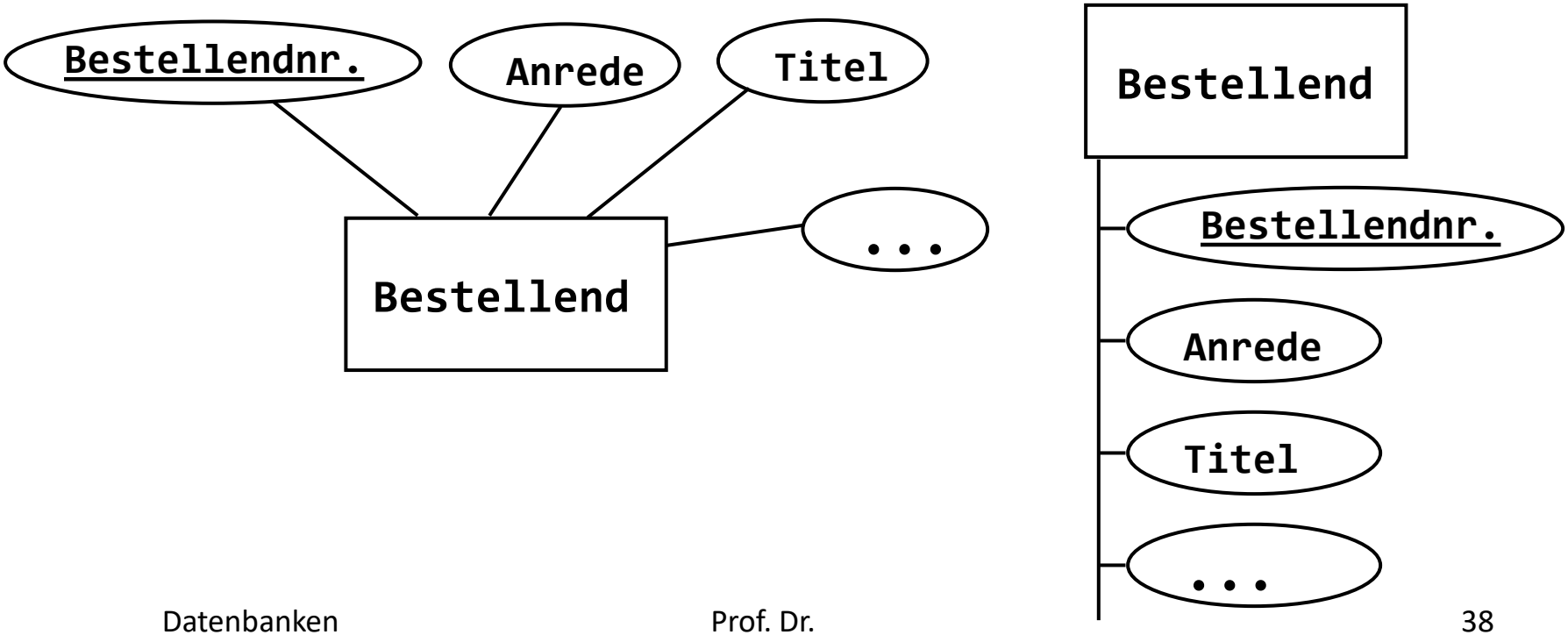
- Schlüssel *key (vorläufige Definition)*
  - identifizierende Attributkombination

Grundbegriffe der Entity-Relationship-Modelle

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- Beispiel zur Visualisierung: Entitätstyp Bestellend
- jede bestellende Person hat Eigenschaften: Bestellendnummer, Anrede, Titel, …
- Beispiel: (42, 'Frau', 'Prof. Dr.', …)
- zu jeder (Bestellend)nummer gibt es maximal eine bestellende Person

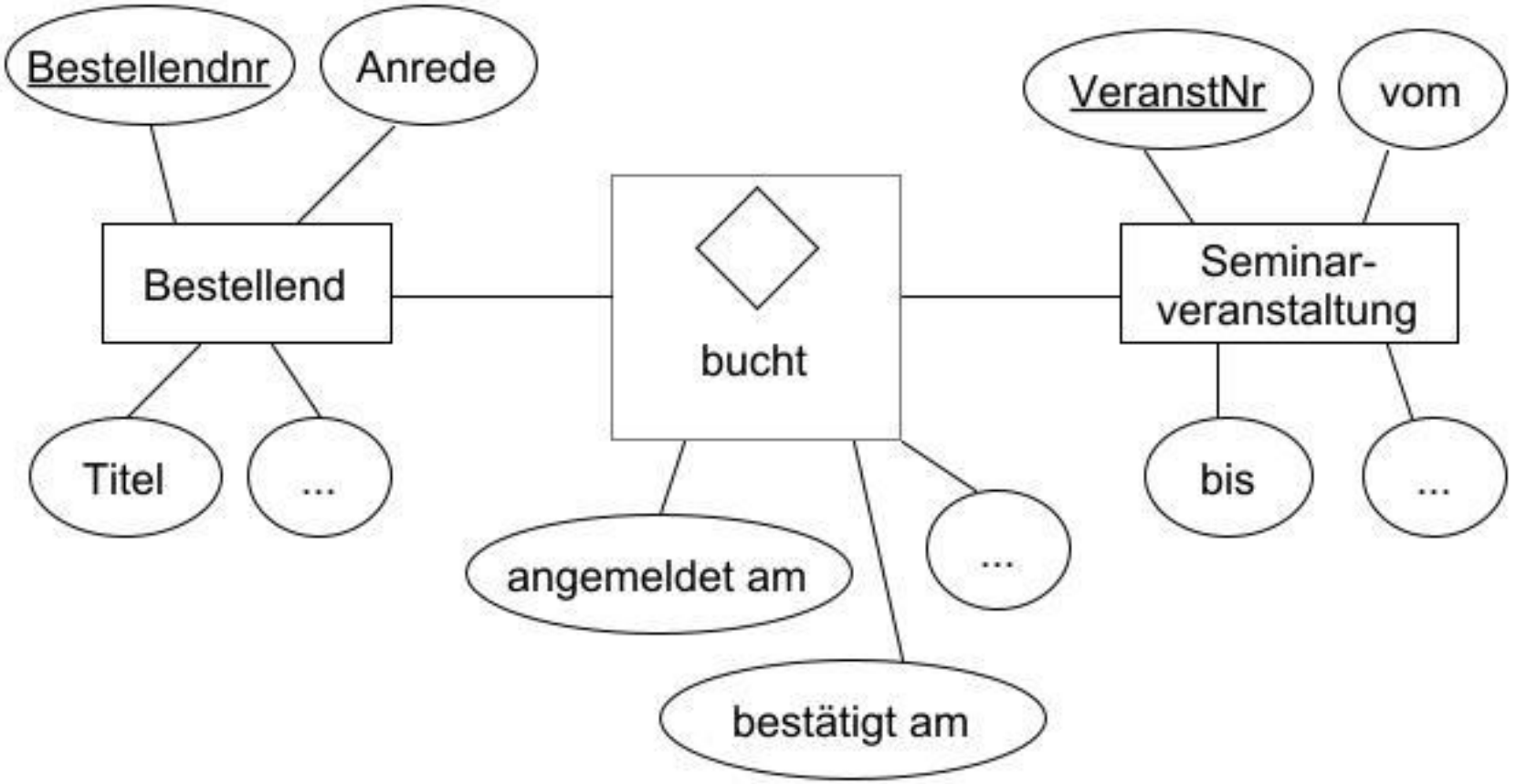


Grundbegriffe der Entity-Relationship-Modelle

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Assoziation *relationship*
  - Zusammenfassung von gleichartigen Beziehungen zwischen Entitäten
  - Name (Verbform) als Oberbegriff für die gleichartigen Relationen zwischen den Entitäten zweier Entitätsmengen
  - kann ebenfalls Attribute haben
  - Auch nur Beziehung oder Relation genannt

Graphische Darstellung (Skizze, unvollständig)



identifizierendes Attribut einer Entität ist unterstrichen (Hinweis: Es ist möglich, dass mehrere Attribute zur Identifizierung benötigt werden oder es identifizieren können)

Konzept Kardinalitäten (1/2)

- Kardinalitäten beschreiben, wieviele Entitäten mit wieviel anderen Entitäten in Beziehung stehen
- In UML oft auch Multiplizitäten genannt
- Folgende Kardinalitäten werden betrachtet
  - C : null oder eine Entität
  - 1: genau eine Entität
  - N: eine oder mehrere Entitäten
  - NC: beliebig viele Entitäten (null, eine oder mehrere)
- Angaben wie „genau 2“ werden als Randbedingungen eines ER-Diagramms notiert, im Diagramm steht nur N
- Tritt N oder NC mehrfach bei einer Relation auf wird oft auch M oder MC geschrieben, um zu betonen, dass die Anzahlen nicht gleich sein müssen

Konzept Kardinalitäten (2/2)

Video

Leserichtung (X, Y ∈ {C, 1, N, NC})

- Jede Entität vom Typ A steht mit Y Entitäten vom Typ B in Beziehung

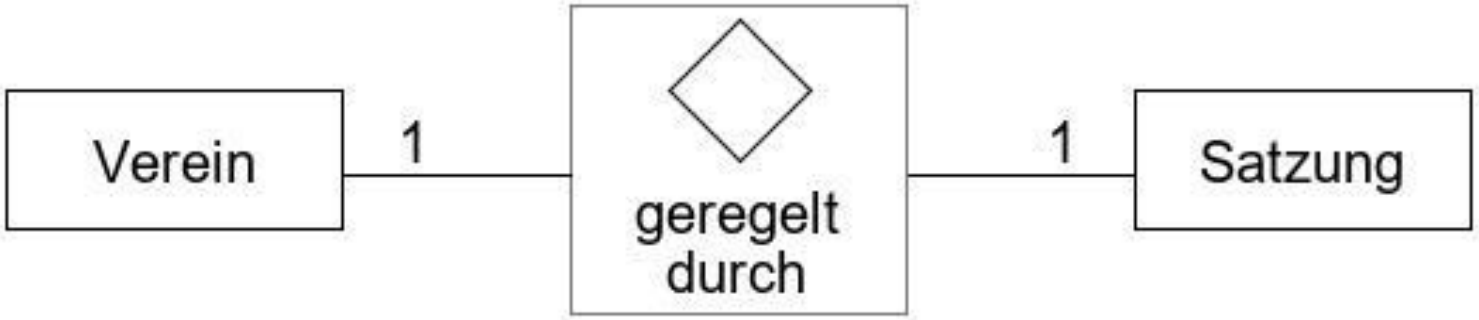
- Jede Entität vom Typ B steht mit X Entitäten vom Typ A in Beziehung

Genutzt wird die modifizierte Chen-Notation

- P. P.-S. Chen: The Entity-Relationship Model-Toward a Unified View of Data, in: ACM Transactions on Database Systems, Vol 1, No 1, März 1976

Kardinalitäten von Assoziationen (1/10)

1:1-Assoziation



jeder Verein hat genau eine Satzung

jede Satzung hat (gehört zu) genau einem Verein

Hinweis: Die Richtigkeit eines Entity-Relation-Ship-Diagramms hängt auch von der individuellen Aufgabenstellung ab

Kardinalitäten von Assoziationen (2/10)

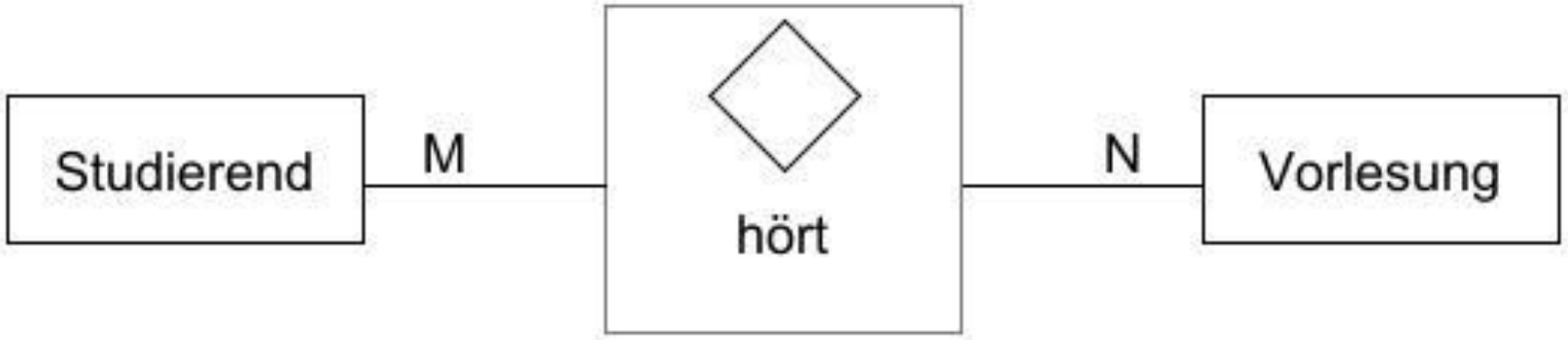
1:N-Assoziation

jede Mutter kann mehrere Kinder gebären
jede Mutter hat mindestens ein Kind geboren
jedes Kind hat genau eine (biologische) Mutter

Kardinalitäten von Assoziationen (3/10)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

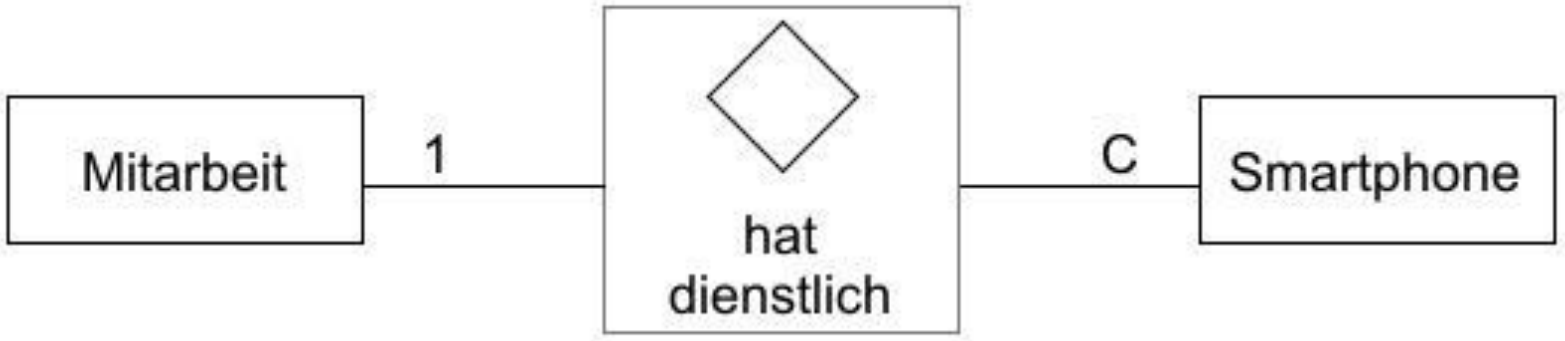
M:N-Assoziation



jede studierende Person kann mehrere Vorlesungen hören
jede studierende Person hört mindestens eine Vorlesung
jede Vorlesung kann von mehreren Studierenden gehört werden
jede Vorlesung wird von mindestens einer studierenden Person gehört

Kardinalitäten von Assoziationen (4/10)

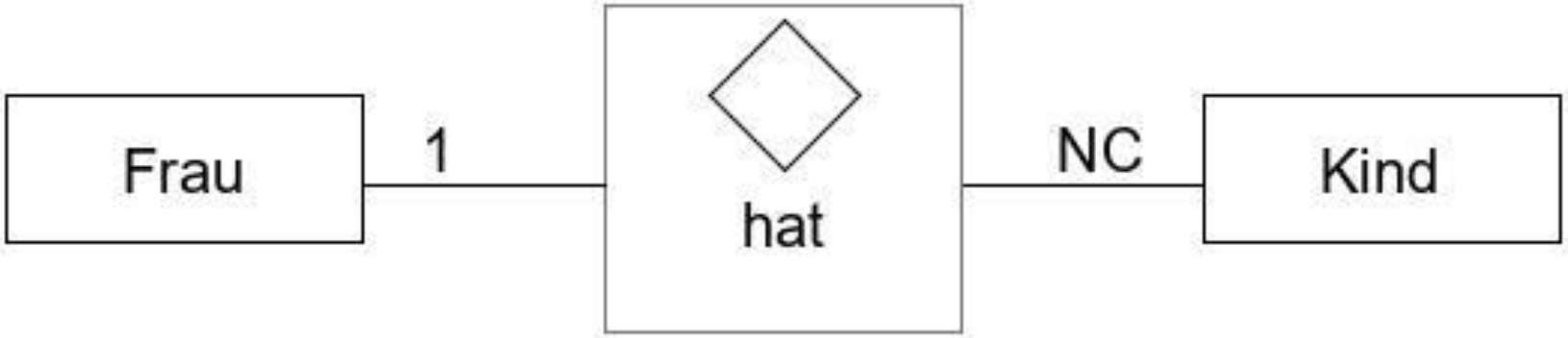
1:C-Assoziation



jede mitarbeitende Person hat dienstlich entweder
genau ein Smartphone oder
kein Smartphone
(konditionelle Beziehung *conditional relation*)
jedes (dienstliche) Smartphone gehört zu genau einer mitarbeitenden Person

Kardinalitäten von Assoziationen (5/10)

1:NC-Assoziation



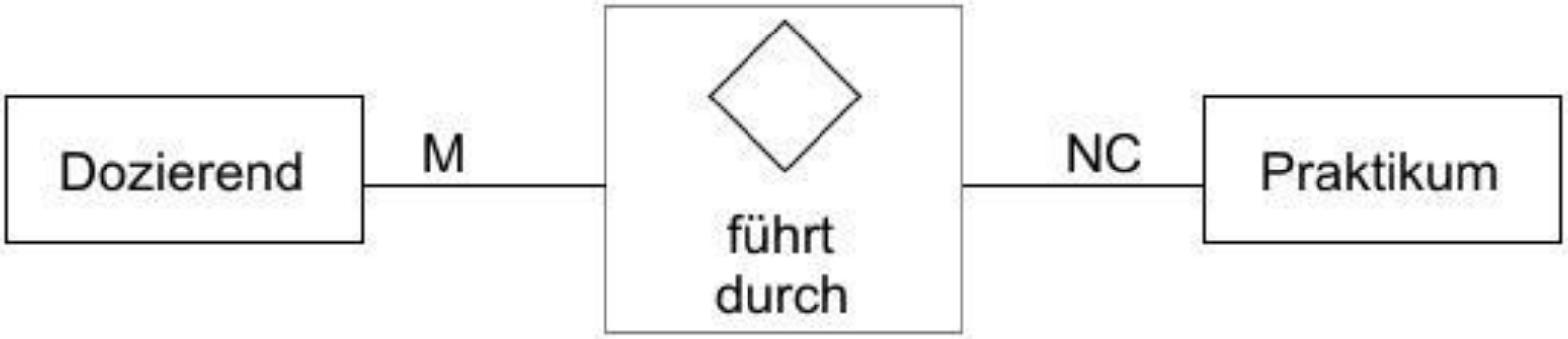
jede Frau kann
- **kein Kind,**
- **ein Kind oder**
- **mehrere Kinder haben**

jedes Kind hat genau eine biologische Mutter (genauer Frau)

Kardinalitäten von Assoziationen (6/10)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

M:NC-Assoziation

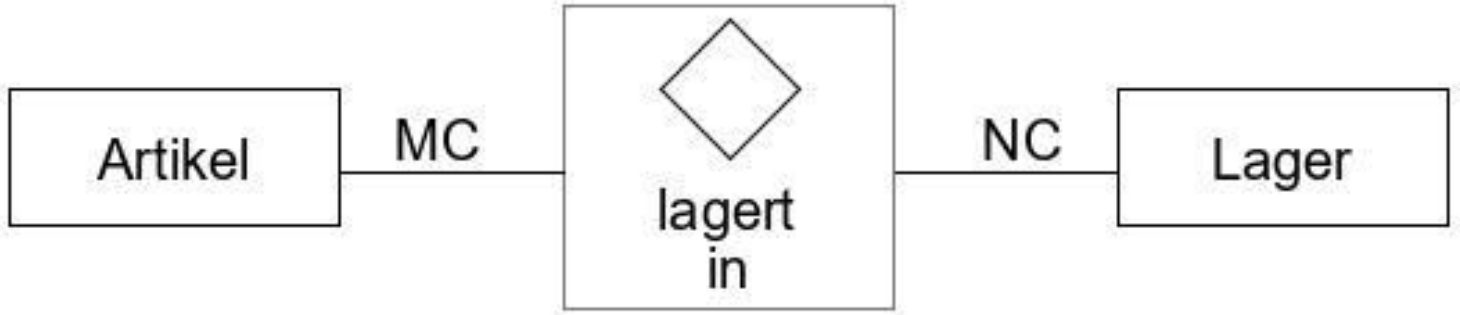


jede dozierende Person kann
- **kein Praktikum,**
- **ein Praktikum oder**
- **mehrere Praktika durchführen**

jedes Praktikum hat
- **eine dozierende Person oder**
- **mehrere dozierende Personen**

Kardinalitäten von Assoziationen (7/10)

MC:NC-Assoziation



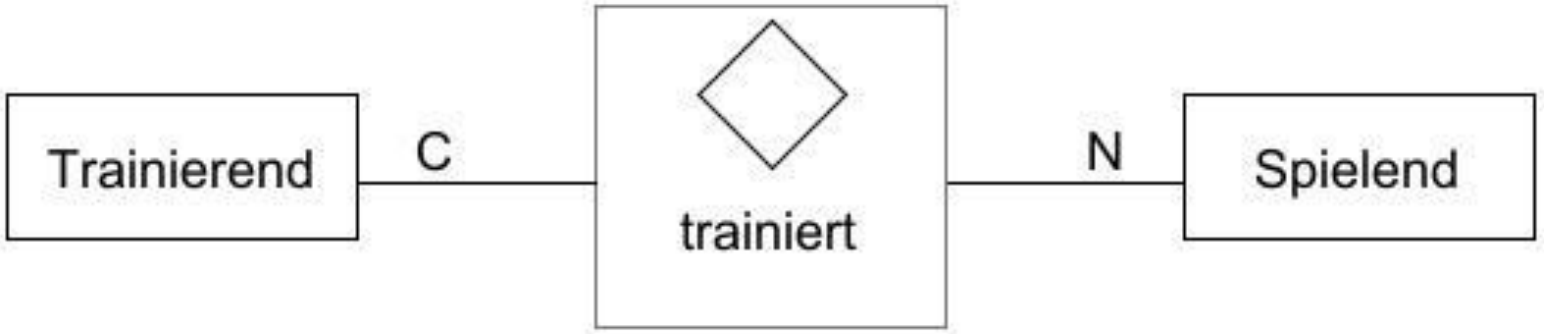
jeder Artikel kann
- **in keinem Lager lagern (ausverkauft)**
- **in einem Lager lagern**
- **in mehreren Lagern lagern**

jedes Lager kann
- **keine Artikel lagern (wird saniert)**
- **einen Artikel lagern**
- **mehrere Artikel lagern**

Kardinalitäten von Assoziationen (8/10)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

C:N-Assoziation



jede trainierende Person trainiert

mindestens eine spielende Person (sonst nicht Trainierend)

eventuell mehrere spielende Personen

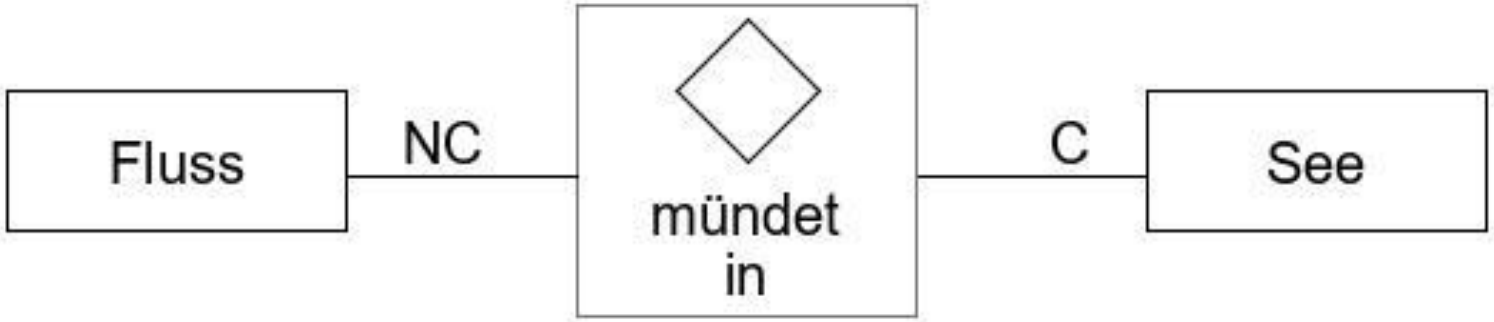
jede spielende Person hat

entweder keine sie trainierende Person

oder eine sie trainierende Person

Kardinalitäten von Assoziationen (9/10)

C:NC-Assoziation



jeder Fluss mündet entweder
genau in einem oder
in keinem See
In jeden See mündet
kein
ein
oder mehrere Flüsse

Kardinalitäten von Assoziationen (9.5/10)

C:C-Assoziation

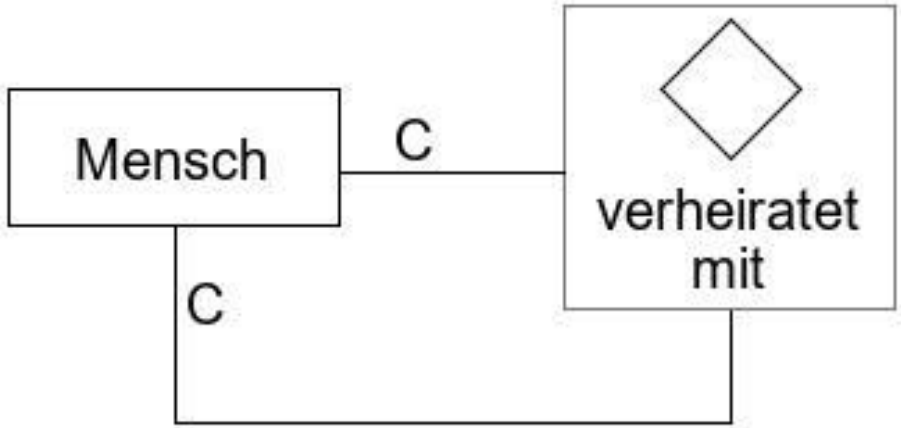
jede Frau ist verheiratet mit entweder
genau einem oder
keinem Mann
jeder Mann ist verheiratet mit entweder
genau einer oder
keiner Frau

Diskriminieren mit Modellierung

Kardinalitäten von Assoziationen (10/10)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

C:C-Assoziation



jeder Mensch ist verheiratet mit entweder
genau einem oder
keinem Menschen
(reflexive Modellierung später genauer)

Muss- und Kann-Assoziationen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Übersicht

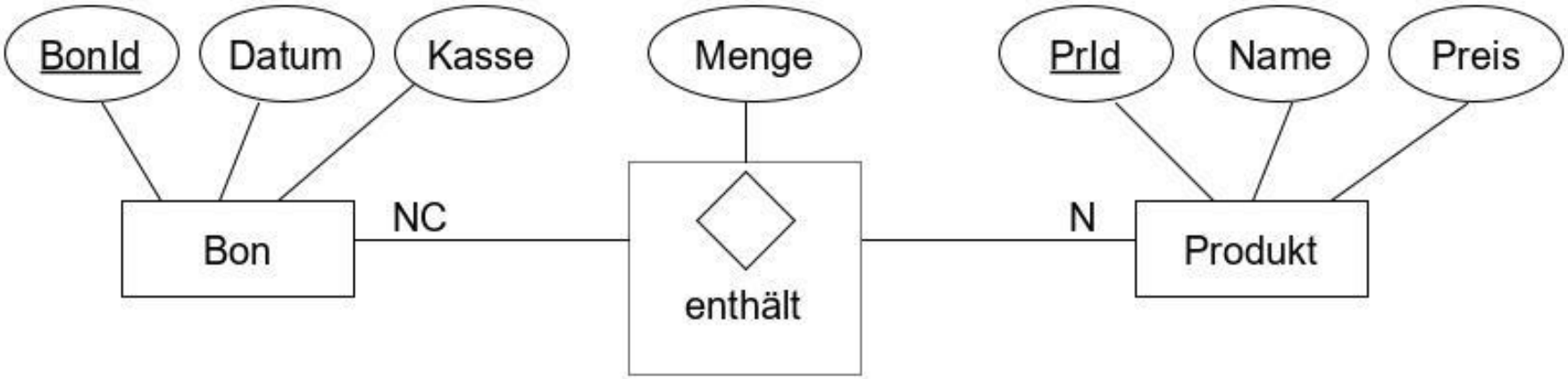
	B	muss		kann	
A		1	N	C	NC
muss	1 M	1:1 M:1	1:N M:N	1:C M:C	1:NC M:NC
kann	C MC	C:1 MC:1	C:N MC:N	C:C MC:C	C:NC MC:NC

Assoziationen mit Attributen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- Attribute, die keiner Entität zugeordnet werden können, sind bei Assoziationen (Verknüpfungspunkt von zwei Entitäten) gut aufgehoben

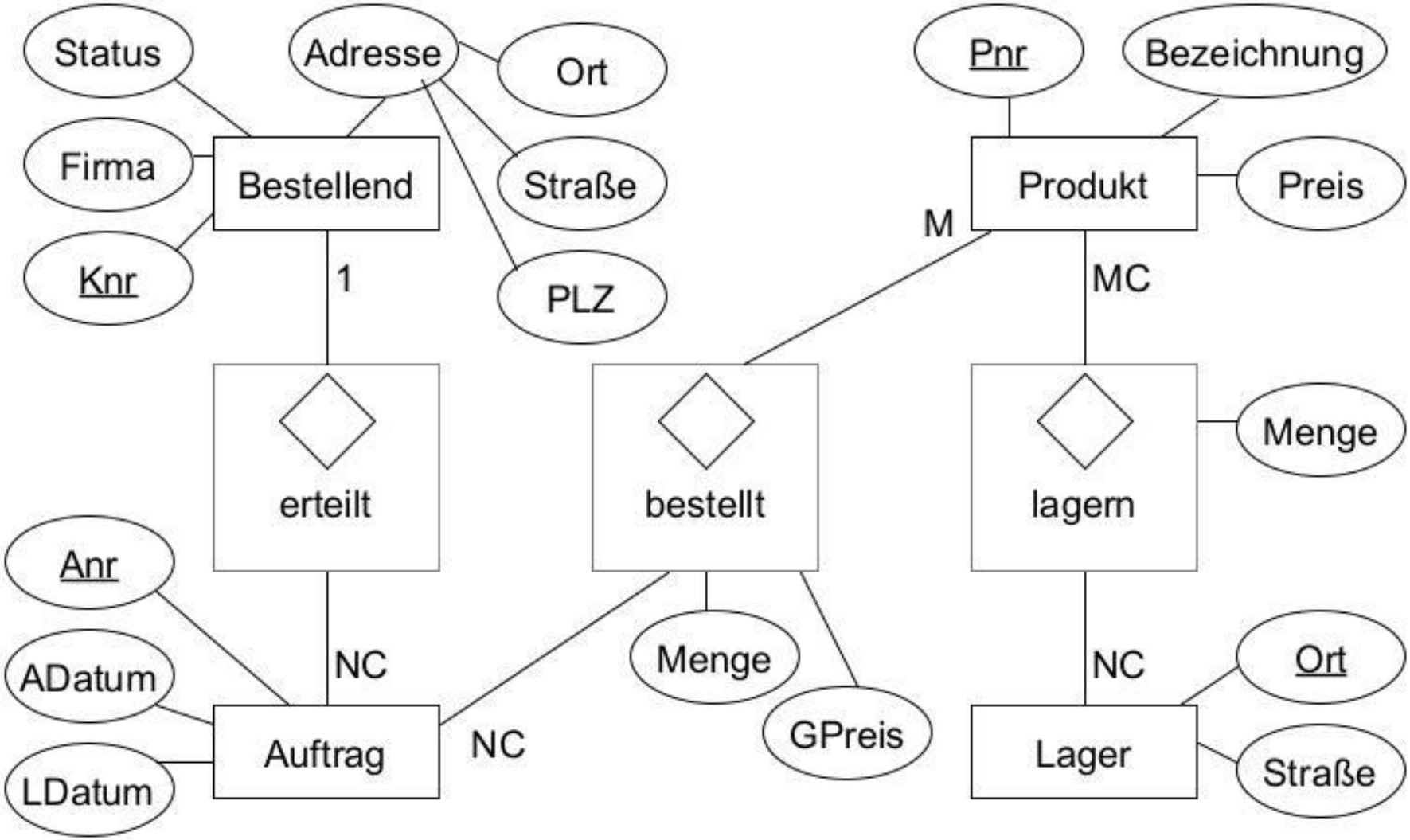


- Die Menge kann nicht beim Bon stehen, da sie für jedes Produkt verschieden sein kann
- Die Menge kann nicht beim Produkt stehen, da jeder Bon eine andere Menge enthalten kann

Lesebeispiel Verkauf

Video

Entwicklungsbeispiel



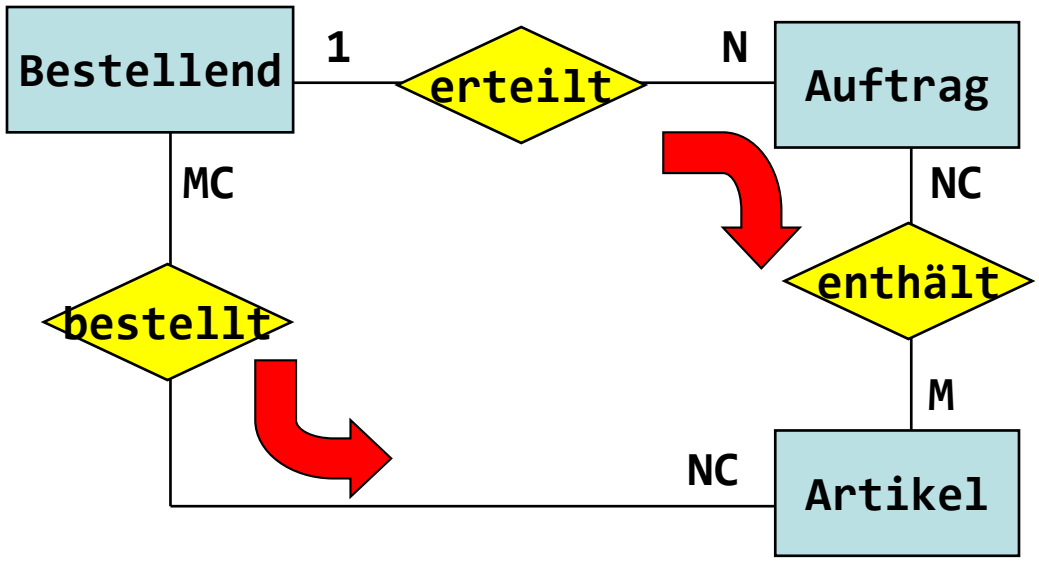
Textanalyse zur Modellfindung

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Nomen können Entitäten oder Attribute sein
- Adjektive deuten auf Attribute hin
- Verben stellen häufig Entitäten (aber auch Attribute und Relationen) in Beziehung
- Hinweis: Qualität des Ausgangstextes zur Analyse ist ein eigenes Thema
- typische Problemfälle:
  - Synonyme: verschiedene Worte für den selben Begriff (Buchtitel, Exemplar)
  - Homonyme: gleiches Wort mit verschiedenen Bedeutungen (Bank, unsaubere Definitionen z. B. Entität)

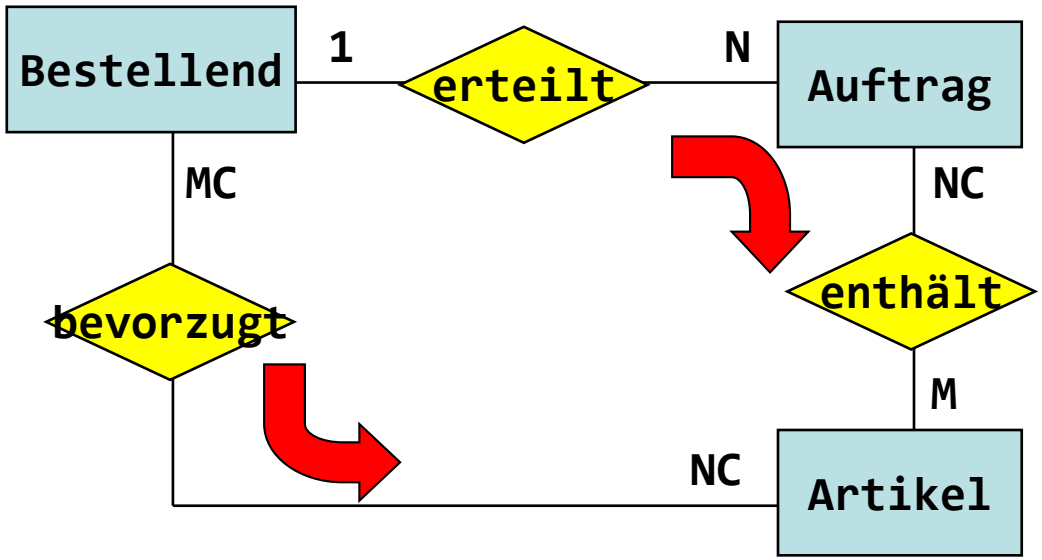
Zyklen in ER-Diagrammen

Video



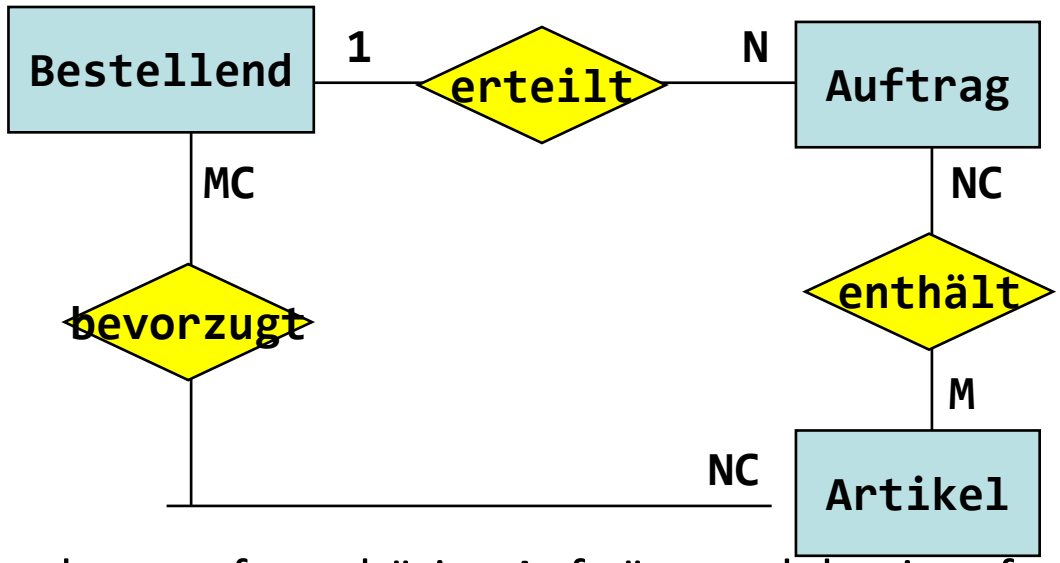
Zyklen in ER-Diagrammen sind zu untersuchen, es darf *generell* keine redundante (auf anderem Weg berechenbare Information) modelliert werden.

Zyklen zur Darstellung unterschiedlicher Zusammenhänge sind dagegen sinnvoll



Analyse von Zyklen

Damit ein Zyklus erlaubt ist, muss für jede Relation geprüft werden, ob sie durch andere ersetzbar ist
(ist hier nicht der Fall)

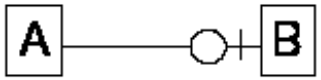
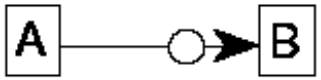
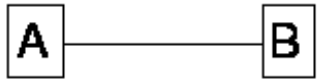
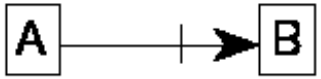
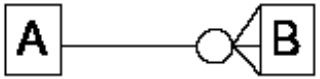
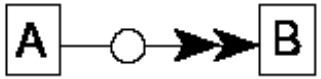
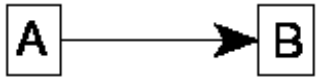
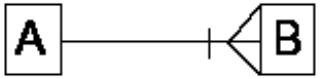
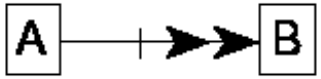


bevorzugt Von bestellenden Personen kann auf zugehörige Aufträge und damit auf bestellte Artikel geschlossen werden, diese müssen aber nicht bevorzugt werden

erteilt Von bestellenden Personen kann auf bevorzugten Artikel geschlossen werden, die in unterschiedlichen Aufträgen vorkommen, die aber nicht von der bestellenden Person erteilt werden müssen.

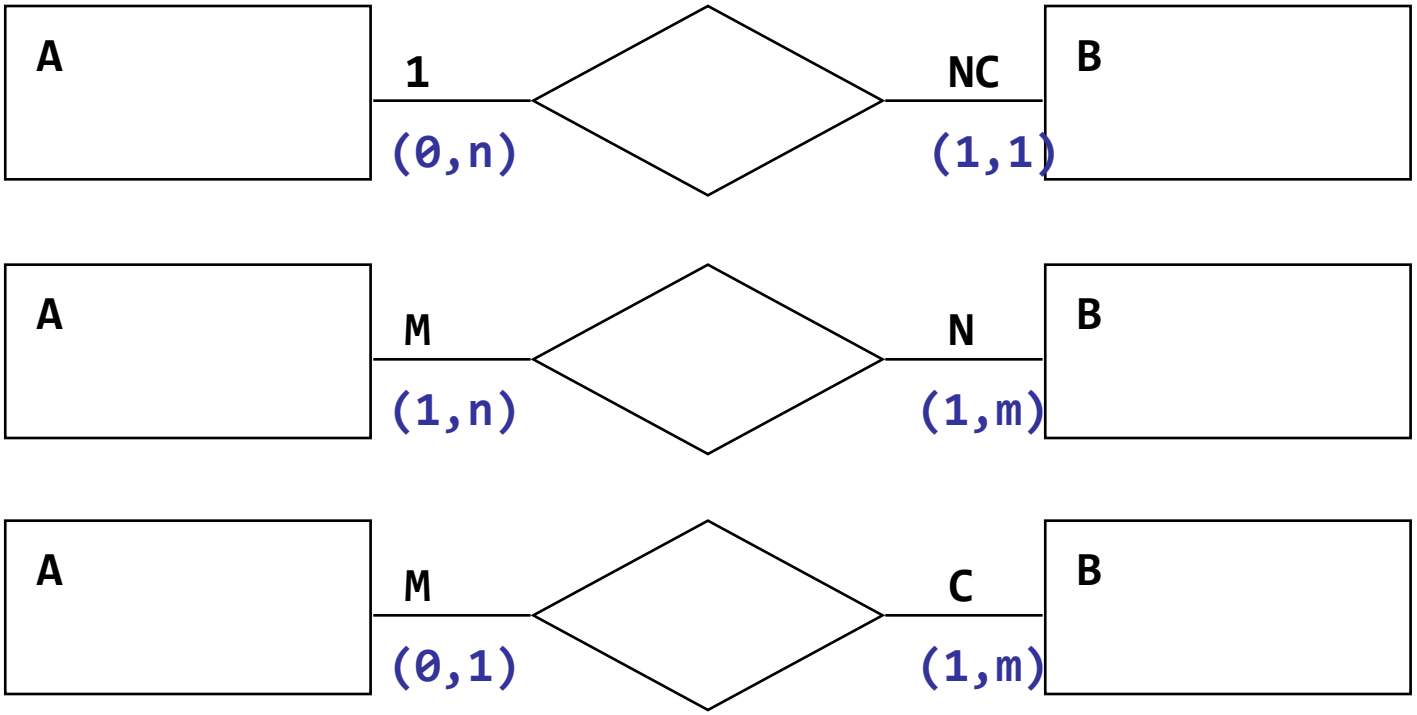
enthält Jeder Auftrag hat eine eindeutige bestellende Person, die viele Artikel bevorzugt, die aber nicht unbedingt in dem Auftrag enthalten sein müssen.

Verschiedene Notationen

NC-Notation	Numerische Notation	Krähenfuß-notation	Pfeil-notation	Bachman-notation
C	(0,1)			
1	(1,1)			
NC	(0,n)			
N	(1,n)			

aus: H. Balzert, Lehrbuch Grundlagen der Informatik, Spektrum Akademischer Verlag, 1999

Alternative Kardinalitätendarstellung



UML-Notation

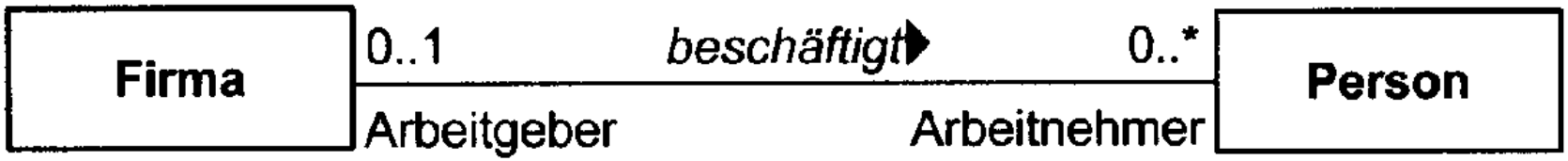
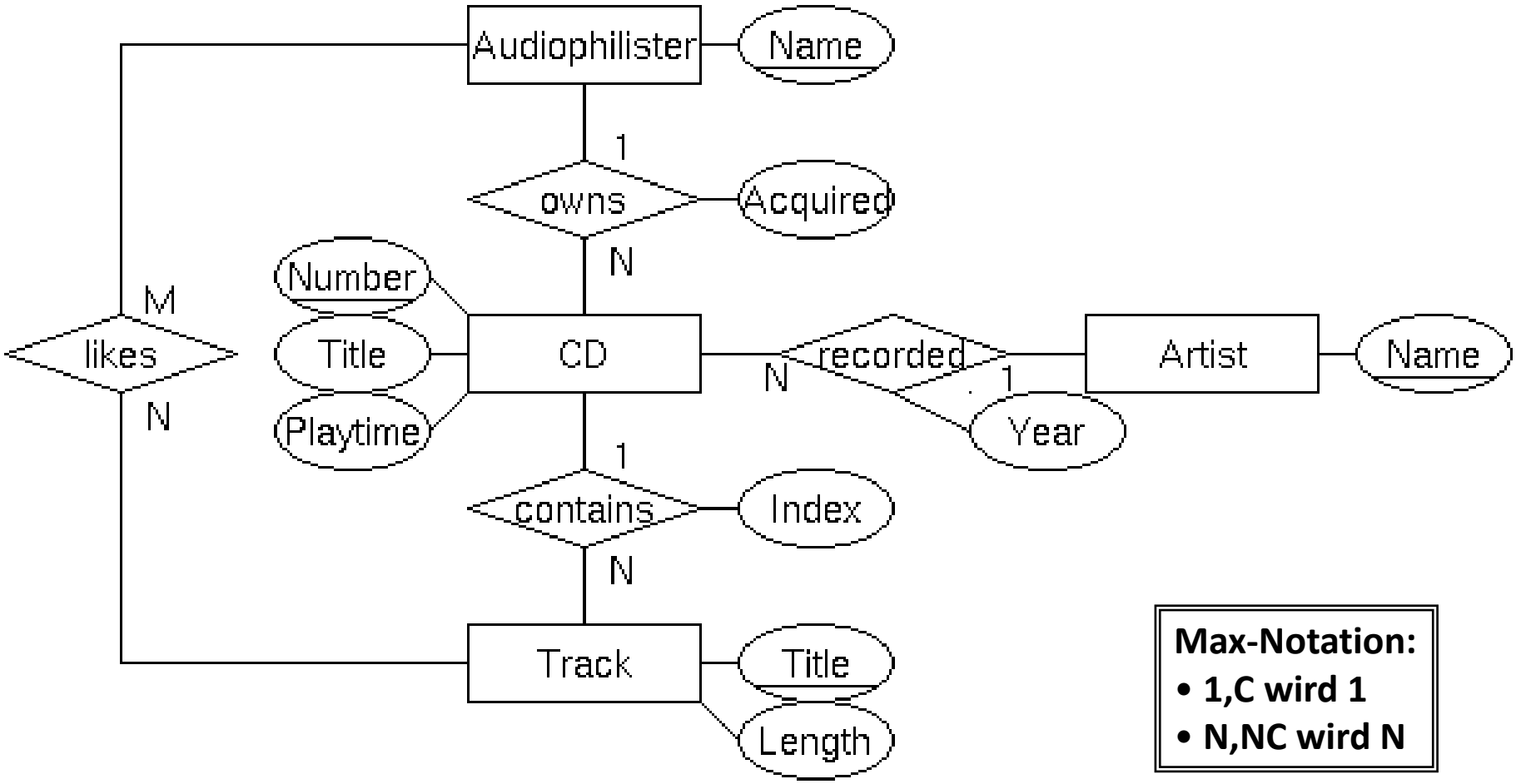


Abb. 2.17: Rollennamen in Assoziation.

Aus B. Oestereich, Objektorientierte Softwareentwicklung

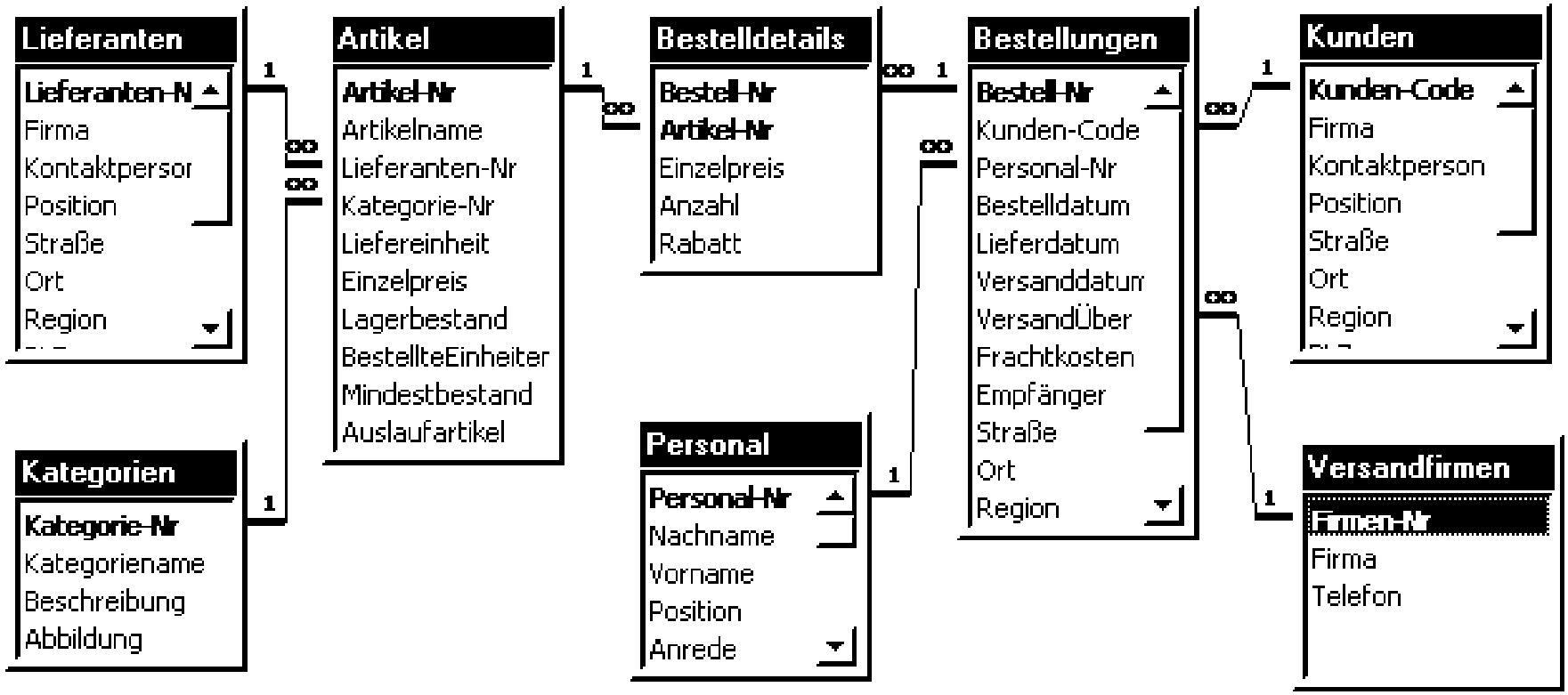
Hinweis: Oft werden nur Teile der Annotationen gezeigt

Lesebeispiel (CD-Sammler)



Max-Notation:
- **1,C wird 1**
- **N,NC wird N**

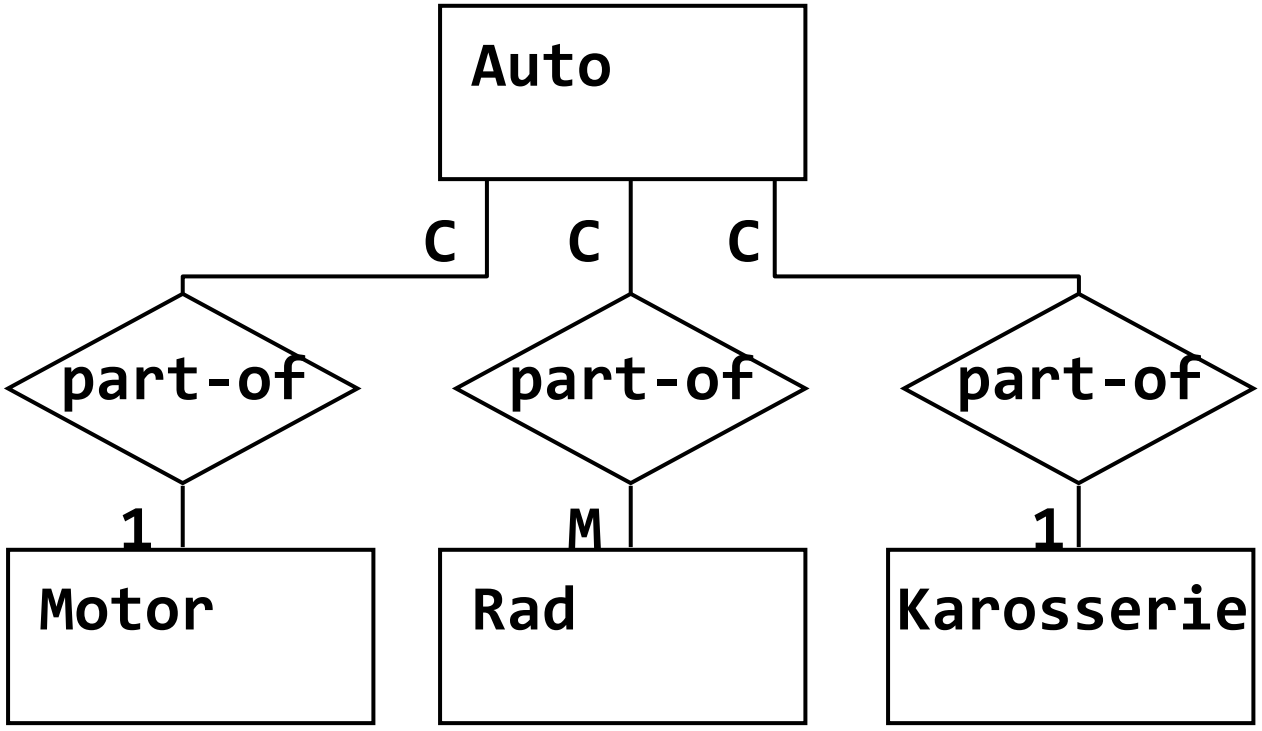
Lesebeispiel Access (Realisierungmodell !)



Spezielle Relationen (1/3)

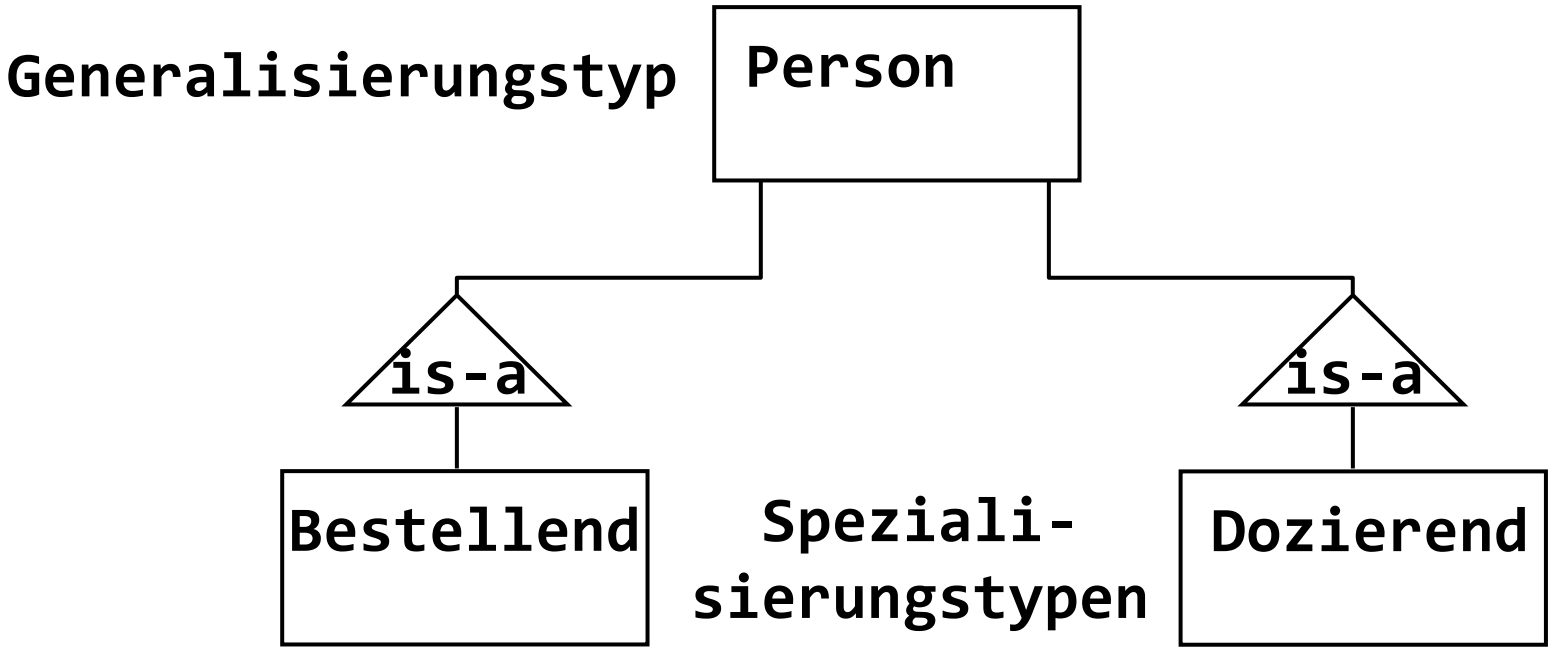
Video

- Aggregation
  - Relation, die Über-/Unterordnung beschreibt
  - meist „ist-Teil-von“ ("*is-part-of* ")



Spezielle Relationen (2/3)

- Generalisierung
  - Relation, die Hierarchie beschreibt
  - meist „ist-ein“ ("*is-a*")
  - Attribute werden vererbt
  - weitere Attribute kommen hinzu

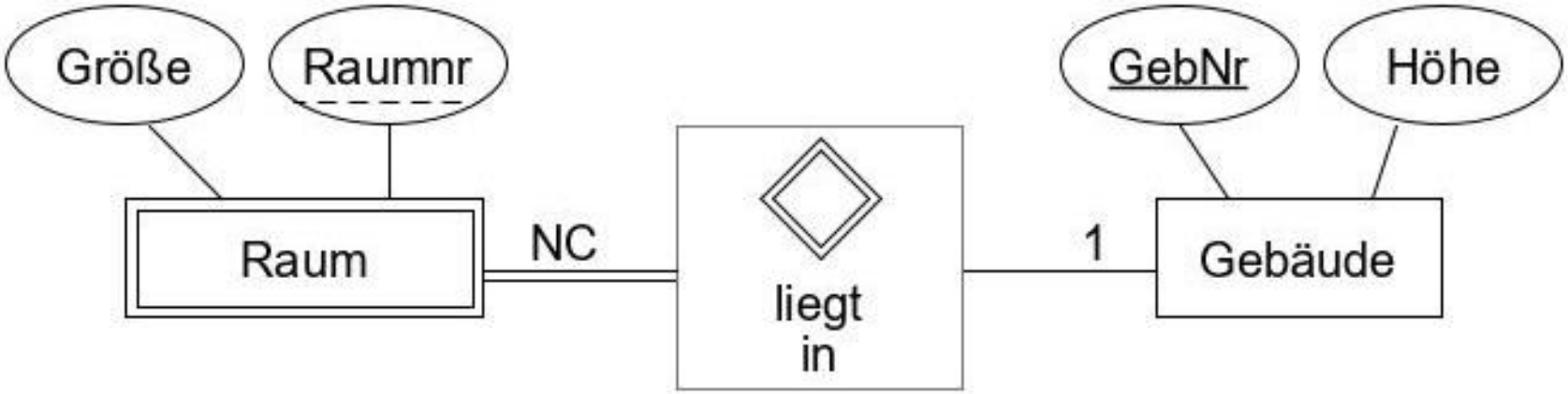


Spezielle Relationen (3/3) [wichtig]

- Schwache Abhängigkeit

  eine Entität kann ohne die Existenz einer anderen Entität nicht existieren (meist hat nur Entität eine doppelte Linie)

  (Darstellung einer Integritätsregel)



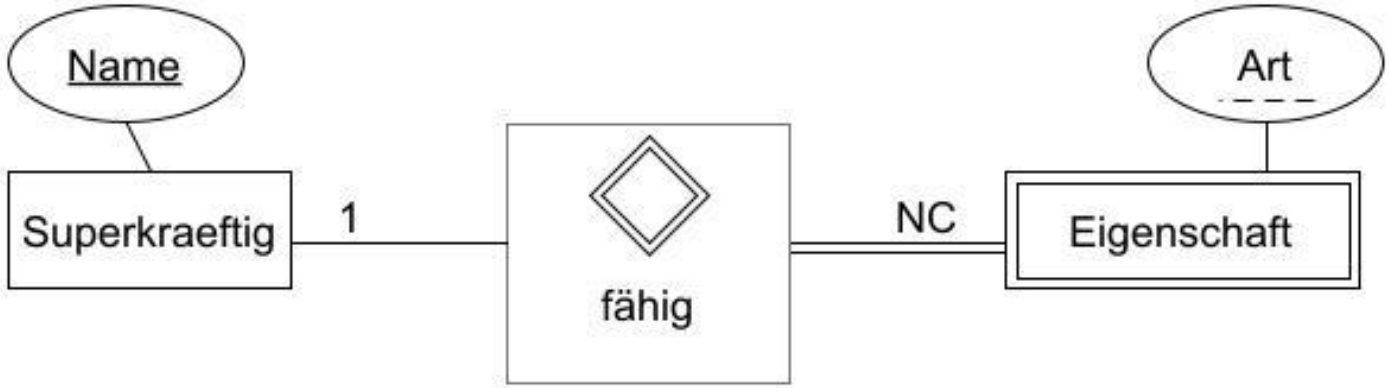
Jedes Gebäude beinhaltet beliebig viele Räume,

jeder Raum gehört zu einem Gebäude und kann ohne dieses nicht existieren,

ein Raum wird eindeutig durch GebNr und RaumNr identifiziert

Beispiel: Modellierung von Sammlungen (1/3)

- Eine Person mit Superkräften hat eine Menge von Eigenschaften
- Programmierung (eine Variante, alternativ Klasse Eigenschaft):
  `private Set<String> eigenschaften = new HashSet<>();`
- falls `List<>` genutzt: Programmierung achtet darauf, dass jeder Wert nur einmal in der Liste steht
- Modellierung



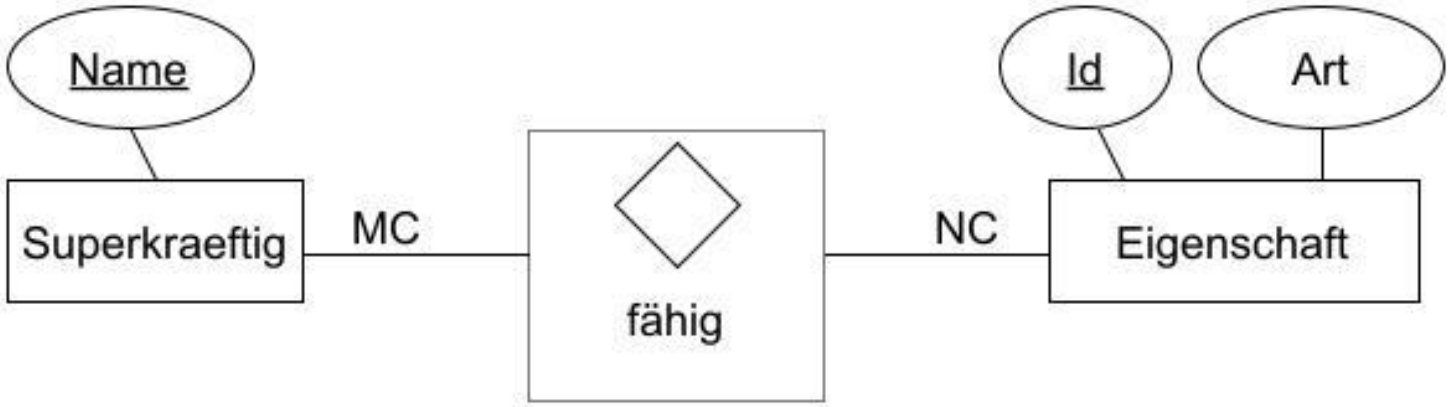
- wird eine Person mit Superkräften gelöscht, dann auch zugehörige Eigenschaften
- nur sinnvoll , wenn jede Eigenschaft einzigartig (hier eher nicht)

Beispiel: Modellierung von Sammlungen (2/3)

- Variante: Es ist wichtig feststellen zu können, dass mehrere Personen mit Superkräften gleiche Eigenschaften haben, dann

```
public enum Eigenschaft{FLIEGEN, STAERKE, ROENTGENBLICK};
private Set<Eigenschaft> eigenschaften = new HashSet<>();
```

- Modellierung:



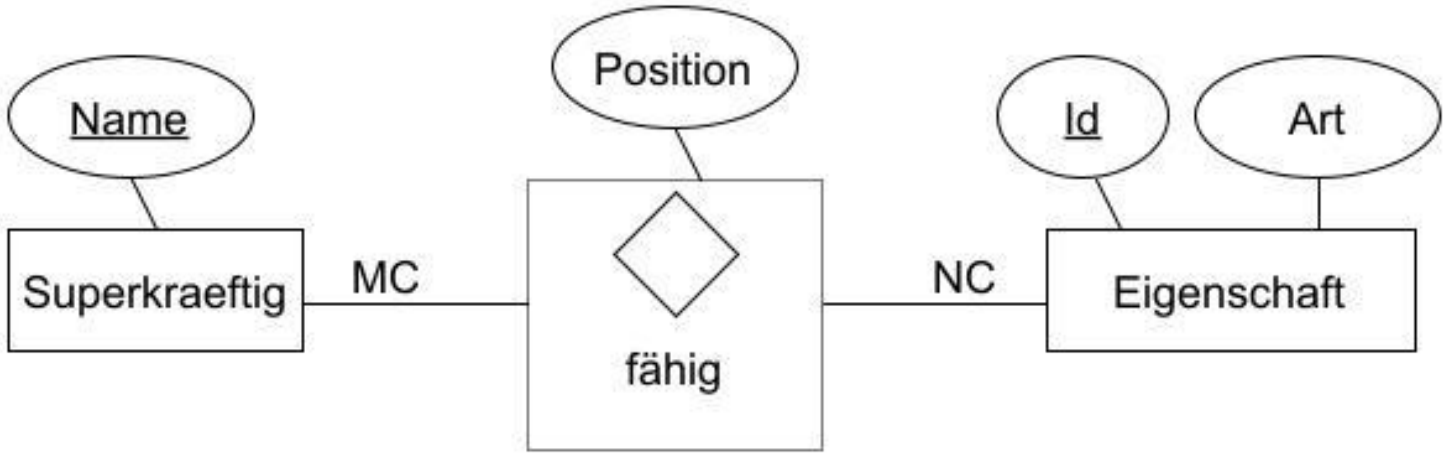
- wird eine Person mit Superkräften gelöscht, dann auch seine Beziehungen zu Eigenschaften

Beispiel: Modellierung von Sammlungen (3/3)

- bisher wurden keine Reihenfolgen berücksichtigt (Set)
- ist Reihenfolge wichtig, ist diese als Attribut zu definieren

```
private List<Eigenschaft> eigenschaften = new ArrayList<>();
```

- Modellierung:

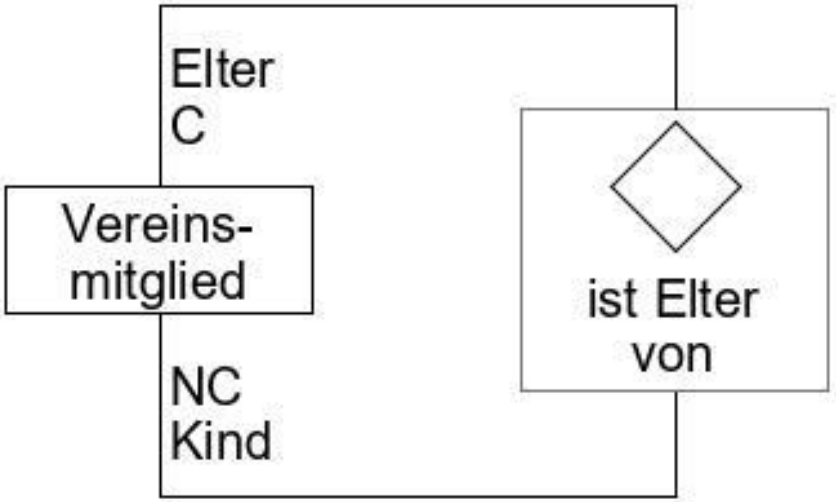


- weitere Entwicklung muss sicherstellen, dass Position eindeutig und lückenlos bleibt (-> weitere Randbedingungen als Dokumentation festhalten)

Rekursive Relation (1/2)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Entitätsmenge in Beziehung mit sich selbst



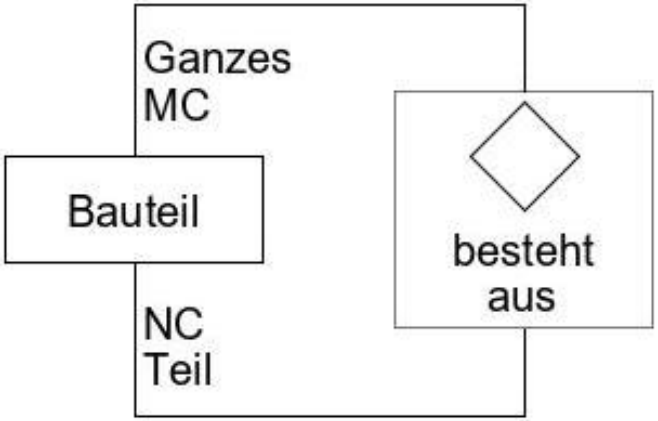
Jedes Vereinsmitglied ist Elter von 0 bis m Kindern im Verein. Für jedes Vereinsmitglied kann ein Elter im Verein sein oder nicht.

Diagramm wird um Rollennamen erweitert, kann auch sonst in ER-Diagrammen genutzt werden

Rekursive Relation (2/2)

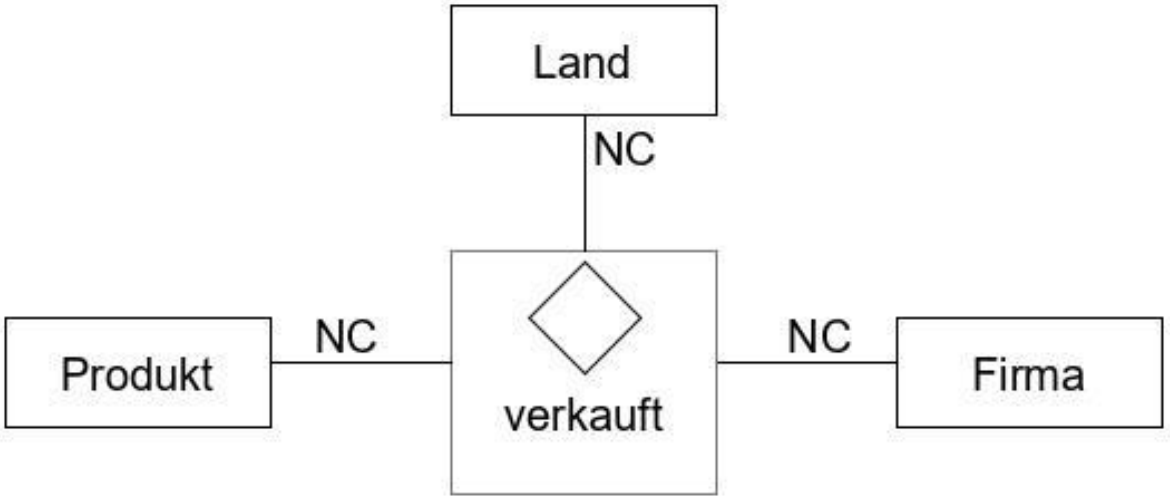
- rekursive Relationen sind teilweise notwendig (auch reflexiv genannt)
- Beispiel: ein Bauteil setzt sich aus anderen Bauteilen zusammen, die sich wiederum aus Bauteilen zusammen setzen, ...
- schlechte Modellierung, auch wenn Ebenenanzahl bekannt:

- korrekte Modellierung:



Beziehungen höheren Grades (1/2)

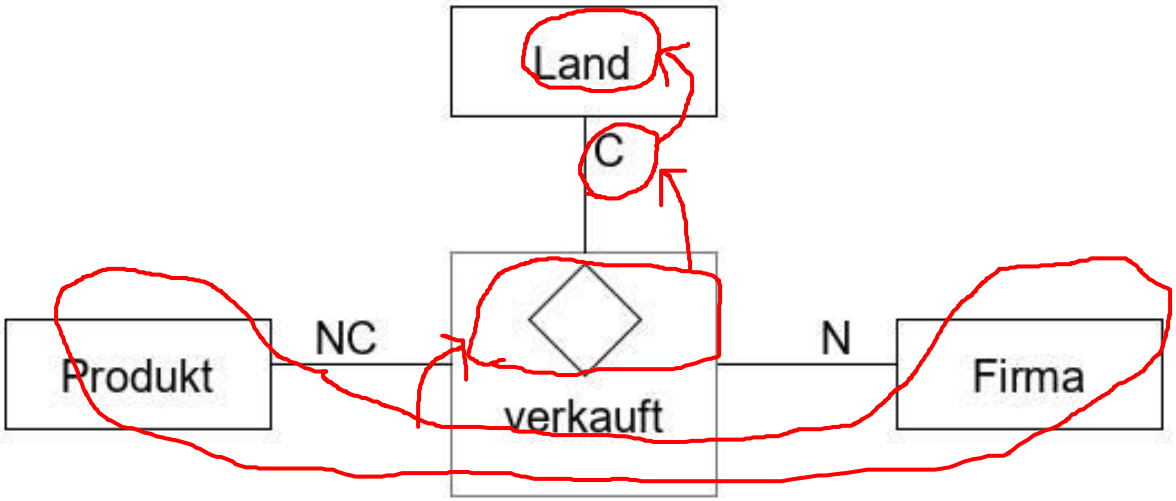
Video



Beispiel für einen ternären (Grad 3) Beziehungstyp:
Es sollen Informationen über Firmen, ihre Produkte und die Länder, in die sie die Produkte exportieren, gespeichert werden, dabei wird nicht jedes Produkt einer Firma in jedem Land verkauft

Hinweis: höhergradige Beziehungen sind eher selten, und müssen, *wenn möglich*, durch Beziehungen zweiten Grades beschrieben werden

Beziehungen höheren Grades (2/2)



gelesen:
Jede Kombination aus Land und Firma verkauft beliebig viele Produkte.
Jede Kombination aus Land und Produkt wird von einer oder mehreren Firmen verkauft (ausgeliefert).
Jede Kombination aus Firma und Produkt verkauft in maximal einem Land.

Bewertung des ER-Modell

- 👍 semantische Datenmodellierung ist Standard in kaufmännischen Anwendungen
- 👍 bereitet relationalen Datenbankentwurf weitgehend vor
- 👍 Unterstützung durch CASE-Werkzeuge
- 👎 bei umfangreichen Datenmodellen unübersichtlich, da keine Verfeinerung

Anmerkung: ER-Diagramme können als Spezialfall von Klassen-diagrammen gesehen werden, diese sind in der UML genormt

3. Tabellenableitung

Video

- Grundlage, was sind Relationen
- Transformation von ER-Diagrammen in Relationen

Relationen (Grundlagen)

Seien W_1, W_2, ..., W_n Wertebereiche (beliebige Mengen). Eine Relation R über einer Menge von Mengen ist definiert als Teilmenge des kartesischen Produkts (Kreuzprodukts) dieser Mengen:

$$R \subseteq W_1 \times W_2 \times \ldots \times W_n$$

n heißt Grad der Relation, man spricht von einer n-stelligen Relation oder einer Menge von n-Tupeln

Relationen lassen sich sehr anschaulich mit folgender Zuordnung als Tabellen interpretieren:

- W_i sind Spaltenüberschriften
- Tupel sind einzelne Zeilen der Tabelle
- Relationen sind Tabellen

Relationen veranschaulicht

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

R

A	B	C	D
xyz	2	blo	4.6
dfg	5	bli	2.4
⋮	⋮	⋮	⋮
ggg	7	bum	4.2

Beispiel zeigt Tabellendarstellung der Relation $R \subseteq A \times B \times C \times D$. Die Anzahl der Zeilen (Tupel) der Tabelle heißt Mächtigkeit der Relation, die Anzahl der Spalten ist der Grad der Relation

Aus der formalen Beschreibung einer Tabelle als Relation ergeben sich folgende Konsequenzen:

1. Alle Einträge einer Spalte sind vom selben Typ.
2. Alle Zeilen sind verschieden (Relationen sind Mengen).
3. Die Reihenfolge der Zeilen ist beliebig.
4. Die Bedeutung jeder Spalte wird durch einen Namen (dem Wertebereichsnamen) gekennzeichnet.

Transformation von ERM in Relationen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- ERM lassen sich leicht ohne Informationsverlust in Relationen abbilden

- Die Ansätze für die verschiedenen Beziehungen
  - 1:1
  - 1:N
  - M:N

  unterscheiden sich

- Wichtig ist der Umgang mit Schlüsseln (identifizierenden Attributen) und *Fremdschlüsseln*
- *Begriff des Schlüssels wird später formalisiert*

Ziele der Transformation

1. Bei der Füllung der Tabellen mit Daten sollen redundante Daten vermieden werden.

2. Wenn es nicht aus praktischer Sicht notwendig ist, soll keine Notwendigkeit zur Nutzung von NULL-Werten (leeren Tabelleneinträgen) entstehen.

3. Es soll unter Berücksichtigung von 1. und 2. eine möglichst minimale Anzahl von Tabellen entstehen.

Transformationen von Entitätsmengen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- **Entitätsmenge wird Tabelle**
- **Attribute werden Spalten**
- **Einzelne Entitäten entsprechen Zeilen bzw. Datensätzen**

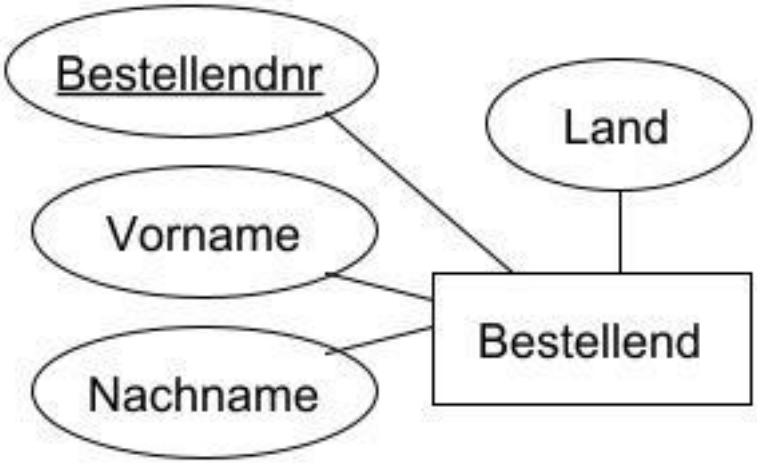
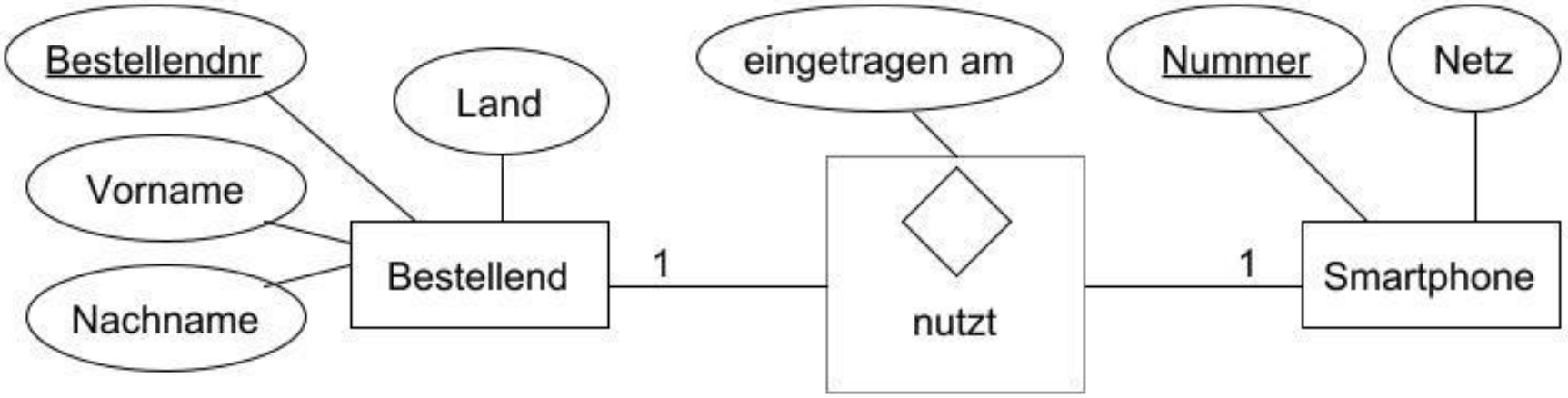


Tabelle *Bestellend*

Bestellendnr	**Vorname**	**Nachname**	**Land**
0001	**Max**	**Meier**	**AUT**
0002	**Nina**	**Petrova**	**D**

Transformation von 1:1-Beziehungen



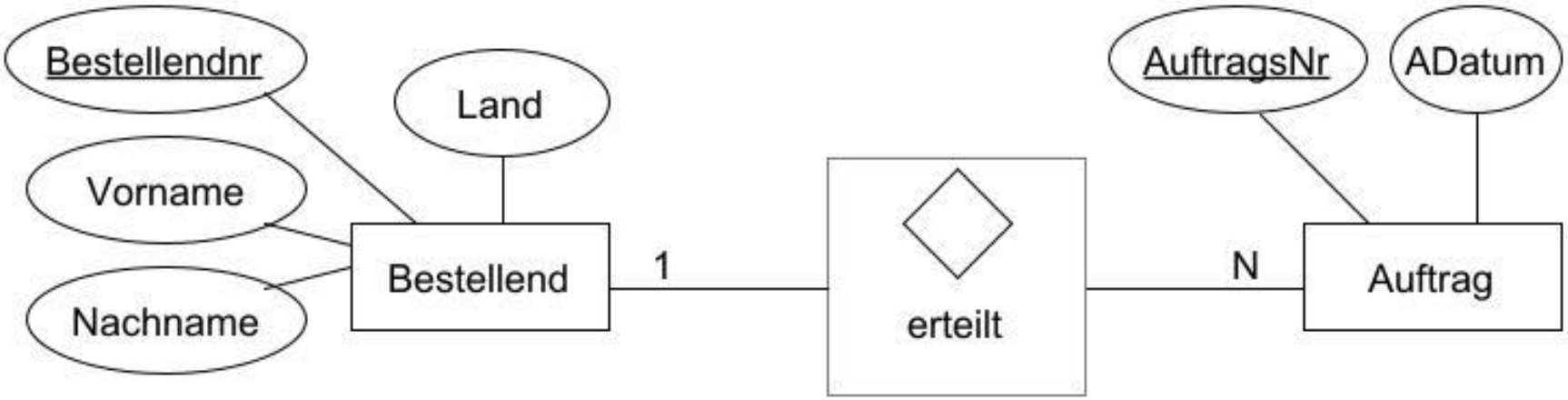
Die Informationen werden in einer Tabelle zusammengefasst:

Tabelle *Bestellend*

Bestellendnr	Vorname	Nachname	Land	eingetragen	Nummer	Netz
0001	Max	Stein	AUT	5.6.2019	0152052	D1
0002	Nina	Pedrova	D	7.6.2019	0166243	E
...						

Alternative: Zwei Tabellen, Schlüssel einer Tabelle wird als Fremdschlüssel in anderer Tabelle eingetragen

Transformation von 1:N-Beziehungen



2 Tabellen sind notwendig:

- **Tabelle *Bestellend***
- **Tabelle *Auftrag:***
  enthält Primärschlüssel der übergeordneten Tabelle (entspricht Objekt mit Beziehung "1"), der als "*Fremdschlüssel*" bezeichnet wird (und Attribute der Assoziation)

Transformation von 1:N-Beziehungen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Tabelle *Bestellend*

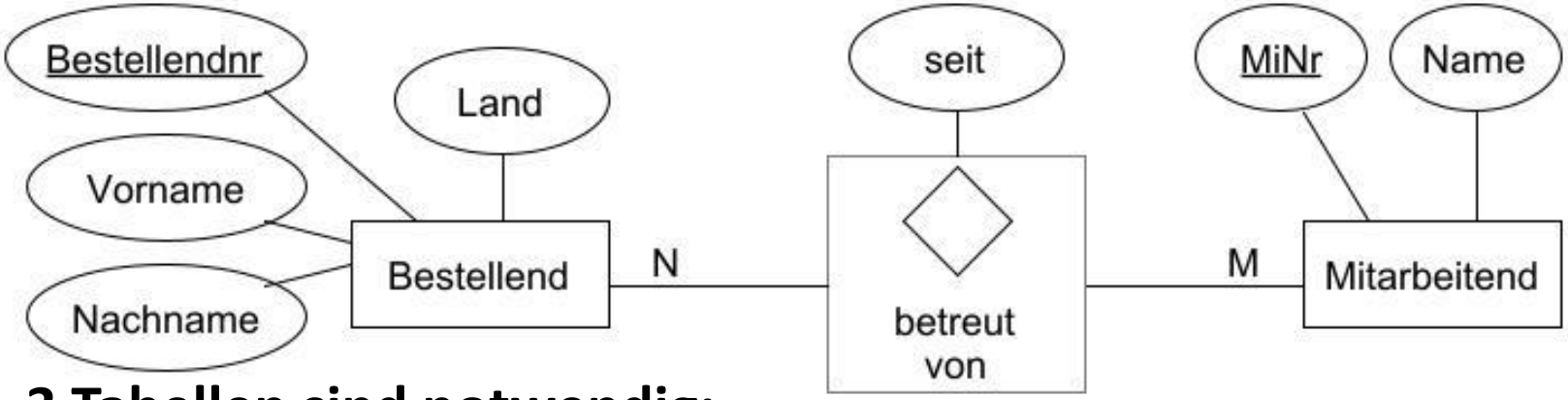
Bestellendnr	Vorname	...
0001	Max	
0002	Nina	
...		

Tabelle *Auftrag*

AuftragNr	Bestellendnr	ADatum	...
00000001	0001	2.1.2022	
00000002	0001	3.1.2022	
00000003	0002	3.1.2022	
00000004	0007	5.1.2022	
...			

BestellendNr wird zum Fremdschlüssel in Auftrag

Transformation von M:N-Beziehungen



3 Tabellen sind notwendig:

- **Tabelle *Mitarbeitend***
- **Tabelle *Bestellend***
- **Beziehungstabelle *Bestellendbetreuung*: enthält Primärschlüssel der beiden Ausgangstabellen → Fremdschlüssel (und Attribute der Assoziation)**
- **Primärschlüssel dieser Tabelle kann, muss aber nicht aus den beiden Fremdschlüsseln zusammengesetzt sein**

Transformation von M:N-Beziehungen (Koppeltabelle)

Tabelle *Mitarbeitend*

MiNr	Name
01	Achmad
02	Herbert
...	

Tabelle *Bestellend*

Bestellendnr	Vorname	...
0001	Max	
0002	Nina	
...		

Tabelle *Bestellendbetreuung*

MiNr	Bestellendnr	seit
02	0001	1.10.2019
01	0002	5.10.2019
02	0002	18.10.2019
23	0963	23.10.2019
...		

zusammengesetzter Schlüssel

N:M – Beziehungen werden somit in zwei 1:N – Beziehungen zerlegt:

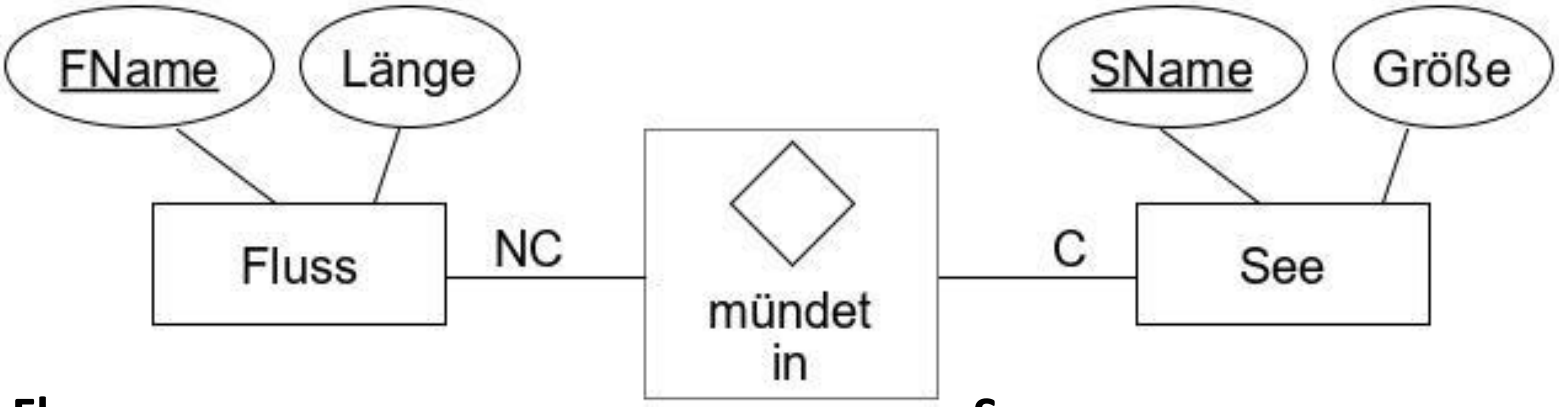
1. **Tabelle Mitarbeitend – Tabelle Bestellendbetreuung**
2. **Tabelle Bestellend – Tabelle Bestellendbetreuung**

Umgang „c“-Kardinalitäten

Video

- Liegt eine kann-Beziehung vor, gibt es verschiedene Ansätze, dies in Relationen auszudrücken:
  - technisch aufwändig (und unüblich) für die Umsetzung von C ist die Aufspaltung der Relation für Elemente, die eine Beziehung haben und Elemente, die (noch) keine Beziehung haben
  - wird C als 1 interpretiert, lässt man NULL-Werte (leere Tabelleneinträge) zu, wobei NULL-Werte so lange wie möglich in der DB-Entwicklung vermieden werden sollen
  - *alternativ* ist die Interpretation von NC als N und von C als NC (und damit als N), da man dann die vorgestellten Übersetzungsschritte nutzen kann (und NULL-Einträge werden vermieden)
- letzten beiden Wege werden genutzt (weniger Tabellen oder keine NULL-werte)

Beispiel: Transformation von C-Kardinalitäten



Fluss

FName	Laenge	SName

See

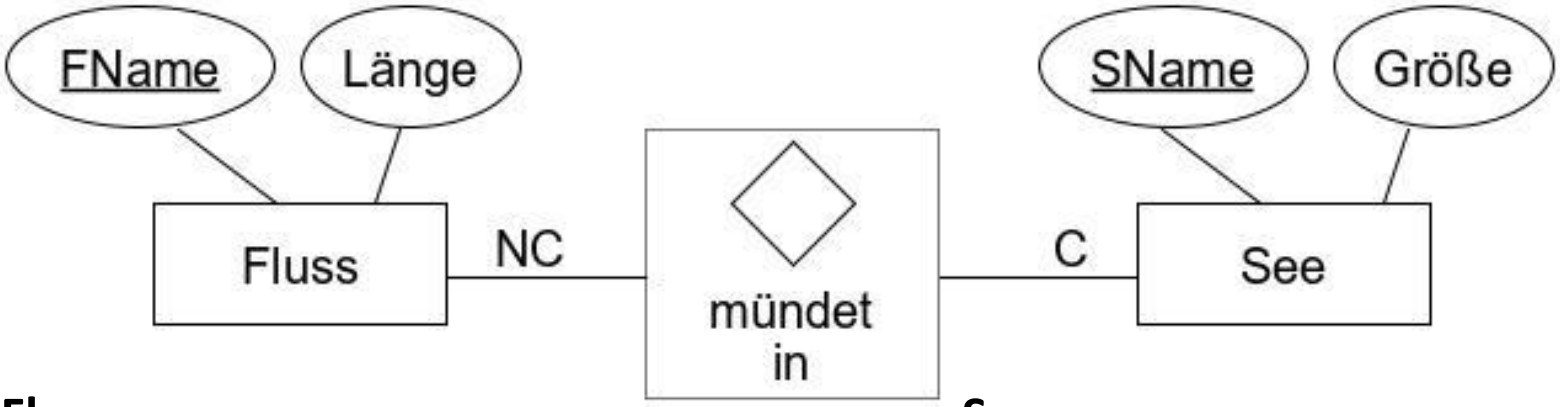
SName	Groesse	FName

Muendung

FName	SName

Zentraler Ansatz: leere Einträge solange wie möglich vermeiden
Achtung: kein zusammengesetzter Schlüssel

Variante: Transformation von C-Kardinalitäten



Fluss

FName	Laenge	SName

See

SName	Groesse

- **Variante: Interpretation als 1:N**
- **in Literatur häufiger genutzt!**
- **zentraler Vorteil: weniger Tabellen**
- **Nachteil: leere Felder, d. h. NULL-Einträge mit Fehlerpotenzial**
- **in dieser VL/Klausur nur Variante mit NULL-Vermeidung relevant!**

Transformation nichtbinärer Relationstypen

Übersetzungsbeispiel

Video

- Beziehungstypen, die mehr als zwei Entitätstypen miteinander in Beziehung setzen, werden in einer eigenen Relation abgebildet. Die Relation erhält als Fremdschlüsselattribute die Schlüssel der Entitätstypen, die dadurch verbunden werden und deren Kardinalität N oder NC ist
- Existiert eine Kardinalität, deren Maximalwert 1 ist, ist der Schlüssel des an der Kardinalität stehenden Entitätstyps nicht Teil des Schlüssels der neuen Tabelle
- Hinweis: Sinnvolle Alternative ist es, die n-Stellige Relation durch einen neuen Entitätstypen zu ersetzen, der dann n zweistellige Relationen mit den zugehörigen Entitätstypen hat.

Beispieltransformation nichtbinärer Typ

Firma

FID	FName

Land

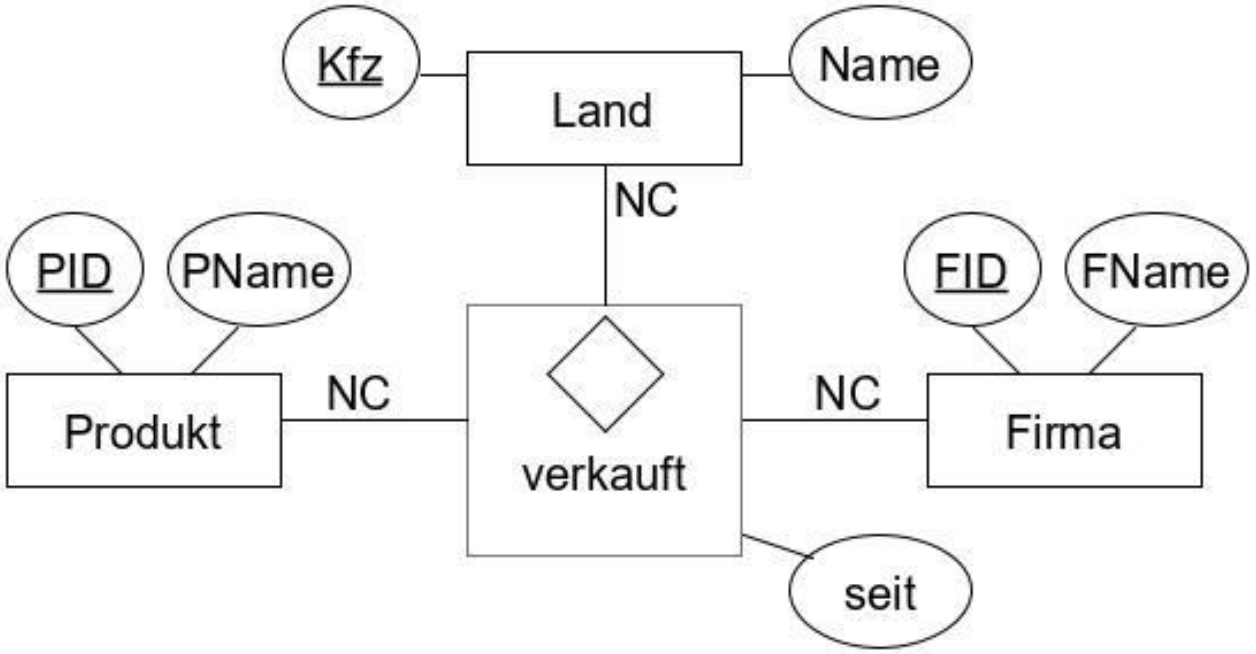
Kfz	Name

Produkt

PID	PName

Verkauf

FID	Kfz	PID	seit



bevorzugte Variante als binärer Typ

Firma

FID	FName

Land

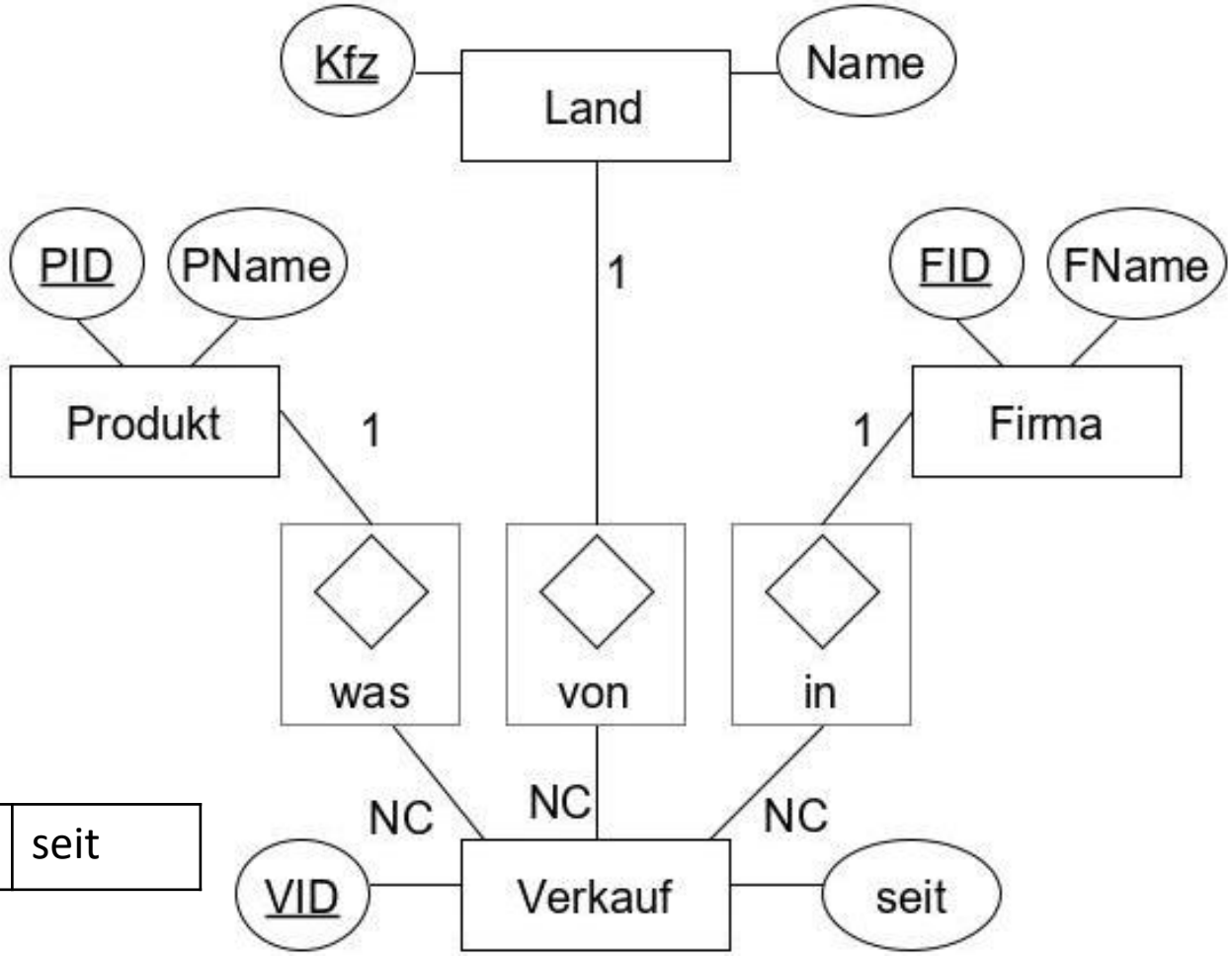
Kfz	Name

Produkt

PID	PName

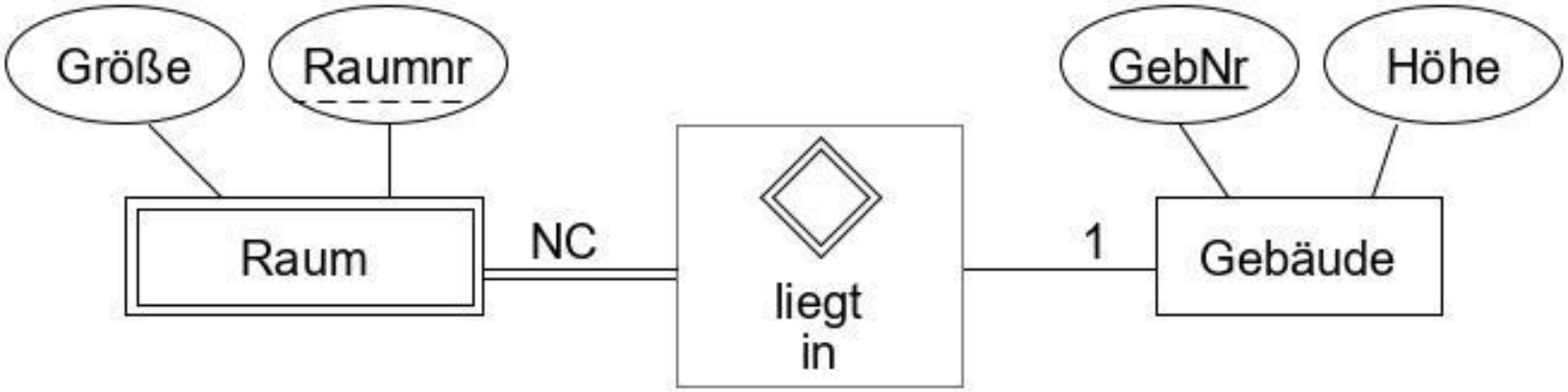
Verkauf

VID	FID	Kfz	PID	seit



Transformation schwacher Abhängigkeiten

- Bei schwachen Abhängigkeiten kann es notwendig sein, den Schlüssel des existenzbestimmenden Entitätstypen beim abhängigen Entitätstypen zu ergänzen



Gebäude

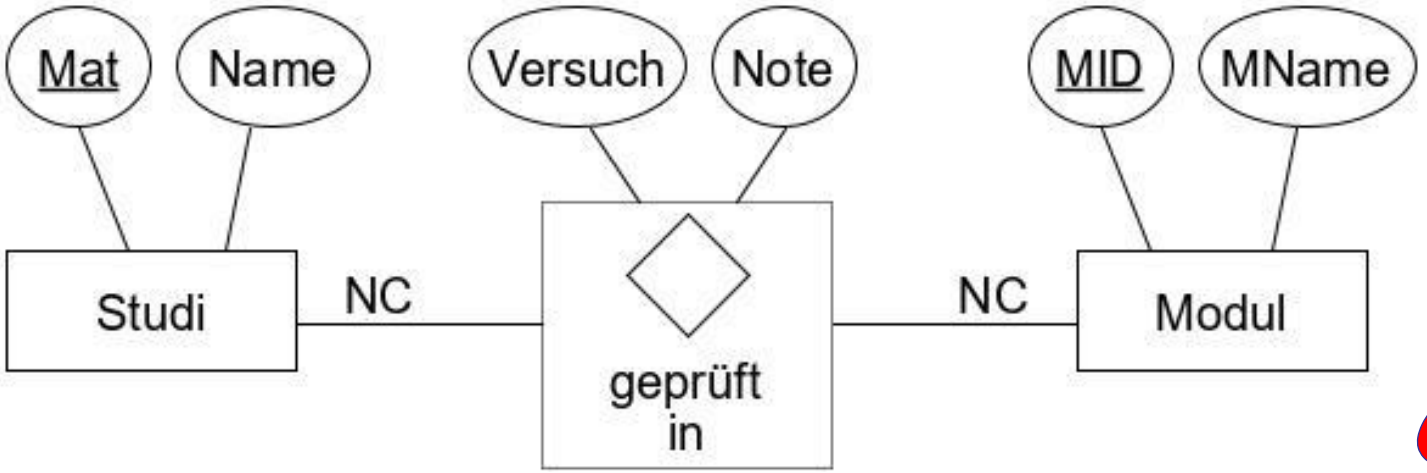
GebNr	Hoehe
A	17
B	12

Raum

RaumNr	GebNr	Groesse
17	A	42
18	A	34
17	B	32

Transformationen zeigen Modellierungsfehler (1/2)

- Anforderung: Studierende Person macht Prüfungen in einem Modul und erhält eine Note, es gibt maximal drei Versuche



falsch !

Studi

Mat	Name
42	Ute
43	Uwe

Geprueft

Mat	MID	Versuch	Note
43	101	1	5
43	101	2	3.7

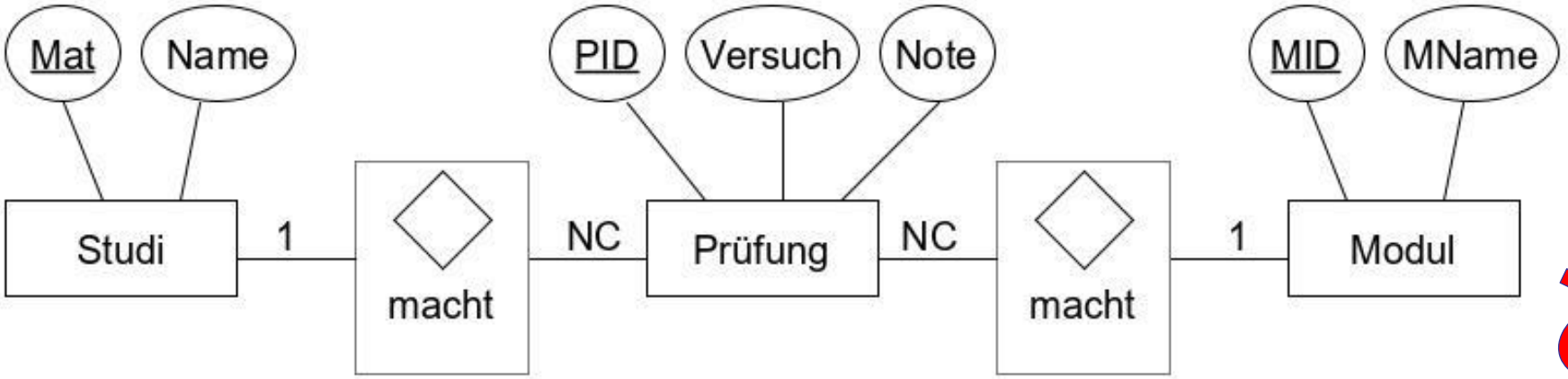
Modul

MID	MName
100	Prog1
101	DB

(Mat,MID) ist kein Schlüsel

Transformationen zeigen Modellierungsfehler (2/2)

- Anforderung: Student macht Prüfungen in einem Modul und erhält eine Note, es können maximal drei Versuche sein



Studi

Mat	Name
42	Ute
43	Uwe

Pruefung

PID	Mat	MID	Versuch	Note
731	43	101	1	5
992	43	101	2	3.7

Modul

MID	MName
100	Prog1
101	DB

Zusammenfassung und Alternativen der Übersetzung

Beziehung	Übersetzung	Alternative
1:1	Tabellen zusammenfassen	getrennte Tabellen, Fremdschlüssel auf einer Seite (wählbar)
1:N, N:1	Fremdschlüssel bei N	
1:NC, NC:1	Fremdschlüssel bei NC	
1:C, C:1	Fremdschlüssel bei C	
C:N, N:C	Koppeltabelle	Fremdschlüssel bei N, NULL-Werte möglich
C:NC, NC:C	Koppeltabelle	Fremdschlüssel bei N, NULL-Werte möglich
C:C	Koppeltabelle	Fremdschlüssel auf einer Seite
N:M	Koppeltabelle	
N:MC, M:NC	Koppeltabelle	
NC:MC	Koppeltabelle	

- Details: H. Jarosch, Grundkurs Datenbankentwurf, 3. Auflage, Vieweg + Teubner, Wiesbaden, 2010

4. Normalformen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- Qualitätsanforderungen an Tabellen
- Klassische Normalformen (1., 2., 3.)
- Spezielle Normalformen

Normalisierungsgründe

- Verständlicheres Datenmodell für Anwender und Entwickler
- Vermeidung von Anomalien beim Einfügen, Löschen oder Ändern von Daten
- Eliminierung von Redundanzen
- Robusteres Datenmodell gegenüber Änderungen oder Erweiterungen
- Korrekte Abbildung der Realität
- man kann sich mal so richtig schön systematisch mit den Daten beschäftigen...

1. Normalform (1NF)

Eine Tabelle ist in der ersten Normalform (1NF)

- wenn alle Attribute nur atomare Werte beinhalten

  mit anderen Worten, pro Attribut/Zeile (also „in jedem Kästchen“) gibt es nur einen Wert

Tabellen in 1NF werden auch als *flache Relationen* bezeichnet

Definition *erste Normalform*: Eine Tabelle ist in erster Normalform, wenn zu jedem Attribut ein für Spalten zugelassener einfacher Datentyp gehört.

Normalisierung in erste Normalform

nicht 1NF →

Bestellzettel				
BestellNr		17		
Name		Meier		
ProdNr	PName	Farbe	Anzahl	EPreis
42	Schraube	weiß	30	1,98
		blau	40	
45	Dübel	weiß	30	2,49
		blau	40	

umgeformt in 1NF ↓

BestellNr	Name	ProdNr	PName	Farbe	Anzahl	EPreis
17	Meier	42	Schraube	weiß	30	1,98
17	Meier	42	Schraube	blau	40	1,98
17	Meier	45	Dübel	weiß	30	2,49
17	Meier	45	Dübel	blau	40	2,49

Einschub: Beispiel für Anomalien

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

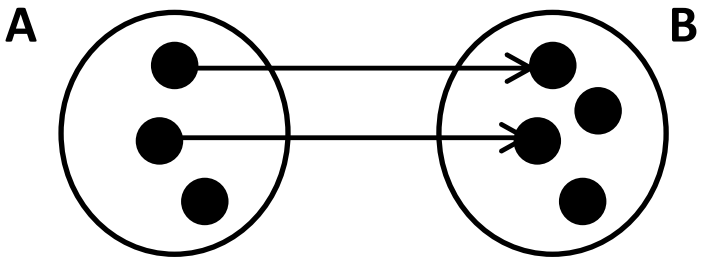
Projektmitarbeit

MiNr	Name	AbtNr	Abteilung	ProNr	Projekt
1	Egon	42	DB	1	Infra
1	Egon	42	DB	2	Portal
2	Erna	42	DB	2	Portal
2	Erna	42	DB	3	Frame
3	Uwe	43	GUI	1	Infra
3	Uwe	43	GUI	3	Frame

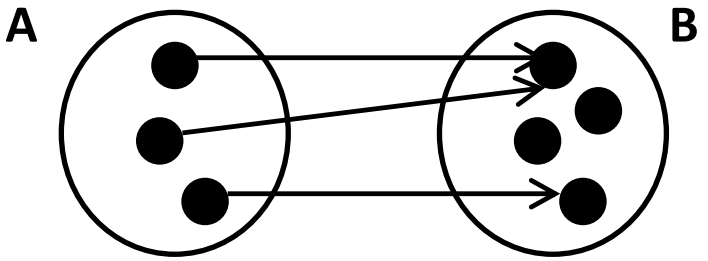
Probleme bei UPDATE, DELETE, INSERT

Wiederholung: mathematische Grundbegriffe

- *Funktion* bildet Elemente der Menge A nach Elemente der Menge B ab, mathematisch f: A → B



partielle Funktion



totale Funktion

- Für zwei Mengen A und B beschreibt das *Kreuzprodukt* (kartesische Produkt) die Menge, die alle Kombinationen aus A und B enthält, mathematisch $A \times B$

  A = {Ute,Uwe}, B = {41,42,43}

  $A \times B$ = {(Ute,41), (Ute,42), (Ute,43),
  (Uwe, 41), (Uwe 42), (Uwe 43)}

Funktionale Abhängigkeit

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

Definition *funktionale Abhängigkeit*: Gegeben sei eine Tabelle. Eine Menge von Attributen B der Tabelle ist funktional abhängig von einer Menge von Attributen A der Tabelle, wenn es zu jeder konkreten Belegung der Attribute aus A nur maximal eine konkrete Belegung der Attribute aus B geben kann. Für funktionale Abhängigkeiten wird die Schreibweise

$A \rightarrow B$ genutzt.

„Immer wenn ich die Werte bestimmter Spalten kenne, weiß ich genau, was in den anderen Spalten steht“

z. B. „Wenn ich die Matrikelnummer kenne, kann ich den Namen, Wohnort, etc. eindeutig bestimmen“

Definition *Belegung von Attributen*: Gegeben seien die Attribute A_1, ..., A_n einer Tabelle T. Alle Elemente aus $A_1 \times ... \times A_n$ werden mögliche Belegungen der Attribute genannt. In der Tabelle T sind die für diese Tabelle geltenden Belegungen aufgeführt.

Beispiel: Funktionale Abhängigkeit

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Projektmitarbeit

MiNr	Name	AbtNr	Abteilung	ProNr	Projekt
1	Egon	42	DB	1	Infra
1	Egon	42	DB	2	Portal
2	Erna	42	DB	2	Portal
2	Erna	42	DB	3	Frame
3	Uwe	43	GUI	1	Infra
3	Uwe	43	GUI	3	Frame

{MiNr} → {Name}

{AbtNr} → {Abteilung}

{ProNr} → {Projekt}

{Name} → {Name}

{MiNr,Abteilung} → {Name}

{MiNr} → {AbtNr,Abteilung}

Wichtig: Funktionale Abhängigkeiten werden nicht aus den Beispieleinträgen, sondern aus formulierten Randbedingungen („Jede mitarbeitende Person hat eindeutige Minr“) abgeleitet

Beispiel: Keine Funktionale Abhängigkeit

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Projektmitarbeit

MiNr	Name	AbtNr	Abteilung	ProNr	Projekt
1	Egon	42	DB	1	Infra
1	Egon	42	DB	2	Portal
2	Erna	42	DB	2	Portal
2	Erna	42	DB	3	Frame
3	Uwe	43	GUI	1	Infra
3	Uwe	43	GUI	3	Frame

Wichtig: einige Verstöße gegen die funktionale Abhängigkeit sind bereits aus Beispieldaten ablesbar

`{Name}` → `{ProNr}` gilt nicht

zu Egon gibt es zwei verschiedene zugeordnete ProNr

Funktionale Abhängigkeit - Rechenregeln

Seien A,B,C,D Mengen von Attributen, es gilt:

- es gilt immer $A \cup B \rightarrow A$
- aus $A \rightarrow B \cup C$ folgt $A \rightarrow B$ und $A \rightarrow C$
- aus $A \rightarrow B$ folgt $A \cup C \rightarrow B \cup C$
- aus $A \rightarrow B$ und $B \rightarrow C$ folgt $A \rightarrow C$
- aus $A \rightarrow B$ und $A \rightarrow C$ folgt $A \rightarrow B \cup C$
- aus $A \rightarrow B$ und $B \cup C \rightarrow D$ folgt $A \cup C \rightarrow D$

Volle Funktionale Abhängigkeit

Definition *volle funktionale Abhängigkeit*: Gegeben sei eine Tabelle. Eine Menge von Attributen B der Tabelle ist voll funktional abhängig von einer Menge von Attributen A der Tabelle, wenn A → B gilt und für jede echte Teilmenge A' von A nicht A' → B gilt.

(anschaulich: man kann auf der linken Seite kein Element entfernen)

{MiNr} → {Name}

{AbtNr} → {Abteilung}

{ProNr} → {Projekt}

{MiNr,ProNr} → {Name,Projekt}

Schlüssel, Schlüsselkandidat

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

Definition *Schlüssel*: Gegeben sei eine Tabelle und eine Menge M, die alle Attribute der Tabelle enthält. Gegeben sei weiterhin eine Attributsmenge A der Tabelle. Wenn A → M gilt, dann heißt A Schlüssel der Tabelle.

Definition *Schlüsselkandidat*: Gegeben sei eine Tabelle und eine Menge M, die alle Attribute der Tabelle enthält. Gegeben sei weiterhin eine Attributsmenge A der Tabelle. Wenn dann A → M gilt und eine volle funktionale Abhängigkeit ist, dann heißt A Schlüsselkandidat der Tabelle.

Primärschlüssel

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Definition *Schlüsselattribute und Nichtschlüsselattribute*:
Gegeben sei eine Tabelle. Die Menge der Schlüsselattribute der Tabelle enthält alle Attribute, die in mindestens einem Schlüsselkandidaten der Tabelle vorkommen. Die Menge der Nichtschlüsselattribute der Tabelle enthält alle Attribute, die in keinem Schlüsselkandidaten vorkommen.

Definition *Primärschlüssel*: Ein Primärschlüssel ist ein willkürlich ausgewählter Schlüsselkandidat einer Tabelle.

Anmerkungen zu Primärschlüsseln

Schlüsselkandidat gewählt, mit folgenden Eigenschaften:

- Belegungen der Attribute werden sich im Verlaufe der Tabellennutzung nicht oder zumindest nur selten ändern
- Datentypen der Attribute verbrauchen wenig Speicherplatz und können deshalb schnell gefunden werden, besonders geeignet sind Attribute mit ganzzahligen Werten, Attribute mit kurzen Texten sind auch geeignet
- Primärschlüssel soll möglichst wenige Attribute enthalten

Beispiel: Tabelle Freundschaft

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- Bestellende Personen haben einen Namen und sind eindeutig durch eine Nummer
- Alle Freundschaften sind eindeutig durch eine laufende Nummer
- Jede Freundschaft verbindet zwei Bestellende. dabei wird eine Freundschaft durch die Bestellendnummer und den Namen angegeben
- Die Tabelle hat zwei Schlüsselkandidaten

Nr	BestellendNr	Bestellendname	FreundNr	Freundname
1	42	Ute	43	Ugur
2	43	Ugur	44	Ulla
3	44	Ulla	42	Ute
4	42	Ute	45	Anna
5	43	Ugur	46	Ulf

2. Normalform (2NF)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Definition *zweite Normalform*: Sei eine Tabelle in erster Normalform. Dann ist diese Tabelle in zweiter Normalform, wenn jede nicht leere Teilmenge der Nichtschlüsselattribute von jedem Schlüsselkandidaten voll funktional abhängig ist.

- Tabellen mit nur einelementigen Schlüsselkandidaten sind immer in 2NF

- Intuitiv: die 2NF wird verletzt, wenn in einer Tabelle mit einem mehrelementigen Schlüsselkandidaten mehr als ein Konzept verwaltet wird

Beispiel zu 2NF

- Projektmitarbeit, jede mitarbeitende Person (eindeutig über Minr) hat in jedem Projekt (eindeutig über ProNr) maximal eine Aufgabe (nicht in 2NF)

ProNr	PName	MiNr	Name	Aufgabe	Werkzeug
42	DBX	1	Ulla	Analyse	Word
42	DBX	2	Ivan	ER-Modell	ERWin
43	Gui	1	Ulla	Analyse	Word
43	Gui	13	Joe	ER-Modell	ERWin

Projektmitarbeit, Projekt, Mitarbeit (in 2NF)

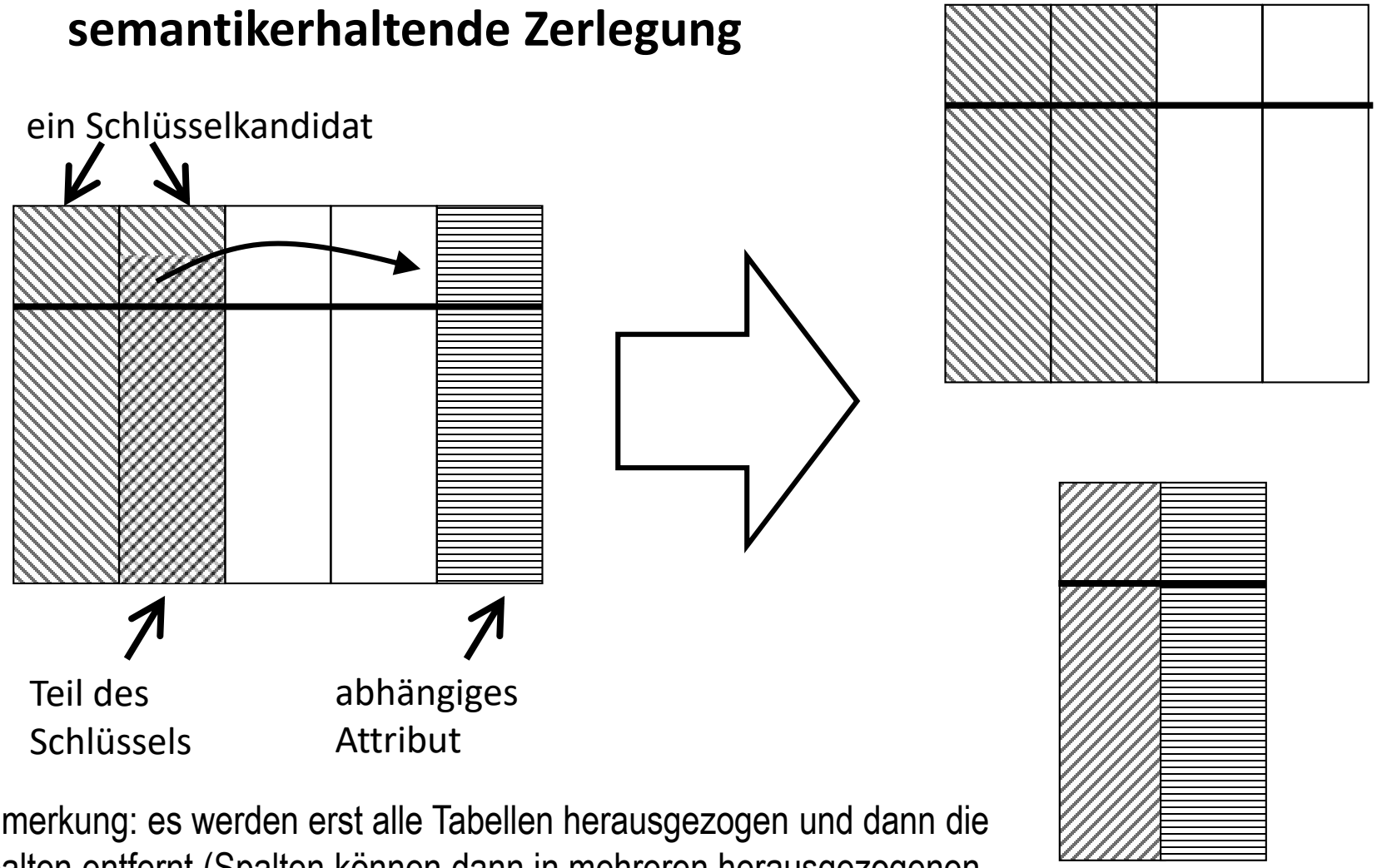
ProNr	MiNr	Aufgabe	Werkzeug
42	1	Analyse	Word
42	2	ER-Modell	ERWin
43	1	Analyse	Word
43	13	ER-Modell	ERWin

ProNr	PName
42	DBX
43	Gui

MiNr	Name
1	Ulla
2	Ivan
13	Joe

Auflösung partieller Abhängigkeiten

semantikerhaltende Zerlegung



Anmerkung: es werden erst alle Tabellen herausgezogen und dann die Spalten entfernt (Spalten können dann in mehreren herausgezogenen Tabellen erscheinen).

Warnung

- Einfach gestrickte DB-Bücher gehen bei Definitionen der 2. und 3. Normalform immer davon aus, dass es nur einen Primärschlüssel gibt, was die Betrachtungen wesentlich erleichtert, da man dann einfach zwischen den Primärschlüsselattributen und den restlichen Attributen unterscheiden kann
- Diese Vereinfachung macht Normalisierung von der Willkür der Wahl des Primärschlüssels abhängig (s. Freundschaftstabelle)
- Rest des Aufbaus der Definitionen stimmt überein.

Nr	BestellendNr	Bestellendname	FreundNr	Freundname
1	42	Ute	43	Ugur
2	43	Ugur	44	Ulla
3	44	Ulla	42	Ute
4	42	Ute	45	Anna
5	43	Ugur	46	Ulf

3. Normalform (3NF)

Video

Definition dritte Normalform: Eine Tabelle in zweiter Normalform ist in dritter Normalform, wenn es keine zwei nicht gleiche und nicht leere Teilmengen A und B der Nichtschlüsselattribute gibt, für die A → B gilt.

mit anderen Worten: keine *transitiven* Abhängigkeiten zwischen Nichtschlüsselattributen!

Beispiel zu 3NF

- Projektmitarbeit, zu jeder Aufgabe gibt es genau ein Werkzeug (nicht in 3NF)

ProNr	MiNr	Aufgabe	Werkzeug
42	1	Analyse	Word
42	2	ER-Modell	ERWin
43	1	Analyse	Word
43	13	ER-Modell	ERWin

- Projektmitarbeit, Werkzeugzuordnung in 3NF

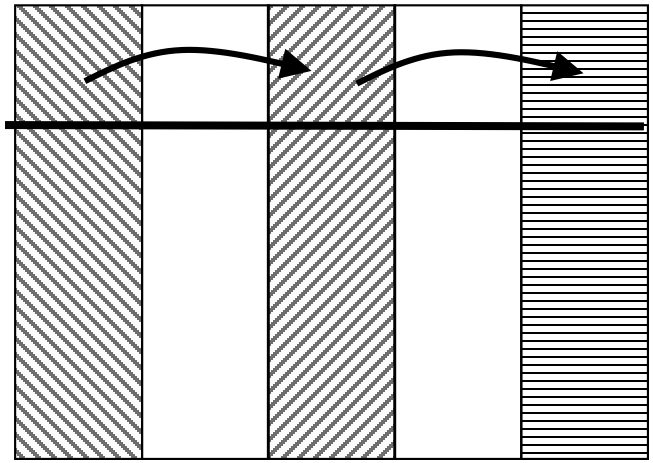
ProNr	MiNr	Aufgabe
42	1	Analyse
42	2	ER-Modell
43	1	Analyse
43	13	ER-Modell

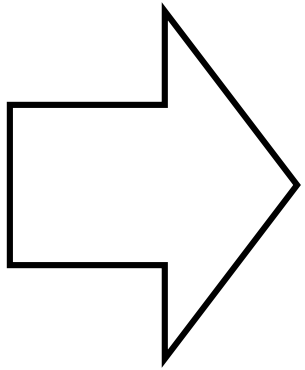
Aufgabe	Werkzeug
Analyse	Word
ER-Modell	ERWin

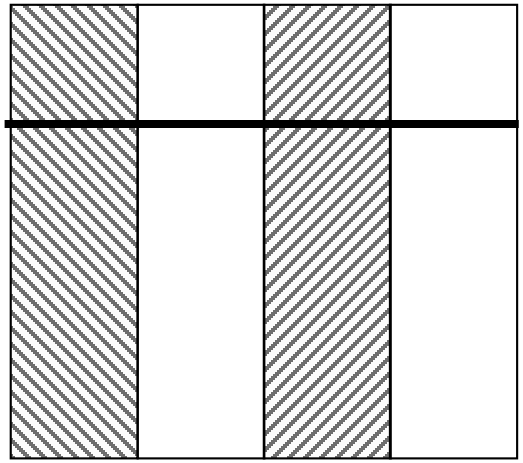
Auflösung von Abhängigkeiten

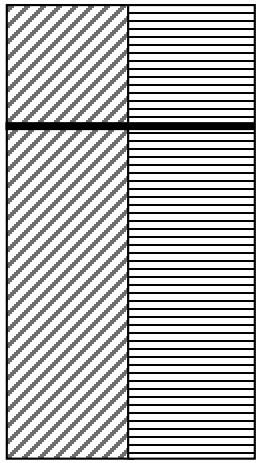
semantikerhaltende Zerlegung

Schlüssel-
kandidat









Attribut(menge) X
[voll abhängig vom
Schlüsselkandidaten]

von X
abhängiges
Attribut

Anmerkung: es werden erst alle Tabellen herausgezogen und dann die Spalten entfernt (Spalten können dann in mehreren herausgezogenen Tabellen erscheinen).

Systematik der Normalisierung (1/6)

- Klassische Aufgabe: Bringen Sie folgende Tabelle schrittweise in ein System in dritter Normalform

Lehrveranstaltung

LVNr	ModulNr	Name	Semester	Lehrt	Büro
1	42	DB	WS13	Ute	SI3
2	43	OOAD	WS13	Ulf	SI4
3	42	DB	WS14	Ulf	SI4
4	43	OOAD	WS14	Ute	SI3
5	44	Prog1	WS14	Ute	SI3

- Schritt 1: Bestimme volle funktionale Abhängigkeiten

  Nie, nie nur aus Beispielinhalt raten! Es muss konkrete Beschreibung dazu geben

Systematik der Normalisierung (2/6)

Lehrveranstaltung					
LVNr	**ModulNr**	**Name**	**Semester**	**Lehrt**	**Büro**

- Jede Lehrveranstaltung ist durch die laufende Veranstaltungsnummer (LVNr) eindeutig.

  {LVNr} → {ModulNr, Name, Semester, Lehrt, Büro}
- Aus der Modulnummer (ModulNr) folgt der Modulname (Name).

  {ModulNr} → {Name}
- Pro Semester wird jedes Modul nur von maximal einer lehrenden Person (Lehrt) als Lehrveranstaltung angeboten.

  {ModulNr, Semester} → {Lehrt, LVNr}
- Jede lehrende Person hat ein Büro, dass nicht verändert wird, aber durchaus mehrere Lehrende enthalten kann. Die Namen der Lehrenden sind eindeutig.

  {Lehrt} → {Büro}

Systematik der Normalisierung (3/6)

Lehrveranstaltung					
LVNr	**ModulNr**	**Name**	**Semester**	**Lehrt**	**Büro**

{LVNr} → {ModulNr, Name, Semester, Lehrt, Büro}
{ModulNr} → {Name}
{ModulNr, Semester} → {Lehrt, LVNr}
{Lehrt} → {Büro}

2. Schritt: Bestimme Schlüsselkandidaten, Schlüsselattribute, Nichtschlüsselattribute

Schlüsselkandidaten: {LVNr}, {ModulNr, Semester}

Schlüsselattribute: {LVNr, ModulNr, Semester}

Nichtschlüsselattribute: {Name, Lehrt, Büro}

Anmerkung: Mengenklammern sind sehr wichtig

Systematik der Normalisierung (4/6)

Lehrveranstaltung					
LVNr	**ModulNr**	**Name**	**Semester**	**Lehrt**	**Büro**

{LVNr} → {ModulNr, Name, Semester, Lehrt, Büro}

{ModulNr} → {Name}

{ModulNr, Semester} → {Lehrt, LVNr}

{Lehrt} → {Büro}

Schlüsselkandidaten: {LVNr}, {ModulNr, Semester}

Schlüsselattribute: {LVNr, ModulNr, Semester}

Nichtschlüsselattribute: {Name, Lehrt, Büro}

3a. Schritt: Prüfe, ob es Nichtschlüsselattribute gibt, die nur von Teilen von Schlüsselkandidaten abhängen (Verstoß 2NF), kann im Beispiel nur {ModulNr, Semester} betreffen

hier ja: {ModulNr} → {Name}

Systematik der Normalisierung (5/6)

Lehrveranstaltung					
LVNr	**ModulNr**	**Name**	**Semester**	**Lehrt**	**Büro**

3b. Schritt: Bei Verstoß teile Tabelle auf (neue Tabelle für jeden bei Verstoß genutzten Teil des Schlüsselkandidaten)

{ModulNr} → {Name}

Lehrveranstaltung				
LVNr	**ModulNr**	**Semester**	**Lehrt**	**Büro**

Modul	
ModulNr	**Name**

4a. Schritt: Prüfe für *jede Tabelle*, ob es Nichtschlüsselattribute gibt, die von anderen Nichtschlüsselattributen abhängen (Verstoß 3NF)

hier ja: {Lehrt} → {Büro}

Systematik der Normalisierung (6/6)

Lehrveranstaltung				
LVNr	**ModulNr**	**Semester**	**Lehrt**	**Büro**

Modul	
ModulNr	**Name**

4b. Schritt: Bei Verstoß teile Tabelle auf (neue Tabelle für jede Kombination von Nichtschlüsselattributen auf linker Seite)

{Lehrt} → {Büro}

Lehrveranstaltung			
LVNr	**ModulNr**	**Semester**	**Lehrt**

Modul	
ModulNr	**Name**

Lehrend	
Lehrt	**Büro**

Hinweis: Nie zu früh umformen; es muss nicht immer Verstöße geben

Boyce-Codd Normalform (BCNF)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

Die BCNF ist eine „verschärfte“ 3NF.

Es gilt: Eine Tabelle T ist in BCNF,

- wenn sie in 3NF ist und
- wenn für jede volle funktionale Abhängigkeit X→Y von T gilt: X ist Schlüsselkandidat

  Mit anderen Worten: es ist verboten, dass Schlüsselattribute von Nichtschlüsseln funktional abhängen.

Beispiel: die Tabelle Stadt (wobei jeder Name nur einmal pro Bundesland vorkommt, Ministerpräsidenten eindeutig sind):

Stadt(Name, Bundesland, Ministerpräsident, Einwohner)

ist in 3NF, aber nicht in BCNF (also zerlegen!)

zu BCNF

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- äquivalente Definition:

  Gilt A→B in einer Relation, mit A und B disjunkt und nicht leer, so heißt A *Determinante*. Eine Relation ist in BCNF, wenn jede Determinante einen Schlüssel enthält

- Anmerkung: Einen Unterschied zwischen BCNF und 3NF gibt es nur, wenn es mehrere Schlüsselkandidaten mit überlappenden Attributen gibt

Ist folgende Relation in BCNF?

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Relation

Lernend	Fach	Lehrend
Ute	D	Anna
Ute	Ma	Anja
Ulla	D	Anna
Ulla	Ma	Antje

- Nebenbedingungen:
  - Jede lernende Person hat pro Fach nur eine lehrende Person
  - Jede lehrende Person hat nur ein Fach
  - Zu jedem Fach kann es mehrere lehrende Personen geben

4. Normalform (4NF, Ausblick)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- 4NF ist Verschärfung von BCNF
- es wird durch „mehrwertige Abhängigkeiten“ verursachte Redundanz ausgeschlossen
- keine zwei voneinander unabhängigen mehrwertigen Fakten
- Beispiel für Problem:

PersNr	Fremdsprache	Programmiersprache
42	englisch	Java
42	französisch	Basic
42	englisch	Basic
42	französisch	Java
45	englisch	Java

- Schlüsselkandidat besteht aus allen drei Attributen, trotzdem Trennung in zwei Tabellen sinnvoll

Typischer Hintergrund für Verletzung der 4NF

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

PersNr	Hobbies	Kinder
2733	Kochen	Susanne
2733	Kochen	Horst
2733	Malen	Susanne
2733	Malen	Horst
5176	Lesen	Maria
5176	Segeln	Maria
5176	Golfen	Maria

alternative Darstellung:

PersNr	Hobbies	Kinder
2733	{Kochen, Malen}	{Susanne, Horst}
5176	{Lesen, Segeln, Golfen}	{Maria}

5. Normalform (5NF)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

5NF nur anschaulich, eine Relation ist in 5NF:

- wenn sie in 4NF ist und
- nicht durch eine Verschmelzung einfacherer (weniger Attribute aufweisender) Relationen mit unterschiedlichen Schlüsseln rekonstruiert werden kann

Hier kann man noch mal das Problem der verlustfreien Zerlegung anschaulich machen.

Beispiel: biertrinkende Dozierende

- BiertrinkendDozierend (Kneipe, Dozierend, Bier)

dies ist nicht verlustfrei in zwei Relationen zerlegbar (wenn das Lieblingsbier einer dozierenden Person von der Kneipe abhängt)

Normalisierung - Fazit

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Normalisierungsregeln helfen dem Datenbankdesign, ein konsistentes und robustes Datenmodell aufzubauen

Aber: Normalisierung hat auch Nachteile

- beim Zerlegen des Modells in viele Einzelrelationen leidet irgendwann die Übersichtlichkeit
- in einer Anwendung müssen (durch Join-Operationen [später]) die Relationen erstmal wieder zusammengesetzt werden; das kostet Zeit

Wichtig ist daher ein guter Kompromiss aus Normalisierung und Zerlegung, dafür braucht man aber etwas Erfahrung... (typisches Ziel 3NF)

5. SQL: Erstellen von Tabellen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- Erzeugen und Löschen von Tabellen
- Umgang mit Bedingungen (Constraints)
- Einfügen und Löschen von Daten
- Änderungen von Tabellenstrukturen

SQL

- Structured Query Language

Historie: Anfänge ca. 1974 als SEQUEL (IBM, System R)

SQL 86 und SQL 89: Schnittmenge existierender Implementierungen ist ANSI-Standard

SQL 92 (SQL 2): z.B.
- expliziter Verbund
- Integritätsbedingungen
- referenzielle Integrität

SQL 99 (SQL 3): z.B. (Standard besteht aus 5 Teilen)
- aktive Regeln
- Stored Procedures
- objektorientierte Konzepte

SQL 2003 (SQL 4): z.B.
- MERGE- Befehl
- Datentyp boolean (optional)
- SQL/XML Zusammenhänge

SQL 2006
SQL 2008
SQL 2011
SQL 2016
SQL 2019
…

Anmerkungen zur Syntax

- SQL (Derby) unterscheidet bei Befehlen, Tabellennamen und Attributen keine Groß- und Kleinschreibung, es bezeichnen z.B. **`CITY, city, City, cItY`** die gleiche Tabelle
- Innerhalb von Strings (Texten) unterscheidet SQL Groß- und Kleinschreibung, z.B. **`NAME = 'Berlin'`** entspricht nicht

  `NAME = 'berlin'`
- Strings stehen in einfachen Hochkommata (neben Ä auf der Tastatur)
- Kommentare werden in `/* ... */` eingeschlossen, oder, wenn nur einzeilig, mit `--` eingeleitet
- Wenn man statt Apache Derby eine andere DB nutzt, müssen die genannten Standards auf ihre Gültigkeit überprüft werden
- In Derby werden mehrere Befehle durch ein „ `;` “ getrennt

Apache Derby

- 1996 Entwicklungsstart Cloudscape Inc (Oakland, USA)
- Von Anfang an in Java entwickelt
- Neben Standard-Variante auch Embedded Version
- 1999 Informix kauft Cloudscape
- 2001 IBM kauf DB-Anteil von Informix
- 2004 unter Open Source Lizenz gestellt ; von Apache Software Foundation als Projekt akzeptiert
- Entwicklung von IBM (und früher Sun) unterstützt
- Als Java DB mit anderer Lizenz Teil jedes JDK bis 1.8.0_181

- Embedded einfach für Web- und Standalone-Programme als DB nutzbar (einfach eine Jar-Datei hinzufügen)

Tabellen nach SQL (ohne Randbedingungen)

Verkaufend

VNR	Vname	Status	Gehalt
1001	Meier	Junior	1000
1002	Schmidt	Senior	3000

Bestellend

KNR	Name	Name
1	Olm	Olm
2	Mai	Mai

```
CREATE TABLE Verkaufend(
  VNR INTEGER,
  Vname VARCHAR(12),
  Status VARCHAR(10),
  Gehalt NUMERIC
);

CREATE TABLE Bestellend(
  KNR INTEGER,
  Name VARCHAR(12),
  Betreuung INTEGER
);
```

Einfacher Aufbau und Beispiel

```
CREATE TABLE <tabellenname>(
   <attributsname> <datentyp>,
   ...
   <attributsname> <datentyp>
)

CREATE TABLE City(
   Name VARCHAR(35),
   Country VARCHAR(4),  --Länderkürzel
   Province VARCHAR(32),
   Population INTEGER,
   Longitude DOUBLE,
   Latitude DOUBLE
)
```

Typspielerei in Derby (1/4)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

```
CREATE TABLE Typen1(
  xinteger INTEGER, /* 4 Bytes */
  xint INT,  /* 4 Bytes */
  xsmallint SMALLINT, /* 2 Bytes */
  xbigint BIGINT, /* 8 Bytes */
  xreal REAL, /* 4 Bytes */
  xdouble DOUBLE, /* 8 Byte */
  xdoubleprecision DOUBLE PRECISION /* 8 Bytes */
);
INSERT INTO Typen1 VALUES(1, 1, 1, 1, 1, 1, 1);
INSERT INTO Typen1 VALUES(2147483647, 1
          , 32767, 92233720368547758O7, 3.402E+38, 2.0, 2.0);
INSERT INTO Typen1 VALUES(1, 1, 32768, 1, 1, 1, 1);
INSERT INTO Typen1 VALUES(2147483648, 1, 1, 1, 1, 1, 1);
INSERT INTO Typen1 VALUES(1, 1, 1, 922337203685477580S, 1, 1, 1);
INSERT INTO Typen1 VALUES(1, 1, 1, 1, 3.403E+38, 1, 1);
```

Typspielerei in Derby (2/4)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
CREATE TABLE Typen2(
  xdecimal DECIMAL(3,2),
  x1numeric NUMERIC,
  x2numeric NUMERIC(3),
  x3numeric NUMERIC(3,2)
);

INSERT INTO Typen2 VALUES(1, 1, 1, 1);
INSERT INTO Typen2 VALUES(1.11, 1.11, 1.111, 1.11);
INSERT INTO Typen2 VALUES(1, 1111, 111.99, 9.99);
INSERT INTO Typen2 VALUES(1, 1, 1111, 1);
INSERT INTO Typen2 VALUES(1, 1, 1, 11);
INSERT INTO Typen2 VALUES(0.1234, 1, 1, 1.111);

SELECT *
  FROM Typen2;
```

XDECIMAL	X1NUMERIC	X2NUMERIC	X3NUMERIC
1.00	1	1	1.00
1.11	1	1	1.11
1.00	1111	111	9.99
0.12	1	1	1.11

Typspielerei in Derby (3/4)

```
CREATE TABLE Typen3(
  xchar CHAR,
  xvarchar VARCHAR(4),  -- max 32672
  xclob CLOB, -- character large object
  xdate DATE,
  xtime TIME,
  xtimestamp TIMESTAMP
);

-- beide ok
INSERT INTO Typen3 VALUES('', 'Hai', 'Ho'
     ,'2015-08-06', '00:00', '2015-08-30
23:03:20.123456');
INSERT INTO Typen3 VALUES('   ', 'Hai', 'Ho'
     ,'2015-09-30', '23:59:59', '2015-08-30 23:03:20');
```

Typspielerei in Derby (4/4)

```
INSERT INTO Typen3 VALUES('', 'Haino', ''
   ,'2015-08-06', '13:00', '2015-08-30 23:03:20.123456');
INSERT INTO Typen3 VALUES('', 'Hai', 'Ho'
   ,'2015-09-31', '23:59:59', '2015-08-30 23:03:20');
INSERT INTO Typen3 VALUES('N', 'Hai', ''
   ,'2015-09-30', '24:00:00', '2015-08-30 24:00:00');
INSERT INTO Typen3 VALUES('Nu', 'Hai', ''
   ,'2015-09-30', '24:00:00', '2015-08-30 24:00:00');
INSERT INTO Typen3 VALUES('N', 'Hai', ''
   ,'2015-09-30', '24:00:00', '2015-08-30 24:00:01');
INSERT INTO Typen3 VALUES('', 'Hai', 'Ho'
   ,'2015-09-30', '23:59:59', '2015-08-30 23:03');
SELECT * FROM Typen3;
```

XCHAR	XVARCHAR	XCLOB	XDATE	XTIME	XTIMESTAMP
X	Hai	Ho	2015-08-06	00:00:00	2015-08-30 23:03:20.123456
	Hai	Ho	2015-09-30	23:59:59	2015-08-30 23:03:20.0
N	Hai		2015-09-30	00:00:00	2015-08-31 00:00:00.0

Überblick Bedingungen (Constraints)

Beispiel Video

Mit Tabellendefinitionen können Bedingungen für konkrete Attributwerte formuliert werden, die bei Eintragungen überprüft werden

- Wertebereichseinschränkungen
- Wert muss angegeben werden
- (Angabe eines Default-Wertes)
- Angaben von Schlüsseln und Fremdschlüsseln
- Forderungen an einzelne Tabelleneinträge (Datensätze) in Form von Prädikaten

Bedingungen (Constraints) (1/2)

- Syntax:

  [CONSTRAINT <name>] <bedingung>

- Typischer Aufbau einer <bedingung>:

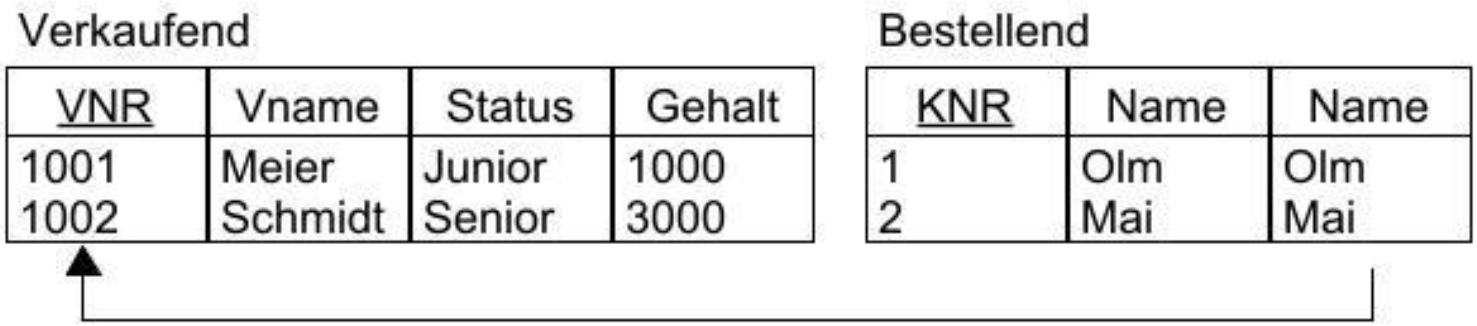
  CHECK (<boolesche_bedingung>)

- Besondere <bedingung> mit anderen Formen:
  - Primärschlüssel, Fremdschlüssel, eindeutige Attributwerte
  - Spalten-Constraints zur Angabe ob Null-Werte erlaubt sind
    - **Name VARCHAR(10) NOT NULL**
    - äquivalent als Tabellen-Constraint

      CHECK(Name IS NOT NULL)

Hinweis: Man kann es sich einfach machen und alle Constraints als Tabellen-Constraints aufschreiben

Beispiel



Randbedingungen:

- Verkaufend-Nummer mindestens vier-stellig
- Name und Status immer angegeben
- ein „Junior“ verdient maximal 2500

```
CREATE TABLE Verkaufend(
  VNR INTEGER CHECK(VNR >= 1000),
  Vname VARCHAR(12) NOT NULL,
  Status VARCHAR(10) NOT NULL,
  Gehalt NUMERIC,
  CONSTRAINT MaxJunior CHECK
       (NOT(Status = 'Junior')
        OR Gehalt <= 2500)
);
```

Syntax von Tabellen mit Constraints

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
CREATE TABLE <tabellenname>(
   <attributsname> <datentyp> [DEFAULT <wert>]
       [<spaltenconstraint>...
             <spaltenconstraint>],
   ...
   <attributsname> <datentyp> [DEFAULT <wert>]
       [<spaltenconstraint>...
             <spaltenconstraint>],
   [<tabellenconstraint>]
   ...
   [<tabellenconstraint>]
)
```

- Teile in eckigen Klammern können weggelassen werden
- **<spaltenconstraint>** bezieht sich nur auf einen Spaltenwert
- **<tabellenconstraint>** kann sich auf eine Zeile beziehen

Erinnerung: Boolesche Logik

A	B	NOT(A)	A AND B	A OR B	NOT(A) OR B
T	T	F	T	T	T
T	F	F	F	T	F
F	T	T	F	T	T
F	F	T	F	F	T

Auswertung von Constraints – Dreiwertige Logik

- Wenn Änderungen an Attributwerten durchgeführt oder neue Zeilen eingefügt werden, findet Überprüfung der Constraints statt
- Wird ein Constraint nach FALSE ausgewertet, wird die Änderung verworfen
- Achtung !! Datenbanken haben eine drei-wertige Logik (TRUE (T), FALSE (F), UNKNOWN (U)), findet z.B. eine Prüfung `VNR>=1000` statt und ist der Wert von VNR NULL, wird die Bedingung nach UNKNOWN ausgewertet

A	B	NOT(A)	A AND B	A OR B	NOT(A) OR B
T	T	F	T	T	T
T	F	F	F	T	F
T	U	F	U	T	U
F	T	T	F	T	T
F	F	T	F	F	T
F	U	T	F	U	T
U	T	U	U	T	T
U	F	U	F	U	U
U	U	U	U	U	U

Bedingungen (Tabellen-Constraints) (2/2)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

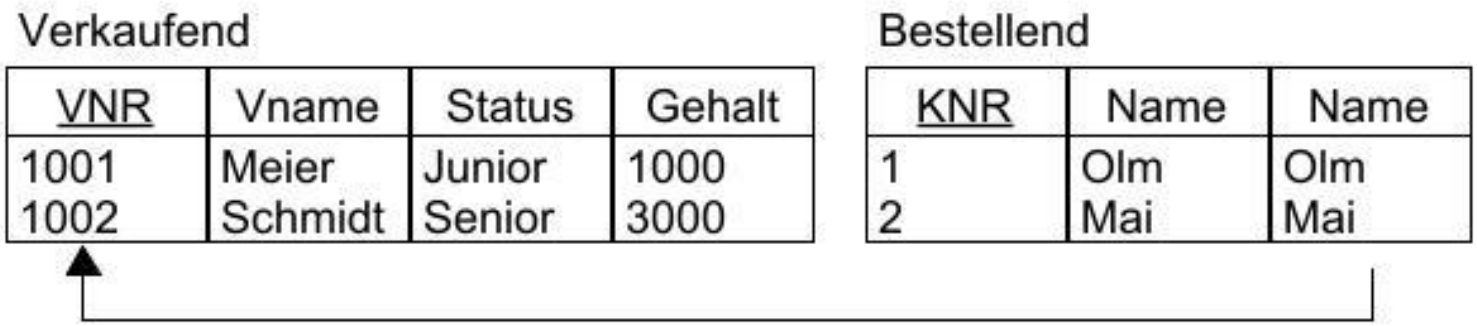
- Angabe von Primärschlüsseln (ausgewählter Schlüsselkandidat):
  PRIMARY KEY(<attributsname>[,...,<attributsname>])
- Fremdschlüssel:

```
FOREIGN KEY (<attributsname>[,...,<attributsname>])
     REFERENCES <tabellenname>
                (<attributsname>[,...,<attributsname>])
     [ON DELETE CASCADE]
```

  – In den Attributslisten steht, wie die Attribute in der zu erstellenden und in der referenzierten Tabelle heißen (müssen dort **PRIMARY KEY** sein, Tabelle muss vorher existieren)
  – **REFRENCES**-Bedingung wird durch NULL-Eintrag nicht verletzt

- Eindeutigkeit von Attributswerten (Attributskombinationen)
  UNIQUE ((<attributsname>[,...,<attributsname>])

Beispiel



Randbedingungen:

- KNR ist Schlüssel
- Bestellend hat Name
- Betreuung ist Schlüssel in Verkäufer-Tabelle

```
CREATE TABLE Bestellend(
  KNR INTEGER,
  Name VARCHAR(12)
   CONSTRAINT Kname NOT NULL,
  Betreuung INTEGER,
  PRIMARY KEY(KNR),
  CONSTRAINT FK_Bestellend
   FOREIGN KEY (Betreuung)
   REFERENCES Verkaufend(VNR)
);
```

Schlüsselkandidaten

- Primary Key ist Schlüsselkandidat, der aus der Menge der Schlüsselkandidaten durch Tabellenersteller ausgewählt wird
- Primary Keys sind eindeutig, dürfen keine NULL-Werte enthalten
- Das Beispiel erfüllt **`UNIQUE(Eins, Zwei)`**, aber nicht **`UNIQUE(Eins)`** und nicht **`UNIQUE(Zwei)`**
- Mit **`UNIQUE`** kann man z. B. festhalten, dass es Alternativen zum Primary Key gibt
- Beispiel:

```
CREATE TABLE Country(
    Name VARCHAR(32) NOT NULL UNIQUE,
    Code VARCHAR(4) PRIMARY KEY,
    ...)
```

Eins	Zwei
a	b
a	NULL
NULL	b
NULL	NULL

- Nur wenn ein Attribut Primary Key, dann als Spalten-Constraint formulierbar

Beispiel

```
CREATE TABLE is_member(
   Country VARCHAR(4) REFERENCES Country(Code),
   Organization VARCHAR(12)
               REFRENCES Organization(Abbreviation),
   Type VARCHAR(30),
   CONSTRAINT MemberKey
               PRIMARY KEY(Country,Organization)
)
```

- Anmerkungen: Hier sind die FOREIGN KEY-Constraints direkt den Attributen zugeordnet worden, könnten auch getrennt aufgeführt werden
- Aus einer Design-Entscheidung für Mondial folgt, dass es Attribute gibt, die wie Tabellen heißen, die sie referenzieren (eher ungewöhnlich, aber machbar)

Einfügen von Daten (1/2)

- einfache Variante:

  INSERT INTO <tabelle> VALUES (<werteliste>)

  Die **<werteliste>** muss für jedes Attribut einen Wert enthalten, für undefinierte Werte wird **NULL** geschrieben

- mit ausgewählten Attributen:

  INSERT INTO <tabelle>
  (<attributsname>[,...,<attributsname>])
  VALUES (<wert>[,...,<wert>])

  Werte werden in die ausgewählten Spalten geschrieben, Rest mit NULL-Werten oder DEFAULT-Werten (s. später) gefüllt

Einfügen von Daten (2/2)

- Ergebnisse einer Anfrage:

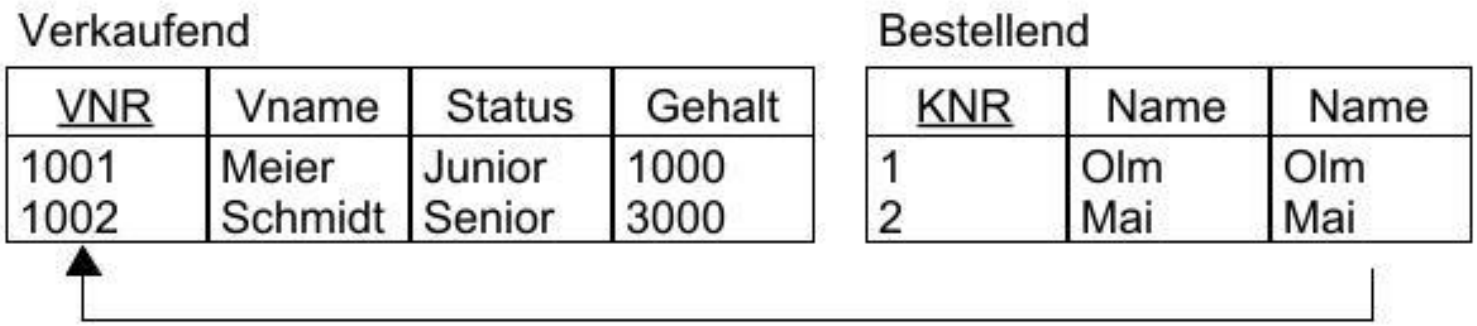
  INSERT INTO <tabelle>[(attributsliste)] <anfrage>

  (wird später deutlich)

- Beispiel:

```
INSERT INTO Country (Name, Code, Population)
                    VALUES('Lummerland','LU',4)
```

Beispiel




```
CREATE TABLE Verkaufend( ...);
CREATE TABLE Bestellend(...);

INSERT INTO Verkaufend VALUES
   (1001,'Meier','Junior',1000);
INSERT INTO Verkaufend VALUES
   (1002,'Schmidt','Senior',3000);

INSERT INTO Bestellend VALUES (1,'Olm',1001);
INSERT INTO Bestellend VALUES (2,'Mai',1002);
```

Default-Werte

- Durch Default-Werte kann man darauf verzichten, einen Wert für ein Attribut anzugeben
- Beispiel:

```
CREATE TABLE is_member(
   Country VARCHAR(4),
   Organization VARCHAR(12),
   Type VARCHAR(30) DEFAULT 'member',
   CONSTRAINT MemberKey PRIMARY KEY
        (Country,Organization)
);
```

- folgende Einfügemöglichkeiten existieren:

```
INSERT INTO is_member
       VALUES ('CZ','EU','membership applicant');
INSERT INTO is_member (Country, Organization)
       VALUES('D','EU');
```

Löschen von Daten

- Mit **DELETE** können eine oder mehrere Zeilen aus jeweils einer Tabelle entfernt werden

```
DELETE FROM <tabelle> WHERE <bedingung>
```

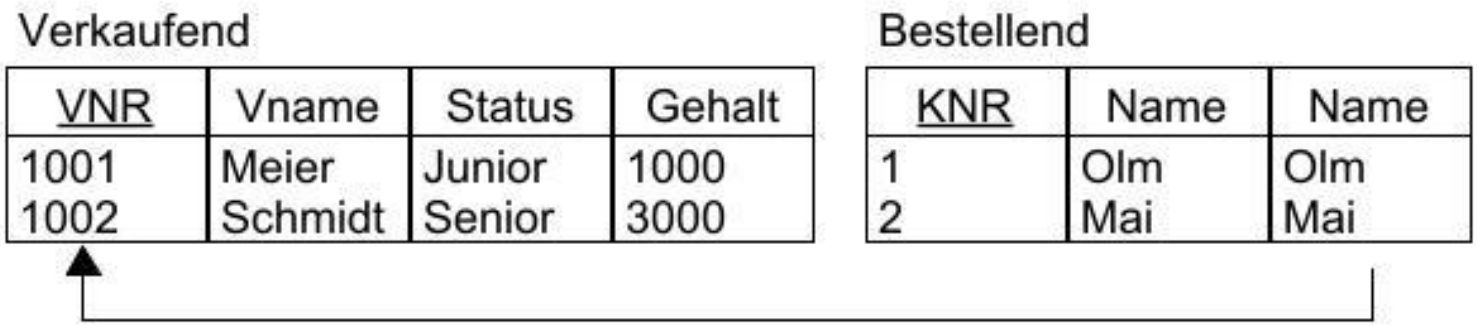
- Ob Zeile gelöscht werden darf, hängt davon ab, ob eine andere Zeile einer anderen Tabelle eine Referenz auf diese Zeile hat und welche Form die Referenz hat (siehe Beispiele)
- Jede Zeile, für die die **<bedingung>** nach TRUE ausgewertet wird, wird (wenn erlaubt) gelöscht
- Löschen aller Städte in Deutschland

```
DELETE FROM City WHERE Country='D';
```

- Wird **<bedingung>** weggelassen, wird sie als TRUE interpretiert -> alle Zeilen werden (wenn erlaubt) gelöscht

```
DELETE FROM City;
```

Einfügen ohne übergeordneten Schlüssel (1/2)




```
CREATE TABLE Bestellend(
  KNR INTEGER,
  Name VARCHAR(12),
  Betreuung INTEGER,
  PRIMARY KEY(KNR),
  CONSTRAINT FK_Bestellend
   FOREIGN KEY (Betreuung)
   REFERENCES Verkaufend(VNR)
);

INSERT INTO Bestellend VALUES(3,'Hai',1003);
```

```
Error code 30000, SQL
state 23503: INSERT in
Tabelle 'Bestellend' hat
für Schlüssel (1003) den
Fremdschlüssel-Constraint
'FK_Bestellend' verletzt.
Die Anweisung wurde
zurückgesetzt.
```

Einfügen ohne übergeordneten Schlüssel (2/2)

Verkaufend

VNR	Vname	Status	Gehalt
1001	Meier	Junior	1000
1002	Schmidt	Senior	3000

Bestellend

KNR	Name	Name
1	Olm	Olm
2	Mai	Mai

```
CREATE TABLE Bestellend(
  KNR INTEGER,
  Name VARCHAR(12),
  Betreuung INTEGER,
  PRIMARY KEY(KNR),
  CONSTRAINT FK_Bestellend
   FOREIGN KEY (Betreuung)
   REFERENCES Verkaufend(VNR)
);

INSERT INTO Bestellend(KNR,Name)
   VALUES(3,'Hai');
SELECT * FROM Bestellend;
```

```
KNR NAME   Betreuung
--- ------ ----------
  1 Olm    1001
  2 Mai    1002
  3 Hai

3 Zeilen ausgewählt.
```

Foreign Key ohne Delete Cascade

Verkaufend

VNR	Vname	Status	Gehalt
1001	Meier	Junior	1000
1002	Schmidt	Senior	3000

Bestellend

KNR	Name	Name
1	Olm	Olm
2	Mai	Mai

```
CREATE TABLE Bestellend(
  KNR INTEGER,
  Name VARCHAR(12),
  Betreuung INTEGER,
  PRIMARY KEY(KNR),
  CONSTRAINT FK_Bestellend
   FOREIGN KEY (Betreuung)
   REFERENCES Verkaufend(VNR)
);

SELECT * FROM Bestellend;
DELETE FROM Verkaufend
       WHERE VNR=1001;
```

```
KNR NAME   Betreuung
--- ------ ----------
  1 Olm    1001
  2 Mai    1002

Error code 30000, SQL state
23503: DELETE in Tabelle
'Verkaufend' hat für
Schlüssel (1001) den
Fremdschlüssel-Constraint
'FK_Bestellend' verletzt.
Die Anweisung wurde
zurückgesetzt.
```

Foreign Key mit Delete Cascade (nicht in Derby!)

Verkaufend

VNR	Vname	Status	Gehalt
1001	Meier	Junior	1000
1002	Schmidt	Senior	3000

Bestellend

KNR	Name	Name
1	Olm	Olm
2	Mai	Mai

```
CREATE TABLE Bestellend(
  KNR INTEGER,
  Name VARCHAR(12),
  Betreuung INTEGER,
  PRIMARY KEY(KNR),
  CONSTRAINT FK_Bestellend
       FOREIGN KEY (Betreuung)
       REFERENCES Verkaufend(VNR)
       ON DELETE CASCADE
);
SELECT * FROM Bestellend;
DELETE FROM Verkaufend
       WHERE VNR=1001;
SELECT * FROM Bestellend;
```

```
KNR NAME   Betreuung
--- ------ ----------
  1 Olm    1001
  2 Mai    1002

2 Zeilen ausgewählt.
1 Zeile wurde gelöscht.

KNR NAME   Betreuung
--- ------ ----------
  2 Mai    1002
1 Zeile wurde
ausgewählt.
```

Formatvorlage

- SQL bietet viele Alternativen, an denen Constraints stehen können => in Projekten Coding-Guideline benötigt

```
CREATE TABLE Bestellend(
  KNR NUMBER,
  Name VARCHAR(12) NOT NULL,
  Betreuung Number,
  PRIMARY KEY(KNR),
  CONSTRAINT FK_Bestellend1
     FOREIGN KEY (Betreuung)
     REFERENCES Verkaufend(VNR),
  CONSTRAINT Bestellend_GrosseKNR
     CHECK(KNR>1000),
);
```

Attribute, nur elementare Constraints und Default-Werte

dann Primärschlüssel

dann Fremdschlüssel (am Constraintnamen erkennbar)

dann weitere Constraints

Tabellen löschen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

DROP TABLE <tabelle> [CASCADE CONSTRAINTS]

- Tabellen müssen nicht leer sein, wenn sie gelöscht werden sollen
- Eine Tabelle, auf die noch eine andere Tabelle mit **REFERENCES** zeigt, kann mit **DROP TABLE <tabelle>** nicht gelöscht werden

- Oracle, nicht Derby: Mit **DROP TABLE <tabelle> CASCADE CONSTRAINTS** wird eine Tabelle immer gelöscht, bei anderen Tabellen, die mit REFERENCES auf diese Tabelle zeigen, wird diese Integritätsbedingung (dieses Constraint) gelöscht
- Achtung, anders als bei Derby! SQL-Standard nur **DROP TABLE <t>** entspricht dann **CASCADE CONSTRAINTS**

Ändern von Tupeln

```
UPDATE <tabelle>
    SET <attributsname> = <wert> | (<Unteranfrage>),
        ...
        <attributsname> = <wert> | (<Unteranfrage>)
    WHERE <Boolesche_Bedingung>
```

- Für jede Tabellenzeile, die die **<Boolesche_Bedingung>** erfüllt, werden die Werte der Attribute wie beschrieben geändert

```
UPDATE City
   SET Name = 'Leningrad',
       Population = Population + 1000
   WHERE Name = 'Sankt-Petersburg'
```

Vorgehensweise bei Übungen

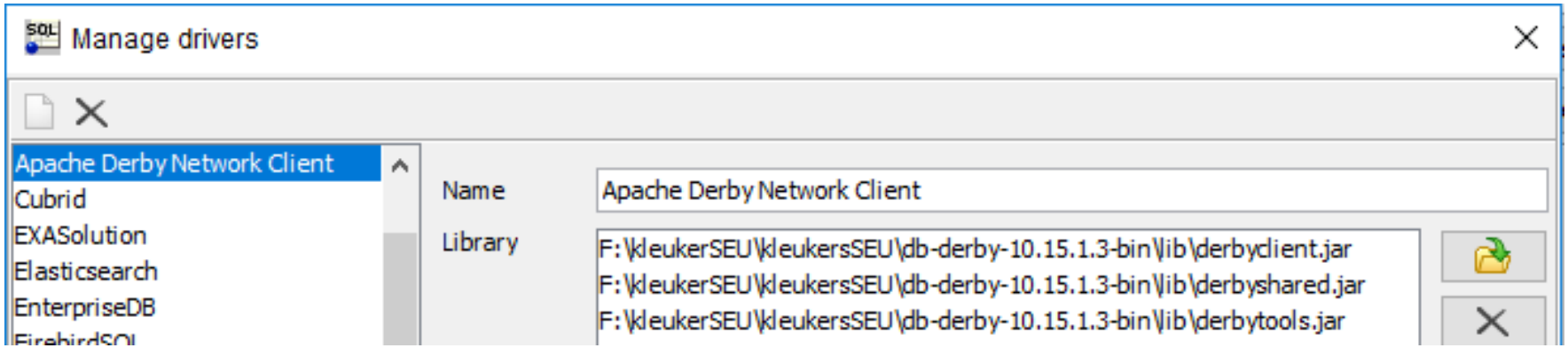
- Will man Definitionen von Tabellen testen, empfiehlt es sich ein SQL-Skript der folgenden Form für zu schreiben:

```
DROP TABLE B;
DROP TABLE A;
CREATE TABLE A(
...);
CREATE TABLE B(
...);
INSERT INTO A VALUES(...); ...
INSERT INTO B VALUES(...); ...
SELECT * FROM A;   // genauer nächste VL
SELECT * FROM B;   // genauer nächste VL
```

- Dieses Skript wird in der DB ausgeführt. Ergebnisse werden von der DB ausgegeben
- Achtung: Scheitert ein SQL-Befehl wird der Fehler ausgegeben und trotzdem der nächste Befehl ausgeführt (deshalb läuft unser Skript)
- Speichern Sie ihre Skripten in *.sql-Dateien

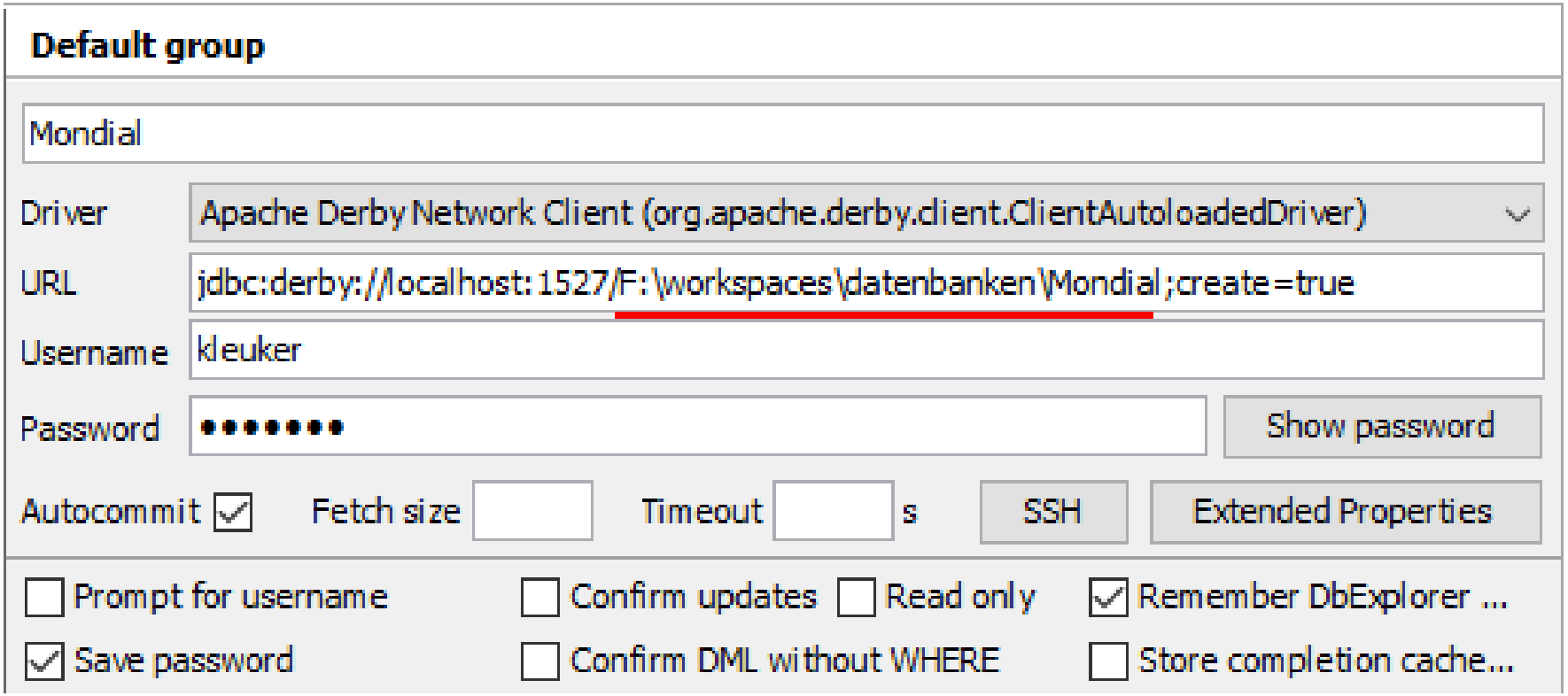
Hilfswerkzeug SQL Workbench/J (1/3)

- https://www.sql-workbench.eu/
- dient zur Einrichtung und Nutzung von Datenbanken
- Einrichtung beschrieben in http://home.edvsz.hs-osnabrueck.de/skleuker/querschnittlich/DBNutzung.pdf
- Zuerst Datenbank-Treiber (drei Jar-Dateien) installieren; Dateien in Derby-Installation im lib-Verzeichnis



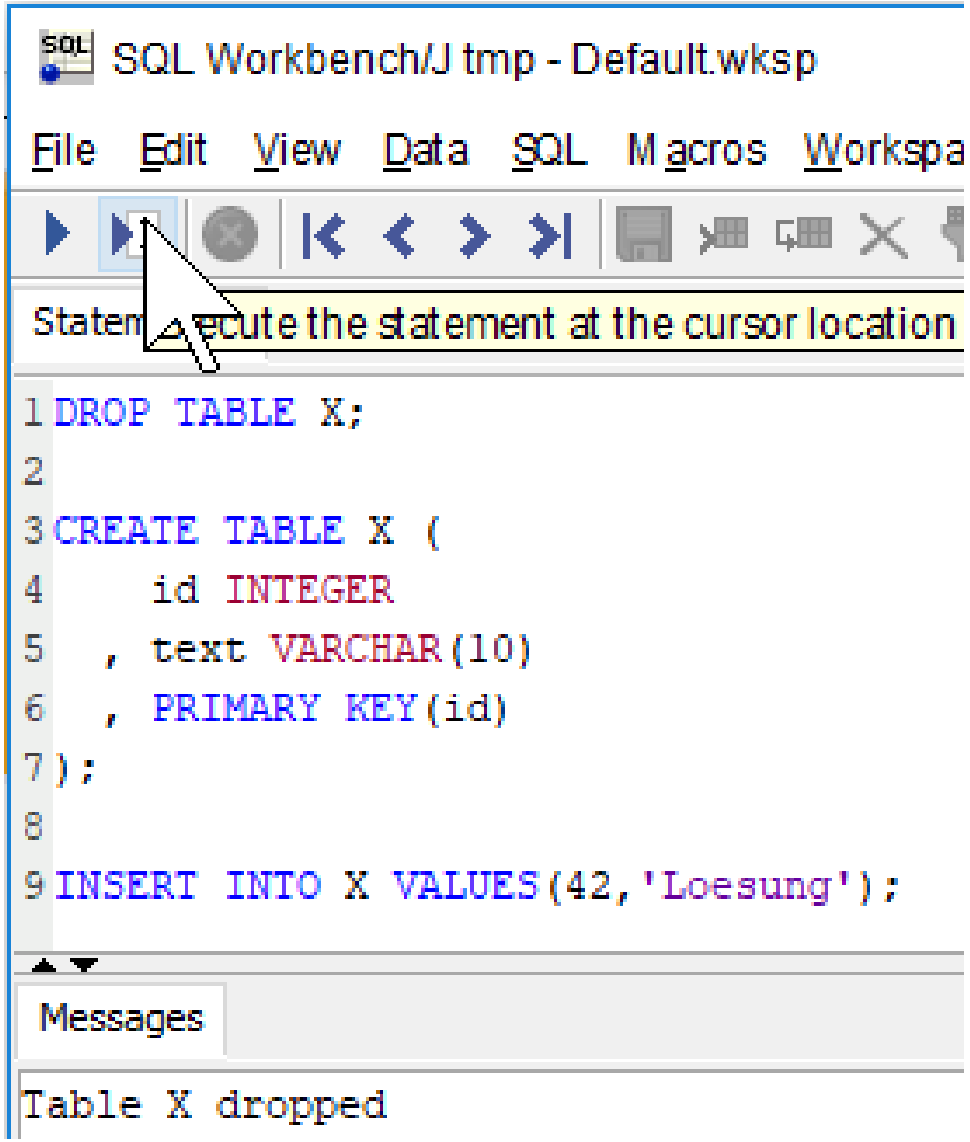
Hilfswerkzeug SQL Workbench/J (2/3)

- Datenbanken immer mit Username und Password anlegen!
- URL / Connection-String beachten (Speicherort)
- Sonderfall Derby: Parameter anhängen `;create=true` ermöglicht DB zu erstellen, wenn nicht vorhanden



Hilfswerkzeug SQL Workbench/J (3/3)

- oben Befehle eingeben
- dann eine der Ausführungs-möglichkeiten nutzen
- unten Ergebnisse interpretieren
- SQL-Skripte speichern!
- Angelegte Tabellen und eingetragene Werte bleiben nach dem Verlassen in der Datenbank erhalten



Syntax von Tabellenänderungen

Video

```
ALTER TABLE <tabelle>
  ADD (<add-zeilen>)
  DROP      PRIMARY KEY
            | UNIQUE (<spaltenliste>)
            | CONSTRAINT <constraintname>
```

- Im Standard auch **DISABLE** und **ENABLE** (z. B. Oracle) enthalten

Beispiel für Tabellenänderungen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
ALTER TABLE Z ADD COLUMN Y INTEGER;
ALTER TABLE Z ADD CHECK(Y <100);
ALTER TABLE Z ADD CONSTRAINT Nr2 CHECK(Y > 0);
ALTER TABLE Z DROP CONSTRAINT nr2;
ALTER TABLE Z ALTER COLUMN Y NULL; --loescht Constraints
ALTER TABLE Z DROP COLUMN Y;
```

- neue Spalten werden mit NULL-Werten gefüllt, damit NOT NULL sinnlos
- später mit **`ALTER TABLE ADD CONSTRAINT ...`** änderbar

Anmerkungen zu Änderungen

- PRIMARY KEY-Bedingung kann nicht gelöscht/disabled werden, solange dieser Schlüssel durch einen Fremdschlüssel in einer REFERENCES-Deklaration referenziert wird

- Bei verketteten Bedingungen ist DISABLEn und späteres ENABLEn sehr aufwändig (ORACLE unterstützt beim DISABLEn den Zusatz CASCADE)

- Oft werden vor dem Update die entsprechenden *Integritätsbedingungen* deaktiviert und nachher wieder aktiviert

Beispiel: Ändern mit Entfernen und Hinzufügen

Verkaufend

VNR	Vname	Status	Gehalt
1001	Meier	Junior	1000
1002	Schmidt	Senior	3000

Bestellend

KNR	Name	Name
1	Olm	Olm
2	Mai	Mai

```
ALTER TABLE Bestellend
  DROP CONSTRAINT FK_Bestellend;
UPDATE Verkaufend
  SET VNR=4444
  WHERE VNR=1001;
UPDATE Bestellend
  SET Betreuung=4444
  WHERE Betreuung=1001;
ALTER TABLE Bestellend
  ADD CONSTRAINT FK_Bestellend
    FOREIGN KEY (Betreuung)
      REFERENCES Verkaufend(VNR);
```

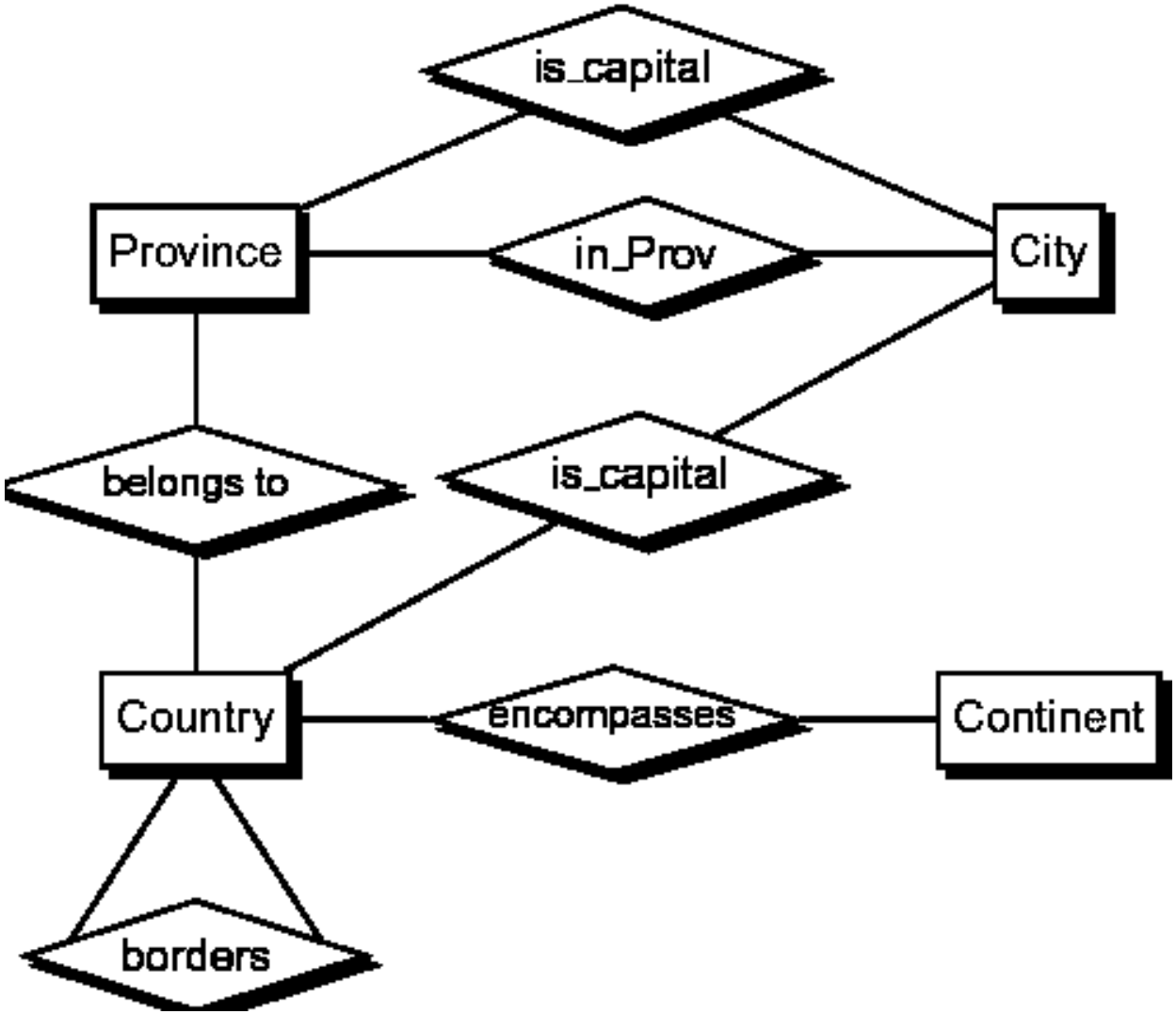
„Meier hat neue VNR 4444“

```
CREATE TABLE Bestellend(
  KNR NUMBER,
  Name VARCHAR(12),
  Betreuung NUMBER,
  PRIMARY KEY(KNR),
  CONSTRAINT FK_Bestellend
    FOREIGN KEY (Betreuung)
    REFERENCES Verkaufend(VNR)
);
```

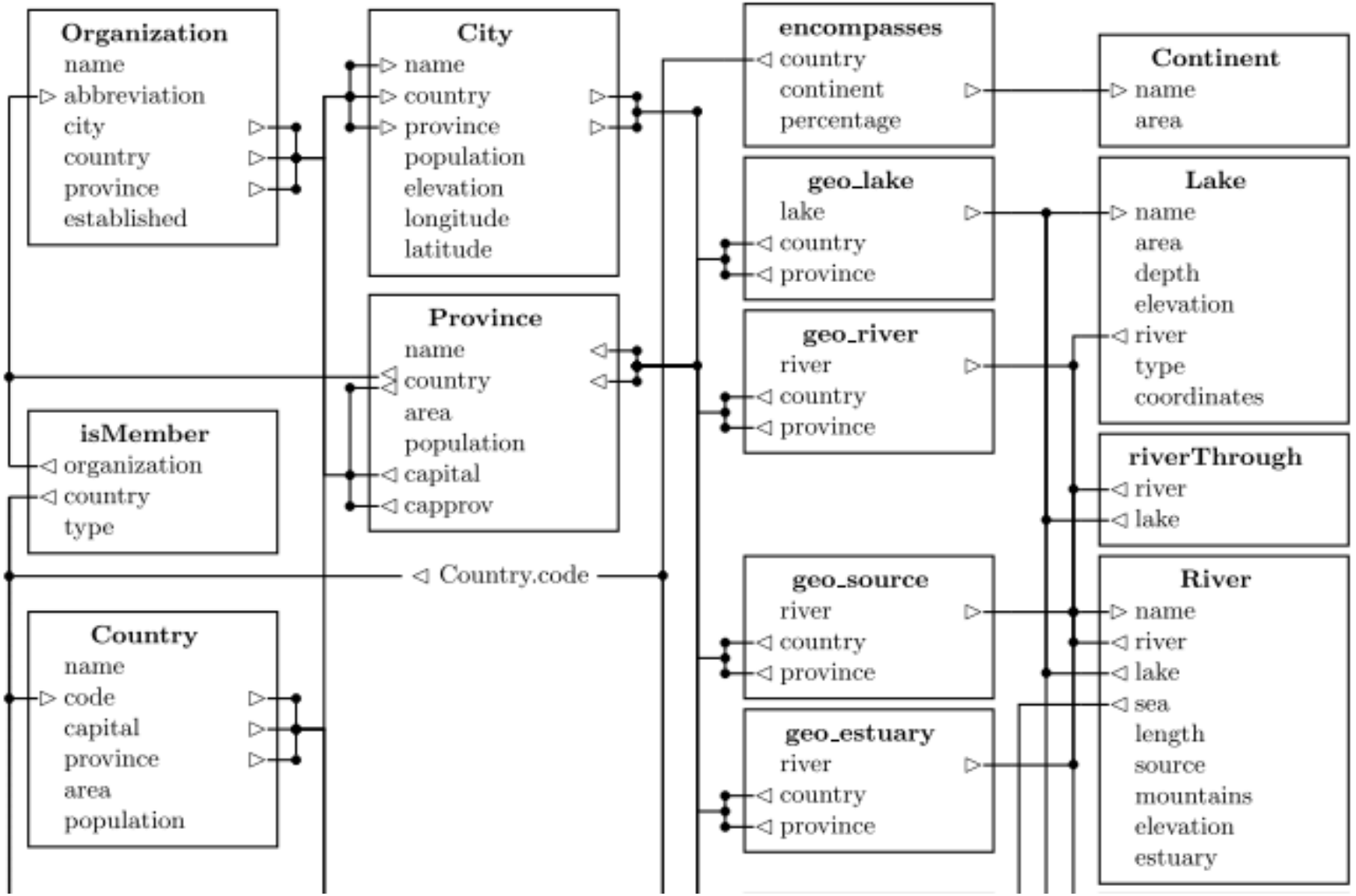
Mondial

- Übungsbeispiel zusammengestellt von der Uni Freiburg („wohnt“ jetzt in Göttingen, Prof. May, [Mon] http://www.dbis.informatik.uni-goettingen.de/Mondial/)
- Zusammenstellung verschiedener geographischer Daten (keine Garantie der Richtigkeit, etwas veraltet)
- Simuliert echte DB mit unvollständigen Daten
- Steht lokal angepasst an Derby auf der Veranstaltungsseite zur Verfügung (Datenbankschema und Inhalt)
- Soll jeder zum Ausprobieren in seine „lokale“ DB-Version einspielen.
- SQL-Aufrufe über das Web auf eine Oracle Datenbank: http://www.semwebtech.org/sqlfrontend/ (evtl. anderer Datenbestand)

Ausschnitt ER-Mondial [Mon]



In Praktika wird Tabellenübersicht genutzt



6. SQL: Einfache Anfragen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- **SELECT**-Anfragen
- Alias
- Aggregatsfunktionen

Grundstruktur der Anfrage in SQL

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

SELECT $A_1, \ldots, A_n$	**Projektionen**
FROM $T_1, \ldots, T_m$	**Betroffene Tabellen**
WHERE F	**Bedingung**

Grundsätzlicher Berechnungsablauf (wird in der DB noch optimiert):

1. Berechne alle Kombinationen der beteiligten Tabellen $T_1, \ldots, T_m$ (kartesisches Produkt)
2. Werte für alle Elemente aus, ob die Bedingung F nach wahr ausgewertet werden kann
3. Für die in 2. berechneten Elemente werden dann die in den Projektionen genannten Werte $A_1, \ldots, A_n$ für das Ergebnis bestimmt

Datenbankschema für folgende Beispiele

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

City

Name	Country	Province	Population	Longitude	Latitude
Aalborg	DK	Denmark	113865	10	57
Aarau	CH	AG	?	?	?
Aarhus	DK	Denmark	194345	10.1	56.1
Abancay	PE	Apurimac	?	?	?
Abeokuta	WAN	Nigeria	377000	?	?
Aberdeen	GB	Grampian	219100	?	?

Is_member

Country	Organization	Type
D	UN	member
D	UNESCO	member
D	UNHCR	member
DK	ICRM	National Society
DK	IEA	member
DK	IFAD	Category I

Country

Name	Code	Capital	Province	Area	Population
Austria	A	Vienna	Vienna	83850	8023244
Afghanistan	AFG	Kabul	Afghanistan	647500	22664136
Albania	AL	Tirane	Albania	28750	3249136
Angola	ANG	Luanda	Luanda	1246700	10342899
Azerbaijan	AZ	Baku	Azerbaijan	86600	7676953
Belgium	B	Brussels	Brabant	30510	10170241

Encompasses

Country	Continent	Percentage
TR	Asia	68
TR	Europe	32
TT	America	100
UAE	Asia	100
USA	America	100
UZB	Asia	100

Einfache Anfragen

Video

- Welche Städtenamen in City gespeichert?

```
SELECT City.Name FROM City
```

Ausgabe (Ausschnitt):

```
NAME
---------------
Alexandria
Cordoba
Cordoba
Cordoba
```

- SQL-Tabellen können im Gegensatz zu Relationen doppelte Einträge enthalten (einziger, aber wichtiger Unterschied)
- Eliminierung von Doppelten:

```
SELECT DISTINCT City.Name FROM City
```

- Welche Millionenstädte sind in City gespeichert

```
SELECT City.Name, City.Population
  FROM City
  WHERE City.Population >= 1000000
```

- Anzeige des gesamten Tabelleninhalts:

```
SELECT * FROM City
```

Umbenennungsmöglichkeiten (1/3)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Ergebnisse von SQL-Anfragen sind Tabellen, d.h. es können Anfragen in Anfragen eingesetzt werden
- für die Kombination von Anfragen gibt es verschiedene Umbenennungsmöglichkeiten (Alias-Einführung)
- Welche Millionenstädte sind in City gespeichert

```
SELECT City.Name, City.Population
    FROM City
    WHERE City.Population >= 1000000
```

- unschöne Kurzform

```
SELECT Name, Population
    FROM City
    WHERE Population >= 1000000
```

Umbenennungsmöglichkeiten (2/3)

- Umbenennung der Ergebniszeilen

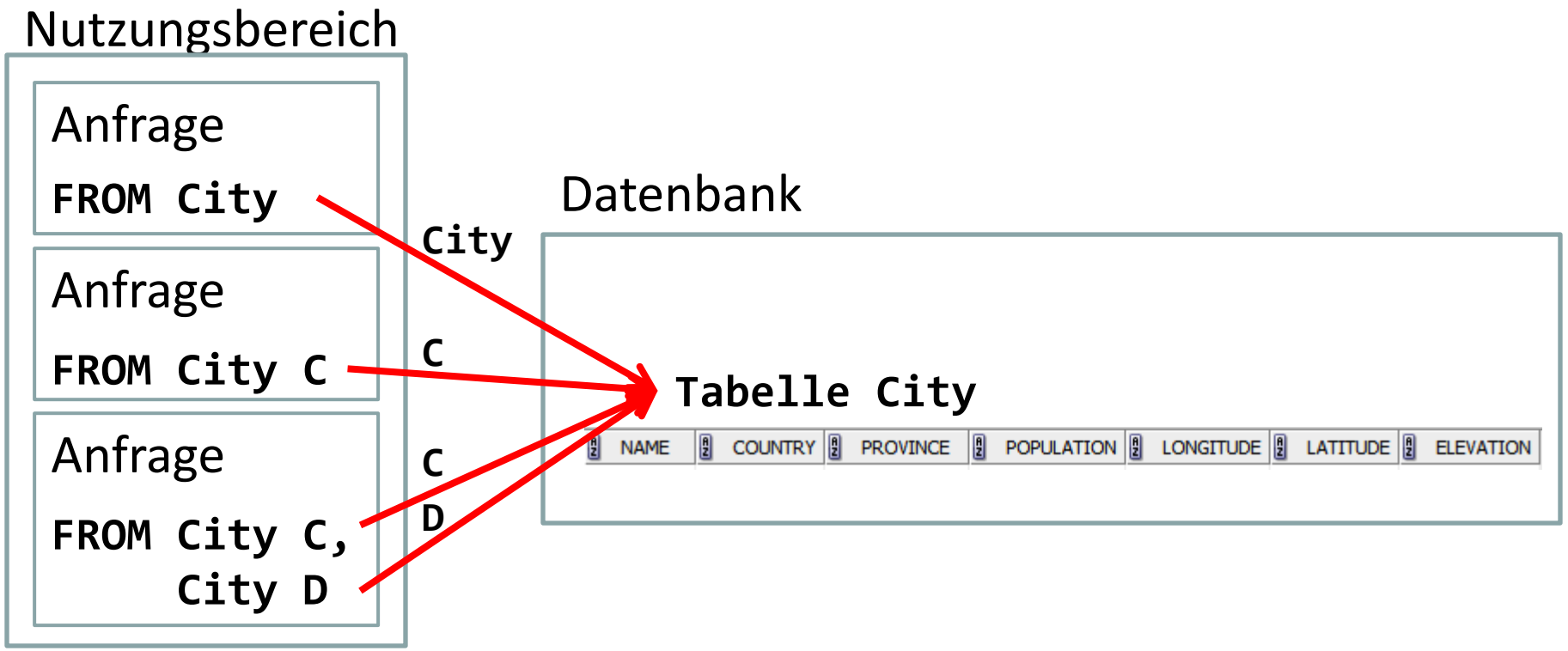
```
SELECT City.Name Stadtname,
       City.Population Einwohnerzahl
  FROM City
  WHERE City.Population >= 1000000
```

STADTNAME	EINWOHNERZAHL
London	6967500
Birmingham	1008400
Belgrade	1407073
Paris	2152423

Umbenennungsmöglichkeiten (3/3)

- lokale Umbenennung der benutzten Tabelle (Referenzen)

```
SELECT C.Name, C.Population
    FROM City C
    WHERE C.Population >= 1000000
```



Erste Möglichkeiten für das WHERE-Prädikat (1/2)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Gib alle Länder aus, deren Kürzel zwischen einschließlich B und ausschließlich D liegen

```
SELECT Country.Name
   FROM Country
   WHERE Country.Code >='B' AND Country.Code < 'D'
```

- Gib alle Städte, die in Ländern mit dem Kürzel F, D oder A liegen

```
SELECT City.Name
FROM City
WHERE City.Country IN ('F', 'D', 'A')
```

- Gib alle Städte, die nicht mehr als 10000 oder mehr als 2000000 Einwohner haben

```
SELECT City.Name
    FROM City
    WHERE NOT(City.Population >= 10000)
          OR City.Population > 2000000
```

Erste Möglichkeiten für das WHERE-Prädikat (2/2)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Hinweis: % steht in Texten für beliebig viele Zeichen, _ für genau ein beliebiges Zeichen, Textähnlichkeit mit **LIKE** geprüft
- Gib alle Länder, die im Namen den Begriff „land“ enthalten

```
SELECT *
    FROM Country
    WHERE Country.Name LIKE '%land%'
```

- Gib alle Länder, deren dritter Buchstabe im Kürzel ein D ist

```
SELECT *
    FROM Country
    WHERE Country.Code LIKE '__D%'
```

- Gib alle Städte, für die keine Einwohnerzahl angegeben ist

```
SELECT *
    FROM City
    WHERE City.Population IS NULL
```

Zugriff auf verschiedene Relationen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Gib alle Städtenamen mit dem zugehörigen Land aus

```
SELECT City.Name, Country.Name
    FROM City, Country
    WHERE City.Country = Country.Code
```

Typisch: Nennung von n Tabellen,
dann Verknüpfung mit n-1 Bedingungen

- Gleiche Anfrage mit Aliassen

```
SELECT T1.Name, T2.Name
    FROM City T1, Country T2
    WHERE T1.Country = T2.Code
```

- Ausgaben können auch sortiert erfolgen:

```
SELECT City.Name, Country.Name
    FROM City, Country
    WHERE City.Country = Country.Code
    ORDER BY City.Name ASC, Country.Name DESC
```

Alternative Verknüpfungsvariante

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Städtenamen mit X am Anfang, mit Ländernamen und Kontinent

```
SELECT City.Name, Country.Name, Encompasses.Continent
  FROM City, Country, Encompasses
 WHERE City.Country = Country.Code
   AND Encompasses.Country = Country.Code
   AND City.Name LIKE'X%';
```

alternativ

```
SELECT City.Name, Country.Name, Encompasses.Continent
FROM City
   JOIN Country ON (City.Country = Country.Code)
   JOIN Encompasses ON (Encompasses.Country = Country.Code)
 WHERE City.Name LIKE'X%';
```

- statt , ein JOIN danach ON(<Boolesche Bedingung>) in ON kann das gleiche wie in WHERE stehen
- neuere Schreibart, kontroverse Diskussion was lesbarer

Alias-Verwendung

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

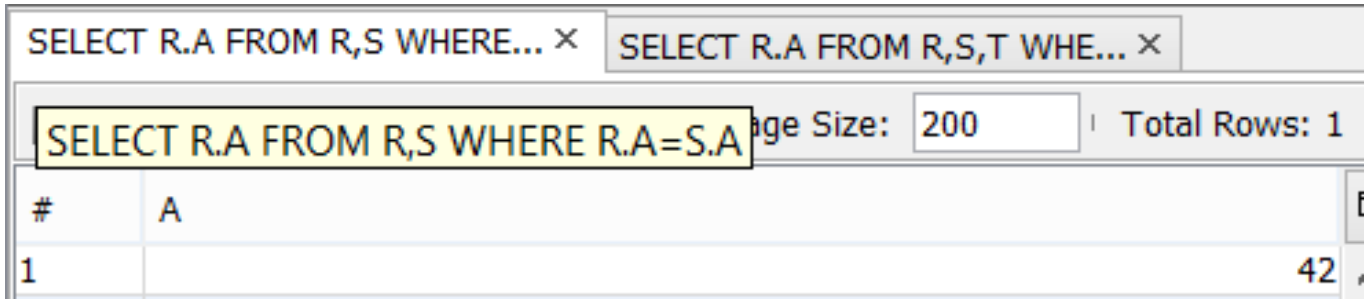
- Bestimme alle Paare von Ländern (deren Länderkürzel), die auf dem gleichen Kontinent liegen

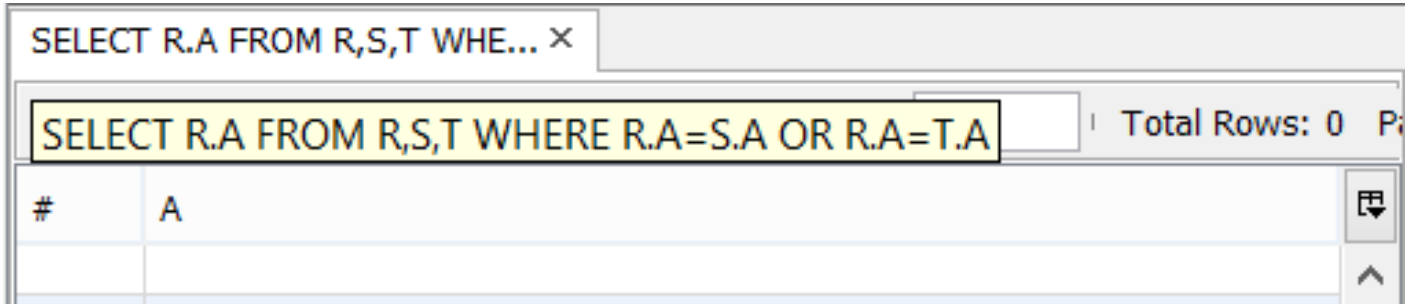
```
SELECT E1.Country fir, E2.Country sec
    FROM Encompasses E1, Encompasses E2
    WHERE E1.Continent = E2.Continent
    AND E1.Country < E2.Country
```

```
FIR  SEC
---  ---
A    AL
A    GR
AL   GR
AND  GR
B    GR
...
4284 Zeilen ausgewählt.
```

Nanu?

```
CREATE TABLE R(A INTEGER);
CREATE TABLE S(A INTEGER);
CREATE TABLE T(A INTEGER);

INSERT INTO R VALUES(42);
INSERT INTO S VALUES(42);
```





Auswertung von SQL-Anfragen

- **SQL-Anfrage:**

```
SELECT A1, ... ,Am
  FROM R1, ... ,Rn
  WHERE F
```

- **Algorithmus (nested-loop-Semantik):**

  FOR each tuple t_1 in relation R_1 DO
  FOR each tuple t_2 in relation R_2 DO
  ...
  FOR each tuple t_n in relation R_n DO
  IF die WHERE-Klausel ist erfüllt nach Ersetzen der Attributnamen durch die entsprechenden Werte von $t_1, \ldots, t_n$ THEN
  werte die SELECT-Klausel bzgl. $t_1, \ldots, t_n$ aus und bilde das Antwort-Tupel.

Der folgende Ausdruck zur Berechnung von $R \cap (S \cup T)$ liefert $\emptyset$, sofern $T = \emptyset$!

```
SELECT R.A FROM R, S, T WHERE R.A = S.A OR R.A = T.A
```

Berechnungen in Ergebniszeilen

- Gib für jede Stadt den Anteil ihrer Einwohnerzahl an der Gesamteinwohnerzahl des Landes an

```
SELECT City.Name,
          City.Population/Country.Population
   FROM City, Country
   WHERE City.Country=Country.Code
```

```
NAME        CITY.POPULATION/COUNTRY.POPULATION
----------- ----------------------------------
Paris                                ,03690873
Vienna                               ,019730172
Berlin                               ,04156297
```

- Man kann auch neue Spalten in die SELECT-Zeile schreiben

```
SELECT Country.Name, 42 DieZahl, 'Hallo' EinText
FROM Country
```

```
NAME           DIEZAHL EINTE
---------- ---------- -----
Albania            42 Hallo
Greece             42 Hallo
Macedonia          42 Hallo
```

Textverknüpfungen

- Texte werden in SQL mit || verknüpft, vgl. + auf Strings in Java
- In Derby müssen Zahlenwerte erst in Strings verwandelt werden, wenn sie mit Texten verknüpft werden sollen

```
SELECT 'Kontinent ' || Continent.name || ' hat Flaeche '
       || CAST(Continent.area AS CHAR(10))
  FROM Continent;
```

1
Kontinent Europe hat Flaeche 9562488
Kontinent Asia hat Flaeche 45095292
Kontinent Australia/Oceania hat Flaeche 8503474
Kontinent Africa hat Flaeche 30254708
Kontinent America hat Flaeche 39872000

Aggregatsfunktionen (1/2)

- Aggregatsfunktionen beziehen sich jeweils auf eine Spalte einer Relation, es gibt die Funktionen **SUM, AVG, MIN, MAX, COUNT**
- Welches in die größte Fläche eines Landes?

```
SELECT MAX(Country.Area)
 FROM Country
```

```
MAX(COUNTRY.AREA)
-----------------
         17075200
```

- Wie viele Länder sind eingetragen?

```
SELECT COUNT(*)
    FROM Country
```

- Wie viele unterschiedliche Organisationen stehen in der Mitgliedertabelle?

```
SELECT COUNT(DISTINCT Ismember.organization)
    FROM Ismember
```

- Schachtelungen wie `MAX(AVG(…))` sind nicht erlaubt

Aggregatsfunktionen (2/2)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Wie viele Einwohner von Deutschland leben in den abgespeicherten Städten?

```
SELECT SUM(City.Population)
FROM City
WHERE City.Country='D'
```

- Wie viele Einwohner leben in Deutschland?

```
SELECT Country.Population
FROM Country
WHERE Country.Name='Germany'
```

- Wie viele Einwohner von Deutschland leben nicht in den aufgeführten Städten?

```
SELECT Country.Population – Tmp.Citypop
FROM Country,
    (SELECT SUM(City.Population) AS Citypop
       FROM City WHERE City.Country='D') Tmp
WHERE Country.Name='Germany'
```

Grobes Kochrezept zur passenden Anfrage

1. Kann ich Aufgabe in Teilanfragen zerlegen? Wenn ja, welche Verknüpfungspunkte gibt es [genauer nächster Block]?
2. Welche Tabellen sind betroffen? Es wird die **FROM**-Zeile festgelegt (wird eine Relation mit sich selbst verknüpft, muss die Relation mehrmals aufgeführt werden)?
3. Verknüpfung der Tabellen aus 2.
4. Basteln des **WHERE**-Anteils aus möglichst elementaren Bausteinen, die (meist mit **AND**) verknüpft werden. So werden Randbedingungen formuliert.
5. Was für Ergebnisanteile brauche ich aus der Anfrage? Es wird die **SELECT**-Zeile festgelegt.
6. Läuft die Anfrage, kann sie optimiert werden.

7. SQL: Komplexere Anfragen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- Geschachtelte Anfragen
- Mengenoperationen
- Verschiedene Arten von JOINs

Schachtelung von Anfragen (Subqueries)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Es können Anfragen in Anfragen eingesetzt werden, eine mögliche Struktur ist:

```
SELECT <attributliste>
    FROM <tabellenliste>
    WHERE <attribut> (<op> [ANY|ALL] | IN) <anfrage>
```

- `<anfrage>` ist eine **`SELECT`**-Anfrage (Subquery)

- Wenn `<op>` aus `{=,<,>,<=,>=,<>}` muss Subquery eine einspaltige Tabelle als Ergebnis haben (bei `IN` sind auch mehrere Spalten erlaubt)

Ansatz für geschachtelte Anfragen

- Komplexe Anfrage wird zerlegt, man überlegt zunächst eine Anfrage für ein Teilergebnis, das dann in eine komplexere Anfrage eingebettet wird – gibt oft alternative Lösungen
- Welche Länder haben eine größere Fläche wie die Türkei?
  1. Teilaufgabe: Bestimme die Fläche der Türkei

```
SELECT Country.Area
    FROM Country
    WHERE Country.Name = 'Turkey'
```

  2. Teilaufgabe: Suche alle Länder mit einer größeren Fläche

```
SELECT Country.Name
    FROM Country
    WHERE Country.Area > (
            SELECT Country.Area
                FROM Country
                WHERE Country.Name='Turkey')
```

Geschachtelte Anfragen (1/3)

- Welche Städte haben mehr Einwohner als alle Städte in Afrika einzeln betrachtet?

```
SELECT City.Name
FROM City
WHERE City.Population > ALL
          (SELECT City.Population
            FROM City, Encompasses
            WHERE City.Country=Encompasses.Country
             AND Encompasses.Continent='Africa'
             AND City.Population IS NOT NULL)
```

- Hinweis: Es gibt andere Lösungsansätze, bei denen man z.B. zuerst die größte Einwohnerzahl einer Stadt in Afrika bestimmt (obere Lösung also hölzern)

Geschachtelte Anfragen (2/3)

- Gib die Namen aller Länder, die sich auf den gleichen Kontinenten wie Russland befinden

```
SELECT DISTINCT Country.name
  FROM Country, Encompasses
  WHERE Country.code=Encompasses.Country
    AND Encompasses.Continent IN
        (SELECT Encompasses.Continent
           FROM Encompasses
           WHERE Encompasses.Country='R')
```

Geschachtelte Anfragen (3/3)

Welche Länder liegen auf zwei Kontinenten?

```
SELECT A.Country
  FROM Encompasses A, Encompasses B
  WHERE A.Country = B.Country AND
        A.Continent <> B.Continent
```

```
COUN
----
R
R
TR
TR
RI
RI
KAZ
KAZ
ET
ET
```

```
SELECT Country.Name
  FROM Country
   WHERE Country.Code IN
   (SELECT A.Country
        FROM Encompasses A, Encompasses B
        WHERE A.Country = B.Country AND
              A.Continent <> B.Continent)
```

```
NAME
-------
Egypt
Kazakstan
Russia
Indonesia
Turkey
```

Exists

Video

- Mit **EXISTS(<anfrage>)** wird überprüft, ob es überhaupt eine Ergebniszeile der <anfrage> gibt, Beispiel:
  Gib die Namen der Länder, für die eine Stadt mit mehr als einer Million Einwohner gespeichert ist
- Vorüberlegung: Berechnung aller Millionen-Städte für ein Land mit Code XYZ:

```
SELECT *
FROM City
 WHERE City.Population>1000000
   AND City.Country='XYZ'
```

- Anfrage zur Untersuchung aller Länder:

```
SELECT Country.Name
FROM Country
WHERE EXISTS( SELECT *
                  FROM City
                  WHERE City.Population>1000000
                    AND City.Country=Country.Code)
```

Schachtelung und FROM

- Innere Anfrage nutzt Tabelle der äußeren Anfrage; äußere WHERE-Bedingung wird einmal für jeden Country geprüft

```
SELECT Country.Name
    FROM Country
    WHERE EXISTS( SELECT *
                      FROM City
                      WHERE City.Population>1000000
                        AND City.Country=Country.Code)
```

- Hier unsinnig, innere Anfrage wiederholt Tabelle der äußeren Anfrage; innere Anfrage hat keinen Zusammenhang mit äußerer Anfrage (zwei Referenzen auf Country)

```
SELECT Country.Name
    FROM Country
    WHERE EXISTS( SELECT *
                      FROM City, Country
                      WHERE City.Population>1000000
                        AND City.Country=Country.Code)
```

EXISTS und FORALL

- FORALL gibt es nicht im Standard!
- Mathematik weiß für natürliche Zahlen: $\forall x: x*x \geq 0$
- äquivalent: $\neg\exists x: \neg(x*x \geq 0)$
- Länder deren Städte alle zwischen dem 0. und 5. östlichen Längengrad liegen
- äquivalent: Länder die keine Stadt haben, die nicht zwischen dem 0. und 5. östlichen Längengrad liegt

```
SELECT Country.Name
  FROM Country
  WHERE NOT EXISTS (SELECT *
                      FROM City
                      WHERE City.Country = Country.CODE
                        AND NOT(    City.Longitude > 0
                                AND City.Longitude < 5));
```

1	Andorra
2	Netherlands
3	Algeria
4	Niger
5	Benin
6	Togo

Anfrageergebnisse als Tabellen nutzen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Kürzel der Länder auf zwei Kontinenten?

```
SELECT A.country Land
  FROM Encompasses A, Encompasses B
  WHERE A.Country = B.Country AND
        A.Continent <> B.Continent
```

```
Land
----
R
R
TR
TR
RI
RI
KAZ
KAZ
ET
ET
```

- Ausgabe der Ländernamen

```
SELECT Country.Name
  FROM Country, (SELECT A.Country Land
    FROM Encompasses A, Encompasses B
    WHERE A.Country = B.Country
    AND A.Continent <> B.Continent) Kuerzel
   WHERE Country.Code = Kuerzel.Land
```

```
NAME
---------
Russia
Russia
Turkey
Turkey
Indonesia
Indonesia
Kazakstan
Kazakstan
Egypt
Egypt
```

Geschachtelte SQL-Anfragen

Anfrage A1

```
SELECT T1.A, T2.B, T1.C+T2.B
FROM T1, T2
WHERE ...
...
```

Auswertung →

Ergebnis: anonyme Tabelle

A	B	C+B

Anfrage A2 (Umbenennung der Ergebnisspalten von A1)

```
SELECT T1.A X, T2.B Y, T1.C+T2.B Z
FROM T1, T2
WHERE ...
...
```

Auswertung →

Ergebnis: anonyme Tabelle

X	Y	Z

Nutzung von Anfragen in FROM-Zeile mit Umbenennung der anonymen Tabelle

```
SELECT ..., Tab.X
FROM T7, (A2) Tab
...
```

Nutzung von Anfragen in der WHERE-Bedingung, z.B.

- WHERE EXISTS (A1)
- WHERE (Att1,Att2,Att3) IN (A1)

CASE WHEN – einfache Fallunterscheidung

Video

```
SELECT CASE Country.Code
         WHEN 'AFG' THEN 'Afghanistan'
         WHEN 'B' THEN 'Belgien'
         WHEN 'B' THEN 'Belgique'
         ELSE Country.Name
       END AS AHA
  FROM Country
  WHERE Country.Code IN ('AFG','AL','B','BZ','CN');
```

AHA
Afghanistan
Albania
Belgien
Belize
China

CASE WHEN – komplexe Fallunterscheidung

```
SELECT Country.Name
  ,CASE
     WHEN Country.Code BETWEEN'A' AND 'B'
          THEN 'XXX'||Code
     WHEN Country.Code BETWEEN'A' AND 'C'
          THEN 'YYY'||Code
     ELSE Country.Code
   END AS AHA
  FROM Country
  WHERE Country.Code IN ('AFG','AL','B','BZ','CN');
```

NAME	AHA
Afghanistan	XXXAFG
Albania	XXXAL
Belgium	XXXB
Belize	YYYBZ
China	CN

Mengenoperationen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- SQL-Anfragen können durch Mengenoperationen verknüpft werden, Syntax:

  `(<anfrage>) <mengen-op> (<anfrage>)`

- Dabei kann `<mengen-op>` folgende Formen haben
  - **`UNION [ALL]`** (Vereinigung)
  - **`EXCEPT [ALL]`** (Differenz)
  - **`INTERSECT [ALL]`** (Durchschnitt)

- Bei einfachen Mengenoperationen ohne **`ALL`** findet automatisch eine Duplikateliminierung statt
- **`ALL`** verhindert die Eliminierung von Duplikaten

Mengenoperationen – Beispiele (1/2)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Gib die Kürzel aller Länder, die in Europa und Asien liegen

```
(SELECT Encompasses.Country
  FROM Encompasses
  WHERE Encompasses.Continent='Europe')
INTERSECT
(SELECT Encompasses.Country
  FROM Encompasses
  WHERE Encompasses.Continent='Asia')
```

- Welche Länderkürzel werden in Country, aber nicht in Encompasses genutzt

```
(SELECT Country.Code FROM Country)
EXCEPT
(SELECT Encompasses.Country FROM Encompasses)
```

Mengenoperationen – Beispiele (2/2)

- Gib alle Städte mit ihrer Einwohnerzahl aus, falls diese nicht angegeben ist, soll 0 ausgegeben werden

```
(SELECT City.Name, City.Population
  FROM City
  WHERE NOT(City.Population IS NULL))
UNION
(SELECT City.Name, 0
  FROM City
  WHERE City.Population IS NULL)
```

Erinnerung: Trick neue Spalte erzeugen (ohne Name, Name von erster Tabelle vorgegeben)

Joins (benutztes Beispiel)

Beispiel Video

Beispiel:

Mitarbeitend

Nachname	Vorname	Geburtsdatum
Meier	Lara	6.6.1966
Huber	Karl	7.7.1977
Schmidt	Helga	5.5.1955

Projekt

Projekt	Nachname	Vorname
Analyse	Huber	Anna
Modell	Schmidt	Helga
Design	Moos	Johann

Cross Join

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
SELECT * FROM Mitarbeitend CROSS JOIN Projekt
SELECT * FROM Mitarbeitend, Projekt   (äquivalent)
```

Mitarbeitend. Nachname	Mitarbeitend. Vorname	Geburtsdatum	Projekt	Projekt. Nachname	Projekt. Vorname
Meier	Lara	6.6.1966	Analyse	Huber	Anna
Meier	Lara	6.6.1966	Modell	Schmidt	Helga
Meier	Lara	6.6.1966	Design	Moos	Johann
Huber	Karl	7.7.1977	Analyse	Huber	Anna
Huber	Karl	7.7.1977	Modell	Schmidt	Helga
Huber	Karl	7.7.1977	Design	Moos	Johann
Schmidt	Helga	5.5.1955	Analyse	Huber	Anna
Schmidt	Helga	5.5.1955	Modell	Schmidt	Helga
Schmidt	Helga	5.5.1955	Design	Moos	Johann

Inner Join (= JOIN)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Inner Join = Equivalent Join
Verbindet Datensätze aus zwei Tabellen mit Hilfe einer Verknüpfungsbedingung.

```
SELECT * FROM Mitarbeitend INNER JOIN Projekt
ON Mitarbeitend.Nachname = Projekt.Nachname  (seit SQL92)

SELECT * FROM Mitarbeitend, Projekt
  WHERE Mitarbeitend.Nachname = Projekt.Nachname
```

Mitarbeitend. Nachname	Mitarbeitend. Vorname	Geburtsdatum	Projekt	Projekt. Nachname	Projekt. Vorname
Huber	Karl	7.7.1977	Analyse	Huber	Anna
Schmidt	Helga	5.5.1955	Modell	Schmidt	Helga

Natural Join

Beim Natural Join werden zwei Tabellen über Spalten mit gemeinsamen Namen verknüpft, die Werte für diese Attribute müssen für das Ergebnis übereinstimmen (teilweise sehr nützlich, man sollte Spalten aber nicht auf Krampf gleich benennen)

```
SELECT *
FROM Mitarbeitend NATURAL JOIN Projekt;
```

Nachname	Vorname	Geburtsdatum	Projekt
Schmidt	Helga	5.5.1955	Modell

Left Join (1/2)

Left Outer Join = Left Join

linke Inklusionsverknüpfung , alle Datensätze aus der ersten (linken) Tabelle im Ergebnis, auch wenn keine entsprechenden Werte für Datensätze in der zweiten Tabelle existieren

Im ON(<Bedingung>) alle für Paarsuche relevanten Verknüpfungsdaten

```
SELECT * FROM Mitarbeitend LEFT JOIN Projekt
            ON (    Mitarbeitend.Nachname = Projekt.Nachname
                AND Mitarbeitend.Vorname = Projekt.Vorname)
```

Mitarbeitend. Nachname	Mitarbeitend. Vorname	Geburtsdatum	Projekt	Projekt. Nachname	Projekt. Vorname
Meier	Lara	6.6.1966	NULL	NULL	NULL
Huber	Karl	7.7.1977	NULL	NULL	NULL
Schmidt	Helga	5.5.1955	Modell	Schmidt	Helga

Left Join (2/2)

- erst ON auswerten, dann WHERE

```
SELECT * FROM Mitarbeitend LEFT JOIN Projekt
            ON (Mitarbeitend.Nachname = Projekt.Nachname
              AND  Mitarbeitend.Vorname = Projekt.Vorname)
```

Mitarbeitend. Nachname	Mitarbeitend. Vorname	Geburtsdatum	Projekt	Projekt. Nachname	Projekt. Vorname
Meier	Lara	6.6.1966	NULL	NULL	NULL
Huber	Karl	7.7.1977	NULL	NULL	NULL
Schmidt	Helga	5.5.1955	Modell	Schmidt	Helga

```
SELECT * FROM Mitarbeitend LEFT JOIN Projekt
            ON (Mitarbeitend.Nachname = Projekt.Nachname)
         WHERE Mitarbeitend.Vorname = Projekt.Vorname
```

Mitarbeitend. Nachname	Mitarbeitend. Vorname	Geburtsdatum	Projekt	Projekt. Nachname	Projekt. Vorname
Schmidt	Helga	5.5.1955	Modell	Schmidt	Helga

Right Join

Right Outer Join = Right Join

Mit einem Right Join wird eine sogenannte rechte Inklusionsverknüpfung erstellt. Rechte Inklusionsverknüpfungen schließen alle Datensätze aus der zweiten (rechten) Tabelle ein, auch wenn keine entsprechenden Werte für Datensätze in der ersten Tabelle existieren.

```
SELECT * FROM Mitarbeitend RIGHT JOIN Projekt
    ON (     Mitarbeitend.Nachname = Projekt.Nachname
         AND  Mitarbeitend.Vorname = Projekt.Vorname)
```

Mitarbeitend. Nachname	Mitarbeitend. Vorname	Geburtsdatum	Projekt	Projekt. Nachname	Projekt. Vorname
NULL	NULL	NULL	Analyse	Huber	Anna
Schmidt	Helga	5.5.1955	Modell	Schmidt	Helga
NULL	NULL	NULL	Design	Moos	Johann

8. SQL: Gruppierung und Analyse von NULL-Werten

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- Gruppierungen
- Null-Werte

Generelle Form der SQL-Anfrage

`SELECT` $A_1, \ldots, A_n$	Projektionen
`FROM` $T_1, \ldots, T_m$	benutzte Tabellen
`WHERE F`	Bedingung
`GROUP BY` $G_1, \ldots, G_m$	Liste der Gruppierungen
`HAVING H`	Gruppierungsbedingung
`ORDER BY O`	Sortierordnung

Auswertungsreihenfolge:

`FROM`-Klausel vor `WHERE`-Klausel vor
`GROUP`-Klausel vor `HAVING`-Klausel vor
`ORDER`-Klausel vor `SELECT`-Klausel

Gruppierung

- Wie hoch ist die Summe der Einwohner in den gespeicherten Städten eines jeden Landes

```
SELECT City.Country, SUM(City.Population)
   FROM City
   GROUP BY City.Country
```

- Durch `GROUP BY` werden alle Ergebniszeilen, die in den aufgeführten Attributen übereinstimmen, zu einer Gruppe zusammengefasst
- Für jede dieser Gruppen können in der `SELECT`-Zeile Aggregatsfunktionen auf den Tabellen-Attributen stehen
- In der `SELECT`-Zeile dürfen nur die Attribute nicht aggregiert auftreten, die in der `GROUP BY`-Zeile zur Gruppierung genutzt werden (gleiches gilt für eine `ORDER BY`–Zeile)

Gruppierung mit HAVING

- In welchen Ländern ist die durchschnittliche Einwohnerzahl in den aufgeführten Städten kleiner als 10000

```
SELECT Country.name, AVG(City.population)
   FROM Country,City
   WHERE Country.code=City.Country
   GROUP BY Country.name
   HAVING AVG(City.population) < 10000
```

NAME	AVG(CITY.POPULATION)
Kiribati	2100
Holy See	392
Montserrat	0
Marshall Islands	7600
Palau	4038
Tuvalu	2120
Belize	3000
Malta	7858
Saint Pierre and Miquelon	5618
Saint Martin	5700
Micronesia	5972
Monaco	1234
San Marino	4416

GROUP BY und HAVING - Randbedingungen (1/2)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
SELECT  [DISTINCT] target-list
FROM    relation-list
WHERE   qualification
GROUP BY grouping-list
HAVING   group-qualification
ORDER BY  order-list
```

- Die *target-list* enthält (i) Attributsnamen (ii) Terme mit Aggregatsfunktionen (z.B. `MIN(Country.Area)`)
  - die Liste der Attribute (i) muss eine Teilmenge der Liste in *grouping-list* sein. Jedes Ergebnistupel korrespondiert mit einer *Group*
  - eine *Group* ist eine Menge von Tupeln, die die gleichen Werte in den Attributen hat, die in *grouping-list* genannt sind

GROUP BY und HAVING - Randbedingungen (2/2)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
SELECT  [DISTINCT] target-list
FROM     relation-list
WHERE    qualification
GROUP BY grouping-list
HAVING   group-qualification
ORDER BY   order-list
```

- die Ausdrücke in *group-qualification* müssen einen einzelnen Wert (Skalar) pro Group liefern
- für die *order-list* gelten die gleichen Bedingungen, wie für die *target-List*

Berechnung von GROUP BY und HAVING (1/2)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Auswahl der Tupel durch die `WHERE`-Klausel

  Das Kreuzprodukt von *relation-list* wird berechnet, Tupel, die die `WHERE`-Bedingung (*qualification*) nicht erfüllen (nicht nach true ausgewertet werden), werden entfernt

- Bildung von Gruppen durch die `GROUP BY`-Klausel

  Die verbleibenden Tupel werden in Gruppen partitioniert, bestimmt durch die Werte der Attribute in *grouping-list*

Berechnung von GROUP BY und HAVING (2/2)

- Auswahl der Gruppen, die die `HAVING`-Klausel erfüllen

  Die `HAVING`-Klausel (*group-qualification*) wird angewandt, um ggfls. einige Gruppen zu entfernen. Ausdrücke in *group-qualification* müssen einen skalaren Wert pro Gruppe liefern!

- Sortieren des Ergebnisses und Ausgabe
- Die angegebenen Attribute oder/und Aggregatsfunktionen in der `ORDER BY`-Klausel werden zum Sortieren des Ergebnisses genutzt. Die Angaben der `SELECT`-Klausel werden zur Auswahl der relevanten Informationen für die Ergebnispräsentation genutzt.

Beispiel: Welcher Umsatz mit welchem Produkt (1/3)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

VK

Verkäufer	Produkt	Käufer
Meier	P1	Schmidt
Müller	P2	Schmidt
Meier	P1	Schulz

PL

Produkt	Preis	Klasse
P1	100	B
P2	200	A

VK × PL

Verkäufer	Produkt	Käufer	Produkt	Preis	Klasse
Meier	P1	Schmidt	P1	100	B
Meier	P1	Schmidt	P2	200	A
Müller	P2	Schmidt	P1	100	B
Müller	P2	Schmidt	P2	200	A
Meier	P1	Schulz	P1	100	B
Meier	P1	Schulz	P2	200	A

Beispiel: Welcher Umsatz mit welchem Produkt (2/3)

```
SELECT VK.Produkt, SUM(PL.Preis)
FROM VK, PL
WHERE VK.Produkt=PL.Produkt
GROUP BY VK.Produkt
```

VK × PL

Verkäufer	Produkt	Käufer	Produkt	Preis	Klasse
Meier	P1	Schmidt	P1	100	B
Meier	P1	Schmidt	P2	200	A
Müller	P2	Schmidt	P1	100	B
Müller	P2	Schmidt	P2	200	A
Meier	P1	Schulz	P1	100	B
Meier	P1	Schulz	P2	200	A

Beispiel: Welcher Umsatz mit welchem Produkt (3/3)

```
SELECT VK.Produkt, SUM(PL.Preis)
FROM VK, PL
WHERE VK.Produkt=PL.Produkt
GROUP BY VK.Produkt
```

VK × PL

Verkäufer	Produkt	Käufer	Produkt	Preis	Klasse
Meier	P1	Schmidt	P1	100	B
Meier		Schulz	P1	100	B
Müller	P2	Schmidt	P2	200	A

Ergebnis:

Produkt	SUM(PL.Preis)
P1	200
P2	200

NULL-Werte (Erinnerung)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- `NULL` steht für unbekannten Wert
- Jeder Datentyp wird durch `NULL`-Wert erweitert
- einzige sinnvolle Prüfungsmöglichkeiten mit `IS NULL` und `IS NOT NULL`
- `NULL` steht in Berechnungen für „Undefiniert“ (unknown),
- z.B. `A+B`, `7+A` liefern `NULL`, wenn `A IS NULL`
- in Vergleichen ist das Ergebnis `NULL` (unknown), wenn einer der Operanden `NULL` ist, z. B. bei `A<B`, `A=B`, `A<>B`, `A!=B`
- Aggregatsfunktionen, z. B. `SUM` ignorieren NULL, Ausnahme `COUNT`
- Dreiwertige Logik (TRUE (t), FALSE (f), UNKNOWN (u)):

AND	t	u	f
t	t	u	f
u	u	u	f
f	f	f	f

OR	t	u	f
t	t	t	t
u	t	u	u
f	t	u	f

NOT	
t	f
u	u
f	t

NULL-Werte

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Wert a	Prüfung	Ergebnis
42	a IS NULL	FALSE
42	a IS NOT NULL	TRUE
NULL	a IS NULL	TRUE
NULL	a IS NOT NULL	FALSE
42	a = NULL	UNKNOWN
42	a != NULL	UNKNOWN
NULL	a = NULL	UNKNOWN
NULL	a != NULL	UNKNOWN
NULL	a = 42	UNKNOWN
NULL	a != 42	UNKNOWN

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- Grundsätzliche Nutzung von JDBC
- Verbindungsaufbau
- Anfragen
- Analyse des erhaltenen Ergebnisses
- Veränderungen des Ergebnisses

Ziel: Verständnis für Konzepte von JDBC aufbauen

Überblick: Datenbankanfragen mit JDBC

Datenbankverbindung herstellen

```
class DriverManager
Connection con =
  DriverManager.getConnection(...);
Statement stmt =
  con.createStatement();
```

Datenbankanfrage

```
ResultSet rs = stmt.executeQuery(...);
```

Ergebnisse verarbeiten

```
rs.next();
int n = rs.getInt("KNr");
```

Verbindung zur DB schließen

```
con.close();
```

Verbindungsaufbau mit einer Datenbank

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Laden des Datenbanktreibers (mit Reflexion)

```
Class.forName("org.apache.derby.jdbc.ClientDriver")
      .getDeclaredConstructor().newInstance();
```

- Aufbau einer Verbindung

```
Connection con = DriverManager.getConnection(
  "jdbc:derby://localhost:1527/F:\\tmp\\Konten"
     ,"user", "password");
Connection con = DriverManager.getConnection("jdbc:derby"
        + "://localhost:1527/Mondial;user=usr;password=pwd");
```

- JDBC-URL identifiziert Datenbank

  Aufbau: jdbc:subprotocol:subname

- Treibermanager übergibt JDBC-URL der Reihe nach an registrierte Treiber

```
Driver.acceptsURL(String url)
```

liefert true, falls der Treiber den String akzeptiert

Derby-Verbindung, genauer

```
try {
  Class.forName("org.apache.derby.jdbc.ClientDriver")
      .getDeclaredConstructor()
      .newInstance();
  this.con = DriverManager.getConnection(
        "jdbc:derby://localhost:1527/"
      + " F:\\workspaces\\dbs\\Mondial;create=true"
      , "kleuker", "kleuker");
  this.stmt = con.createStatement(); // con, stmt
Objektvariablen
} catch (SQLException | ClassNotFoundException
    | InstantiationException | IllegalAccessException
    | IllegalArgumentException | InvocationTargetException
    | NoSuchMethodException | SecurityException ex) {
      System.out.println("Problem: " + ex);
        // geht sinnvoller
}
```

falls DB nicht existiert, dann erzeugen, wird sonst ignoriert

Auslesen der Oracle-Verbindungsdaten

- Oracle „versteckt“ irgendwo Verbindungsdaten in tnsnames.ora
- benötigt wird IP-Adresse, Port (default 1521), Name der DB-Instanz (z. B. in SID) + Username und Passwort
- Daraus abgeleiteter Connection-String für HS-DB (SID: Ora11)

```
Connection con = DriverManager.getConnection(
  "jdbc:oracle:thin:"
  + "@oracle-srv.edvsz.hs-osnabrueck.de"
  + ":1521:Ora11", "username", "passwort");
```

- Connection-String für DB auf lokalem Rechner (Oracle XE)

```
Connection con = DriverManager
    .getConnection(
        "jdbc:oracle:thin:@localhost:1521:xe"
        , "username", "passwort");
```

Connection

- Derby: Port 1527
- Achtung zwei Versionen: Embedded und Client/Server
- Ergebnis von DriverManager.getConnection() ist eine **`Connection con`** (oder eine SQLException)
- **`Connection`** ist Verbindung zur Datenbank
- **`Connection`** ist teure Ressource:
  - Aufbau der Verbindung kostet viel Zeit
  - Anzahl gleichzeitiger Verbindungen häufig beschränkt
- Wichtigste Aufgaben:
  - Erzeugen von Statement-Objekten
  - Beschreibungen von Datenbank und DBMS erfragen
  - Transaktionen handhaben

SQLException

- Jede Methode mit DB-Zugriff kann SQLException werden
- Behandlung von Exception muss in SW-Projekten einheitlich geregelt werden
- Meist sinnvoll Exceptions zu loggen
- Typischer Ansatz (in Klasse DBZugriff)

```
try{
  this.con = DriverManager.getConnection(…);
  …
} catch (SQLException ex) {

Logger.getLogger(DBZugriff.class.getName())
                        .log(Level.SEVERE, null, ex);
}
```

- Achtung: Auf Folien oft notwendige try-catch-Behandlung weggelassen, da immer fast identisch!!!

Verbindungsanalyse durch Metadaten (Ausschnitt)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

```
DatabaseMetaData dbmd = con.getMetaData();
System.out.println("DB-Name: " + dbmd.getDatabaseProductName()
  + "\nDB-Version: " + dbmd.getDatabaseMajorVersion()
  + "\nDB-Release: " + dbmd.getDriverMinorVersion()
  + "\nTransaktionen erlaubt: " + dbmd.supportsTransactions()
  + "\nbeachtet GroßKlein :" + dbmd.storesMixedCaseIdentifiers()
  + "\nunterstützt UNION :" + dbmd.supportsUnion()
  + "\nmax. Prozedurname: " + dbmd.getMaxProcedureNameLength());
```

```
DB-Name: Apache Derby
DB-Version: 10
DB-Release: 15
Transaktionen erlaubt:
true
beachtet GroßKlein :false
unterstützt UNION :true
max. Prozedurname: 128
```

```
DB-Name: Oracle
DB-Version: 11
DB-Release: 2
Transaktionen erlaubt:
true
beachtet GroßKlein :false
unterstützt UNION :true
max. Prozedurname: 30
```

Statement / Anfrage ausführen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- `Statement stmt = con.createStatement();`
- Wird immer aus bestehender **`Connection`** erzeugt
- Aufgabe: Ausführen einer SQL-Anweisung über die **`Connection`**
- Mehrere parallele Statements pro **`Connection`** möglich
- SELECT-Anweisung ausführen:

```
ResultSet rs = stmt.executeQuery(
                              "SELECT * FROM Bestellend");
```

  am Ende mit **`rs.close()`** schließen
- Daten ändern:

```
int updates = stmt.executeUpdate(
                              "DELETE FROM Bestellend ...");
```

Metadaten des Anfrageergebnisses

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
ResultSet rs = stmt.executeQuery("SELECT * FROM Continent");
ResultSetMetaData rsmd = rs.getMetaData();
int spalten = rsmd.getColumnCount();
for (int i = 1; i <= spalten; i++) { // nicht i=0
  System.out.println(i + ". Name: " + rsmd.getColumnName(i)
      + "  Typ: " + rsmd.getColumnTypeName(i)
      + "  Javatyp: " + rsmd.getColumnClassName(i));
}
```

```
1. Name: NAME  Typ: VARCHAR  Javatyp:
java.lang.String
2. Name: AREA  Typ: DECIMAL  Javatyp:
java.math.BigDecimal
```

Analyse von ResultSet

```
ResultSet rs = stmt.executeQuery(
                        "SELECT * FROM Bestellend");
```

- Ergebnis einer Selektionsanweisung: Tabelle
- **`ResultSet`** enthält einen Datensatz-Zeiger (Cursor) zum Durchlauf der Tabelle
- Cursor entspricht Iterator-Idee z. B. aus C++, Java
- Voreinstellung: sequenziell und lesend
- ab JDBC 2: nichtsequenzielle und aktualisierbare ResultSets
- Zeiger steht initial *vor* der ersten Tabellenzeile
- **`rs.next()`** positioniert zur nächsten Zeile, liefert false, falls bereits auf letzter Zeile

Attributwerte auslesen (1/2)

- Spaltenwerte (Attribute) einer Zeile mit getXXX()-Methoden lesen
- Treiber konvertiert Daten, falls möglich, deshalb (fast) immer getString() nutzbar
- Beispiel: Lesen einer ganzen Zahl in DB-Spalte BestellendNr:

```
int n = rs.getInt("BestellendNr");
```

- Effizientere Methode, falls Spaltenindex bekannt:

```
int n = rs.getInt(4);
```

- Spaltenindex zum Spaltennamen finden

```
int findColumn(String columnName)
```

- Strategie: Spaltenindex einmalig ermitteln und merken, Werte danach immer über den Index abrufen

Attributwerte auslesen (2/2)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Methode **`getObject()`**
  - Liest jeden beliebigen SQL-Datentyp
  - Liefert Ergebnis als entsprechenden Java-Typ
- Nullwerte
  - Spalte kann leere Zellen enthalten (Nullwert, SQL-NULL)
  - Bei leerer Zelle liefert **`getInt()`** ebenfalls das Ergebnis 0
  - Unterscheidung zu echter 0 möglich durch **`wasNull()`**
    - true, falls zuletzt mit getXXX() gelesene Zelle SQL-NULL enthielt
    - false sonst

Beispiel: Ausgabe eines Anfrageergebnisses

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
ResultSet rs = stmt.executeQuery("SELECT * FROM Continent");
ResultSetMetaData rsmd = rs.getMetaData();
int spalten = rsmd.getColumnCount();
while (rs.next()) {
  for (int i = 1; i <= spalten; i++) {
    System.out.print(rs.getString(i) + " ");
  }
  System.out.print("\n");
}
```

```
Europe 9562488
Asia 45095292
Australia/Oceania 8503474
Africa 30254708
America 39872000
```

JDBC – Prüfung auf Existenz einer Tabelle

Video

```
public boolean existiertTabelle(String tabelle){
  ResultSet rs = this.connection.getMetaData()
                       .getTables(null, null, "%", null);
  boolean existiert = false;
  while (rs.next() && !existiert) {
    if (rs.getString(3)
          .equalsIgnoreCase(tabelle)) {
      existiert = true;
    }
  }
  return existiert;
}
```

ResultSet: positionieren und ändern

- **`Statement createStatement(int resultSetType , int resultSetConcurrency)`**
- Parameter (ResultSet-Konstanten) ermöglichen beliebiges Positionieren (Scrolling) und Ändern der Datensätze
- **`TYPE_FORWARD_ONLY`**: sequentieller Durchlauf
- **`TYPE_SCROLL_INSENSITIVE`**: positionierbar, Änderungen an Datenbank werden nicht bemerkt
- **`TYPE_SCROLL_SENSITIVE`**: positionierbar, Änderungen an Datenbank werden bemerkt

- **`CONCUR_READ_ONLY`**: nur lesen
- **`CONCUR_UPDATABLE`**: Änderungen möglich

ResultSet: positionieren (1/3)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- **`void beforeFirst()`**

  Positioniert vor den ersten Satz
- **`boolean first()`**

  Positioniert auf den ersten Satz
- **`boolean last()`**

  Positioniert auf den letzten Satz
- **`void afterLast()`**

  Positioniert hinter den letzten Satz

ResultSet: positionieren (2/3)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- **`boolean absolute(int pos)`**

  Positioniert ausgehend vom Anfang (pos positiv) oder vom Ende (pos negativ)
- **`boolean relative(int rows)`**

  Positioniert relativ zum aktuellen Satz vorwärts (rows positiv) oder rückwärts (rows negativ)
- **`boolean next()`**

  Positioniert auf den nächsten Satz
- **`boolean previous()`**

  Positioniert auf den vorigen Satz

ResultSet: positionieren (3/3)

- **`int getRow()`**

  Liefert aktuelle Satznummer
- **`boolean isBeforeFirst()`**

  Liefert true, falls vor dem ersten Satz
- **`boolean isFirst()`**

  Liefert true, falls auf dem ersten Satz
- **`boolean isLast()`**

  Liefert true, falls auf dem letzten Satz
- **`boolean isAfterLast()`**

  Liefert true, falls hinter dem letzten Satz

ResultSet: Datensatz ändern

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- Methoden **`updateXXX()`** ändern Werte in aktueller Zeile.

```
rs.absolute(5);
rs.updateString("Name", "Heinz");
rs.updateInt(2, 42);
rs.updateNull(3);
rs.updateRow();
```

- **`void updateRow()`**
  Schreibt geänderte Zeile in die Datenbank (*)
- **`void cancelRowUpdates()`**
  Macht Änderungen rückgängig – nur vor **`updateRow()`**
- **`void deleteRow()`**
  Löscht aktuelle Zeile aus ResultSet

(*) Ob Bearbeitung des ResultSet sich direkt auf die Datenbank auswirkt, hängt von Autocommit-Einstellung ab [später]

ResultSet: Datensatz einfügen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Einfügezeile: im ResultSet, nicht in der Datenbank

```
rs.moveToInsertRow();
rs.updateString("Name", "Callaghan");
rs.updateInt(2, 42);
rs.insertRow();
rs.moveToCurrentRow();
```

- **`void moveToInsertRow()`**

  Positioniert auf die Einfügezeile

- **`void insertRow()`**

  Schreibt Einfügezeile in ResultSet und Datenbank (abhängig von Autocommit-Einstellung)

- **`void moveToCurrentRow()`**

  Positioniert von der Einfügezeile auf die aktuelle Zeile im ResultSet

Daten ändern

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- **`ResultSet executeQuery(String)`** führt SELECT-Anweisung aus
- **`int executeUpdate(String)`** führt INSERT-, UPDATE- oder DELETE-Anweisung aus
  - Liefert Anzahl betroffener Zeilen (Update count)
  - Auch für sonstige Befehle (CREATE ...) geeignet
- **`boolean execute(String)`** führt beliebige SQL-Anweisung aus
  - Liefert true, falls Ergebnis ein ResultSet ist, dann mit getResultSet() Ergebnis abrufbar
  - Liefert false, falls Ergebnis ein Update count ist

PreparedStatement (1/2)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- **`PreparedStatement`** Objekt enthält vorübersetzte SQL-Befehle
- geeignet, wenn Statement mehr als einmal ausgeführt werden muss
- **`PreparedStatement`** kann Variablen enthalten, die jedesmal bei Ausführung definiert werden
- Ansatz: Erzeuge **`PreparedStatement`**, identifiziere Variablen mit **`?`**

```
PreparedStatement pstmt =
    con.prepareStatement ("UPDATE Bestellend "
                        + "  SET Status = ? "
                        + "  WHERE Umsatz > ?");
```

PreparedStatement (2/2)

1. Variablen-Werte übergeben
   `pstmt.setXXX(index,value);`
2. Statement ausführen
   `pstmt.executeQuery();`
   `pstmt.executeUpdate();`

```
int goldenerBestellend=42000;
PreparedStatement pstmt =
  con.prepareStatement ("UPDATE Bestellend "
       + "SET Status = ? where Umsatz > ?");
pstmt.setString(1, "Gold");
pstmt.setInt(2, goldenerBestellend);
pstmt.executeUpdate();
```

Fallstudie 1 Fallstudie 2 Fallstudie 3

Einschub: Transaktionen informell (später genauer)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- Datenbanken arbeiten standardmäßig mit lokalem Cache!

```
        Nutzung 1                          Nutzung 2
INSERT INTO T VALUES (42)
SELECT * FROM T
        T
        --                          SELECT * FROM T
        42                                T
                                          --

COMMIT

                                    SELECT * FROM T
                                          T
                                          --
                                          42
```

COMMIT

- Für **`Connection con`** ist automatisch ein „Auto-COMMIT“ eingestellt
- damit wird jede Änderung direkt auf DB ausgeführt
- dies kann sehr kritisch sein (siehe später)

- Mit **`con.setAutoCommit(boolean)`** einstellbar
- Starteinstellung über **`con.getAutoCommit()`** ermitteln (default: true !)

- **`con.commit()`** zum erfolgreichen Abschließen
- **`con.rollback()`** zum Verwerfen der Änderungen seit dem letzten Commit

Vergessene offene Verbindungen

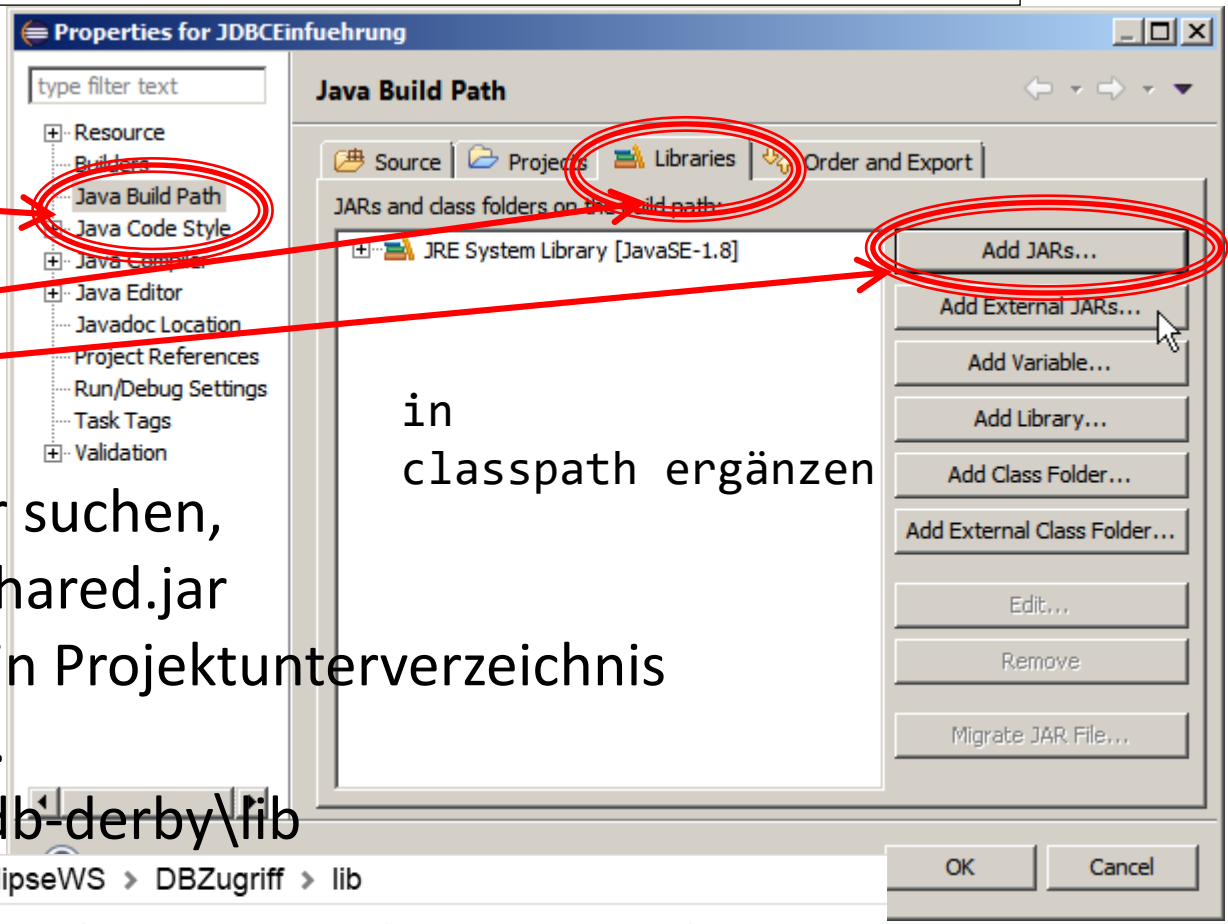
- Grundsätzlich alle Objekte der Typen Connection, Statement, ResultSet nach der Nutzung schließen

```
public void schliessen() {
  try {
    if (!this.stmt.isClosed()) {
      this.stmt.close();
    }
    if (!this.con.isClosed()) {
      this.con.rollback();
      this.con.close();
  } catch (SQLException ex) {
    Logger.getLogger(Main.class
          .getName()).log(Level.SEVERE, null, ex);
  }
}
```

Derby verlangt, dass offene Transaktionen geschlossen werden (bei meisten anderen DB und bei autoCommit(true) nicht notwendig)

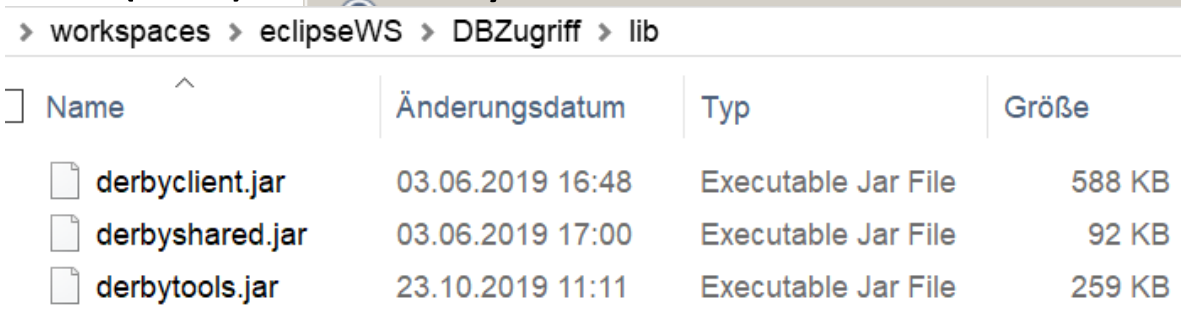
Einbindung von Derby-JDBC in Eclipse

Rechtsklick auf Projekt, dann Properies wählen, „Java Build Path“, dann Reiter Libraries, dann „Add JARs...“



in classpath ergänzen

Dann passenden Treiber suchen, derbyclient.jar + derbyshared.jar + derbytools.jar vorher in Projektunterverzeichnis lib kopieren, Quelle z. B. C:\Program Files (x86)\db-derby\lib



Beispiel: Konten (1/13) - Aufgabe

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Objekte der Klasse Konto (Kontonummer (nr) und aktueller Geldbetrag (betrag)) sollen persistierbar sein

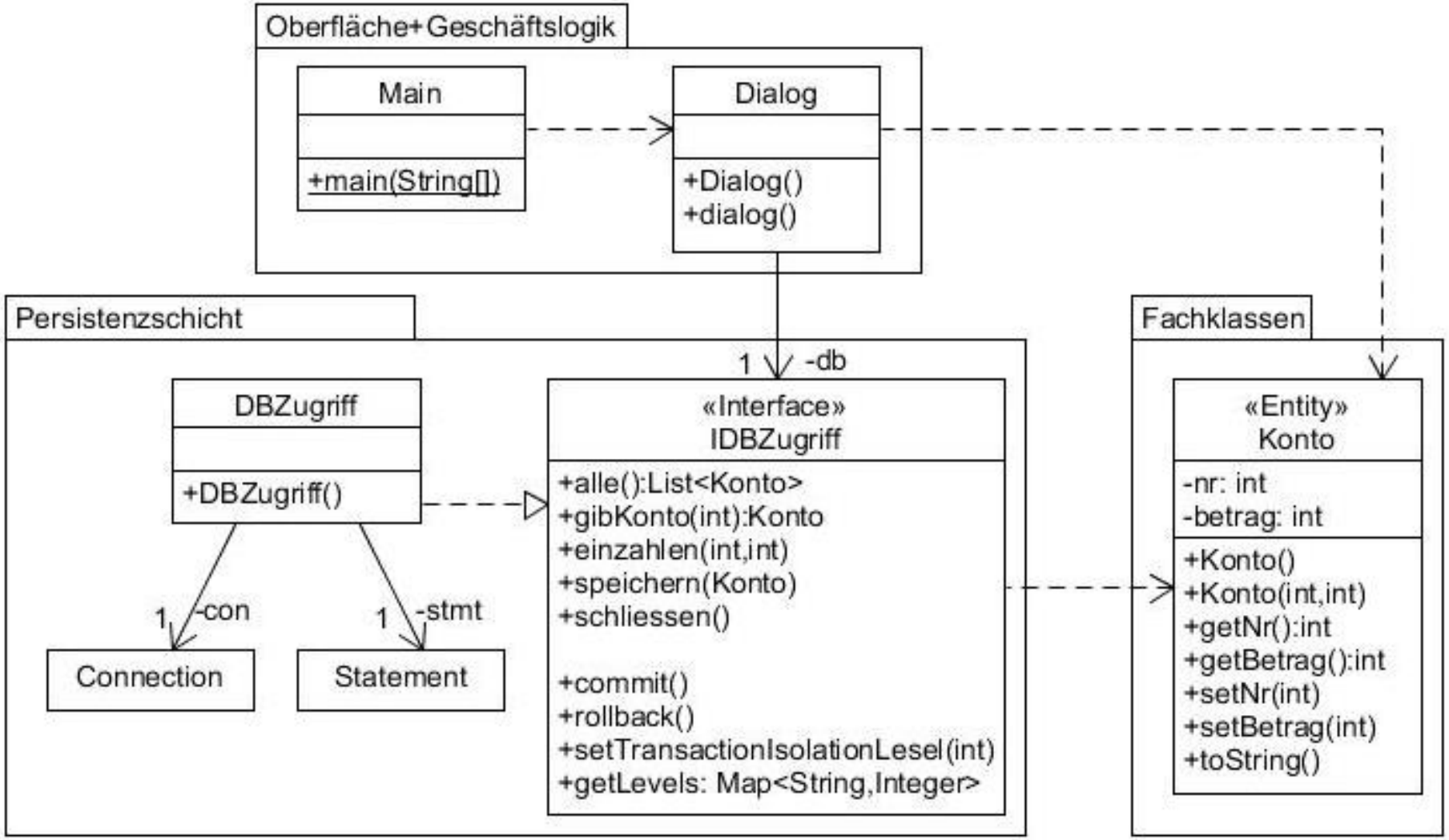
zentrale Use Cases:

- Anlegen eines neuen Kontos mit Startbetrag
- Einzahlen eines Betrags auf ein Konto
- Überweisen eines Betrags von einem Konto zum anderen
- (Individuelle Transaktionssteuerung)

Vereinfachungen

- Textschnittstelle (GUI zusammen mit Fachlogik)
- keine Validierung

Beispiel: Konten (2/13) - Architekturskizze



Beispiel: Konten (3/13) – Fachklasse Konto (1/2)

```
public class Konto {
    private int nr;
    private int betrag;

    public Konto(){}

    public Konto(int nr, int betrag) {
        this.nr = nr;
        this.betrag = betrag;
    }

    public int getNr() {
        return nr;
    }

    public void setNr(int nr) {
        this.nr = nr;
    }
```

klassische Bean, POJO (Plain Old Java Class)

- parameterloser Konstruktor
- get- und set- für alle Exemplarvariablen
- sinnvoll wären noch equals() und hashCode()
- (fehlt: besitzende Person(en))

Beispiel: Konten (4/13) – Fachklasse Konto (2/2)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
    public int getBetrag() {
        return betrag;
    }

    public void setBetrag(int betrag) {
        this.betrag = betrag;
    }

    @Override
    public String toString() {
        return this.nr + " " + this.betrag;
    }

    // equals und hashCode ggfls. sinnvoll
    // (Java besser immer)
}
```

Beispiel: Konten (5/13) – Schnittstelle Persistenz

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
public interface IDBZugriff {
  public List<Konto> alle();
  public void einzahlen(int nr, int betrag);
  public Konto gibKonto(int nr);
  public void speichern(Konto k);
  public void schliessen();
  // folgende Methoden ungewöhnlich; hier zum Experimentieren
  public void commit();
  public void rollback();
  public void setzeTransactionIsolationLevel(int level);
  public Map<String, Integer> getLevels();
}
```

Beispiel: Konten (6/13) – Persistenzklasse (1/5)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
public class DBZugriff implements IDBZugriff {

    private Connection con;
    private Statement stmt;
    private Map<String,Integer> levels; // spaeter relevant

    public DBZugriff() {
        this.starten();
    }
```

- schliessen() schon bekannt, ergänzen: this.con.commit()
- boolean existiertTabelle(name) auch bekannt
- in allen folgenden Methoden try-catch-weggelassen, wird aber immer lokal gefangen und Meldung ausgegeben

Beispiel: Konten (7/13) – Persistenzklasse (2/5)

```
private void starten() {
  Class.forName("org.apache.derby.jdbc.ClientDriver")
       .newInstance();
  this.con = DriverManager.getConnection(
    "jdbc:derby://localhost:1527/Konten","kleuker", "kleuker");
  this.con.setAutoCommit(false);
  this.con.setTransactionIsolation(1);
  this.stmt = con.createStatement();
  this.levels = new HashMap<>();
  this.levels.put("NONE", Connection.TRANSACTION_NONE);
  this.levels.put("READ_COMMITTED"
     , Connection.TRANSACTION_READ_COMMITTED);
  this.levels.put("READ_UNCOMMITTED"
     , Connection.TRANSACTION_READ_UNCOMMITTED);
  this.levels.put("REPEATABLE_READ"
     , Connection.TRANSACTION_REPEATABLE_READ);
  this.levels.put("SERIALIZABLE„
     ,Connection.TRANSACTION_SERIALIZABLE);
}
```

rote Kästen notwendig; Rest für spätere Transaktionsanalyse

Beispiel: Konten (8/13) – Persistenzklasse (3/5)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
private void erzeugeFachklassen() {
  this.stmt.execute("CREATE TABLE Konto("
                    + "nr INTEGER PRIMARY KEY,"
                    + "betrag INTEGER)");
  this.con.commit();
}

@Override
public List<Konto> alle() {
  List<Konto> erg = new ArrayList<>();
  ResultSet rs = stmt.executeQuery("SELECT * FROM Konto");
  while (rs.next()) {
    erg.add(new Konto(rs.getInt(1), rs.getInt(2)));
  }
  return erg;
}
```

Beispiel: Konten (9/13) – Persistenzklasse (4/5)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
@Override
public Konto gibKonto(int nr){
  ResultSet rs = this.stmt.executeQuery(
           "SELECT COUNT(*) FROM Konto WHERE nr=" + nr);
  rs.next();
  if(rs.getInt(1) == 0){
    return null;
  }
  rs.close();
  rs = this.stmt.executeQuery("SELECT * FROM Konto WHERE nr="+nr);
  rs.next();
  return new Konto(rs.getInt(1), rs.getInt(2));
}

@Override
public void speichern(Konto k) {
  this.stmt.execute("INSERT INTO Konto VALUES(" + k.getNr()
                                    + "," + k.getBetrag() + ")");
}
```

Beispiel: Konten (10/13) – Persistenzklasse (5/5)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
@Override
public void einzahlen(int nr, int betrag) {
  if (this.gibKonto(nr) != null){
    this.stmt.execute("UPDATE Konto SET betrag = betrag +"
                      + betrag + " WHERE nr=" + nr);
  }
}

@Override  // folgende Methoden fuer „spaeter“
public void setzeTransactionIsolationLevel(int level){
  this.con.setTransactionIsolation(level);
}
@Override  public void commit() {
  this.con.commit();
}
@Override  public void rollback() {
  this.con.rollback();
}
```

Beispiel: Konten (11/13) – Oberfläche (1/3)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
public class Dialog {
    private IDBZugriff db;

    public Dialog(){
        this.db = new DBZugriff();
        this.dialog();
    }

    public void dialog() {
        int auswahl = 0;
        Scanner scanner = new Scanner(System.in);
        while (auswahl != -1) {
            System.out.println("Was wollen Sie machen?\n"
                    + " (-1) Programm beenden\n"
                    + " (1) vorhandene Konten\n"
                    + " (2) Neues Konto\n"
                    + " (3) Einzahlen (Auszahlen)\n"
                    + " (4) Überweisen\n"
```

Beispiel: Konten (12/13) – Oberfläche (2/3)

```
            + " (5) COMMIT\n"
            + " (6) ROLLBACK\n"
            + " (7) Ändere Transaction Isolation Level");
    auswahl = scanner.nextInt();
    switch (auswahl) {
        case 1: {
           for(Konto k: this.db.alle()){
               System.out.println(k);
           }
           break;
        }
        case 2: {
            System.out.print("Welche Kontonummer? ");
            int nr = scanner.nextInt();
            System.out.print("Welcher Startbetrag? ");
            int betrag = scanner.nextInt();
            this.db.speichern(new Konto(nr, betrag));
            break;
        }
```

Beispiel: Konten (13/13) – Oberfläche (3/3)

```
        case 4: {
            System.out.print("Von welcher Kontonummer? ");
            int nr1 = scanner.nextInt();
            System.out.print("Auf welche Kontonummer? ");
            int nr2 = scanner.nextInt();
            System.out.print("Welchen Betrag? ");
            int betrag = scanner.nextInt();
            this.db.einzahlen(nr1, -betrag);
            this.db.einzahlen(nr2, betrag);
            break;
        }
        case 5: {
            this.db.commit();
            break;
        }
    }
  }
  this.db.schliessen();
}
```

Variante ohne DB (1/5)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Theoretisch kann ohne DB gearbeitet werden
- Daten einfach in Datei abspeichern
- Varianten: Binärcode, XML, JSON, …

XML: eXtensible Markup Language

- Aufbau eines Elements mit Start- und End-Tags als Klammer

```
<Elementname> Inhalt </Elementname>
```

- Inhalt kann wieder aus Elementen bestehen, es ergibt sich Baumstruktur
- Elemente können Attribute enthalten

```
<Elementname att1="bla" att2="blubb" >
    Inhalt
</Elementname>
```

- Java unterstützt automatische XML-Umwandlung

Variante ohne DB (2/5)

```
public class XMLSpeicher implements IDBZugriff {

    private Map<Integer,Konto> konten;
    private final static String DATEI = "konten.xml";

    public XMLSpeicher() {
        this.konten = new HashMap<>();
        File f = new File(DATEI);
        if (f.exists()) {
            this.laden();
        }
    }

    private synchronized void speichern() {
        try (XMLEncoder out = new XMLEncoder(
                new BufferedOutputStream(
                    new FileOutputStream(DATEI)))) {
            out.writeObject(this.konten);
        } catch (FileNotFoundException e) {
        } //wegschauen
    }
```

Variante ohne DB (3/5)

```
private synchronized void laden() {
  try (XMLDecoder in = new XMLDecoder(
       new BufferedInputStream(new FileInputStream(DATEI)))){
    this.konten = (HashMap<Integer,Konto>) in.readObject();
  } catch (FileNotFoundException e) {
  } //wegschauen
}

@Override
public List<Konto> alle() {
  this.laden();
  return new ArrayList(this.konten.values());
}

@Override
public Konto gibKonto(int nr) {
  return this.konten.get(nr);
}
```

Variante ohne DB (4/5)

```
@Override
public void einzahlen(int nr, int betrag) {
    Konto k = this.konten.get(nr);
    if (k != null){
        k.setBetrag(k.getBetrag() + betrag);
    }
    this.speichern();
}

@Override
public void speichern(Konto k) {
    if(this.gibKonto(k.getNr()) == null){
        this.konten.put(k.getNr(), k);
        this.speichern();
    }
}
```

Variante ohne DB (5/5)

```
    @Override
    public void schliessen() {
    }

    // genauso rollback(), setzeTransactionIsolationLevel(int)
    // da fuer diesen Ansatz sinnlos
    @Override
    public void commit() {
        throw new UnsupportedOperationException(
                    "Not supported");
    }
}
```

Anmerkung: Ansatz extrem langsam, da ganze Datei geöffnet, gelesen, geschrieben wird; gibt schnelle File-basierte DB, die aber dann mit Betriebssystem zusammen Datei bearbeiten

Ausschnitt aus resultierender XML-Datei

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.8.0_45" class="java.beans.XMLDecoder">
 <object class="java.util.HashMap">
  <void method="put">
   <int>1</int>
   <object class="entities.Konto">
    <void property="betrag">
     <int>10011</int>
    </void>
    <void property="nr">
     <int>1</int>
    </void>
   </object>
  </void>
  <void method="put">
   <int>2</int>
   <object class="entities.Konto">
    <void property="betrag">
     <int>2</int>
```

10. Effiziente Datenverwaltung

Video (nur grober Überblick)

- Index
- ISAM
- B-Baum
- B*-Baum
- binärer Trie

nicht im Detail besprochen

Index

- wird erstellt, um den schnelleren Zugriff auf einzelne Spalten zu erlauben

```
CREATE INDEX <index_name>
 ON <table>(<col1>[,<coln>]);
```

- Index wird aufdatiert, wenn zugehörige Tabelle geändert wird (→ mehr Indexe heißt nicht unbedingt schnelleren Zugriff)
- wenn auf indizierte Spalten zugegriffen wird, wird der Index genutzt

Wann ist eine Index-Erstellung sinnvoll?

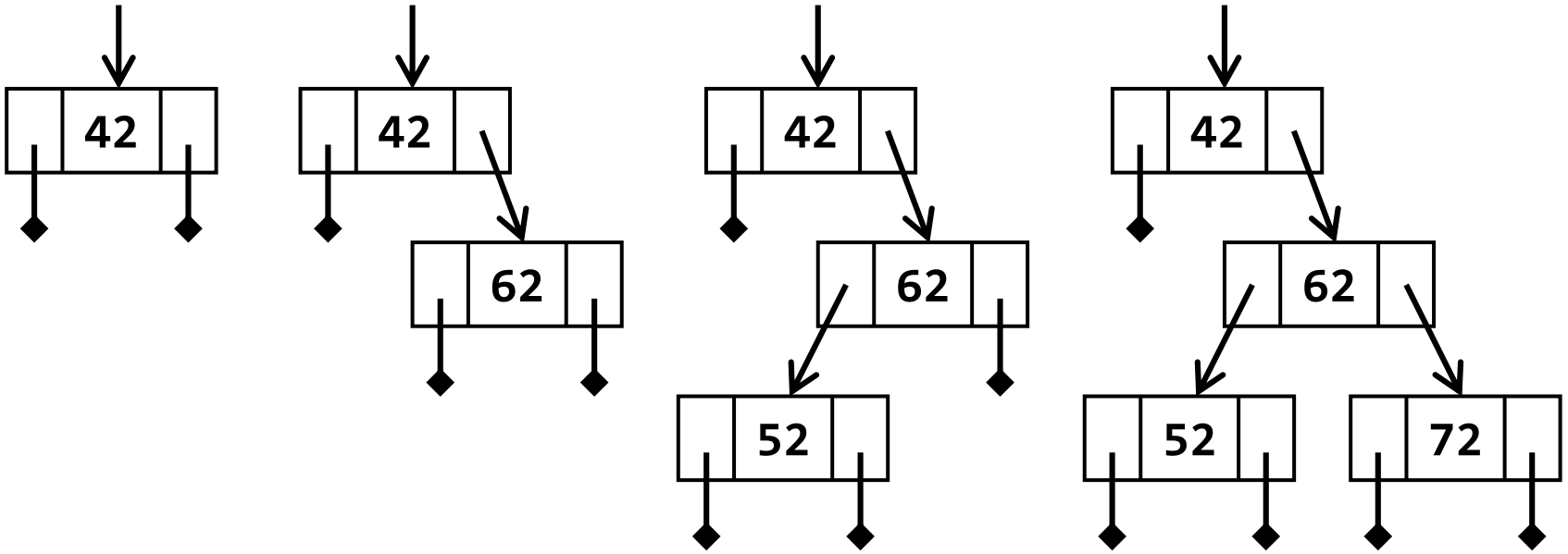
HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Das Attribut wird häufig in WHERE-Klauseln oder JOIN-Bedingungen benutzt
- Das Attribut hat einen großen Bereich von angenommenen Werten
- Die Spalten im Index werden sehr häufig in Anfragen zusammen verwandt
- Die zugehörige Tabelle ist sehr groß und die meisten Anfragen auf diese Tabelle liefern zwischen 2% und 4% der Zeilen als Antwort zurück

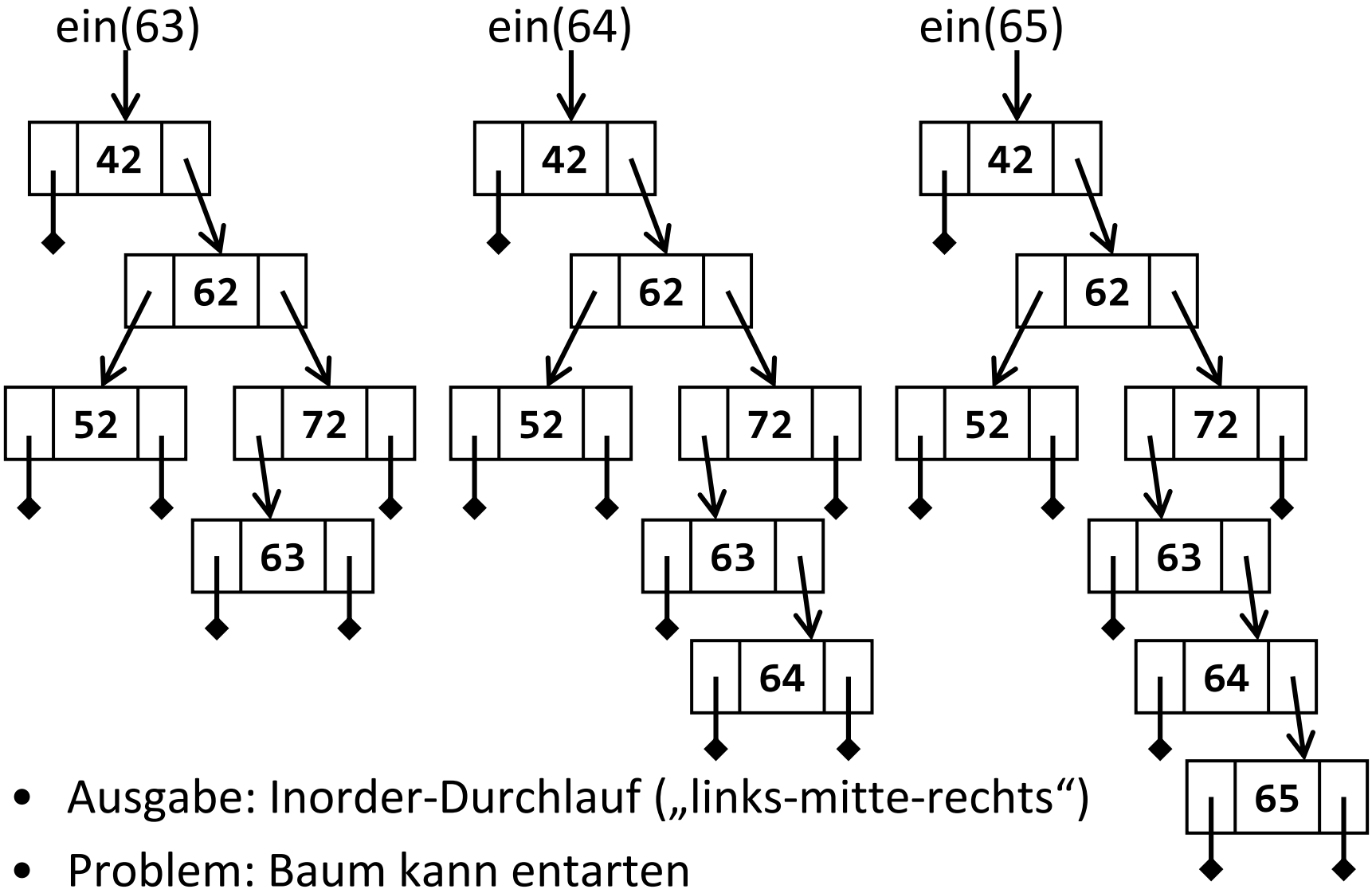
Erinnerung: Sortieren mit Bäumen (1/4)

- Ansatz: Knoten besteht aus Inhalt und linkem Zeiger auf Knoten mit kleinerem und rechtem Zeiger auf Knoten mit größeren Inhalt

- ein(42) ein(62) ein(52) ein(72)

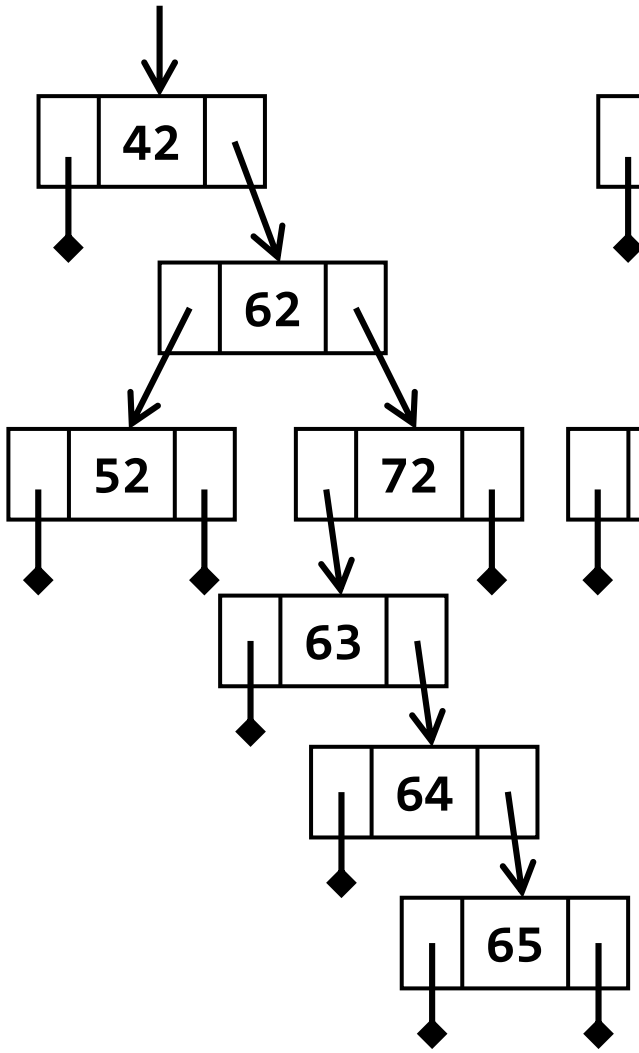


Erinnerung: Sortieren mit Bäumen (2/4)

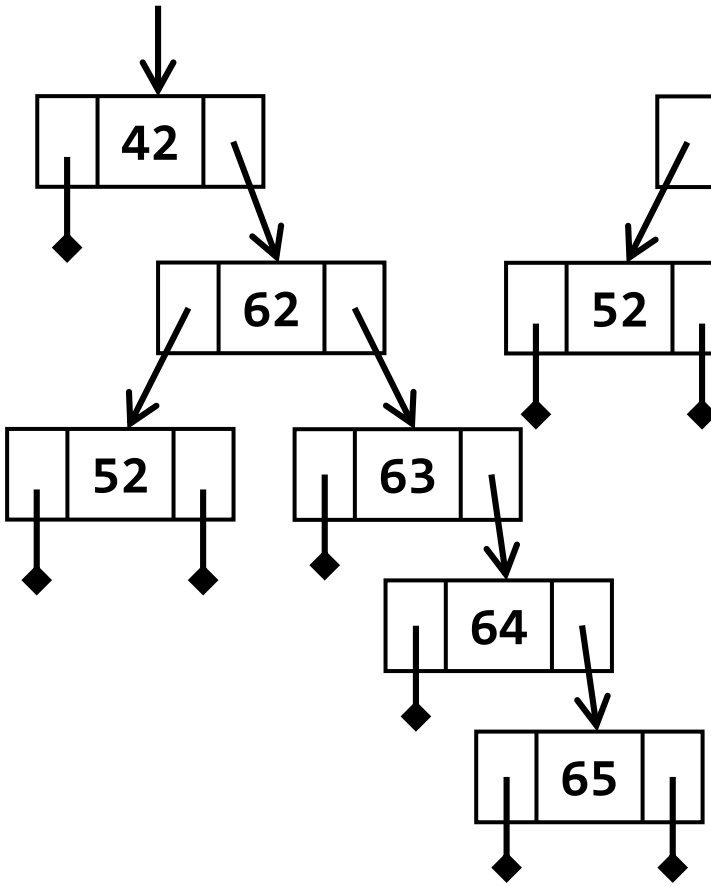


- Ausgabe: Inorder-Durchlauf („links-mitte-rechts“)
- Problem: Baum kann entarten

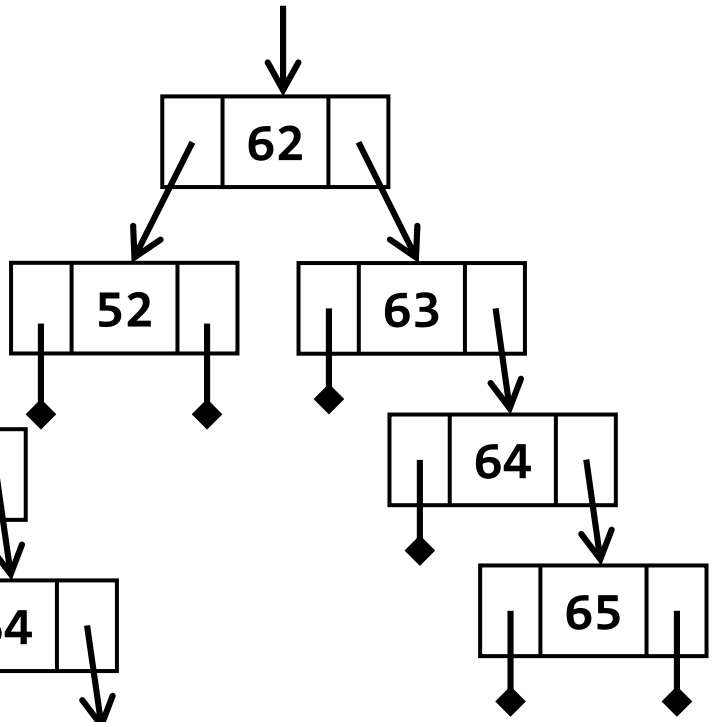
Erinnerung: Sortieren mit Bäumen (3/4)



loesche(72)



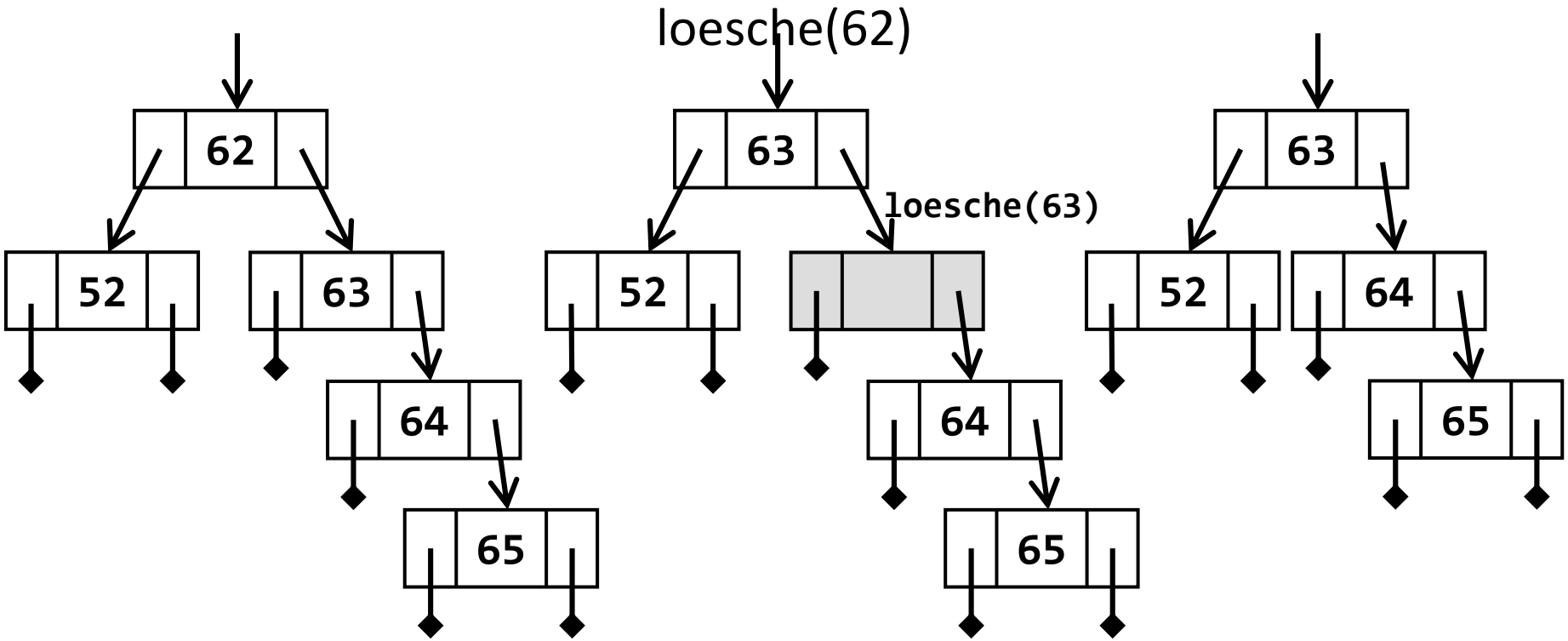
loesche(42)



Löschen: wenn keine zwei Nachfolger,

Knoten löschen (evtl. Verlinkung ändern)

Erinnerung: Sortieren mit Bäumen (4/4)



- Löschen: wenn zwei Nachfolger, durch kleinsten Nachfolger [ein Schritt nach rechts, dann solange wie möglich nach links] ersetzen (dadurch rekursiv)
- Abhilfe bei Entartung: AVL-Bäume
- Variante: m-näre Bäume

Bäume zur Datenverwaltung

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

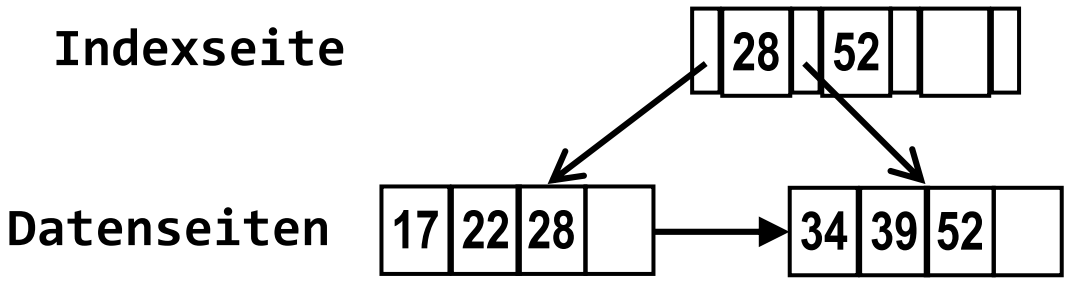
- Binär-Bäume und AVL-Bäume sind für „kleinere“ Speicherstrukturen gut nutzbar
- klein hier: im Hauptspeicher verwaltbar
- gezeigten Binär-Bäume für Datenhaltung auf Platten nicht anwendbar, da sehr viel geschrieben und gelesen werden muss

- Erinnerung: Plattenverwaltung passiert in Seiten
- Seiten werden mit Ersetzungsstrategie in Hauptspeicher geladen

- Ansatz: ein Knoten steht für eine Seite
  - dadurch Baum auch über mehrere Platten verteilbar
- (im erklärenden Beispiel, wieder Zahlen statt Seiten genutzt)

ISAM

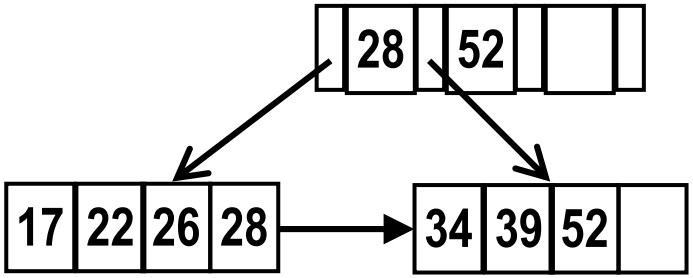
- Index Sequential Access Method
- zweistufige Verwaltung:
- Indexseite: Referenz auf Datenseiten
- Datenseiten: enthalten eigentliche Informationen



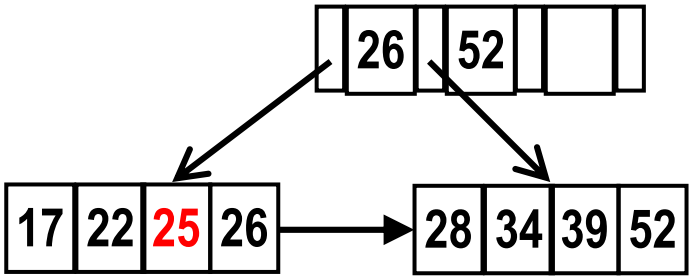
- Index und Datenseiten sind sortiert, zwischen zwei Indexwerten Referenz auf Datenseite
- linker Zeiger auf Werte kleiner gleich Index
- rechter Zeiger auf Werte größer Index
- Datenseiten untereinander verlinkt: erlaubt schnellen Durchlauf

Einfügen in ISAM (1/2)

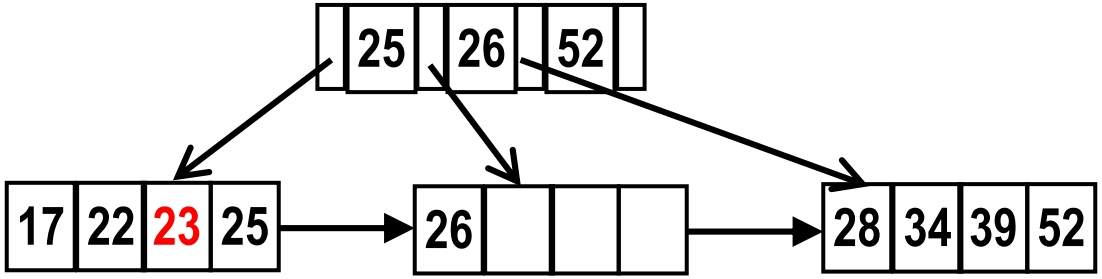
- Einfügen erfolgt orientiert an Indexseite
- ein(26)



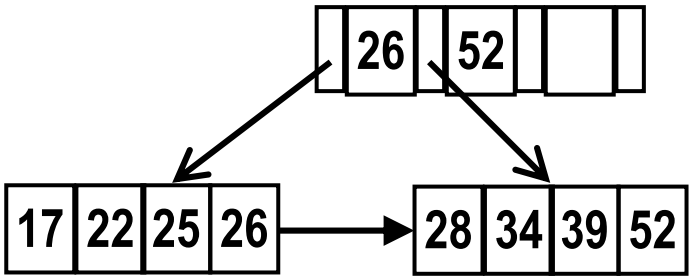
- ein(25): Datenseite voll, Nachfolger nicht, Ausgleich mit Folgeseite



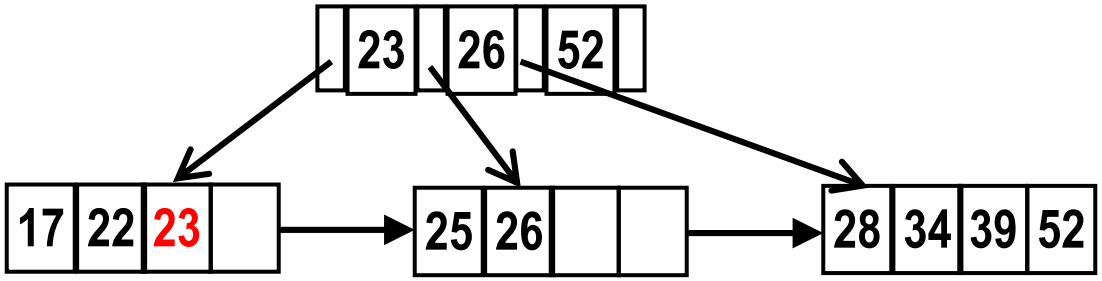
- ein(23): Datenseite voll, kein Ausgleich möglich, neues Blatt



Einfügen in ISAM (2/2)



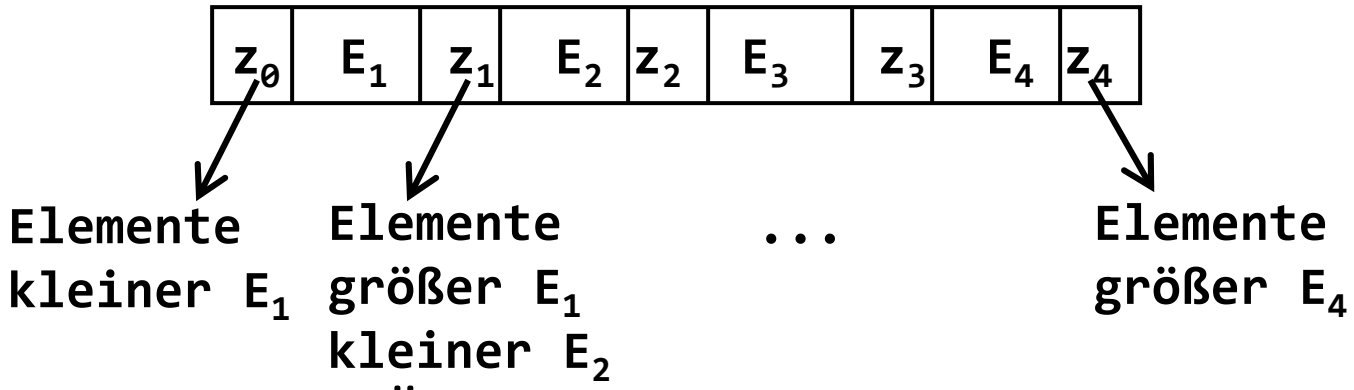
- ein(23) - Variante bei Erstellung neuer Seiten: Seitenausgleich



- Generelles Ziel möglichst viele gleichmäßig gut ausgelastete Seiten, neue Knoten immer rechenaufwändig
- Beim Löschen kann über Verschmelzungsalgorithmen nachgedacht werden
- Insgesamt ist ISAM bei sehr großen und sich dynamisch ändernden Daten wegen seiner nur zwei Schichten zu langsam

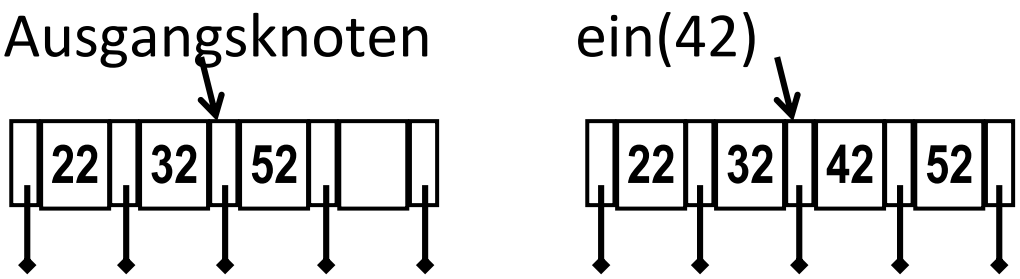
Idee von B-Bäumen

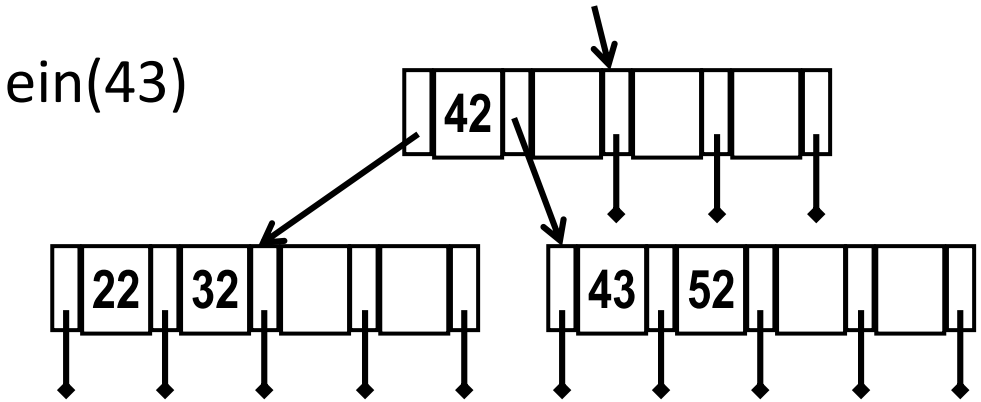
- Jeder Knoten kann maximal 2*k Elemente aufnehmen (B-Baum vom Grad k) , 1972 von Bayer und McCreight
- Jeder Knoten außer der Wurzel enthält mindestens k Elemente
- Die Elemente sind im Knoten sortiert
- Um jedes Element herum gibt es zwei Zeiger, der linke zeigt auf Knoten mit kleineren Werten, der rechte auf Elemente mit größeren Elementen (k=2)



- Klare Regeln, wie bei Überlauf und Unterlauf der Baum verändert wird

Einfügen in B-Bäumen (k=2) (1/3)



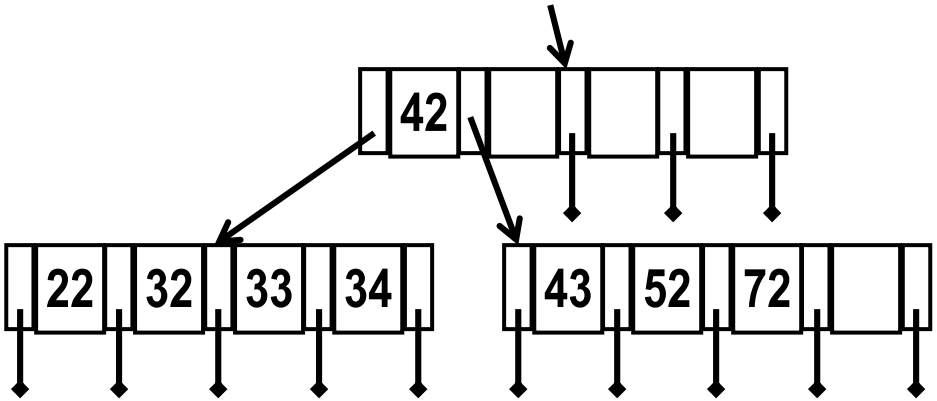


Beim Überlauf wird Knoten aufgeteilt:

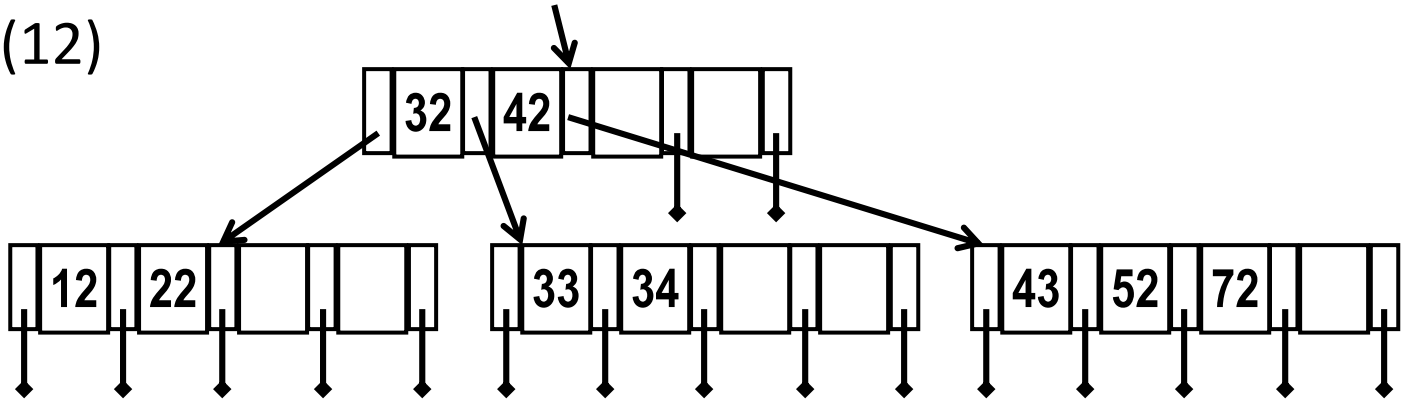
- Zwei neue Knoten, einer mit Elementen kleiner als die Mitte, einer mit Elementen größer als die Mitte
- „Mitte“ wandert einen Knoten nach oben, evtl. neue Wurzel

Einfügen in B-Bäumen (k=2) (2/3)

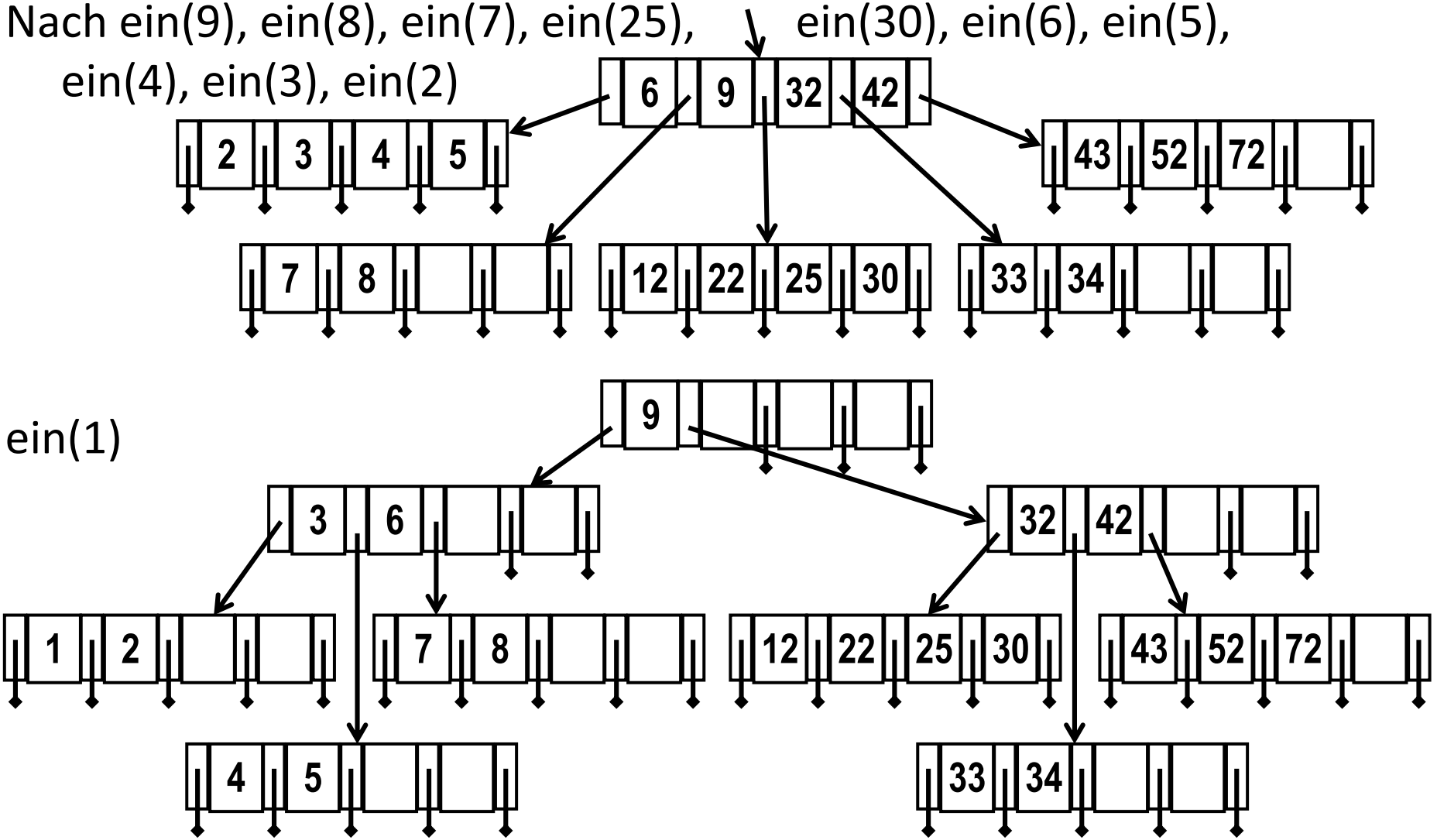
Nach ein(33), ein(34), ein(72)



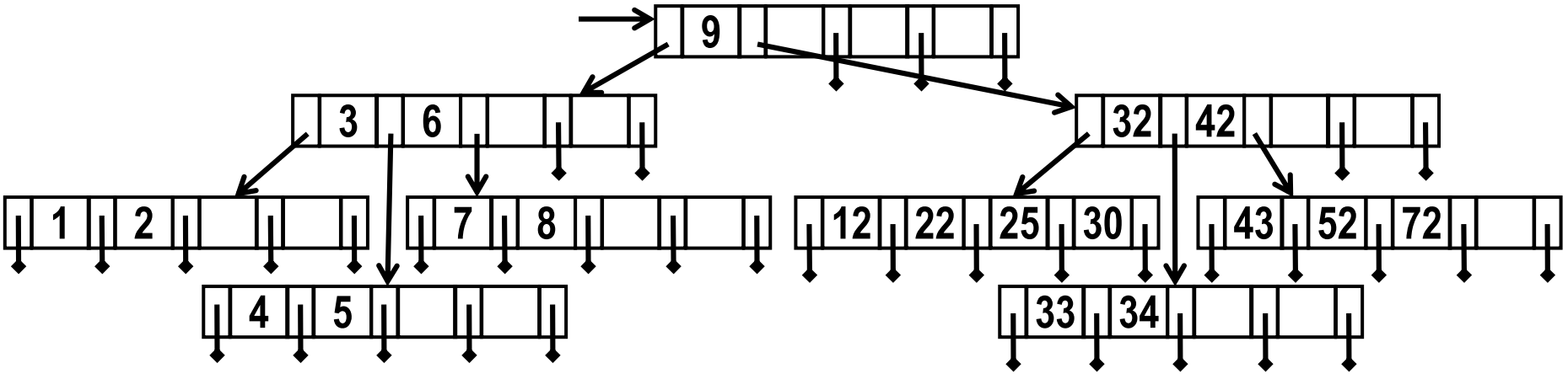
ein(12)



Einfügen in B-Bäumen (k=2) (3/3)

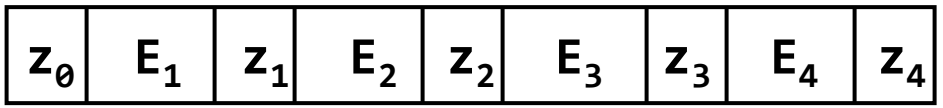


Suche



Suchen im Baum (Grad k=2, dann n=4): suche(x)

Suche x im Knoten



x gefunden, dann gefunden

x nicht gefunden und Knoten ist Blatt: dann nicht gefunden

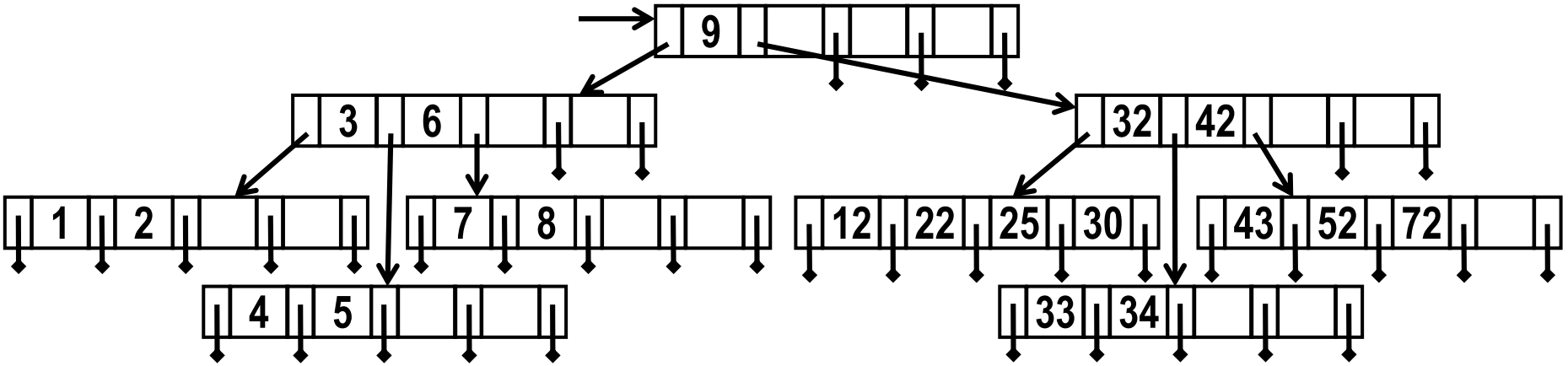
wenn Knoten nicht Blatt

und $x<E_1$, dann suche in Knoten, der an z_0 hängt

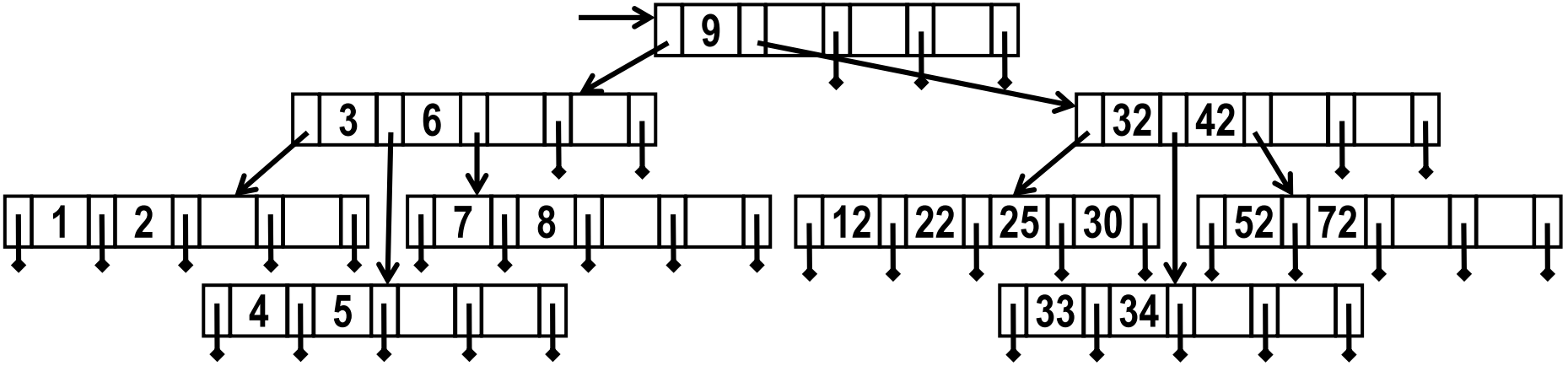
und $E_i<x<E_{i+1}$, dann suche in Knoten, der an z_i hängt

und sei E_i der maximale Inhalt und $E_i<x$, dann suche in Knoten, der an z_{i+1} hängt

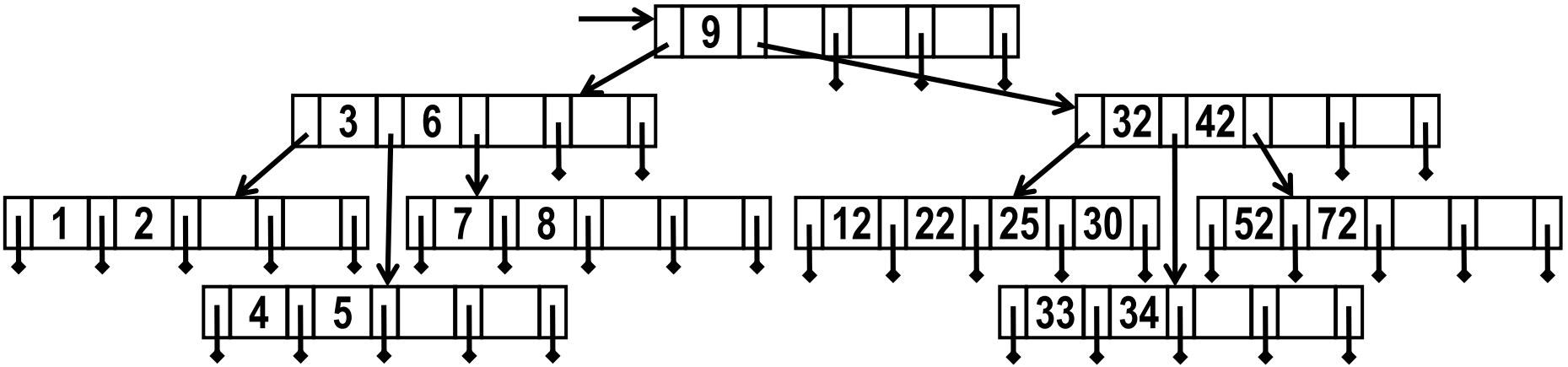
Löschen (1/5)



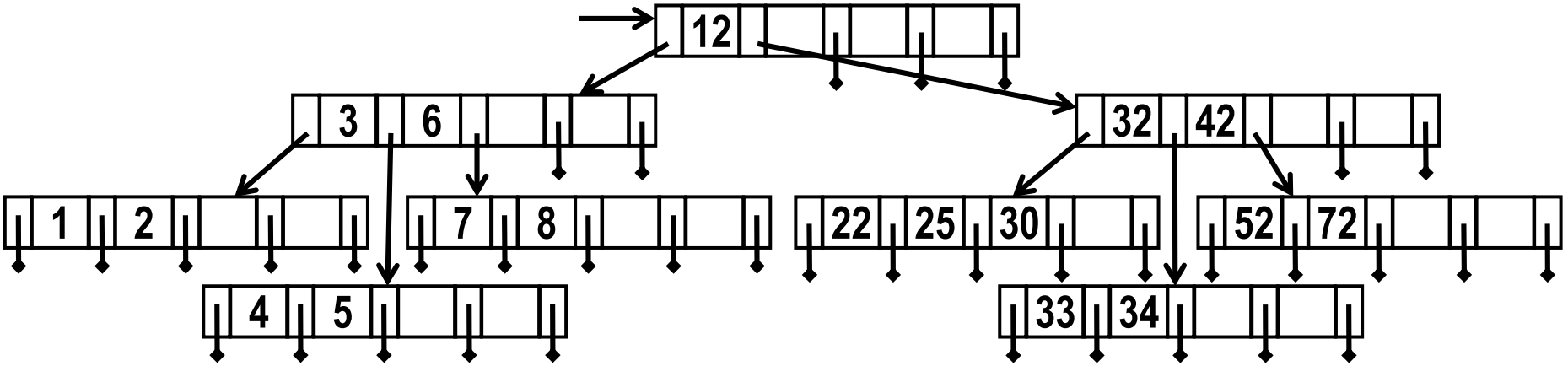
Einfacher Fall: Löschen im Blatt, kein Unterlauf: loesche(43)



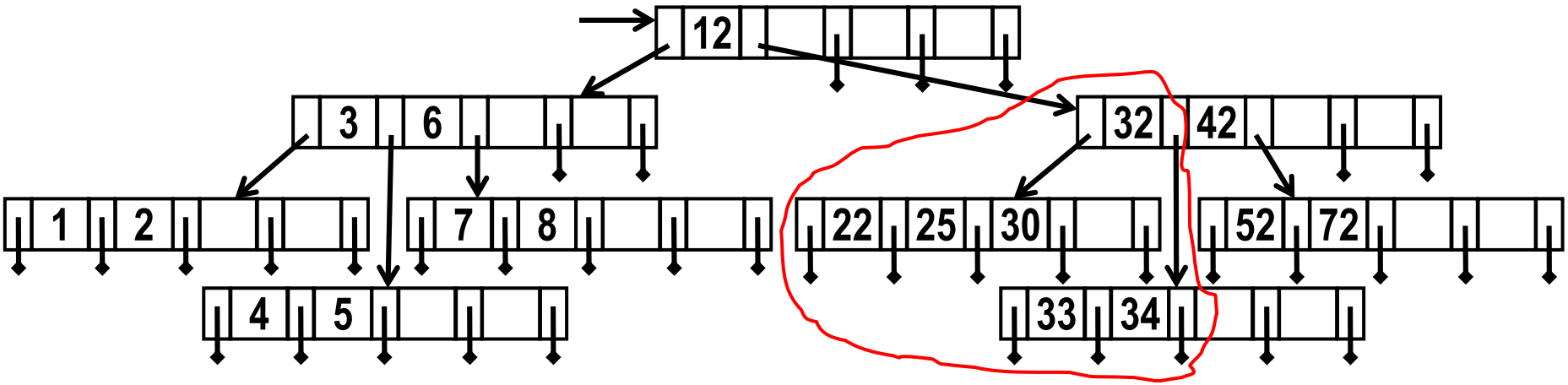
Löschen (2/5)



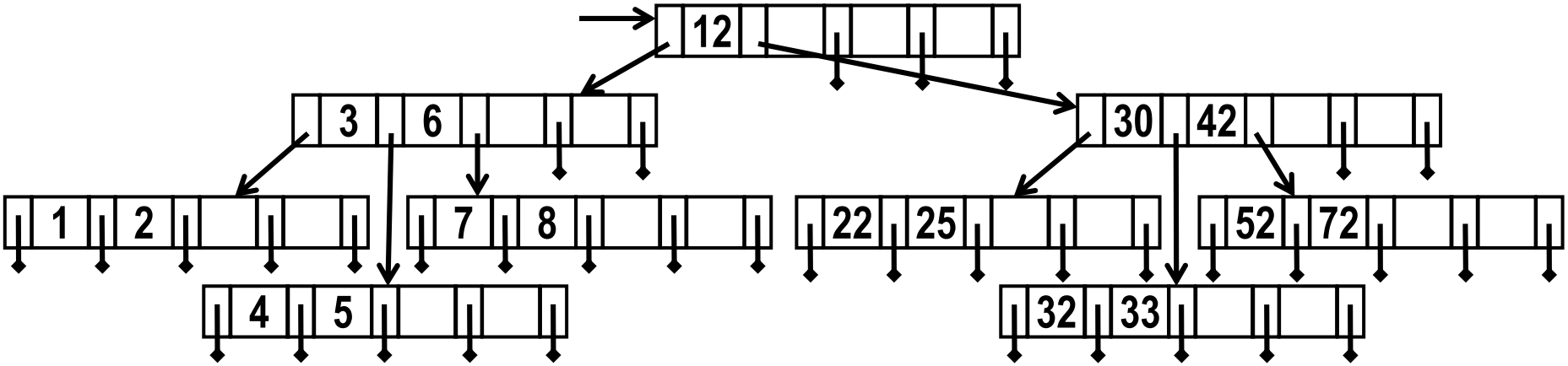
Relativ einfacher Fall: Löschen im Nicht-Blatt, ersetze durch kleinsten Nachfolger und es entsteht kein Unterlauf: loesche(9)
[Zeiger auf rechter Seite, dann solange wie möglich nach links]



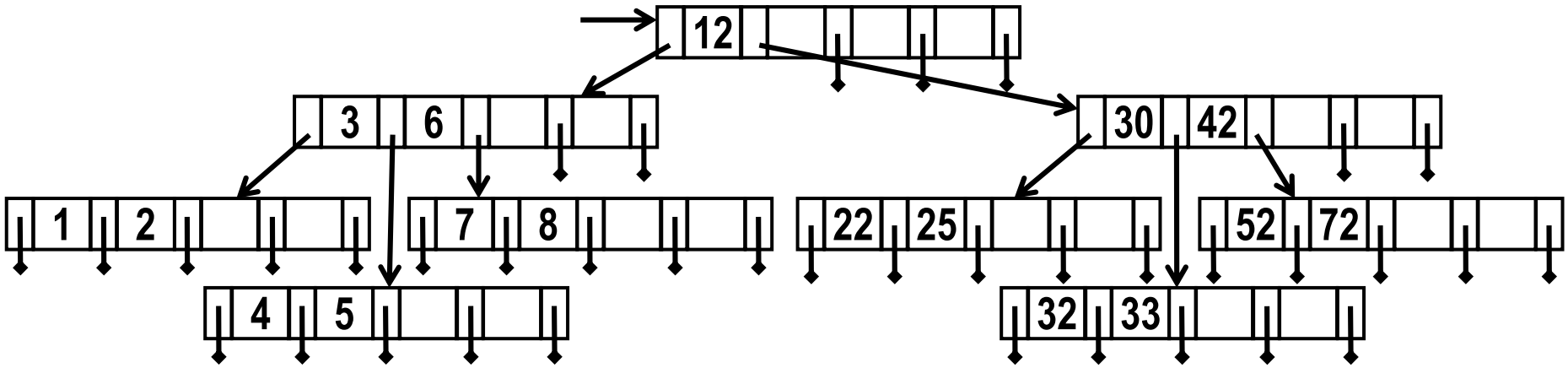
Löschen (3/5)



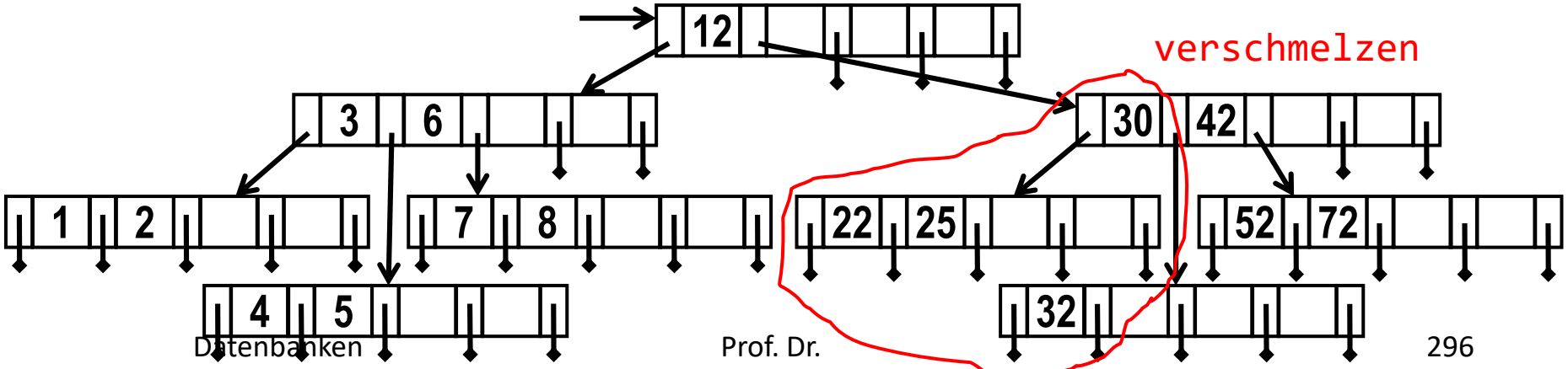
Bisheriges Löschverfahren führt zu Unterlauf im Blatt, dann schauen, ob man mit Nachbarn ausgleichen kann: loesche(34)



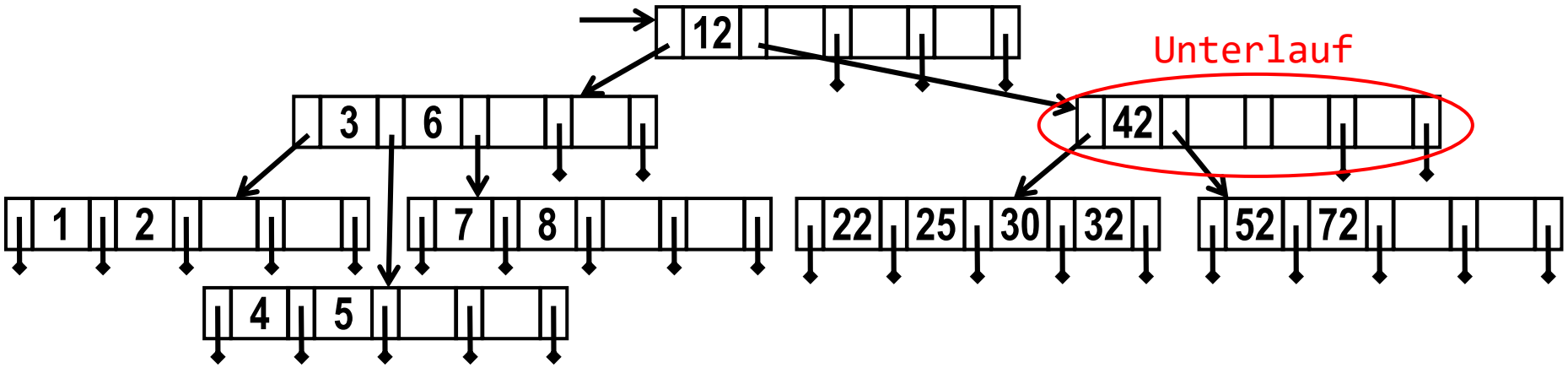
Löschen (4/5)



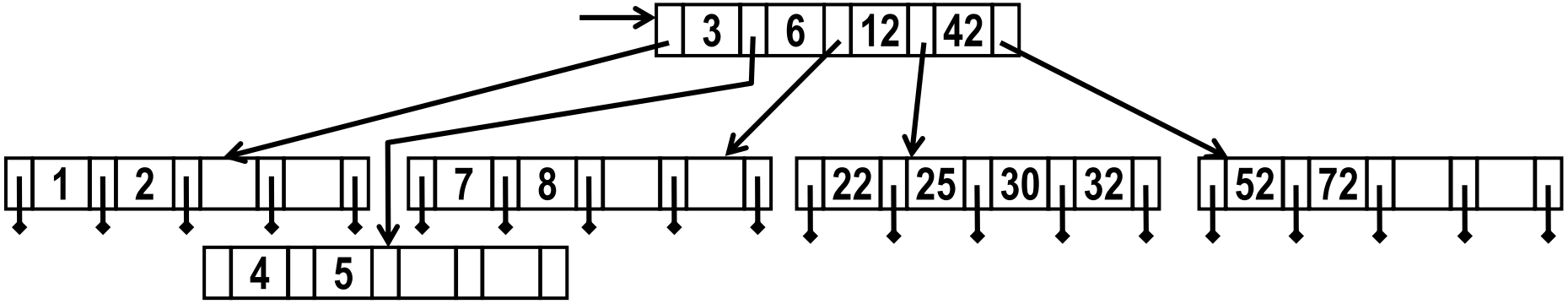
Bisheriges Löschverfahren führt zu Unterlauf im Blatt und kein Nachbar zum Ausgleichen da, dann verschmelze Knoten mit einem Nachbar und zugehörigem Element des Oberknotens (dies kann sich rekursiv fortsetzen) dann schauen, ob man mit Nachbarn ausgleichen kann: loesche(33)



Löschen (5/5)



Unterlauf: wieder versuchen auszugleichen mit Nachbarn (hier nicht möglich), erneut verschmelzen

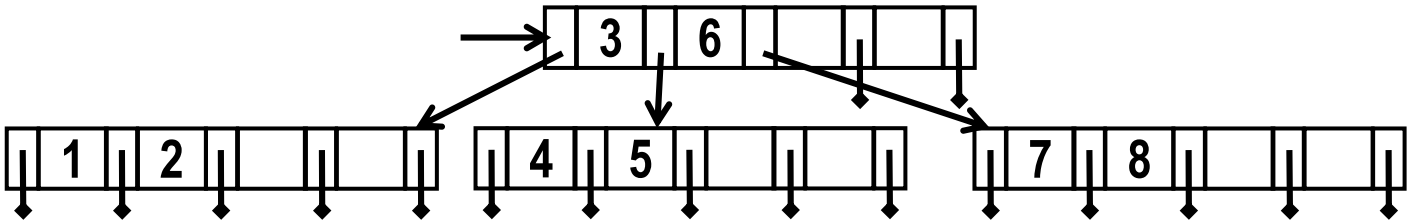


Zusammenfassung B-Baum

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Vorteile
  - Bis auf Wurzel ist jedes Blatt mindestens zu 50% gefüllt
  - Der Weg von der Wurzel zu jedem Blatt ist gleich lang
  - Schnelles Suchen, Recht Schnelles Einfügen (-> log n)
- Löschen kann aufwändig sein
- schmutziger Trick: beim Löschen Unterlauf akzeptieren, nutzender Person Möglichkeit zur Reorganisation der Daten bieten
- unschöner Spezialfall: wenn sich Einfügen und Löschen häufig abwechseln, kann Baum häufig aufwändig wachsen und gleich wieder schrumpfen
- Frage: Geht es besser als mindestens 50% Auslastung?
- Antwort: Ja, der Verschmelzungsalgorithmus wird angepasst (etwas aufwändiger, dafür mehr Auslastung, Knuth: 2/3-Auslastung)

Erinnerung zur Darstellung



- In Beispielen werden Zahlen als Elemente in B-Baum eingefügt (veranschaulicht Algorithmen)
- Real besteht Datensatz aus mehreren Elementen (eine Zeile einer Tabelle), z. B.

  3 Udo Otter Informatik m DE 1986
- 3 als Matrikelnummer ist eindeutig (Schlüsselkandidat); dieser Wert wird zur Einordnung des Datensatzes genutzt

B*-Baum

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

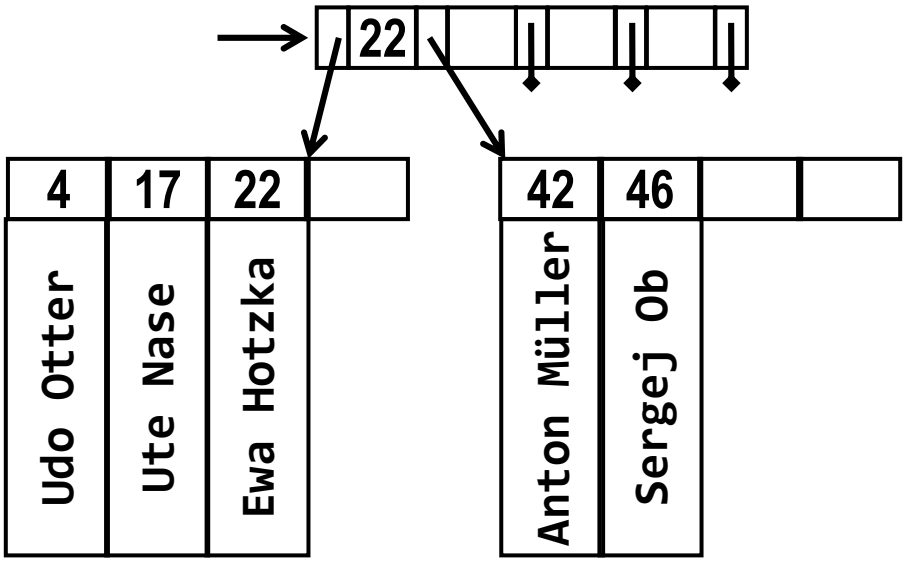
Ideen:

- Knoten speichern nur Index-Informationen
- Analog zu B-Baum, Zeiger mit links auf kleinere (oder gleiche), rechts auf größere Indices
- Alle Informationen werden in Blättern gespeichert
- Dadurch sind alle Wege zum Blatt gleich lang
- Einfügen: Analog zum B-Baum
- Löschen: Nur im Blatt Löschen (Unterlauf wird akzeptiert)
- Optimierung: Verkettung der Blätter zu linearer Liste
- Wichtig: Im folgenden stehen in Blättern nur Zahlenwerte, diese stehen für die vollständige Information
- In der Literatur auch B+-Baum genannt (B* ursprünglich von Knuth für Optimierung des B-Baums)

Einfügen in B*-Baum (1/3)

- Am Anfang Spezialfall: Wurzel=Blatt

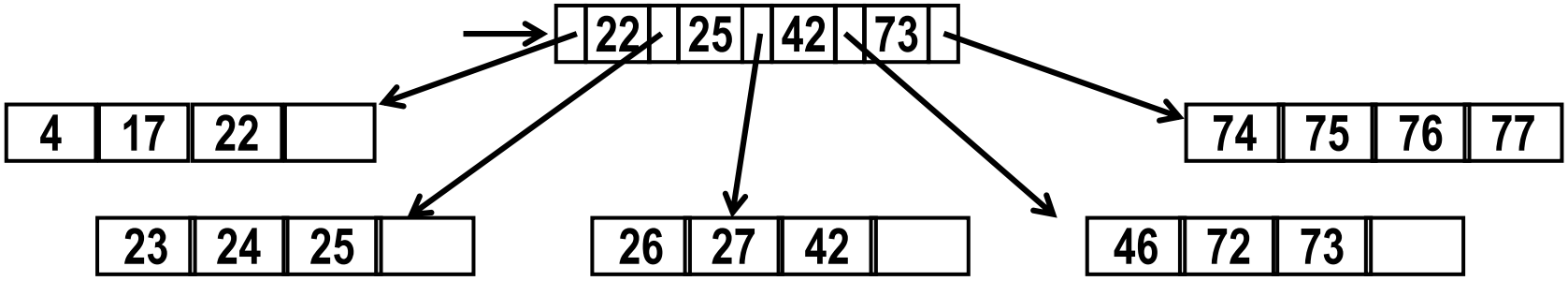
- ein(22)



- Da es sich im Inneren des Baumes nur um Index-Informationen handelt, wiederholen sich Werte, deren Inhalte in den Blättern stehen

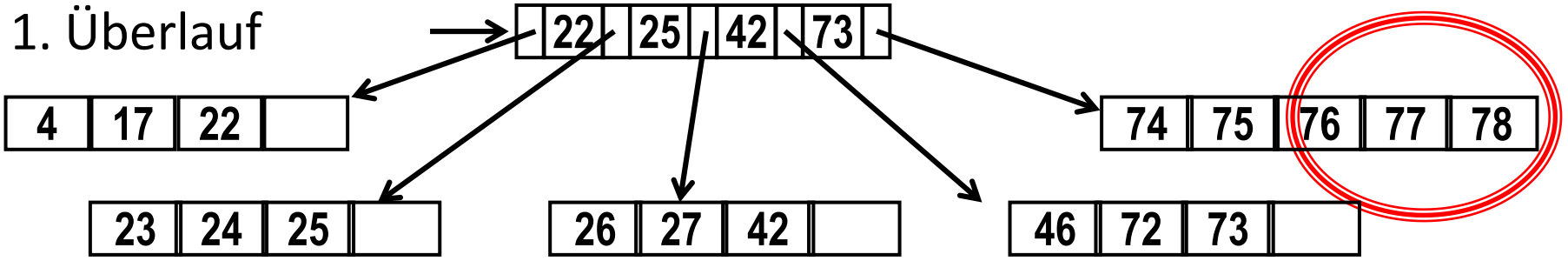
Einfügen in B*-Baum (2/3)

- nach ein(23),ein(24),ein(25),ein(26),ein(27),ein(72), ein(73), ein(74), ein(75), ein(76), ein(77)



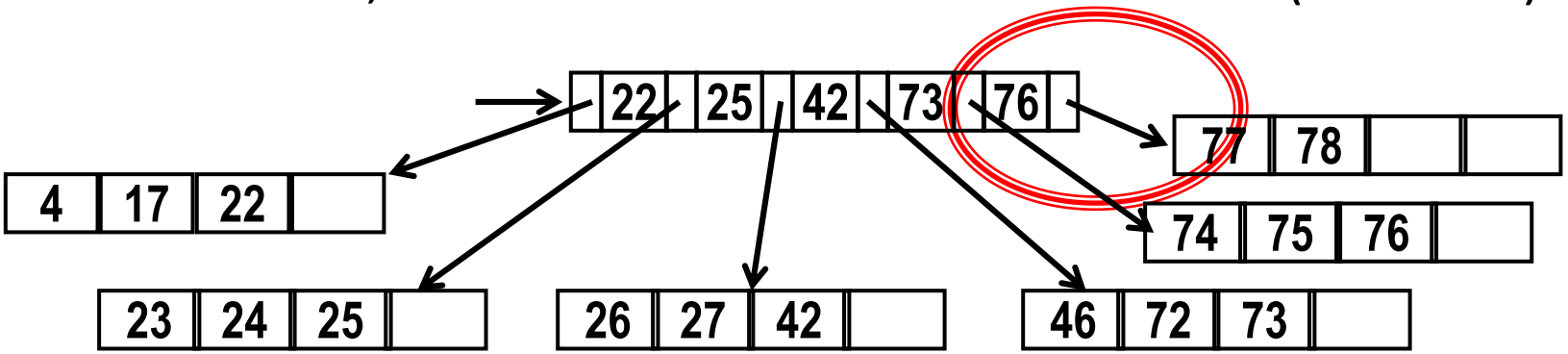
- ein(78): Baum wächst wie B-Baum nach oben zur Wurzel hin

1. Überlauf

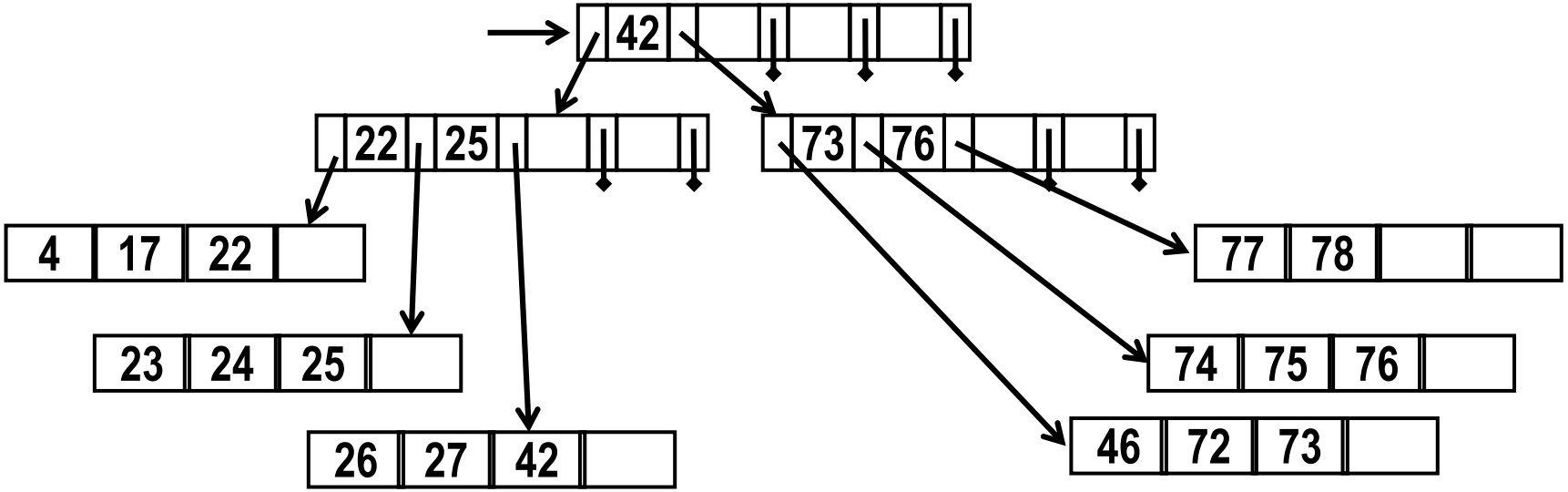


Einfügen in B*-Baum (3/3)

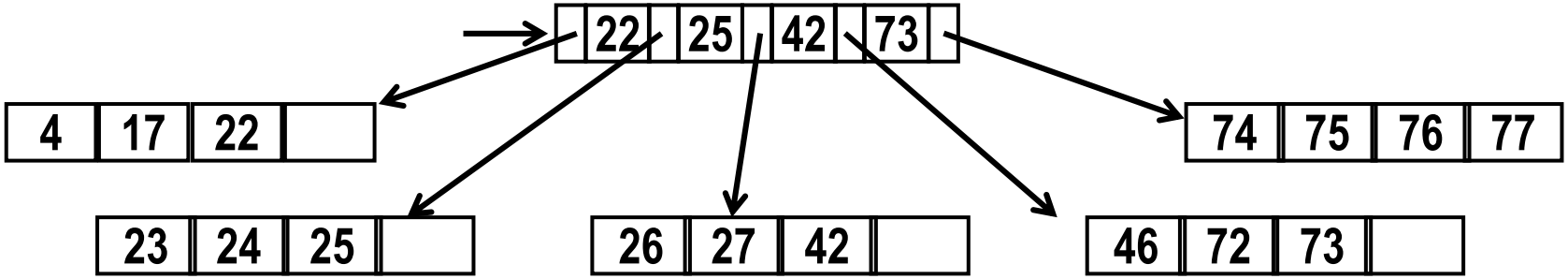
2. Blatt aufteilen, mittlerer Index wandert nach oben (Überlauf)



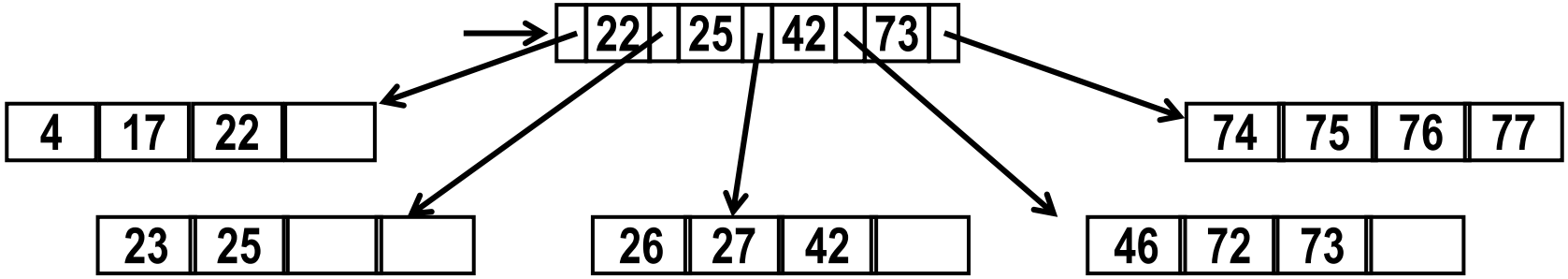
3. Aufspalten des inneren Knotens (evtl. neue Wurzel [wie hier])



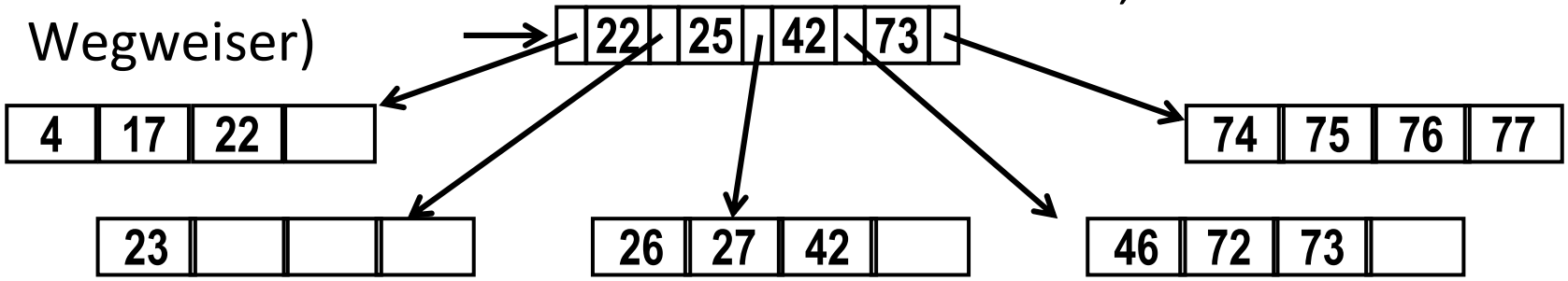
Löschen in B*-Bäumen [ohne Verschmelzen]



- loesche(24): einfach im Blatt löschen

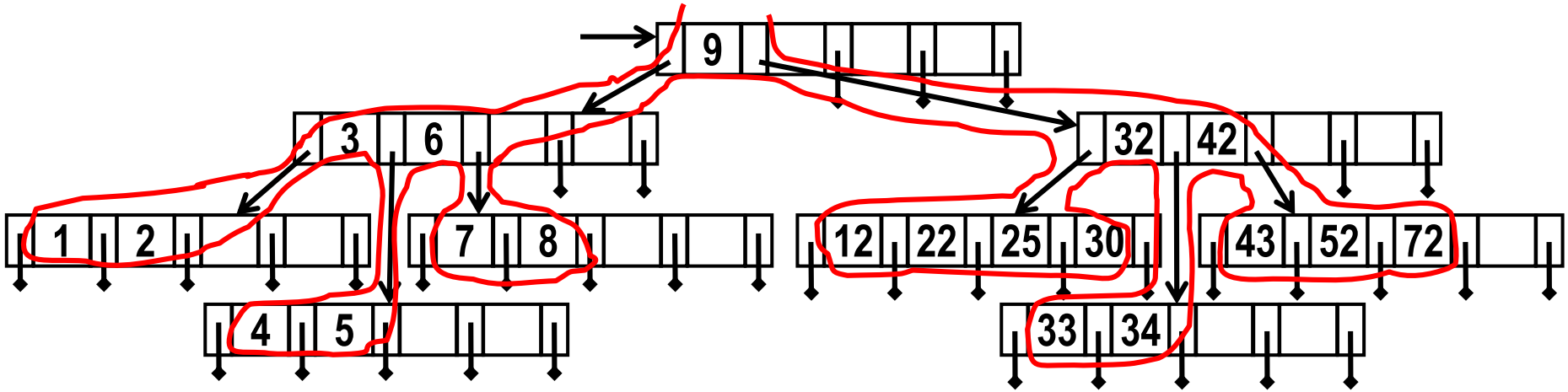


- loesche(25): einfach im Blatt löschen (Unterlauf ignorieren, Wert muss im Inneren nicht entfernt werden, da nur Wegweiser)

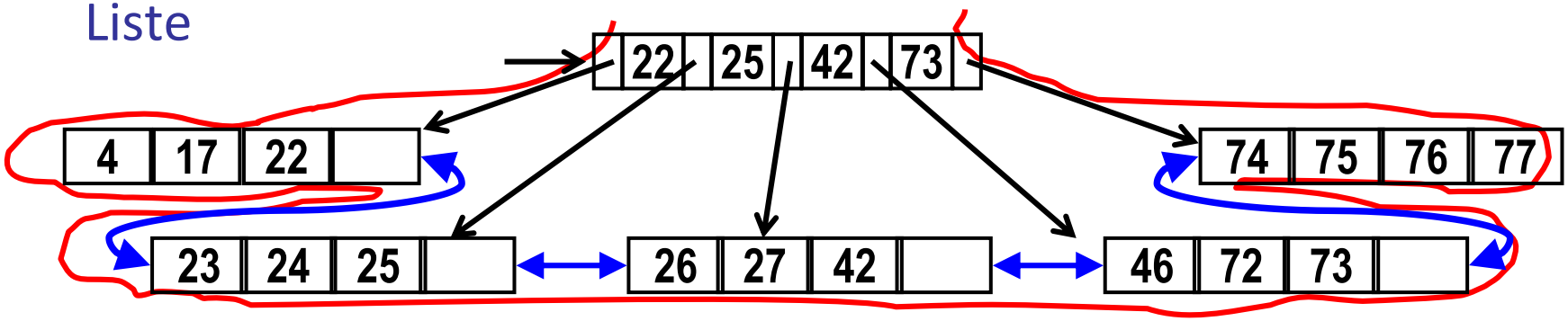


Effizientes Durchlaufen

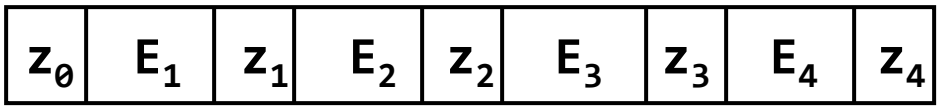
- Häufig sollen alle Daten (oft sortiert) abgearbeitet werden
- In B-Baum häufiges auf und ab, Knoten immer wieder laden



- Trick bei B*-Bäumen: Blattenden ergeben doppelt verkettete Liste



Generelle Anmerkungen zu B*-Bäumen



(k=2)

- Die Größe der Index-Knoten und der Blätter muss in keinem Zusammenhang stehen

Typische Definition: B*-Baum vom Typ (k,k^*), mit Eigenschaften:

- Jeder Weg von der Wurzel zu einem Blatt ist gleich lang
- Jeder innere Knoten (weitere Ausnahme die Wurzel) hat mindestens k und höchstens 2*k Einträge
- Jedes Blatt hat maximal 2*k^* Einträge
- Die Wurzel hat entweder maximal 2*k Einträge oder sie ist ein Blatt mit maximal 2*k^* Einträgen
- Jeder innere Knoten mit n Einträgen hat n+1 Kinder
- Zeiger z_0 zeigt auf Elemente kleiner-gleich Index-Wert E_1, z_{2*k} zeigt auf Elemente größer als Index-Wert E_{2k}, z_i zeigt auf Elemente kleiner-gleich Index-Wert E_{i+1}, und auf Elemente größer als Index-Wert E_i

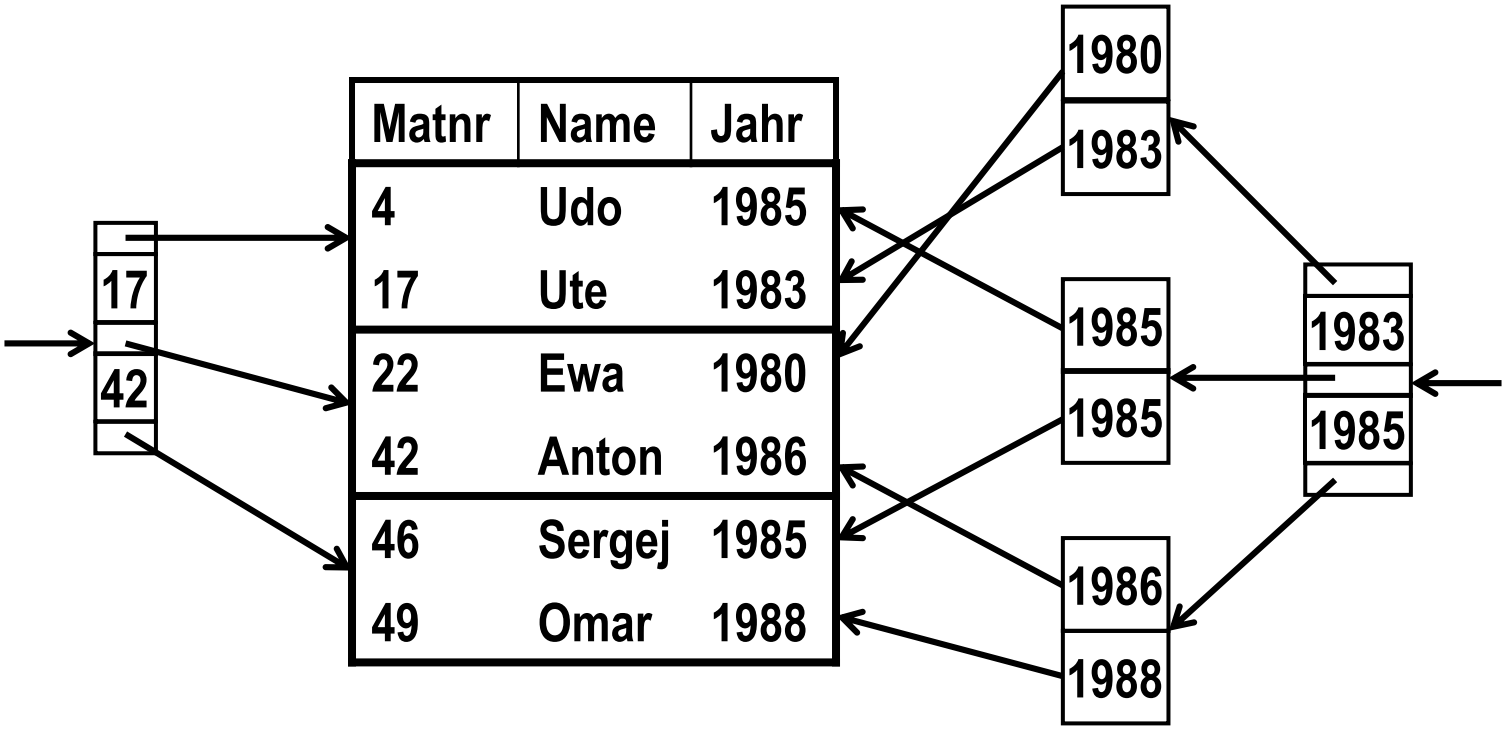
Index

- Häufig möchte man Daten nicht nur nach ihrem Schlüsselkriterium (z. B. Matrikelnummer) analysieren
- z. B. Überblick anhand von Geburtsjahrgängen gewünscht
- ist dies häufiger der Fall, wird effizienter Suchindex für dieses Attribut oder diese Attributkombination genutzt

- Index sinnvoll, wenn Attribute (Attributkombinationen) sehr häufig zum Suchen verwendet werden
- weiterhin sollten diese Attribute viele unterschiedliche Werte haben (z. B. Geburtsjahrgang, aber nicht Geschlecht)
- Index auf Attributen macht das Einfügen in Tabellen wesentlich langsamer, da alle Indices für dieses Attribut (bei Kombinationen mehrere möglich) aufdatiert werden müssen

Index mit B*-Bäumen

- Neben Datenverwaltung mit B*-Bäumen kann man diese auch für einen Index erstellen. Blätter enthalten dann statt Daten Referenzen auf Daten im anderen Baum

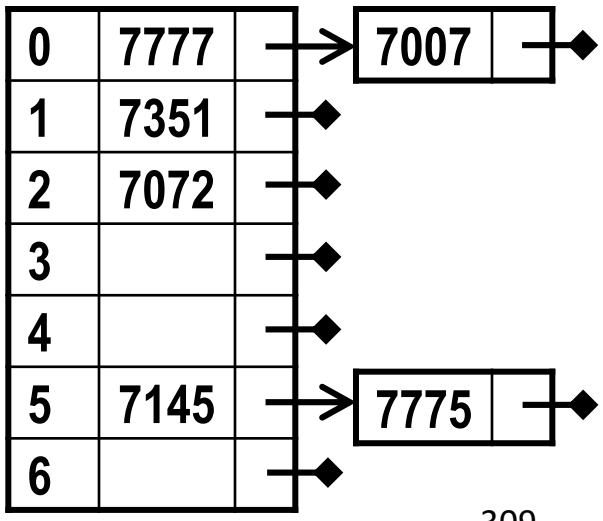


(Erinnerung) an Hash-Funktion

- Ansatz für schnelles Suchen und Ablegen von Daten
- Ansatz: Es gibt eine Menge B von Behältern (einfachste Variante Array), auf die Daten mit Hilfe einer Funktion h abgebildet werden h: Daten $\rightarrow$ B
- Beispiel: Abspeichern großer Zahlen, h: *N* $\rightarrow$ {0,...,6} mit h(x)= x mod 7, Abspeichern: 7777, 7072, 7351, 7145

0	7777
1	7351
2	7072
3	
4	
5	7145
6	

- Problem, wenn Behälter bereits besetzt
- Abspeichern von 7775 und 7007
- Lösungsansätze
  - wähle nächsten freien Platz
  - hänge Liste für Überlauf an (siehe Beispiel rechts)



Hash-Funktion und Index

- Hash-Funktion kann zur Indizierung eines Attributes genutzt werden
- h: Wertebereich $\rightarrow$ B
- statt gesamten Attributwert z. B. bei langen Strings zu betrachten, wird meist nur Teilstück genutzt
- Werden Tabellen über lange Zeit kontinuierlich gefüllt, sind Kollisionen wahrscheinlich (macht Index-Nutzung langsam)
  - Lösung 1: sehr große Hash-Tabellen (zu teuer)
  - Lösung 2: Ab bestimmten Füllungsgrad neue Hash-Funktion nutzen (machbar, wird auch genutzt, ist aber aufwändig, da jeder Wert neu platziert werden muss)
  - Lösung 3: vorgestelltes Verfahren heißt statisches Verfahren, Variante ist dynamisches (erweiterbares) Hashing

Erweiterbares Hashing (1/2)

- Nutzung einer Hashfunktion, Ergebnis wird binär interpretiert
- Immer minimale Anzahl binärer Stellen nutzen
- Jedem Hashwert wird ein Behälter bestimmter Größe für Daten (bzw. Referenzen) zugeordnet
- bei Kollision wird Hashtabelle ohne großen Rechenaufwand (aber Speicheraufwand) verdoppelt
- im Beispiel wird die Rückwärtsdarstellung des Binärformats der Zahlen genutzt (nur bedingt geeignet, da Bitmuster sehr ähnlich)
- Start: Behältergröße 2
- Einfügen 5 = 10100, d. h. Behälter 1
- Einfügen 6 = 01100, d. h. Behälter 0
- Einfügen 9 = 10010, d. h. Behälter 1

0	6 (01100)
1	5 (10100)
	9 (10010)

Erweiterbares Hashing (2/2)

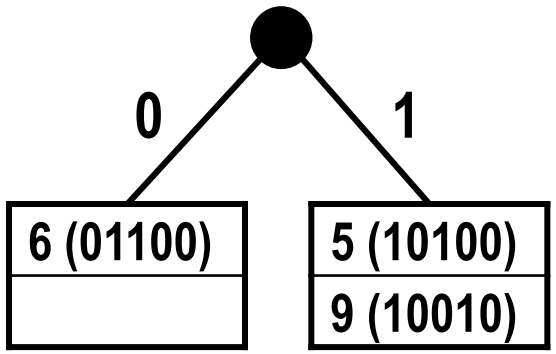
- Das Einfügen von 11 (11010) führt zu einem Überlauf des Behälters
- Hashwertebereich wird verdoppelt, wobei wieder zu jedem Wert ein Behälter gehört
- Alle (!) eingetragenen Werte müssen zunächst erneut betrachtet werden (allerdings einfach, da nur Bitmuster relevant)
- Es ergibt sich Hashtabelle auf rechter Seite
- Worst-Case: Alle Elemente des überfüllten Behälters würden zusammen mit neuem Element in einen Behälter fallen -> erneuter Überlauf

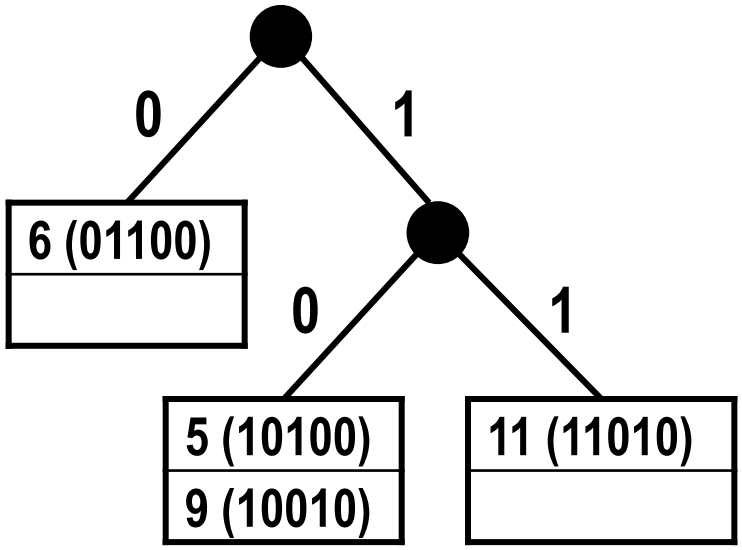
0	6 (01100)
1	5 (10100)
	9 (10010)

00	
01	6 (01100)
10	5 (10100)
	9 (10010)
11	11 (11010)

Variante für erweiterbares Hashing

- Statt die Behälterzahl zu verdoppeln, kann man dies auch nur für kritischen Behälter durchführen
- Man merkt sich Hashwert dazu in einem Baum
- ein(5), ein(6), ein(9), ein(11)

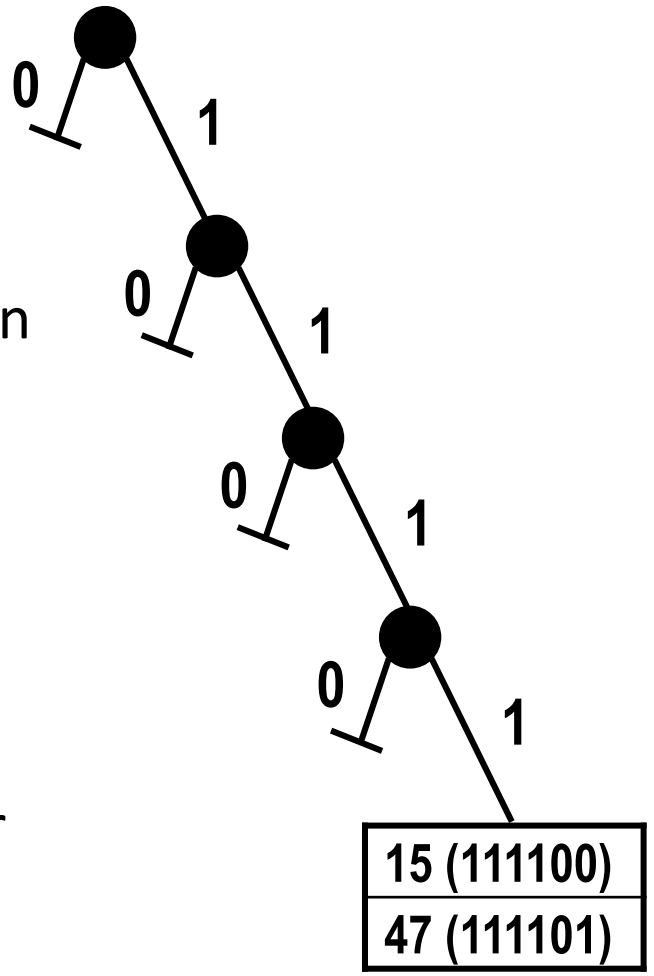




- heißt auch Entscheidungsbaum (binärer trie, von Re*trie*val)

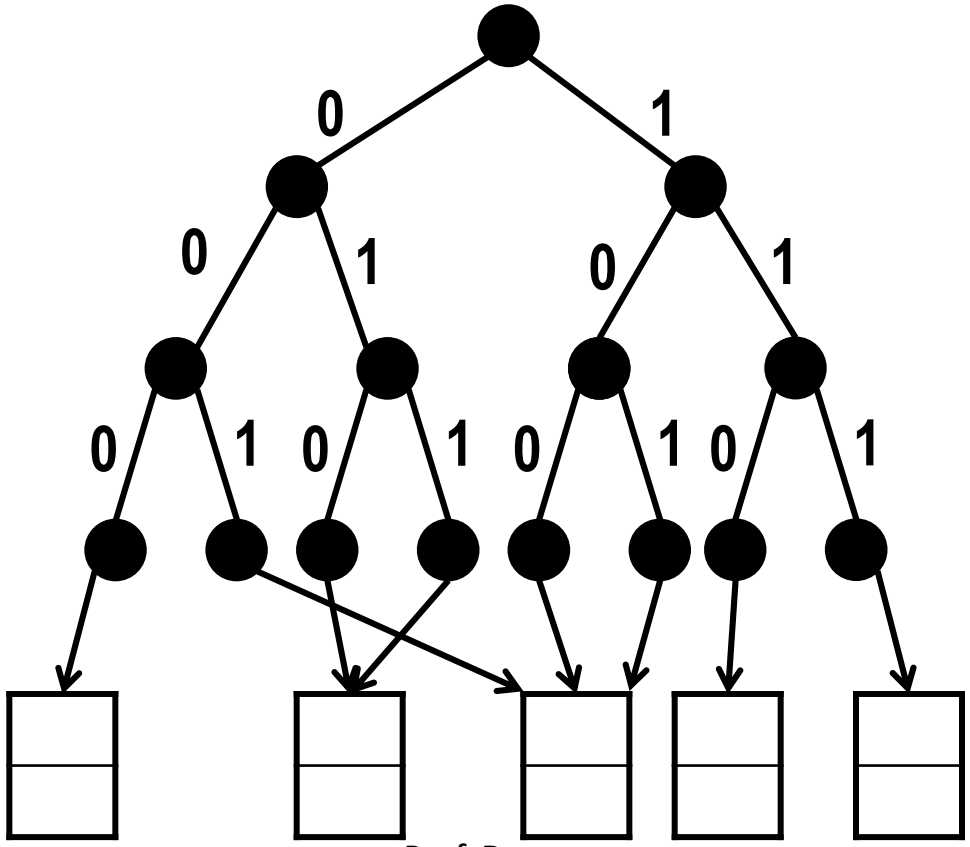
Besonderheit beim binären Trie

- ungeschickte Hash-Funktion kann zu langen Bäumen mit wenig Blättern führen
- Erinnerung: hier wir nur das Einfügen von Zahlen betrachtet, geht genauer um (Zahl, Inhalt) Paare
  - Inhalt kann Tabellenzeile sein
  - Inhalt kann Speicheradresse des eigentlichen Inhalts sein
  - (Inhalt könnte Kopie häufig genutzter Werte sein)
  - …



Variante der Variante für erweiterbares Hashing

- schlecht gefüllte Behälter sind immer ein Problem
- Ansatz: unterschiedliche Bit-Folgen führen zum gleichen Behälter
- Am Start bereits einzelne Bitfolgen zunächst auf gleichen Behälter abbilden



11. Programmierung in der Datenbank

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- Motivation für Programme in der Datenbank
- Aufbau von serverseitigen Programmen
- Ausnahmebehandlung

- Erstellung von Triggern

Erweiterung relationaler Datenbanken

- Einbettung von SQL in prozedurale oder objektorientierte Wirtssprachen *(embedded SQL);* meistens C, C++, oder Java (JDBC)
- Erweiterung von SQL um prozedurale Elemente *innerhalb* der SQL-Umgebung
- Vorteile der internen Erweiterung: Bessere Integration der prozeduralen Elemente in die Datenbank; Nutzung in Prozeduren, Funktionen und Triggern

Warum Erweiterung von SQL sinnvoll ist

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- keine prozeduralen Konzepte im klassischen SQL (Schleifen, Verzweigungen, Variablendeklarationen)
- viele Aufgaben nur umständlich über Zwischentabellen oder überhaupt nicht in SQL zu realisieren.
  - Transitive Hülle
- Programme repräsentieren anwendungsspezifisches Wissen, das nicht in der Datenbank enthalten ist

Ansätze zur Erweiterung

- DB-System erhält eigene Programmiersprache, z. B.
  - PL/SQL für Oracle
  - Transact SQL für MS SQL Server

- Variante: Bringe Programmiersprache mit Ausführungsumgebung in die Datenbank, z. B. virtuelle Maschine von Java läuft mit/ „im“ Datenbankserver
  - z. B. von Oracle und Derby unterstützt

Nutzung gegebener Funktionen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- parameterlose Funktionen werden ohne Klammern aufgerufen
- Beispiel: Aufrufe von fünf Funktionen

```
SELECT Current_date, Current_time
     , Current_timestamp, Upper(Continent.name), abs(area)
  FROM Continent;
```

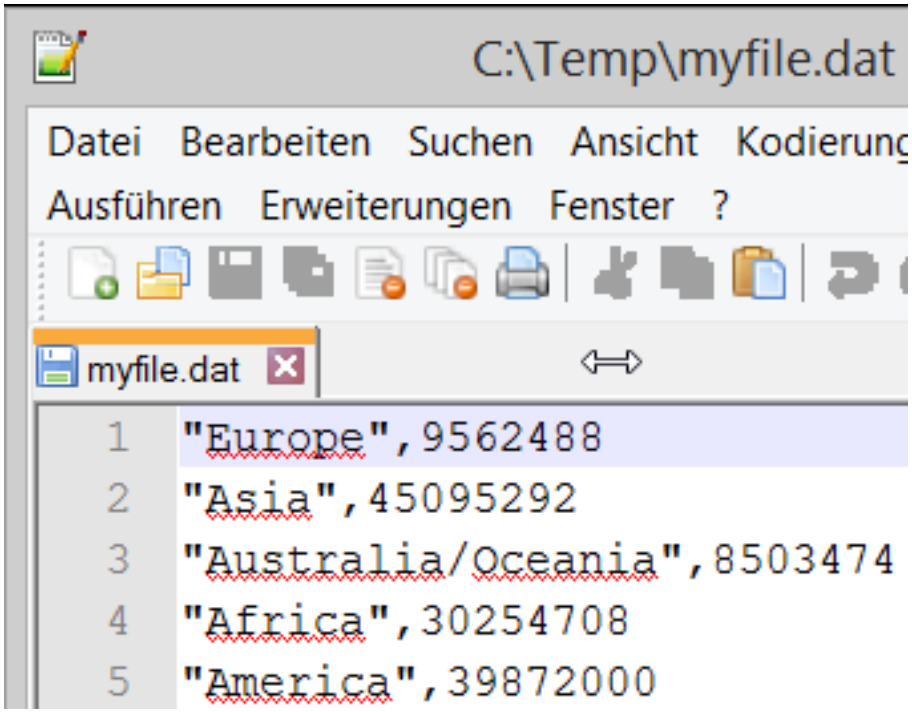
1	2	3	4	5
2015-08-20	14:13:44	2015-08-20 14:13:44.816	EUROPE	9562488
2015-08-20	14:13:44	2015-08-20 14:13:44.816	ASIA	45095292
2015-08-20	14:13:44	2015-08-20 14:13:44.816	AUSTRALIA/OCEANIA	8503474
2015-08-20	14:13:44	2015-08-20 14:13:44.816	AFRICA	30254708
2015-08-20	14:13:44	2015-08-20 14:13:44.816	AMERICA	39872000

- http://db.apache.org/derby/docs/10.14/ref/rrefsqlj29026.html

Nutzung gegebener Prozeduren

- sogenannte Stored Procedures, Aufruf mit CALL
- Parameter können IN, OUT, INOUT sein
- wenn nur IN, dann direkt aufrufbar, ansonsten aus Programm

```
CALL SYSCS_UTIL.SYSCS_EXPORT_QUERY('select * from continent'
,  'c:/Temp/myfile.dat', NULL, NULL, NULL);
```



Entwicklung eigener Prozeduren und Funktionen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- Funktion: nur IN-Parameter, nur lesende SQL-Befehle!

1. Realisiere Funktionalität in **Klassen**methoden (static) einer Java-Klasse
2. Packe ausführbaren Code als JAR-Datei
3. Lade Jar-Datei unter einem Pseudo-Namen in die Datenbank
4. Füge Pseudo-Namen zu den ausführbaren Paketen hinzu
5. Lege Funktion/Prozedur an, die auf Java-Methode zugreift

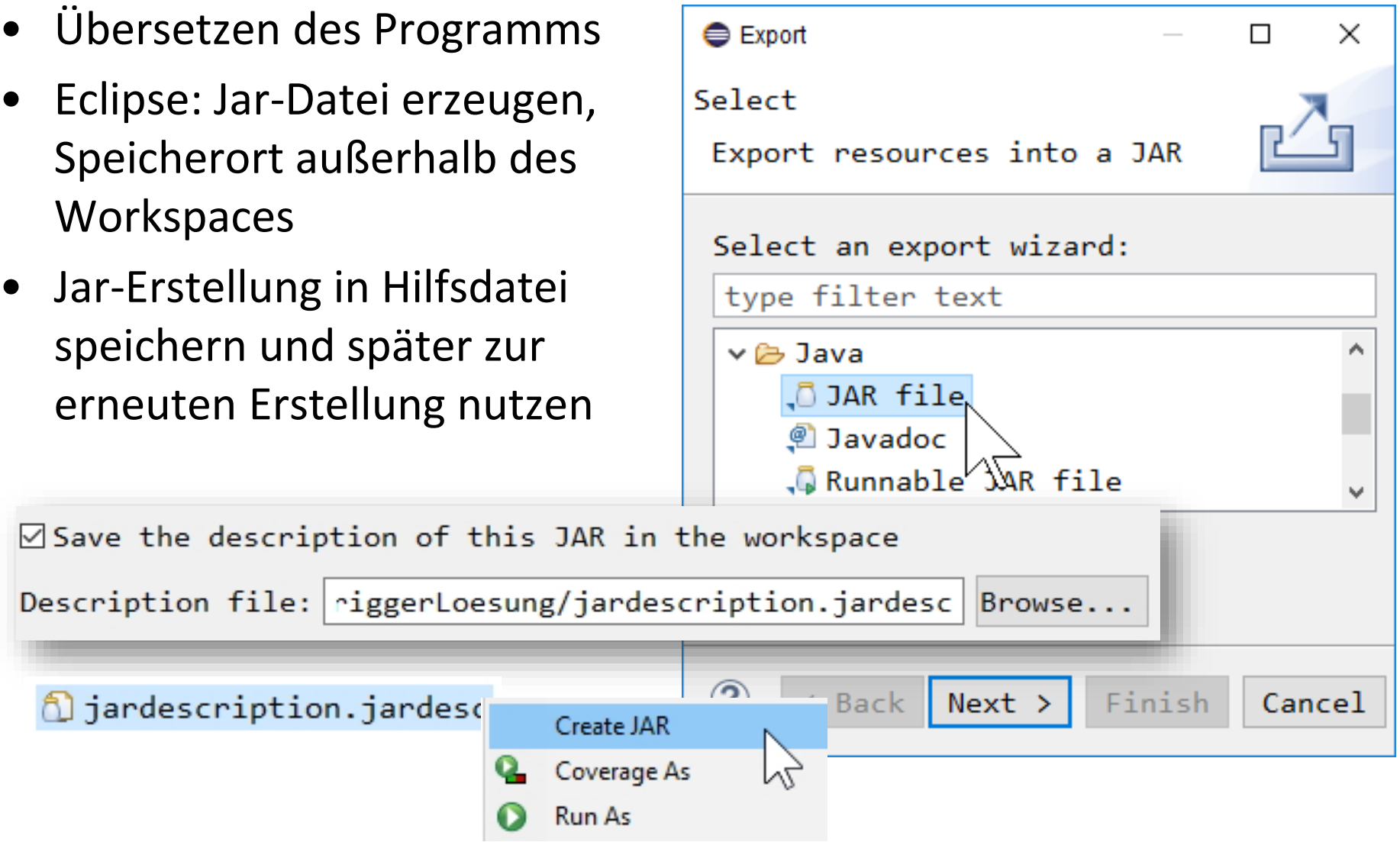
6. Nutze Funktion oder Prozedur

Erstes Beispiel (1/5) – Java Code

```
package added;

public class Work {
    public static int hallo(String txt){
        // Ausgaben auf DB-Konsole unüblich
        System.out.println("Hallo " + txt);
        return txt.length();
    }

    public static void auchHallo(String txt){
        System.out.println("Hai " + txt);
    }
}
```

Erstes Beispiel (2/5) – Erzeugung einer jar-Datei

- Übersetzen des Programms
- Eclipse: Jar-Datei erzeugen, Speicherort außerhalb des Workspaces
- Jar-Erstellung in Hilfsdatei speichern und später zur erneuten Erstellung nutzen



Erstes Beispiel (3/5) – jar-Datei laden, Fkt erstellen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
CALL sqlj.install_jar(
   'F:\workspaces\NetbeansWork_WS15\ErsteDBErweiterung'
   ||'\dist\ErsteDBErweiterung.jar', 'APP.Erstes', 0);

CALL SYSCS_UTIL.SYSCS_SET_DATABASE_PROPERTY(
     'derby.database.classpath',
     'APP.Erstes');

CREATE FUNCTION Halo(t VARCHAR(40))
RETURNS INT
LANGUAGE JAVA
PARAMETER STYLE JAVA
NO SQL
EXTERNAL NAME 'added.Work.hallo';
```

statt install auch replace möglich (dann kein letzter Parameter)

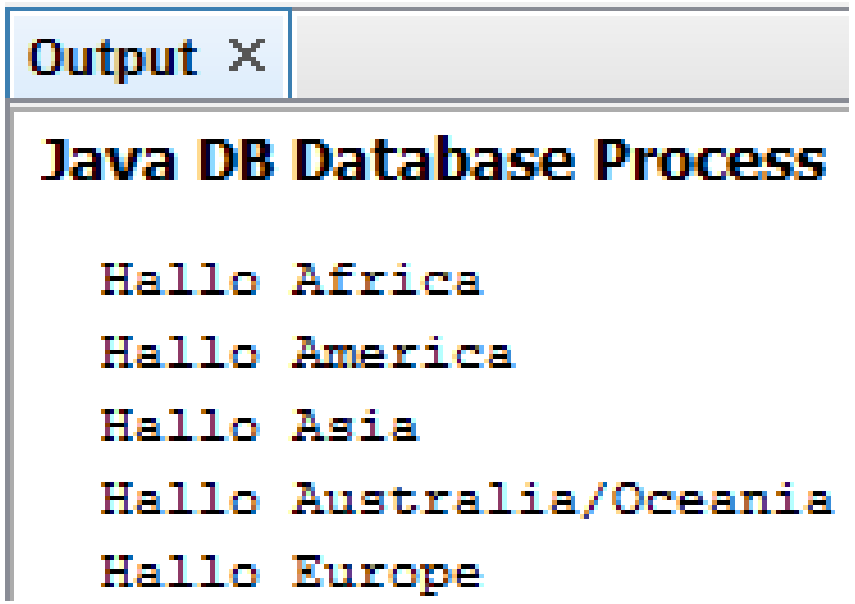
kein DB-Zugriff in der Methode, alternativ READS SQL DATA

voll qualifizierter Methodenname

Erstes Beispiel (4/5) – Nutzung von Funktionen

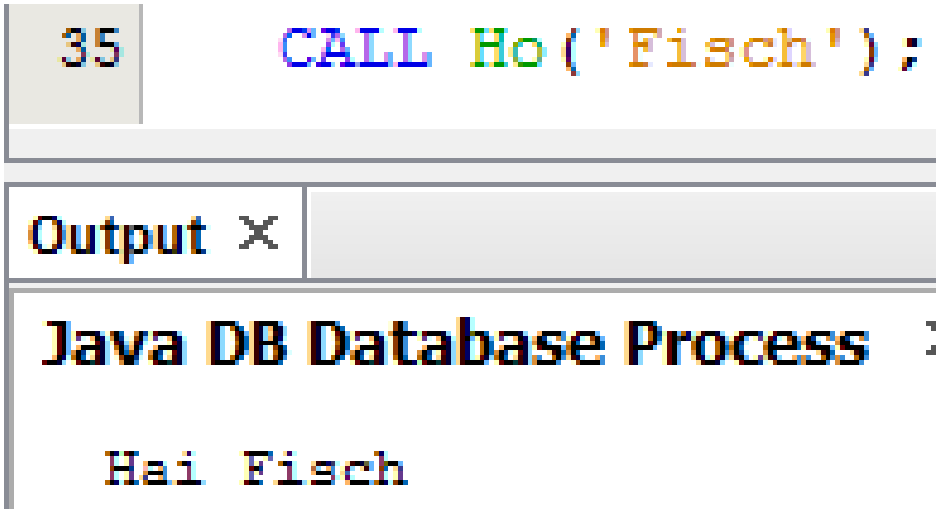
```
SELECT name, Halo(Continent.name) FROM Continent;
```

NAME	2
Africa	6
America	7
Asia	4
Australia/Oceania	17
Europe	6



Erstes Beispiel (5/5) – Prozedur-Erstellung / Nutzung

```
CREATE PROCEDURE Ho(IN t VARCHAR(40))
LANGUAGE JAVA
PARAMETER STYLE JAVA
READS SQL DATA
EXTERNAL NAME 'added.Work.auchHallo';
```



Systematischer Umgang mit Statement-Objekten

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- Statement-Objekte sollen nach ihrer Nutzung geschlossen werden
- seit Java 7 gibt es „AutoClosable“ – Interface

```
try (Statement stmt = con.createStatement()) {
    // stmt nutzen
}
```

- stmt wird mit oder ohne Exception geschlossen (stmt.close())
- catch und finally weiterhin möglich

- JavaDoc: „ A ResultSet object is automatically closed when the Statement object that generated it is closed, re-executed, or used to retrieve the next result from a sequence of multiple results. “

Zugriff auf Datenbank

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- der Zugriff auf die aktuell in Transaktion genutzter Verbindung:

```
DriverManager.getConnection("jdbc:default:connection");
```

- generell damit alle JDBC-Befehle zugänglich
- typischerweise keine Erzeugung neuer Tabellen erwünscht bzw. möglich
- Transaktionssteuerung (später) erlaubt, kann aber durch verschachtelte Aufrufe zu Problemen führen, wenn z. B. geschlossenes ResultSet genutzt wird

- Funktionen und Prozeduren laufen mit den Rechten der nutzenden Person, der sie importiert hat; man kann so Zugriffe ausschließlich auf diese Form einschränken (administrierende Person importiert und gibt Rechte an nutzende Person weiter; diese hat bei der Ausführung Rechte der administrierenden Person)

Methoden für neue Konten

```
public static void neuesKonto(int nr, int betrag)
    throws SQLException{
  Connection con = DriverManager
                   .getConnection("jdbc:default:connection");
  try (Statement stmt = con.createStatement()){
    stmt.execute("INSERT INTO Konto VALUES(" + nr
                                    + "," + betrag +")");
} }
```

```
public static void neuesKonto(int betrag) throws SQLException{
  Connection con = DriverManager
                   .getConnection("jdbc:default:connection");
  try (Statement stmt = con.createStatement()) {
    ResultSet rs = stmt.executeQuery("SELECT MAX(nr) FROM Konto");
    rs.next();
    int nr = rs.getInt(1)+1;
    neuesKonto(nr,betrag);
  }
}
```

Einspielen und Erstellen

```
CALL sqlj.replace_jar(
    'F:\workspaces\NetbeansWork_WS15\ErsteDBErweiterung'
    ||'\dist\ErsteDBErweiterung.jar', 'APP.Erstes');

CREATE PROCEDURE neuesKonto(IN nr INTEGER
                          , IN betrag INTEGER)
LANGUAGE JAVA
PARAMETER STYLE JAVA
MODIFIES SQL DATA
EXTERNAL NAME 'added.Kontoverwaltung.neuesKonto';

CREATE PROCEDURE neuesKonto2(IN betrag INTEGER)
LANGUAGE JAVA
PARAMETER STYLE JAVA
MODIFIES SQL DATA
EXTERNAL NAME 'added.Kontoverwaltung.neuesKonto';
```

keine Polymorphie möglich

Nutzen

```
CALL neuesKonto(10000,50);
CALL neuesKonto2(51);

SELECT * FROM Konto
  WHERE Konto.nr > 9999;
```

#	NR	BETRAG
1	10000	50
2	10001	51

Exception-Handling

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Java-Methoden werfen bekannte SQLExceptions
- zwei relevante Methoden
  - getMessage(): Fehlermeldung, kann null sein
  - getSQLState(): Fehleridentifikator (String)
- man kann Exceptions intern behandeln, oft werden sie nach außen weitergereicht
- man kann eigene SQLExceptions definieren
  - nach Standard ist SQLState dann zwischen 30000 und 38000
- umgebendes Programm kann dann SQLException auswerten

Einzahlung mit Prüfung

```
public static void einzahlen(int nr, int betrag)
                                throws SQLException{
  Connection con = DriverManager
                    .getConnection("jdbc:default:connection");
  try (Statement stmt = con.createStatement()){
    ResultSet rs = stmt.executeQuery(
                  "SELECT COUNT(*) FROM Konto  WHERE nr="+nr);
    rs.next();
    if(rs.getInt(1) == 0){
      throw new SQLException("Konto fehlt","30000");
    }
    rs.close();
    rs = stmt.executeQuery("SELECT * FROM Konto  WHERE nr="+nr);
    rs.next();
    if(rs.getInt(2) + betrag < 0){
      throw new SQLException("zu wenig Geld","30001");
    }
    stmt.execute("UPDATE Konto SET betrag = betrag+" + betrag
                + " WHERE nr=" + nr);
  }
}
```

geht einfacher, untere Anfrage, rs.next()=false

Ausführung

```
CALL einzahlen(10001,49);
SELECT * FROM Konto
  WHERE Konto.nr > 9999;
```

#	NR	BETRAG
1	10000	50
2	10001	100

```
CALL einzahlen(10002,49);
```

```
Error code 30000, SQL state 38000: Bei der Auswertung
eines Ausdrucks wurde die Ausnahme 'java.sql.SQLException:
Konto fehlt' ausgelöst.
```

```
Error code 99999, SQL state 30000: Konto fehlt
```

```
CALL einzahlen(10001,-149);
```

```
Error code 30000, SQL state 38000: Bei der Auswertung
eines Ausdrucks wurde die Ausnahme 'java.sql.SQLException:
zu wenig Geld' ausgelöst.
```

```
Error code 99999, SQL state 30001: zu wenig Geld
```

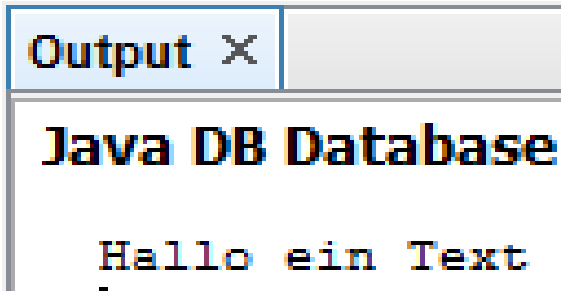
Aufruf einer Funktion durch den DB-Client

Video

```
public void rufeFunktion(String text) {
  CallableStatement stmt
                = this.con.prepareCall("{? = call halo(?)}");
  stmt.registerOutParameter(1, Types.INTEGER);
  stmt.setString(2, text);
  stmt.execute();
  int erg = stmt.getInt(1);
  System.out.println("erg: " + erg);
}
```

```
…
db.rufeFunktion("ein Text");
```

```
erg: 8
```



Aufruf einer Prozedur durch den DB-Client

```
public void rufeProzedur(int nr,int betrag){
  try( CallableStatement stmt = this.con
         .prepareCall( "{call einzahlen(?,?)}" ) ){
    stmt.setInt(1, nr);
    stmt.setInt(2, betrag);
    stmt.execute();
  } catch (SQLException e) {
    SQLException s = e.getNextException();
    System.out.println(s.getMessage() + ": "
                       + s.getSQLState());
  }
}
```

…

```
db.rufeProzedur(10001, 50);
db.rufeProzedur(10002, 50);
db.rufeProzedur(10001, -500);
```

```
Konto fehlt: 30000
zu wenig Geld: 30001
```

Nutzung von OUT-Parametern

- Prozeduren können über OUT-Parameter Werte zurückliefern
- Erster Java-Ansatz

```
public static void kontostand(int nr, int stand){
  …
  stand = …
}
```

- geht nicht, da **`stand`** keine Referenz in Java ist
- Trick: Nutze Array als Parametertyp und davon nur das erste Element; bei Arrays von Referenzen sind Änderungen möglich

Prozedur für Kontostand

```
public static void kontostand(int nr, int[] stand)
                      throws SQLException{
  Connection con = DriverManager
                     .getConnection("jdbc:default:connection");
  try (Statement stmt = con.createStatement()){
    ResultSet rs = stmt.executeQuery("SELECT * FROM Konto WHERE nr="
                                     + nr);
    if(!rs.next()){
      throw new SQLException("Konto "+nr+" fehlt","30000");
    }
    stand[0] = rs.getInt(2);
  }
}
```

```
CREATE PROCEDURE kontostand(IN nr INTEGER
                     , OUT stand INTEGER)
LANGUAGE JAVA
PARAMETER STYLE JAVA
MODIFIES SQL DATA
EXTERNAL NAME 'added.Kontoverwaltung.kontostand';
```

Nutzung der Prozedur

```
public int kontostand(int nr){
    try(CallableStatement stmt = this.con
              .prepareCall( "{call kontostand(?,?)}" )){
    stmt.setInt(1, nr);
    stmt.registerOutParameter(2, Types.INTEGER);
    stmt.execute();
    return stmt.getInt(2);
  } catch (SQLException e) {
    SQLException s = e.getNextException();
    System.out.println(s.getMessage() + ": "
                       + s.getSQLState());
  }
  return -1;
}
```

```
150
Konto 10002 fehlt: 30000
-1
```

```
…
System.out.println(db.kontostand(10001));
System.out.println(db.kontostand(10002));
```

Zugriffsrechte bei Prozeduren / Funktionen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Benutzungsrechte vergeben:

```
GRANT EXECUTE ON <procedure/function>
    TO <user>;
```

- Prozeduren und Funktionen werden jeweils mit den Zugriffsrechten des *Besitzers* ausgeführt
- d.h. nutzende Person kann die Prozedur/Funktion auch dann aufrufen, wenn sie kein Zugriffsrecht auf die dabei benutzten Tabellen hat

- Anmerkung: Sieht man SQLJ als serverseitige Programmierung, ist dies neben der Portabilität ein zentrales Argument für SQLJ

Trigger (1/2)

Beispiel Video

- typischer Einsatz bei Prüfung komplexer Bedingungen, die nicht mit Constraints überwachbar sind
- spezielle Form der Nutzung von Prozeduren
- werden beim Eintreten eines bestimmten Ereignisses ausgeführt
- Spezialfall aktiver Regeln nach dem Event-Condition-Action-Paradigma
- Werden einer Tabelle zugeordnet
- Bearbeitung wird durch das Eintreten eines Ereignisses (Einfügen, Ändern oder Löschen von Zeilen der Tabelle) ausgelöst (Event)

Trigger (2/2)

- Ausführung von Bedingungen für den Datenbankzustand abhängig (Condition)
- Action:

  vor oder *nach* der Ausführung der entsprechenden aktivierenden Anweisung ausgeführt
- einmal pro auslösender Anweisung (Statement-Trigger) oder einmal für jede betroffene Zeile (Row-Trigger) ausgeführt
- Trigger-Aktion kann auf den alten und neuen Wert der gerade behandelten Zeile zugreifen
- Aktion sorgt mit Exception dafür, dass keine Änderung stattfindet

Einschränkungen bei Trigger-Aktionen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

It must not contain any dynamic parameters (?).

It must not create, alter, or drop the table upon which the trigger is defined.

It must not add an index to or remove an index from the table on which the trigger is defined.

It must not add a trigger to or drop a trigger from the table upon which the trigger is defined.

It must not commit or roll back the current transaction or change the isolation level.

Before triggers cannot have INSERT, UPDATE or DELETE statements as their action.

Before triggers cannot call procedures that modify SQL data as their action.

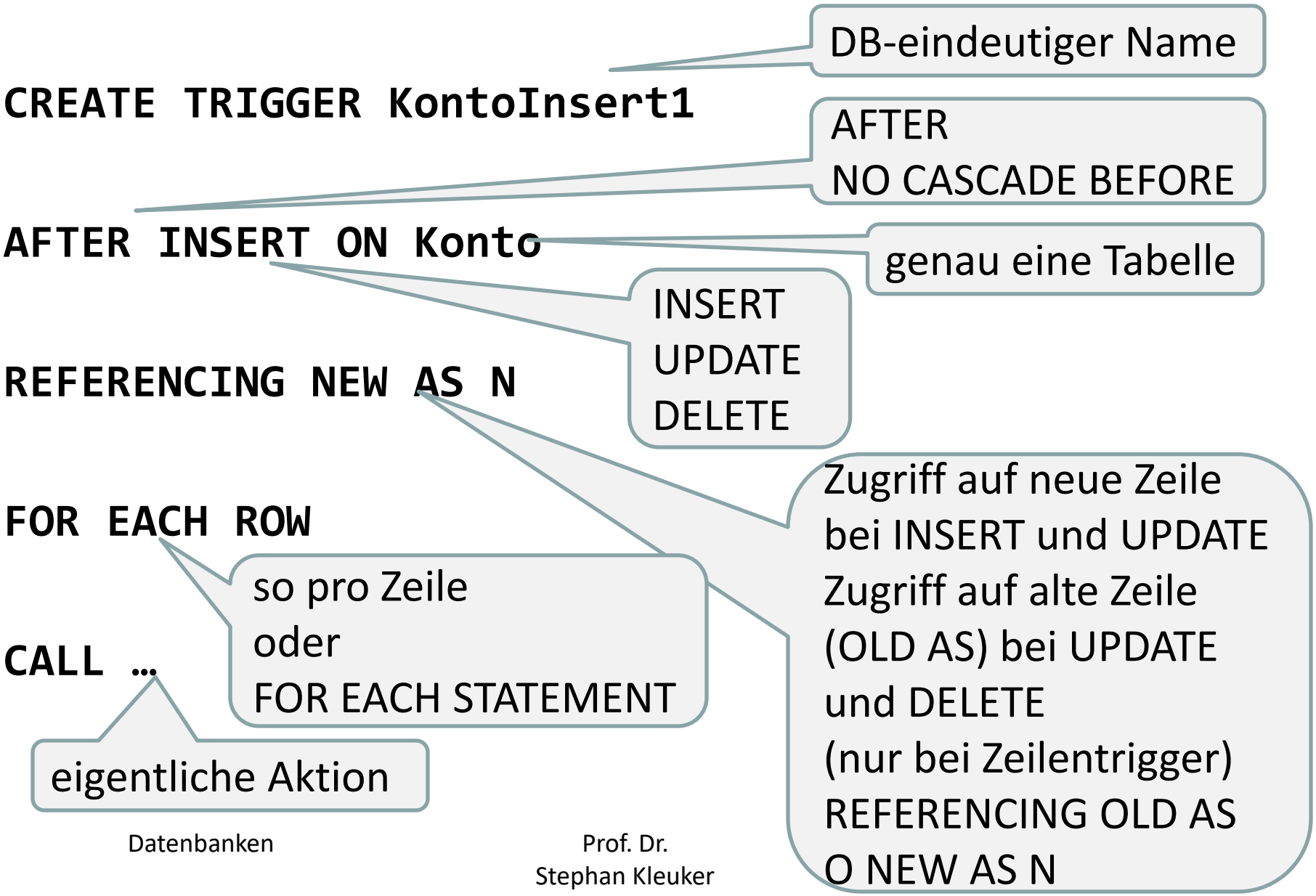
http://db.apache.org/derby/docs/10.14/ref/rrefsqlj43125.html

Erster Trigger (noch ohne Prozedur)

```
CREATE TABLE Protokoll(
  wer VARCHAR(30),
  wann TIMESTAMP,
  was VARCHAR(30)
);

CREATE TRIGGER KontoInsert1
AFTER INSERT ON Konto
REFERENCING NEW AS N
FOR EACH ROW
INSERT INTO Protokoll VALUES(USER, Current_timestamp
    ,'INSERT '|| CAST (N.nr AS CHAR(7)));

INSERT INTO Konto VALUES(17,0);
INSERT INTO Konto VALUES(177,0);
INSERT INTO Konto VALUES(1777,0);

SELECT * FROM Protokoll;
```

WER	WANN	WAS
KLEUKER	2015-08-21 19:39:02.285	INSERT 17
KLEUKER	2015-08-21 19:39:02.297	INSERT 177
KLEUKER	2015-08-21 19:39:02.307	INSERT 1777

Trigger genauer analysiert



FOR EACH ROW oder nicht

```
CREATE TABLE Tr(
  X INTEGER,
  Y INTEGER
);
INSERT INTO Tr VALUES (1,3);
INSERT INTO Tr VALUES (1,4);
INSERT INTO Tr VALUES (1,5);
SELECT * FROM Tr;

CREATE TRIGGER TrOhneEach
NO CASCADE BEFORE UPDATE ON Tr
CALL Ho('TrOhneEach');

CREATE TRIGGER TrMitEach
NO CASCADE BEFORE UPDATE ON Tr
FOR EACH ROW
CALL Ho('TrMitEach');

UPDATE TR SET Y=Y+1 WHERE X=1;
SELECT * FROM Tr;
```

```
 X           Y
---------- ----------
         1          3
         1          4
         1          5

Hai TrOhneEach
Hai TrMitEach
Hai TrMitEach
Hai TrMitEach

3 Zeilen wurden
aktualisiert.
         X          Y
---------- ----------
         1          4
         1          5
         1          6
```

Auswahl BEFORE oder AFTER

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Gibt keine generelle Regel, was besser ist
- Ansatz: finde möglichst einfachen Algorithmus, der eine Prüfung intuitiv verständlich macht
- BEFORE: berechne genau, ob gewünschte Aktion möglich ist
- AFTER: nehme an Aktion wurde ausgeführt; suche nach einem Fehler in den Daten; wenn vorhanden, brich ab
- EACH ROW oder nicht: interessiert jede einzelne Aktion oder nur das Gesamtergebnis der Aktion

- Konto-Beispiel: Die Summe aller Beträge der Konten soll nicht negativ sein
- Jeweils Trigger für INSERT, UPDATE und DELETE benötigt
- hier einfache Prüfung nachdem Aktion ausgeführt wurde

Prüfmethode

```
public static void nichtNegativ() throws SQLException{
  Connection con = DriverManager
                   .getConnection("jdbc:default:connection");
  try(Statement stmt = con.createStatement()){
    ResultSet rs = stmt.executeQuery("SELECT SUM(betrag) "
                                     + " FROM Konto");
    rs.next();
    System.out.println("Summe: " + rs.getInt(1));
    if(rs.getInt(1) < 0){
      throw new SQLException("Pleite verhindern","30002");
    }
  }
}

CREATE PROCEDURE nichtNegativ()
LANGUAGE JAVA
PARAMETER STYLE JAVA
MODIFIES SQL DATA
EXTERNAL NAME 'added.Kontoverwaltung.nichtNegativ';
```

Trigger erstellen und ausprobieren

```
CREATE TRIGGER nichtNegativInsert
AFTER INSERT ON Konto
CALL nichtNegativ();

CREATE TRIGGER nichtNegativUpdate
AFTER UPDATE ON Konto
CALL nichtNegativ();

CREATE TRIGGER nichtNegativDelete
AFTER DELETE ON Konto
CALL nichtNegativ();

INSERT INTO Konto VALUES(42,50);
INSERT INTO Konto VALUES(43,50);
INSERT INTO Konto VALUES(44,-100);
SELECT * FROM Konto;

INSERT INTO Konto VALUES(45,-1);

UPDATE Konto SET betrag = betrag - 10;

DELETE FROM Konto WHERE nr = 43;
```

NR	BETRAG
42	50
43	50
44	-100

```
Summe: 50
Summe: 100
Summe: 0
Summe: -1
Summe: -30
Summe: -50
```

NR	BETRAG
42	50
43	50
44	-100

Problem der Mutating Tables

Trigger können die Inhalte von Tabellen ändern (der Tabelle, auf der sie definiert sind und andere),

- d.h. jede Ausführung des Triggers sieht eventuell einen anderen Datenbestand der Tabelle, auf der er definiert ist, sowie der Tabellen, die er evtl. ändert
- d.h. Ergebnis *abhängig von der Reihenfolge* der veränderten Tupel
- Ansatz: Betroffene Tabellen werden während der gesamten Aktion als „mutating“ gekennzeichnet, können nicht erneut von Triggern gelesen oder geschrieben werden
- Nachteil: Oft ein zu strenges Kriterium
- Derby: maximale Kettenlänge bei Aufrufen

Beispiel (1/4): Aufgabe

Video

- Tabelle speichert Gebote eines Mitglieds (mnr) für eine Ware (ware) als Preis (gebot)
- Forderung: bei neuen Geboten (insert oder update erlaubt) für die gleiche Ware muss das eigene Gebot erhöht werden

```
CREATE TABLE Gebot(
      id INTEGER
    , mnr INTEGER
    , ware INTEGER
    , gebot DECIMAL(8, 2)
    , PRIMARY KEY(id)
);
```

Beispiel (2/4) : Methode

```
public static void gebotErhoehen(int mnr, int ware
                    , double gebot) throws SQLException{
  Connection con = DriverManager
                   .getConnection("jdbc:default:connection");
  try(Statement stmt = con.createStatement()){
    ResultSet rs = stmt.executeQuery("SELECT MAX(gebot) "
                 + "FROM Gebot "
                 + "WHERE mnr =" + mnr
                 + "  AND ware =" + ware);
    rs.next();
    double max = rs.getDouble(1);
    System.out.println(rs.wasNull()+ " " + max);
    if (!rs.wasNull() && max >= gebot){
      throw new SQLException("Gebot erhöhen!","30009");
    }
  }
}
```

Beispiel (3/4): Trigger anlegen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
CREATE PROCEDURE gebotErhoehen(m INTEGER, w INTEGER, g DOUBLE)
LANGUAGE JAVA
PARAMETER STYLE JAVA
READS SQL DATA
EXTERNAL NAME 'added.Gebotspruefung.gebotErhoehen';

CREATE TRIGGER gebotErhoehenInsert
NO CASCADE BEFORE INSERT ON Gebot
REFERENCING NEW AS N
FOR EACH ROW
CALL gebotErhoehen(N.mnr, N.ware, N.gebot);

CREATE TRIGGER gebotErhoehenUpdate
NO CASCADE BEFORE UPDATE ON Gebot
REFERENCING NEW AS N
FOR EACH ROW
CALL gebotErhoehen(N.mnr, N.ware, N.gebot);
```

Beispiel (4/4): Probieren

```
INSERT INTO Gebot VALUES(100, 42, 99, 1.00);
```

ID	MNR	WARE	GEBOT
100	42	99	1.00

```
UPDATE Gebot SET gebot = 1.01
    WHERE mnr = 42 AND ware = 99;
```

ID	MNR	WARE	GEBOT
100	42	99	1.01

```
INSERT INTO Gebot VALUES(101, 42, 99, 1.00);
```

ID	MNR	WARE	GEBOT
100	42	99	1.01

```
UPDATE Gebot SET gebot = 1.01
    WHERE mnr = 42 AND ware = 99;
```

ID	MNR	WARE	GEBOT
100	42	99	1.01

```
true 0.0
false 1.0
false 1.01
false 1.01
```

12. Testen von DB-Software

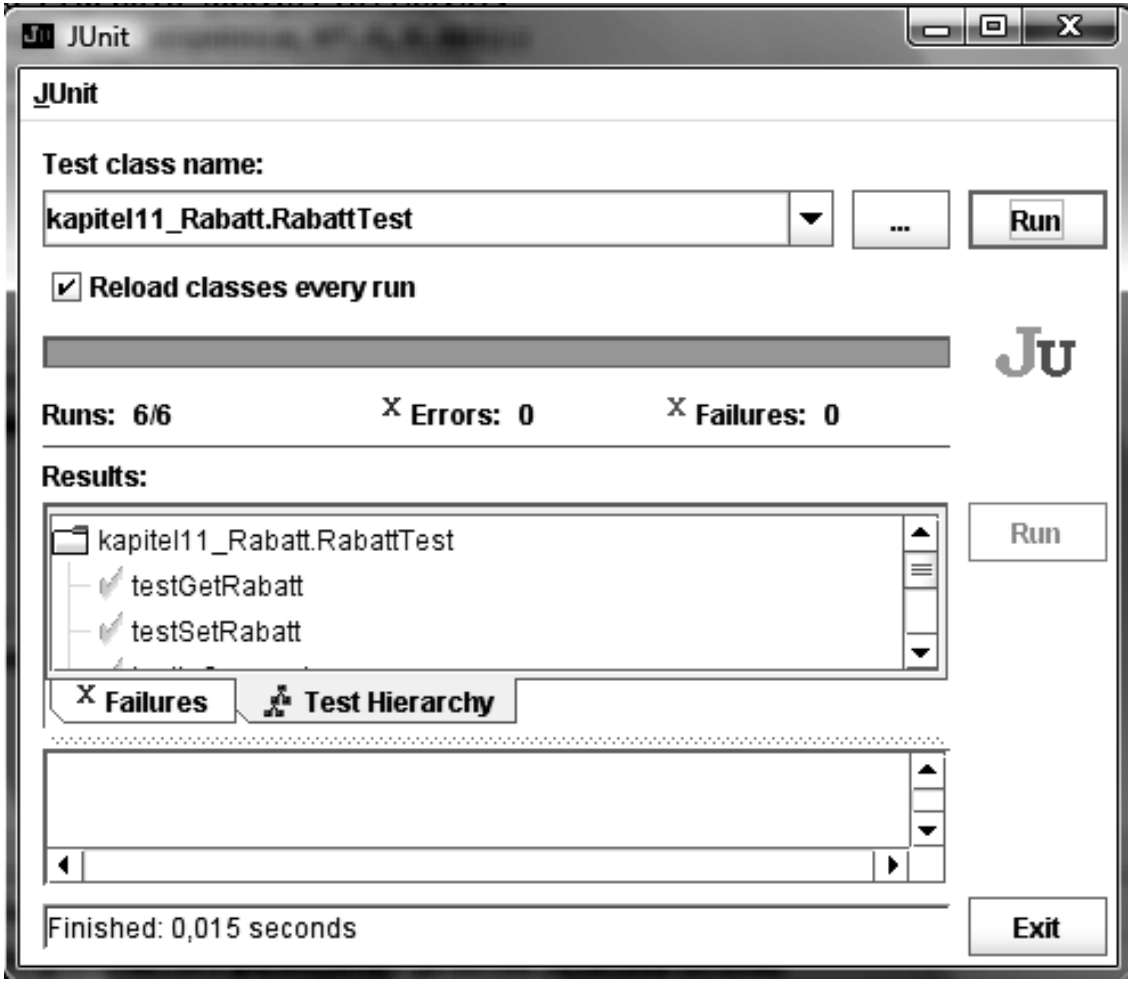
Beispiel Video

- Kurzeinführung JUnit
- Regeln zur Testerstellung
- Nutzung von DBUnit

Einschub: Testen mit JUnit

- Framework, um den Unit-Test eines Java-Programms zu automatisieren
- einfacher Aufbau
- leicht erlernbar

- geht auf SUnit (Smalltalk) zurück
- mittlerweile für viele Sprachen verfügbar (NUnit, CPPUnit)



S. Kleuker, Qualitätssicherung durch Softwaretests, 2. Auflage, Springer Vieweg, Wiesbaden, 2019

Testfall

- Vor dem Testen müssen Testfälle spezifiziert werden
- Vorbedingungen (Arrange)
  - Zu testende Software in klar definierte Ausgangslage bringen (z. B. Objekte mit zu testenden Methoden erzeugen)
  - Angeschlossene Systeme in definierten Zustand bringen
  - Weitere Rahmenbedingungen sichern (z. B. HW)
- Ausführung (Act)
  - Was muss wann gemacht werden (einfachster Fall: Methodenaufruf)
- Nachbedingungen (Assert)
  - Welche Ergebnisse sollen vorliegen (einfachster Fall: Rückgabewerte)
  - Zustände anderer Objekte / angeschlossener Systeme

Einführendes Beispiel – JUnit Jupiter (1/11)

- zu testendes System

```
package controller;

public class Adder {
  public int add(int x, int y) {
    if ((x == 42) || (y == 42)) {
      throw new IllegalArgumentException("Nicht mit 42");
    }
    return x + y;
  }
}
```

Einführendes Beispiel – JUnit Jupiter (2/11)

- JUnit Jupiter / JUnit5 in verschiedene Module zerlegt
- auch in klassischen Projekten bis Java 8 nutzbar

```
open module jUnit5Einfuehrung {
  exports controller;
  requires org.junit.jupiter.api;
  requires org.junit.jupiter.params;
}
```

Einführendes Beispiel – JUnit Jupiter (3/11)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
import java.util.Arrays;
import java.util.stream.Stream;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;

public class AdderTest {
```

Beliebiger Klassenname, Endung „Test“ üblich

Einführendes Beispiel – JUnit Jupiter (4/11)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Test ist beliebige mit @Test annotierte Methode

Methodenname ist beliebig, beginnt typischerweise mit „test“ und beinhaltet Name der Methode oder Sinn des Tests

```
@Test
 void testAdderOk() {
   // Assemble (vorbereiten)
   Adder a = new Adder();
   // Act (zu Testendes ausfuehren)
   int erg = a.add(21, 21);
   // Assert (pruefe mit Zusicherungen)
   Assertions.assertEquals(42, erg
         , "erwartet 42, gefunden " + erg);
 }
```

Experimente

Prüfmethode, Methodenname zeigt Prüfart ; ist Bedingung „false“ wird Test als gescheitert festgehalten und Text ausgegeben

Einführendes Beispiel – JUnit Jupiter (5/11)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
private Adder sut; // System under Test

@BeforeEach
public void setUp() {
  this.sut = new Adder();
  System.out.println("setUp");
} // Ausgaben extrem unueblich

@Test
public void testAdderNegativ() {
  Assertions.assertEquals(0, this.sut.add(21, -21));
}

@AfterEach
public void tearDown() {
  // aufraeumen
  System.out.println("tearDown");
}
```

normale Objektvariablen

Methode wird vor jedem Test ausgeführt, Idee: einheitliche Ausgangssituation schaffen

einmal nach jeden Tests (lokal aufräumen)

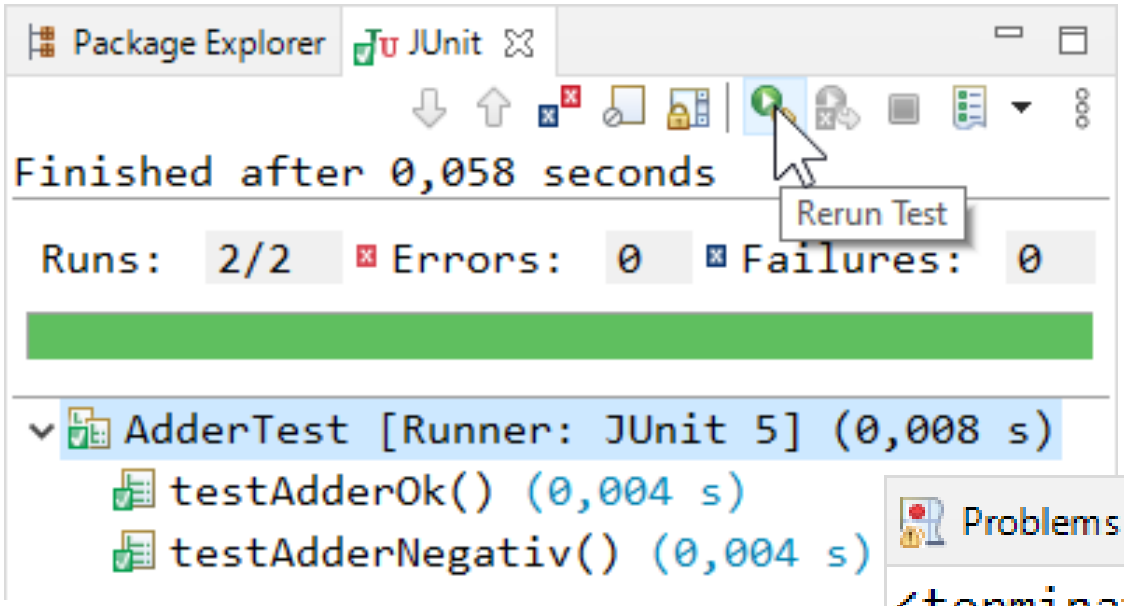
Einführendes Beispiel – JUnit Jupiter (6/11)

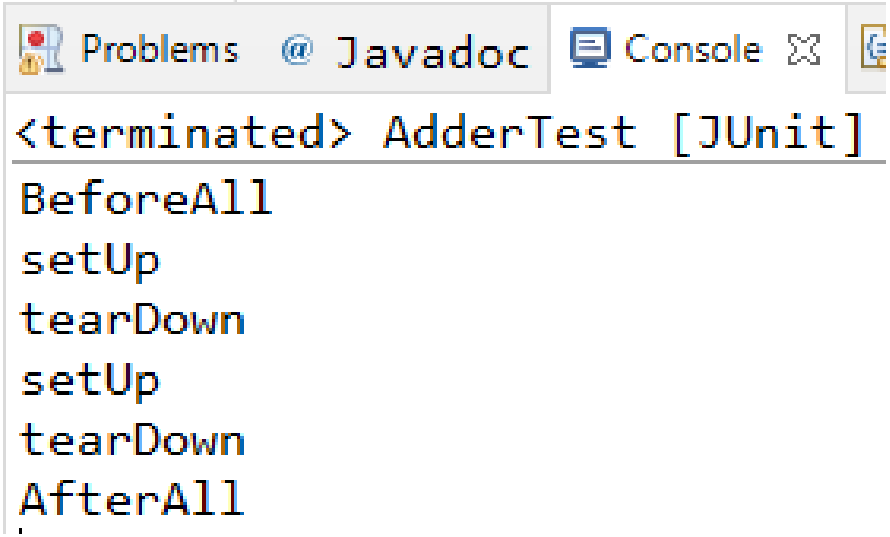
- Methoden zum Aufbau der Testumgebung und zum Abbau werden einmal vor allen Tests und einmal nach allen Tests ausgeführt
- Testfixture wird jeweils durchlaufen
- Reihenfolge der Tests ist beliebig

```
@BeforeAll
public static void setUpClass() {
   System.out.println("BeforeAll");
}

@AfterAll
public static void tearDownClass() {
  System.out.println("AfterAll");
}
```

Einführendes Beispiel – JUnit Jupiter (7/11)

- Ausführung bisheriger Klasse





Einführendes Beispiel – JUnit Jupiter (8/11)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
// Viele Varianten des Exception-Tests; hier klassisch
@Test
public void testAdderKeineException() {
  try {
    Assertions.assertEquals(-1, this.sut.add(20, -21));
  } catch (IllegalArgumentException e) {
    Assertions.fail();
  }
}

@Test
public void testAdderEineException1() {
  try {
    this.sut.add(42, 0);
    Assertions.fail();
  } catch (IllegalArgumentException e) {
    Assertions.assertTrue(e.getMessage().contains("42"));
  }
}
```

Einführendes Beispiel – JUnit Jupiter (9/11)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Jeder Fehlerfall wird einzeln geprüft
- nach Assertions keine andere Art von Anweisung
- nicht gefangene Exceptions als „Errors“ von JUnit gezählt

```
@Test
 public void testAdderEineException2() {
   try {
     this.sut.add(0, 42);
     Assertions.fail();
   } catch (IllegalArgumentException e) {
     Assertions.assertTrue(e.getMessage().contains("42"));
   }
 }
```

Einführendes Beispiel – JUnit Jupiter (10/11)

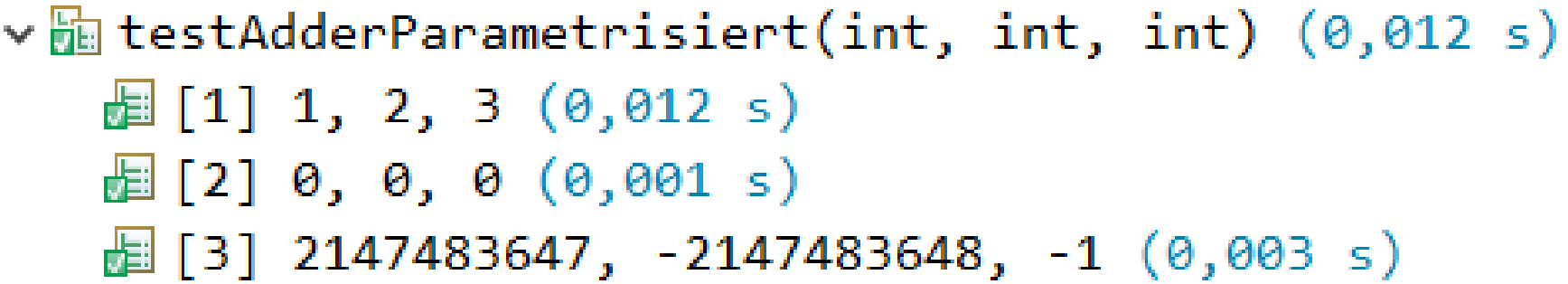
HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- sehr viele weitere Möglichkeiten in JUnit Jupiter
- wichtig: parametrisierbare Tests; eine große Menge von Testdaten, z. B. aus Datei, sollen in gleichartigem Test genutzt werden
- eine Methode stellt Daten zur Verfügung

```
public static Stream<Arguments> daten() {
 Arguments[] testdaten = {
     Arguments.of(1, 2, 3)
   , Arguments.of(0, 0, 0)
   , Arguments.of(Integer.MAX_VALUE, Integer.MIN_VALUE, -1)
 };
 return Arrays.asList(testdaten).stream();
}
```

Einführendes Beispiel – JUnit Jupiter (11/11)

```
@ParameterizedTest
@MethodSource({"daten"})
public void testAdderParametrisiert(int arg1
        , int arg2, int erg) {
  Assertions.assertEquals(erg, this.sut.add(arg1, arg2));
  Assertions.assertEquals(erg, this.sut.add(arg2, arg1));
}
```



Basisideen zur Testerstellung

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- viele kleine Tests, da nachfolgende Assertions nicht geprüft, wenn eines vorher abbricht
- erwartetes Verhalten kann zusammen geprüft werden
- jede mögliche Ausnahme in getrenntem Testfall
- Extremwerte prüfen

- Testklassen können weitere Hilfsmethoden enthalten
- typisch: am Anfang auf „leerem Feld“ neue Testausgangssituation (@SetUp) erstellen

Test von Datenbanken

- Nie, nie mit laufender Geschäftsdatenbank testen; es wird immer ein Testsystem benötigt
- generell direkt mit JUnit machbar
- Detailproblem: nach den Tests so aufräumen, dass Ausgangssituation wieder hergestellt (abhängige Daten !)
- Detailproblem: aufwändiger Vergleich zwischen aktuellem und erwartetem Tabelleninhalt

Vereinfachung mit DBUnit als Ergänzung von JUnit

- einfaches Leeren und Neueinspielen von Datensätzen
- einfacher Vergleich von Mengen von Tabelleneinträgen
- Tabellen z. B. auf Basis von XML-Dateien definierbar
- (hier nur zentrale Konzepte) http://www.dbunit.org/

DBUnit – erste Schritte (1/7)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Beispieldatenbank

```
CREATE TABLE Person(
   id INTEGER
 , name VARCHAR(20)
 , PRIMARY KEY(id)
);

INSERT INTO Person VALUES(42, 'James');
INSERT INTO Person VALUES(43, 'Jana');
INSERT INTO Person VALUES(52, 'Kevin');
INSERT INTO Person VALUES(53, 'Leila');
```

DBUnit – erste Schritte (2/7)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- DBUnit nutzt eigene Connection-Variante zum Zugriff auf Datenbank

```
public class Spielereien {

  private Connection con;
  private IDatabaseConnection conDBU;

  public Spielereien() throws Exception {
    this.con = this.verbinden();
    this.conDBU = new DatabaseConnection(this.con, null, true);
  }

  public Connection verbinden() throws Exception {
    Class.forName("org.apache.derby.jdbc.ClientDriver")
         .getDeclaredConstructor().newInstance();
    return DriverManager
              .getConnection("jdbc:derby://localhost:1527/"
                   + "F:\\workspaces\\datenbanken\\dbunit"
                          ,"kleuker", "kleuker");
  }
```

DBUnit – erste Schritte (3/7)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Trennen der Datenbank NIE vergessen

```
public void verbindungTrennen() throws Exception {
   if (this.con != null) {
     this.con.close();
   }
   if (this.conDBU != null) {
     this.conDBU.close();
   }
 }
```

- generell verschiedene Varianten zur Gewinnung von Testdaten
  - Daten aus realer Datenbank kopieren (Datenschutz beachten)
  - selbst Testdaten erstellen

DBUnit – erste Schritte (4/7)

- alle Daten aus Datenbank auslesen

```
public void alleDatenAuslesen() throws DataSetException
      , FileNotFoundException, IOException, SQLException {
  FlatXmlDataSet.write(this.conDBU.createDataSet(),
      new FileOutputStream("testdaten\\alles.xml"));
}
```

- Daten mit Anfrage auslesen

```
public void testdatenAusDBauslesen()
    throws DataSetException, FileNotFoundException
        , IOException {
   QueryDataSet dataSet = new QueryDataSet(this.conDBU);
   dataSet.addTable("Person",
       "SELECT * FROM Person WHERE Person.Id < 50");
   FlatXmlDataSet.write(dataSet,
       new FileOutputStream("testdaten\\unter50.xml"));
 }
```

DBUnit – erste Schritte (5/7)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Beispieldatei alles.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<dataset>
  <PERSON ID="42" NAME="James"/>
  <PERSON ID="43" NAME="Jana"/>
  <PERSON ID="52" NAME="Kevin"/>
  <PERSON ID="53" NAME="Leila"/>
</dataset>
```

- Tag ist Tabellenname, Spalten sind Attributsnamen
- wird Spalte nicht angegeben, dann NULL-Wert
- Daten werden in angegebener Reihenfolge eingefügt
- mehrere Tabellen in einer XML-Datei möglich
- Testdaten in dieser Form selbst erstellbar

DBUnit – erste Schritte (6/7)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Daten einspielen (Dateinamen als Parameter)

```
public void datenWiederEinspielen()
    throws DatabaseUnitException, SQLException
          , FileNotFoundException, MalformedURLException {
   FlatXmlDataSetBuilder builder
                              = new FlatXmlDataSetBuilder();
   builder.setColumnSensing(true); // fuer NULL-Werte
   IDataSet dataSet = builder
       .build(new File("testdaten\\alles.xml"));
   DatabaseOperation.CLEAN_INSERT
                     .execute(this.conDBU, dataSet);
 }
```

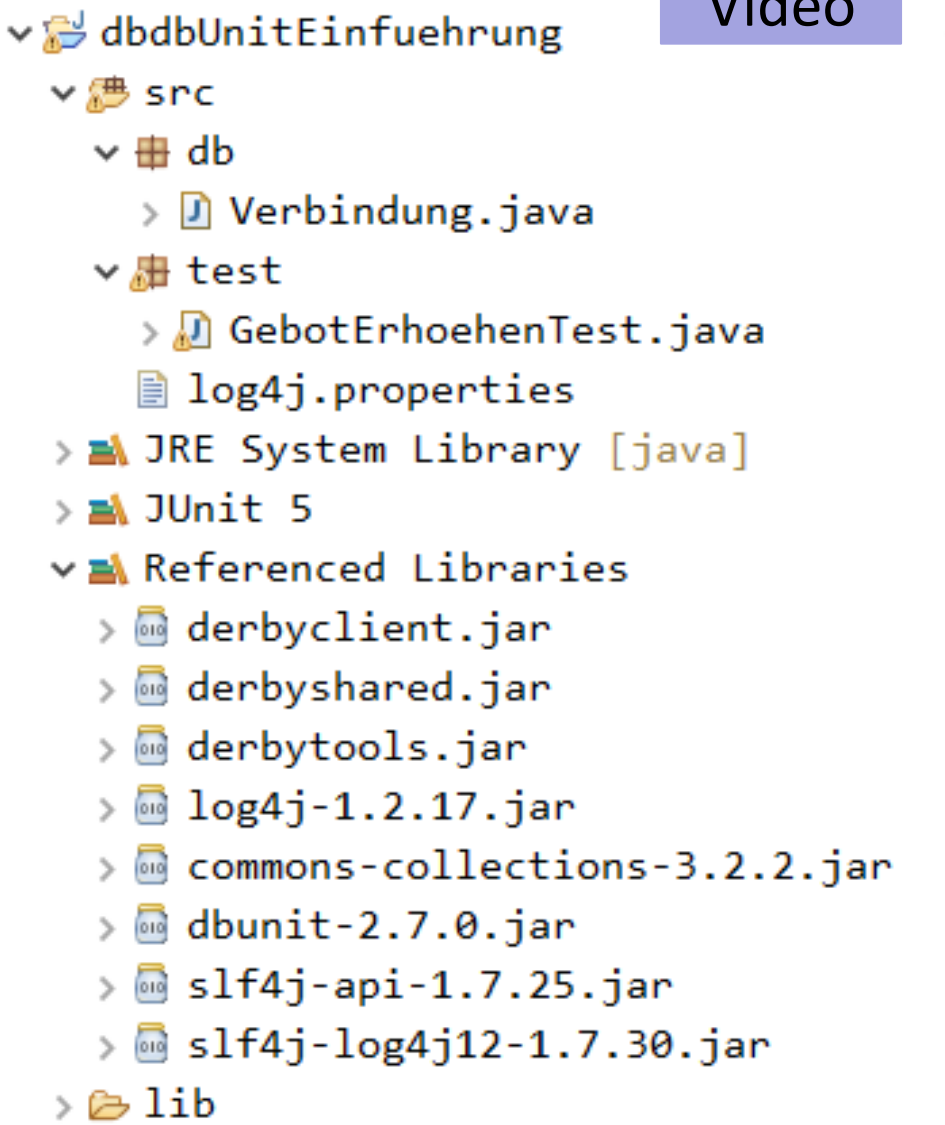
DBUnit – erste Schritte (7/7)

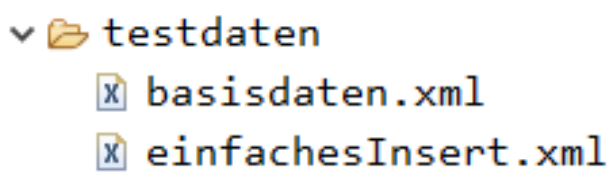
HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- verschiedene Varianten zum Einfügen von Daten
  - CLEAN_INSERT: alle Daten zuerst löschen, dann einfügen
  - DELETE_ALL: löscht alle Daten in den Tabellen
  - DELETE : löscht die übergebenen Daten
  - INSERT: fügt die übergebenen Daten in die Tabellen ein
  - UPDATE: aktualisiert die vorhandenen Daten mit den übergebenen Daten
  - REFRESH: aktualisiert vorhandene Daten, fügt nicht vorhandene Daten hinzu

Projektaufbau – Eclipse – Test von Trigger

Video





Konfiguration des Loggers (log4j.properties)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
log4j.rootLogger=WARN, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.conversionPattern=%5p [%t]
(%F:%L) - %m%n
```

- ermöglicht flexibles Schreiben von Meldungen in Log-Dateien
- sehr einfach ein- und ausschaltbar (hier Level WARN)

- Erinnerung: Fallstudie Gebot
- Tabelle speichert Gebote eines Mitglieds (mnr) für eine Ware (ware) als Preis (gebot) mit eigener id
- Forderung: bei neuen Geboten (insert oder update erlaubt) für die gleiche Ware muss das eigene Gebot erhöht werden

Verbindung

```
public class Verbindung {

  public static Connection verbinden() throws Exception {
    Class.forName("org.apache.derby.jdbc.ClientDriver")
         .newInstance();
    return DriverManager
           .getConnection("jdbc:derby://localhost:1527/"
                     + "F:\\workspaces\\datenbanken\\Gebot",
                ,"kleuker", "kleuker");
  }

  public static void verbindungTrennen(Connection con
        , IDatabaseConnection conDBU) throws Exception {
    if (con != null) {
      con.close();
    }
    if (conDBU != null) {
      conDBU.close();
    }
  }
}
```

Testbed: Verbindungs- auf und Abbau

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
public class GebotErhoehenTest {

  private static Connection con = null; // direkte DB-Verbindung
  private static IDatabaseConnection conDBU; // DBUnit-Verbindung

  @BeforeAll
  public static void setUpBeforeClass() throws Exception {
    con = Verbindung.verbinden();
    conDBU = new DatabaseConnection(con, null, true);
    //DatabaseConfig config = conDBU.getConfig();
    //config.setProperty(DatabaseConfig.PROPERTY_DATATYPE_FACTORY,
    //    new OracleDataTypeFactory());
  }

  @AfterAll
  public static void tearDownAfterClass() throws Exception {
    Verbindung.verbindungTrennen(con, conDBU);
  }
```

DBUnit hat DB-individuelle Einstellungen

basisdaten.xml

- Möglichkeit zur Spezifikation von Testdaten mit XML

```
<?xml version="1.0" encoding="UTF-8"?>
<dataset>
<Gebot id="1" mnr="1" ware="100" gebot="1.00" />
<Gebot id="2" mnr="1" ware="101" gebot="1.00" />
<Gebot id="3" mnr="1" ware="100" gebot="2.00" />
<Gebot id="4" mnr="2" ware="100" gebot="2.01" />
<Gebot id="5" mnr="3" ware="101" gebot="1.01" />
</dataset>
```

```
@BeforeEach
public void setUp() throws Exception {
  IDataSet dataSet = new FlatXmlDataSetBuilder()
    .build(
       new FileInputStream(".\\testdaten\\basisdaten.xml"));
  DatabaseOperation.CLEAN_INSERT.execute(conDBU, dataSet);
}
```

Test: erlaubtes Insert

```
@Test
public void testGebotAufNeueWareOk() {
  try {
    int anzahl = con.createStatement().executeUpdate(
        "INSERT INTO Gebot VALUES (42, 4, 102, 3.00)");
    Assertions.assertTrue(anzahl == 1
        , "statt 1," + anzahl + " Datensaetze geaendert", );
  } catch (SQLException e) {
    Assertions.fail("erlaubtes INSERT gescheitert: "
        + "VALUES (42, 4, 102, 3.00)");
  }
}
// Hinweis: Test mit weiteren erlaubten INSERT, UPDATE,
// DELETE sinnvoll
```

- Etwas kritisch: es wurde eine Zeile eingefügt; wird nicht überprüft, ob sie so im Ergebnis steht

Erlaubtes Insert, präzise Prüfung (1/2)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- einfachesInsert.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<dataset>
<Gebot id="1" mnr="1" ware="100" gebot="1.00" />
<Gebot id="2" mnr="1" ware="101" gebot="1.00" />
<Gebot id="3" mnr="1" ware="100" gebot="2.00" />
<Gebot id="42" mnr="4" ware="102" gebot="3.00" />
<Gebot id="4" mnr="2" ware="100" gebot="2.01" />
<Gebot id="5" mnr="3" ware="101" gebot="1.01" />
</dataset>
```

- Erinnerung: SQL speichert ohne Reihenfolge, was beim Vergleich zu beachten ist
- DBUnit ermöglicht lexikographische Sortierung

Erlaubtes Insert, präzise Prüfung (2/2)

```
@Test
public void testGebotAufNeueWareVarianteOk()
                                              throws Exception {
  con.createStatement().executeUpdate(
      "INSERT INTO Gebot VALUES (42, 4, 102, 3.00)");
  IDataSet databaseDataSet = conDBU.createDataSet();
  ITable actualTable = databaseDataSet.getTable("Gebot");
  IDataSet expectedDataSet = new FlatXmlDataSetBuilder()
     .build(new File(".\\testdaten\\einfachesInsert.xml"));
  ITable expectedTable = expectedDataSet.getTable("Gebot");

  Assertion.assertEquals(new SortedTable(expectedTable)
                        , new SortedTable(actualTable));
}
```

DBUnit Klasse Assertion

Test, ob Primary Key noch existiert

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
@Test
public void testPrimaryKeyVerstoss(){
  try {
    con.createStatement().executeUpdate(
        "INSERT INTO Gebot VALUES (2, 42, 100, 2.01)");
    Assertions.fail("verbotenes INSERT durchgefuehrt: "
        + "VALUES (2, 42, 100, 2.01)");
  } catch (SQLException e) {
  }
}
```

Test des Triggers (1/2)

```
@Test
public void testHoeheresGebotOk() {
  try {
    int anzahl = con.createStatement().executeUpdate(
        "INSERT INTO Gebot VALUES (42, 3, 101, 1.02)");
    Assertions.assertTrue(anzahl == 1
        , "statt 1," + anzahl + " Datensaetze geaendert", );
  } catch (SQLException e) {
    Assertions.fail("erlaubtes INSERT gescheitert: "
        + "VALUES (42, 3, 101, 1.02)");
  }
}
```

Test des Triggers (2/2)

```
@Test
public void testGleichesGebotVerboten(){
  try {
    con.createStatement().executeUpdate(
        "INSERT INTO Gebot VALUES (42, 3, 101, 1.01)");
    Assertions.fail("verbotenes INSERT durchgefuehrt: "
        + "VALUES (42, 3, 101, 1.01)");
  } catch (SQLException e) {
    // hier gibt es Unterschiede zwischen DBs
    Assertions.assertTrue(
        e.getNextException().getSQLState().equals("30009")
      , ("Fehler 30009 erwartet, gefunden"
              + e.getNextException().getSQLState());
  }
}
// fehlen Tests für UPDATE
// Anmerkung: Trigger werden vor PRIMARY KEY überprüft
```

Test auf passenden Fehlercode

Fazit DBUnit

- grundsätzlich alles ohne DBUnit machbar
- DBUnit erleichtert wesentlich das Lesen und wieder Einspielen von Daten in die Datenbank
- DBUnit stellt bequeme Überprüfung auf inhaltliche Gleichheit von Tabellen zur Verfügung
- weitere Infos und Funktionalität: http://dbunit.sourceforge.net/
- Varianten von DBUnit in einigen anderen Programmiersprachen
- Erinnerung: NIE, nie auch nur in der Nähe von realen Daten testen

Video

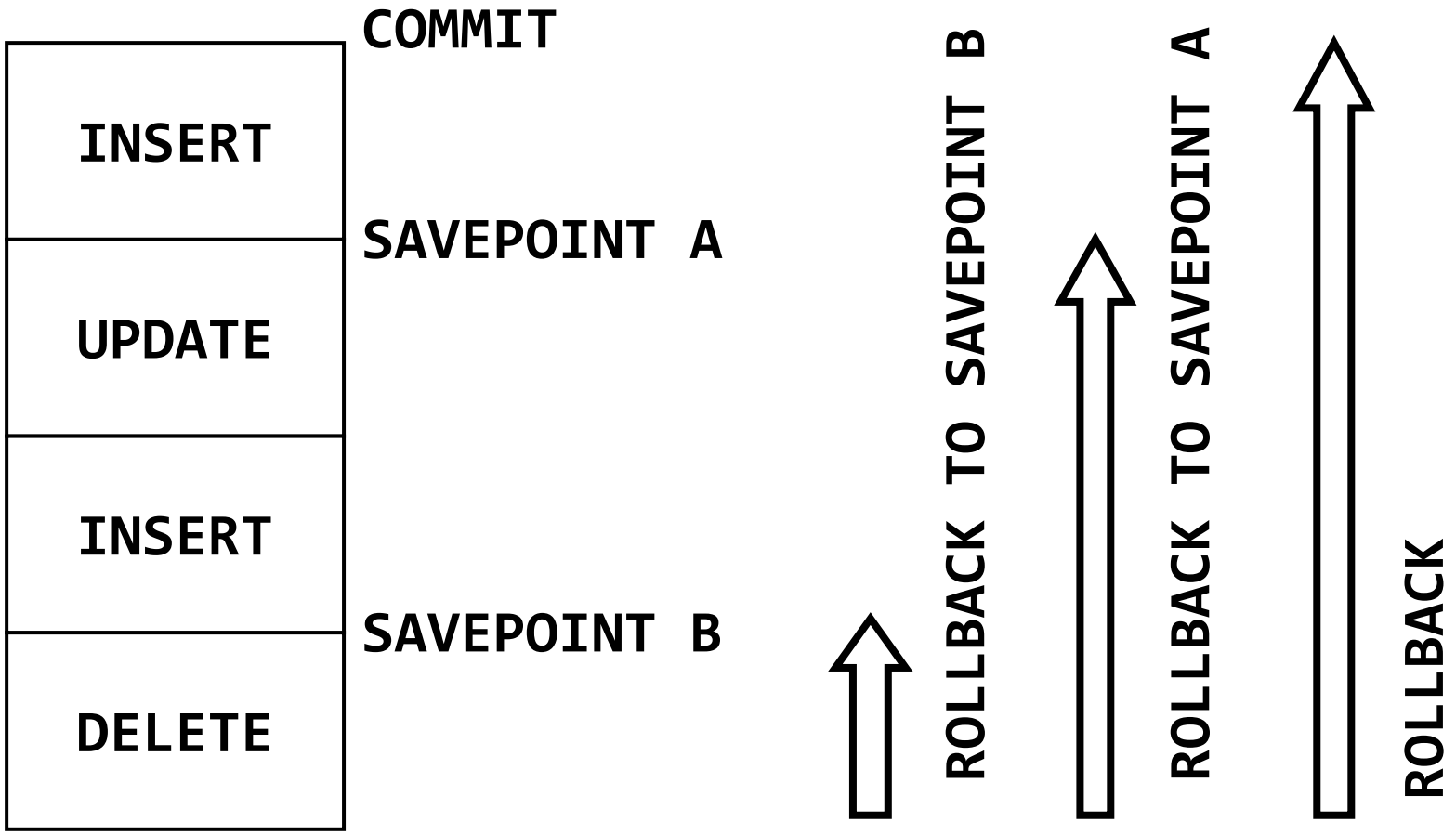
- Motivation
- Problemfälle
- Beispiel
- Lösungsmöglichkeit

Transaktionen (Einleitung)

- bisher: jede Änderung direkt auf der Datenbank; entspricht COMMIT nach jedem DB-Befehl
- ok, wenn alle Befehle unabhängig (Kommentarfunktion)
- kritisch, wenn mehrere Aktion voneinander abhängen (z. B. Geldüberweisung), da Abbruch in der Mitte möglich
- mehrere Schritte werden zu Transaktion zusammengefasst
- Änderungen in der Datenbank finden erst am Ende der Transaktion (in SQL `COMMIT;` oder Element der Definitionssprache, oder Beenden der SQL-Sitzung) statt
- Zwischenzeitlich können Sicherungspunkte angelegt werden `SAVEPOINT <name>` (in Oracle, nicht Standard-SQL)
- Rücksprung zum SAVEPOINT mit `ROLLBACK` (zum Transaktionsanfang) oder `ROLLBACK <name>` zum konkreten SAVEPOINT

Ablaufmöglichkeiten bei Transaktionen

Nutzungsaktionen



Anmerkung: Standard-SQL kennt nur ROLLBACK

Ende einer Transaktion

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- `COMMIT` Befehl macht alle Änderungen persistent
- `ROLLBACK` nimmt alle Änderungen der aktuellen Transaktion zurück
- z. B. Oracle: `ROLLBACK TO` <savepoint> nimmt alle Änderungen bis zum <savepoint> zurück
- Oracle durch DDL-Befehle (`CREATE`, `DROP`, `ALTER`, `RENAME`); in Derby nicht
- Oracle: nutzende Person beendet Sitzung; meldet sich vom System ab; in Derby nicht
- Abbruch des Benutzungsprozesses

ACID-Transaktionen

Atomicity (Atomarität)

Transaktionen werden entweder ganz oder gar nicht ausgeführt

Consistency (Konsistenz)

Transaktionen überführen die Datenbank von einem konsistenten Zustand in einen anderen konsistenten Zustand

Isolation (Isolation)

Nebenläufige (gleichzeitige) Transaktionen laufen jede für sich so ab, als ob sie alleine ablaufen würden.

Durability (Dauerhaftigkeit)

Die Wirkung einer abgeschlossenen Transaktion bleibt (auch nach einem Systemausfall) erhalten.

Zu ACID-Transaktionen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Tatsächlich muss ein DBMS nur garantieren:

- Das Ergebnis einer Transaktion ist *gleichwertig* zu einer ACID-Transaktion

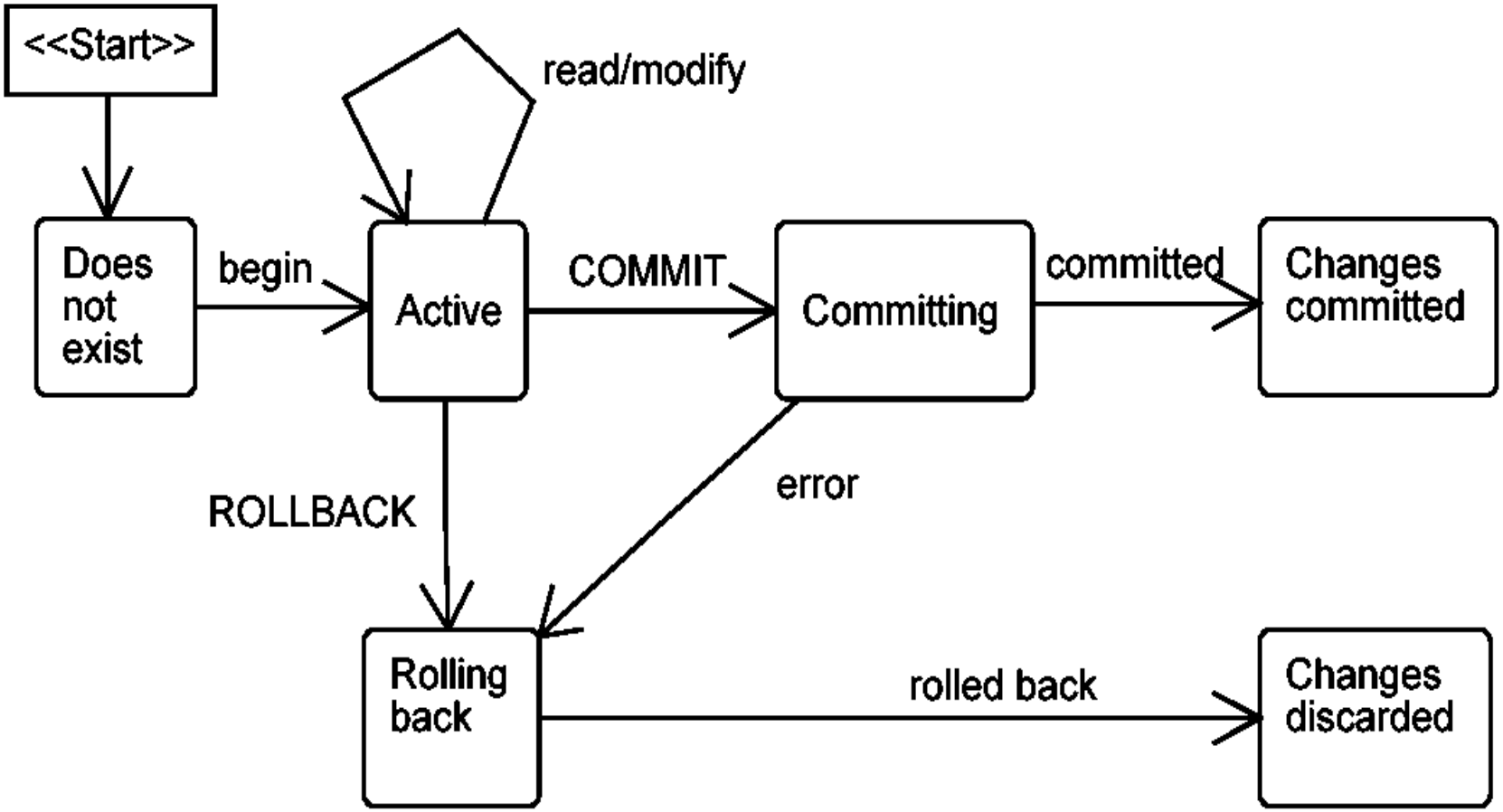
Die Transaktionsbehandlung betrifft:

- *die Synchronisation* von mehreren gleichzeitig ablaufenden Transaktionen
- *das Recovery*, d.h. die Behebung von Fehlern

Transaktionen können erfolgreich (`commit`*)* oder erfolglos *(abort, für DB-Nutzung* `rollback`*)* abgeschlossen werden

Transaktionen

- Zustände einer Transaktion



Transaktion in SQL

Erster SQL-Befehl	Beginn der ersten Transaktion
Folge von SQL-Befehlen	
`commit`	Festschreiben
Nächster SQL-Befehl	Beginn der nächsten Transaktion
Folge von SQL-Befehlen	
`rollback`	Rücksetzen zum vorhergehenden commit
Nächster SQL-Befehl	Beginn der nächsten Transaktion
Folge von SQL-Befehlen	
`commit`	Festschreiben

Atomarität (1/2)

Analogie

Video

Eine Transaktion wird entweder ganz oder gar nicht ausgeführt

- Konto-Beispiel: Umbuchung von K1 auf K2
  - Abbuchung von Konto K1
  - Zubuchung auf Konto K2

  → entweder beide Operationen werden durchgeführt oder keine

Abbruch kann stattfinden aufgrund:

- Selbstaufgabe (z.B. Benutzungsabbruch)
- Systemabbruch durch das DBMS (z.B. wg. Deadlock)
- Crash (Hardware-/Softwarefehler)

Atomarität (2/2)

Wie stellt das Datenbanksystem die Atomarität sicher?

- bei einer lesenden Transaktion: kein Problem
- bei einer schreibenden Transaktion:
  - bei Abbruch: Änderungen müssen rückgängig gemacht werden (bzw. dürfen gar nicht erst sichtbar werden)
  - im Erfolgsfall: alle Änderungen müssen sichtbar werden

Realisierungsmöglichkeit: das *Schattenspeicherverfahren*

Schattenspeicherkopie

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Realisierungsmöglichkeit: das Schattenspeicherverfahren

- Kopie einer DB: Arbeitskopie

Durchführung einer Transaktion:

- Änderung: wird nur auf der Arbeitskopie gemacht
- Commit (erfolgreich): DB := Arbeitskopie
- Abort (oder erfolgloses Commit): Arbeitskopie wegwerfen

Konsistenz

Konsistenz betrifft alle vorgegebenen Regeln:

- Datentypen und Bereiche bei Attributen
- PRIMARY KEY
- FOREIGN KEY
- CONSTRAINTs
- TRIGGER

→ technische Realisierung z.B. wie bei der Atomarität

Isolation

Parallel ablaufende Transaktionen sollen sich gegenseitig nicht beeinflussen, d.h. jede läuft für sich so ab, als sei sie die einzige Transaktion im System

Dient zur Vermeidung div. Probleme:

- lost-update-Problem
- dirty read
- non-repeatable-read
- phantom read

Beispiel: Mehrbenutzungsbetrieb

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- zwei parallel verlaufende Transaktionen:

```
T1:   BEGIN  A=A+100,   B=B-100   END
T2:   BEGIN  A=1.06*A,  B=1.06*B  END
```

- erste Transaktion transferiert 100 € vom Konto B zum Konto A
- zweite Transaktion schreibt beiden Konten 6 % Zinsen gut
- gibt keine Garantie, dass T1 vor T2 ausgeführt wird (oder umgekehrt), wenn beide zusammen gestartet werden. Jedoch gilt: Der Nettoeffekt *muss* äquivalent zu beiden Transaktionen sein, wenn sie seriell in irgendeiner Reihenfolge ablaufen würden

Beispiele für Schedules

- Betrachte folgenden Ablauf mit ineinander geschachtelten Abläufen (Schedules) :

T1:	A=A+100,		B=B-100
T2:		A=1.06*A,	B=1.06*B

- Kein Problem, aber bei diesem Beispiel:

T1:	A=A+100,		B=B-100
T2:		A=1.06*A, B=1.06*B	

- Zweiter Schedule aus Sicht des DBMS:

T1:	R(A), W(A),		R(B), W(B)
T2:		R(A), W(A), R(B), W(B)	

Anomalien im Mehrbenutzungsbetrieb (1/4)

- Verlorengegangene Änderungen (*Lost Update*)
  - WW-Konflikt
  - Gleichzeitige Änderung desselben Objekts durch zwei Transaktionen
  - Erste Änderung (aus nicht beendeter Transaktion) wird durch die zweite überschrieben

T1: **W(A)**, W(B), C
T2: **W(A)**, W(B), C

C für Commit

Anomalien im Mehrbenutzungsbetrieb (2/4)

- Zugriff auf schmutzige Daten (*Dirty Read*)
  - WR-Konflikt
  - “schmutzige“ Daten = geänderte Objekte, deren Änderungen von Transaktionen stammen, die noch nicht beendet sind
  - Dauerhaftigkeit der Änderungen nicht garantiert, da Transaktionen noch zurückgesetzt werden
  - Ungültige Daten werden somit durch andere Transaktion gelesen und (schlimmer noch!) vielleicht noch weiterpropagiert

T1:	R(A), **W(A)**,		R(B), W(B), **Rollback**
T2:		**R(A)**, W(A), C	

Anomalien im Mehrbenutzungsbetrieb (3/4)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Nicht-wiederholbares Lesen (*Unrepeatable Read*)
  - RW-Konflikt
  - Eine Transaktion sieht (bedingt durch parallele Änderungen) während ihrer Ausführung unterschiedliche Zustände des Objekts. Erneutes Lesen in der Transaktion liefert somit anderen Wert

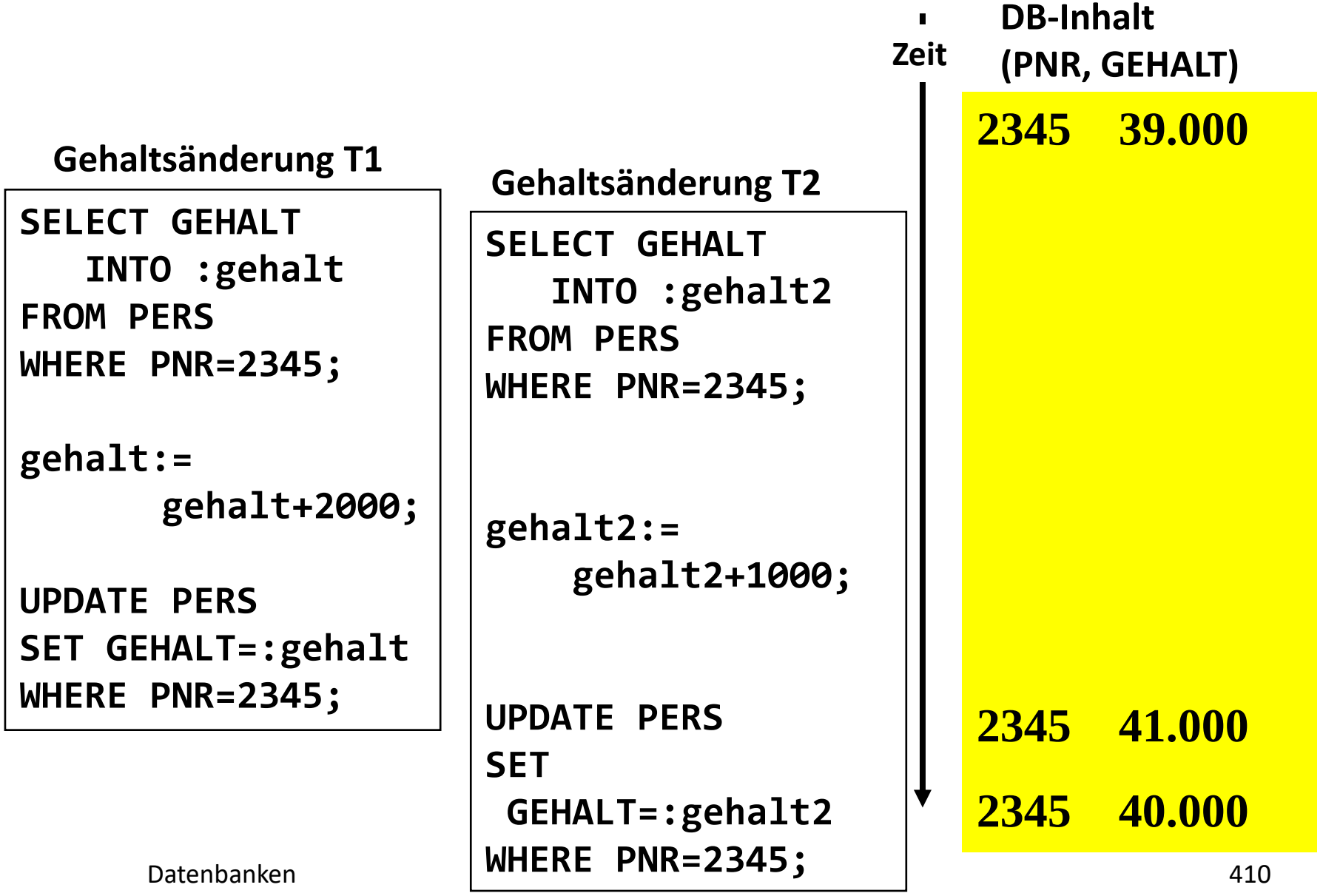
T1:	**R(A)**,		**R(A)**, W(A), C
T2:		R(A), **W(A)**, C	

Anomalien im Mehrbenutzungsbetrieb (4/4)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Phantom-Problem
  - Spezielle Form des Unrepeatable Read
  - Lesetransaktion: Mengenorientiertes Lesen über ein bestimmtes Suchprädikat P
  - Parallel läuft Änderungstransaktion, die die Menge der sich für das Prädikat qualifizierenden Objekte ändert
  - Folge: Phantom-Objekte, die durch parallele Einfüge- oder Löschvorgänge in Ergebnismenge auftauchen und/oder daraus verschwinden

Lost Update Beispiel



Dirty Read Beispiel

Gehaltsänderung T1

```
UPDATE PERS
SET GEHALT=
    GEHALT+1000
WHERE PNR=2345

        . . .

ROLLBACK
```

Gehaltsänderung T2

```
SELECT GEHALT
    INTO :gehalt
FROM PERS
WHERE PNR=2345

gehalt:=gehalt*1.05;

UPDATE PERS
SET GEHALT=:gehalt
WHERE PNR=2345
COMMIT
```

Zeit

DB-Inhalt (PNR, GEHALT)

2345	39.000
2345	40.000
2345	42.000
2345	39.000

Unrepeatable Read Beispiel

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Gehaltsänderung T1

```
UPDATE PERS
SET GEHALT=
   GEHALT+1000
WHERE PNR=2345

       . . .


UPDATE PERS
SET GEHALT=
   GEHALT+2000
WHERE PNR=3456

COMMIT
```

Gehaltssumme T2

```
SELECT SUM(GEHALT)
   INTO :summe
FROM PERS
WHERE PNR IN
      (2345,3456)
```

Inkonsistente Analyse
summe=85.000

DB-Inhalt
(PNR, GEHALT)

2345	39.000
3456	45.000
2345	40.000
3456	47.000

Zeit

Unrepeatable Read Beispiel

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Gehaltsänderung T1

```
UPDATE PERS
SET GEHALT=
   GEHALT+1000
WHERE PNR=2345

UPDATE PERS
SET GEHALT=
   GEHALT+2000
WHERE PNR=3456

COMMIT
```

Gehaltssumme T2

```
SELECT GEHALT INTO :g1
FROM PERS
WHERE PNR=2345

SELECT GEHALT INTO :g2
FROM PERS
WHERE PNR=3456
summe:=g1+g2
```

DB-Inhalt (PNR, GEHALT)

2345	39.000
3456	45.000
2345	40.000
3456	47.000

Zeit

Inkonsistente Analyse ohne schmutziges Lesen in T2

Phantom-Problem Beispiel

Lesetransaktion (Gehaltssumme der Abteilung 17 bestimmen)

```
SELECT SUM(GEHALT) INTO :sum1
FROM PERS
WHERE ANR=17

                . . .


SELECT SUM(GEHALT) INTO :sum2
FROM PERS
WHERE ANR=17

IF sum1<>sum2 THEN
      <Fehlerbehandlung>
```

Änderungstransaktion (Einfügen eines neuen Angestellten in Abteilung 17)

```
INSERT INTO PERS
  (PNR,ANR,GEHALT)
VALUES(4567,17,55.000)
```

Zeit

Isolationsgrade

Die gezeigten Effekte treten je nach Isolationsgrad *(isolation level)* auf:

Isolationsgrad	mögliche Effekte
0	Dirty Read, Non-Repeatable-Read, Phantom
1	Non-Repeatable-Read, Phantom
2	Phantom
3	-

Standard ist Isolationsgrad 3; geringere Grade können für bestimmte Anwendungen Sinn machen (z.B. grobe Statistik)

- man erreicht so eine Leistungssteigerung

Auch hier gilt: man sollte wissen, was man tut...

Isolationsgrade in SQL (eine Version)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Oracle: **`SET TRANSACTION ISOLATION LEVEL`**
`[ READ UNCOMMITED | READ COMMITTED`
`| REPEATABLE READ | SERIALIZABLE];`

in Derby (0 nicht unterstützt): **`{NONE=0, READ_UNCOMMITTED=1,`**
`READ_COMMITTED=2, REPEATABLE_READ=4, SERIALIZABLE=8}`

```
SET ISOLATION READ UNCOMMITTED;
SET ISOLATION READ COMMITTED;
SET ISOLATION REPEATABLE READ;
SET ISOLATION SERIALIZABLE;
```

Einstellung gilt für aktuelle Verbindung, nach **`COMMIT`** oder **`ROLLBACK`** wieder zurückgesetzt

Sperren in Derby: Lock granularity

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Derby can be configured for *table-level* locking. With table-level locking, when a transaction locks data in order to prevent any transaction anomalies, it always locks the entire table, not just those rows being accessed.

By default, Derby is configured for row-level locking. Row-level locking uses more memory but allows greater concurrency, which works better in multi-user systems. Table-level locking works best with single-user applications or read-only applications.

You typically set lock granularity for the entire Derby system, not for a particular application. […]

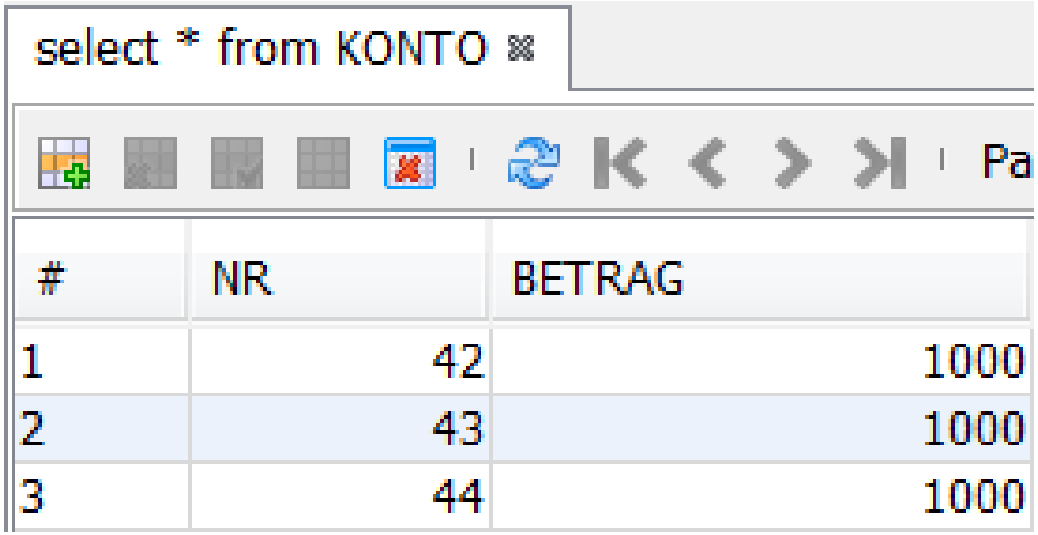
http://db.apache.org/derby/docs/10.15/devguide/cdevconcepts23810.html

Erinnerung: Kontobeispiel

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

```
Was wollen Sie machen?
 (2) Neues Konto
2
Welche Kontonummer? 67
Welcher Startbetrag? 1
Was wollen Sie machen?
 (1) vorhandene Konten
1
42 1000
43 1000
44 1000
67 1
```



#	NR	BETRAG
1	42	1000
2	43	1000
3	44	1000

Möglicher Ablauf mit READ UNCOMMITED

42: 1000	43: 1000
(3) Einzahlen Welche Kontonummer? 42 Welchen Betrag? 100	(3) Einzahlen Welche Kontonummer? 43 Welchen Betrag? 50
	(1) vorhandene Konten 42: 1100
(6) ROLLBACK (1) vorhandene Konten 42: 1000 43: 1050	43: 1050
	(5) COMMIT (1) vorhandene Konten 42: 1000 43: 1050

Möglicher Ablauf mit READ COMMITED (1/2)

	42: 1000	43: 1000
(3) Einzahlen		
Welche Kontonummer? 42		
Welchen Betrag? 100		
		Welche Kontonummer? 43
		Welchen Betrag? 50
		(1) vorhandene Konten
(1) vorhandene Konten		[warten]
[warten]		`java.sql.SQLTransactionRollbackException: Eine Sperre konnte aufgrund eines Deadlocks nicht angefordert werden. Zyklus der Sperren und beantragten Sperren:…`
42: 1100		
43: 1000		

Möglicher Ablauf mit READ COMMITED (2/2)

42: 1000 43: 1000

(1) vorhandene Konten
42: 1000
43: 1000

(3) Einzahlen
Welche Kontonummer? 42
Welchen Betrag? 100
(5) COMMIT

(1) vorhandene Konten
42: 1100
43: 1000

Möglicher Ablauf mit REPEATABLE READ

42: 1000 43: 1000

(1) vorhandene Konten
42: 1000
43: 1000

(3) Einzahlen
Welche Kontonummer? 42
Welchen Betrag? 100
[warten]
`java.sql.SQLTransaction RollbackException: Eine Sperre konnte innerhalb der vorgegebenen Zeit nicht angefordert werden.`

Isolation - Realisierung

Die Isolationsgrade werden erreicht durch Sperren (locks)

- Lesesperren (shared lock, s)
- Schreibsperren (exclusive lock, x)

→ Lesesperren sind mit anderen Lesesperren kompatibel, aber nicht mit Schreibsperren;
Schreibsperren sind mit nichts kompatibel

Die Sperrgranularität ist von der Implementierung eines DBMS abhängig (z.B. Tabellensperren vs. Tupelsperren)

Oracle:

```
LOCK TABLE T IN EXCLUSIVE MODE;
LOCK TABLE T IN EXCLUSIVE MODE NOWAIT;
```

Freigabe der Sperren nach Transaktionsende

2-Phasen-Sperrprotokoll

Relationale Datenbanken arbeiten üblicherweise pessimistisch,

und zwar nach dem *2-Phasen-Sperrprotokoll*

- *eine Transaktion muss jedes Objekt, das sie lesen will, mit einer Lesesperre (s-lock) versehen*
- *eine Transaktion muss jedes Objekt, das sie ändern will, mit einer Schreibsperre (x-lock) versehen*
- *eine Transaktion darf kein Objekt mehr neu sperren, nachdem sie die erste Sperre freigegeben hat*

Probleme mit ACID-Transaktionen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Entwurfsanwendungen: Entwurf startet, Transaktion beginnt, nach drei Monaten ist der Entwurf fertig, Transaktion wird aber beim commit abgebrochen (… bitte von vorne anfangen...)

  →gefragt sind sog. *lange Transaktionen: Syncpoints* oder *Sagas*

- *Mobile Geräte: Daten werden auf Datenbank in (Notebook, PDA) geladen, danach Trennung von Hauptdatenbank, nutzende Person macht Änderungen, verbindet sich wieder mit der DB,*

  → was tun? alle Änderungen zurückweisen? Haupt-DB sperren?
  → Synchronisation ist gefragt, evtl. mit vorheriger Anzeige möglicher Arbeitsbereiche

14. Views und Datenbankverwaltung

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Video

- Views
- Änderungen in Views
- Organisation der DB
- Zugriffsrechte

Sichtkonzept (Views 1/2)

- Sicht (View): mit eigenem Namen bezeichnete, aus Basisrelation abgeleitete, virtuelle Relation (View-Name wie Tabellen-Name verwendbar)
- Views sind das Ergebnis einer Anfrage, auf dem weitere Operationen durchgeführt werden können
- Views können jedes mal neu erzeugt werden oder nur einmal und dann gespeichert (materialized view)
- Gespeicherte Views müssen nach jedem Update der Basisrelationen geändert werden
- Wir betrachten keine materialized Views
- Korrespondenz zum externen Schema bei ANSI SPARC (nutzende Person sieht jedoch mehrere Views und Basisrelationen)

Sichtkonzept (Views 2/2)

```
CREATE VIEW Germany AS
   SELECT name, population
   FROM City
   WHERE Country='D'
```

- Vorteile
  - Erhöhung der Nutzungsfreundlichkeit (z.B. Verbergen komplexer Joins in einer View)
  - Datenschutz
- Löschen eines Views

```
DROP VIEW Germany
```

View-Update-Problem

- Änderungsoperationen auf Sichten erfordern, dass zu jedem Tupel der Sicht zugrunde liegende Tupel der Basisrelationen eindeutig identifizierbar sind

- Sichten auf einer Basisrelation sind nur änderbar, wenn der Primärschlüssel in der Sicht enthalten ist

- Wenn Tupelanteile bei **`INSERT`** eindeutig auf die darunter liegenden Basisrelationen abgebildet werden können, könnten fehlende Werte durch NULL aufgefüllt werden (Constraints sind zu beachten)

  [Allerdings typischerweise keine Veränderung auf zusammengesetzten View möglich]

Problemquellen bei Views

1. Zeilen löschen, wenn der View Gruppenfunktionen (z.B. **`COUNT`**), **`GROUP BY`** oder **`DISTINCT`** enthält oder in der View-Definition **`WITH READ ONLY`** steht
2. Zeilen ändern, wenn 1. oder es berechnete Spalten (z.B. **`A+B`**) gibt
3. Zeilen hinzufügen, wenn 1. oder 2. oder es in den Basistabellen eine Spalte mit **`NOT NULL`** gibt, die nicht im View liegt

Beispiel: View-Probleme

R

A	B
a	b
x	b

S

B	C
b	c
b	z

RS

A	B	C
a	b	c
x	b	c
a	b	z
x	b	z

```
CREATE VIEW RS AS
    SELECT A, R.B, C
        FROM R, S
    WHERE R.B=S.B;
```

- Löschen von (a,b,c) führte zum Verlust von (a,b,z)
- alleiniges Ändern von (a,b,c) nach (a,b,d) geht nicht
- Einfügen von (a,b,d) ebenfalls

Organisation der DB (in Oracle)

- Datenbanken organisieren sich selbst über Tabellen
- gibt z. B. Tabellen in denen die existierenden Tabellen, Prozeduren und Trigger stehen

SELECT * FROM SYS.SYSTABLES;

#	TABLEID	TABLENAME	TABLETYPE	SCHEMAID	LOCK
20	e03f4017-0115-382c-0...	SYSROLES	S	8000000d-00d0-...	R
21	9810800c-0121-c5e2-e...	SYSSEQUENCES	S	8000000d-00d0-...	R
22	9810800c-0121-c5e1-a...	SYSPERMS	S	8000000d-00d0-...	R
23	9810800c-0134-14a5-4...	SYSUSERS	S	8000000d-00d0-...	R
24	0a42c1d3-014f-40d4-1...	KONTO	T	2e31c0ab-014f-4...	R
25	8fe48195-014f-50ce-b4...	PROTOKOLL	T	2e31c0ab-014f-4...	R
26	e0b94027-014f-53fe-c4...	TR	T	2e31c0ab-014f-4...	R
27	1b8a4e72-014f-53fe-c4...	GEBOT	T	2e31c0ab-014f-4...	R

Einrichtung von Usern (Oracle / Derby)

```
CREATE USER <user>
IDENTIFIED BY <password>;

CREATE USER Egon
IDENTIFIED BY ottilie01
QUOTA 5M ON system;
```

- nächster Schritt: Einrichtung der Systemprivilegien für benutzende Person oder Prozess (viele Möglichkeiten), u.a.

```
GRANT create session TO Egon
```

- Derby

```
CALL SYSCS_UTIL.SYSCS_SET_DATABASE_PROPERTY(
    'derby.user.ich', 'ich');
CREATE SCHEMA ich AUTHORIZATION ich;
```

Verwaltung einer Datenbank

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Für DB-Projekte gibt es meist zwei administrierend tätige Personenarten (Rollen)
  - DB-Systemadministration: Physikalische Einrichtung von Datenbanken (z.B. Name, Speicherbereich), Nutzungsverwaltung
  - Projekt-DB-Administration: Verantwortlich für die Tabellen des Projekts, wer hat welche Rechte auf welchen Tabellen

- Abhängig vom DB-System müssen beide eng zusammenarbeiten (auch in mittelgroßen Unternehmen oft eine Person)

Systemprivilegien

```
GRANT <privilege-list>
TO <user-list> | PUBLIC
[WITH ADMIN OPTION];
```

- **PUBLIC:** jeder erhält das Recht
- **ADMIN OPTION:** empfangende Person darf Recht weitergeben

Rechte entziehen:

```
REVOKE <privilege-list> | ALL
FROM <user-list> | PUBLIC;
```

- nur wenn man dieses Recht selbst vergeben hat (im Fall von **ADMIN OPTION** kaskadierend)

Systemprivilegien

- berechtigen zu Schemaoperationen

- **`CREATE [ANY] TABLE / VIEW / TYPE / INDEX / CLUSTER / TRIGGER/ PROCEDURE:`**

  nutzende Person darf die entsprechenden Schema-Objekte erzeugen

- **`ALTER [ANY] TABLE / TYPE/ TRIGGER / PROCEDURE:`**

  nutzende Person darf die entsprechenden Schema-Objekte verändern

Systemprivilegien

- **`DROP [ANY] TABLE / VIEW / TYPE / INDEX / CLUSTER / TRIGGER / PROCEDURE:`**

  nutzende Person darf die entsprechenden Schema-Objekte löschen

- **`SELECT / INSERT / UPDATE / DELETE [ANY] TABLE:`**

  nutzende Person darf in Tabellen Tupel lesen/ erzeugen/ verändern/ entfernen

- **`ANY:`** Operation in *jedem* Schema erlaubt,
- ohne **`ANY:`** Operation nur im eigenen Schema erlaubt

Rollen

- Privilegien können Rollen zugeordnet werden, die dann wieder nutzenden Personen zugeordnet werden können

```
CREATE ROLE manager;

GRANT create table, create view
TO manager;

GRANT manager TO i03d09, i00d02;
```

- Der Entwurf einer sinnvollen Rollenmatrix ist nicht trivial!

Objektprivilegien (1/3) [entspricht Projektebene]

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

berechtigen dazu, Operationen auf existierenden Objekten auszuführen:

- Niemand sonst darf mit einem Datenbankobjekt einer konkreten nutzenden Person arbeiten, außer
- nutzende Person (oder DBA) erteilt explizit entsprechende Rechte:

```
GRANT <privilege-list> | ALL
        [(<column-list>)]
ON <object>
TO <user-list> | PUBLIC
[ WITH GRANT OPTION ];
```

Objektprivilegien (2/3)

- **<object>: TABLE, VIEW, PROCEDURE/FUNCTION, TYPE**
- Tabellen und Views: Genauere Einschränkung für **INSERT, REFERENCES** und **UPDATE** durch **<column-list>**
- **<privilege-list>: DELETE, INSERT, SELECT, UPDATE** für Tabellen und Views,

  INDEX, ALTER und **REFERENCES** für Tabellen

  EXECUTE für Prozeduren, Funktionen und **TYPE**n
- **ALL:** alle Privilegien, die man an dem beschriebenen Objekt hat
- **WITH GRANT OPTION:** empfangende Person darf das Recht weitergeben

Objektprivilegien (3/3)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Rechte entziehen:

```
REVOKE <privilege-list> | ALL
ON <object>
FROM <user-list> | PUBLIC
[CASCADE CONSTRAINTS];
```

- **CASCADE CONSTRAINTS** (bei **REFERENCES**): alle referenziellen Integritätsbedingungen, die auf einem entzogenen **REFERENCES**-Privileg beruhen, fallen weg
- Berechtigung von mehreren nutzenden Personen erhalten: Fällt mit dem letzten **REVOKE** weg
- im Fall von **GRANT OPTION** kaskadierend
- Abfrage von Eigenschaften über Systemtabellen

Beispiele

```
GRANT select,
   update(name,code)
ON Country
TO egon, manager

GRANT select,insert
ON City
TO PUBLIC

REVOKE select,insert
ON Country
FROM manager
```

Object Privilege	Table	View	Sequence	Procedure
ALTER	√		√	
DELETE	√	√		
EXECUTE				√
INDEX	√			
INSERT	√	√		
REFERENCES	√			
SELECT	√	√	√	
UPDATE	√	√		

Zugriffsrechte

- Zugriffsrechte an Account gekoppelt
- Ausgangsrechte von Person mit Administrationsrechten vergeben, Derby: DB-Erzeuger

Schema-Konzept

- Jeder nutzenden Person ist sein *Database Schema* zugeordnet, in dem „ihre" Objekte liegen.
- Bezeichnung der Tabellen *global* durch

  <username>.<table> (z.B. **xmaier.City** für die Tabelle **City** der nutzenden Person **xmaier**),
- im eigenen Schema durch **<table>** oder **<ich>.<table>** nutzbar

Synonyme

- Schemaobjekt unter einem anderen Namen als ursprünglich ansprechbar:

```
CREATE [PUBLIC] SYNONYM <synonym>
   FOR <schema>.<object>;
```

- Ohne **PUBLIC**: Synonym ist nur für nutzende Person definiert
- **PUBLIC** ist das Synonym systemweit verwendbar. Geht nur mit **CREATE ANY SYNONYM** -Privileg

Beispiel:

```
CREATE SYNONYM City2
       FOR db07ws65.City
```

(man muss Zugriffsrechte auf die Tabelle haben)

löschen:

```
DROP SYNONYM <synonym>
```

Zugriffseinschränkung über Views (1/2)

- **`GRANT SELECT`** kann nicht auf Spalten eingeschränkt werden

  Stattdessen: Views verwenden

```
GRANT SELECT [<column-list>] -- nicht erlaubt
ON <table>
TO <user-list> | PUBLIC
[WITH GRANT OPTION];
```

- kann ersetzt werden durch

```
CREATE VIEW <view> AS
  SELECT <column-list> FROM <table>;

GRANT SELECT
ON <view>
TO <user-list> | PUBLIC
[WITH GRANT OPTION];
```

Zugriffseinschränkung über Views (2/2)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- db07ws65 besitzt Tabelle *Country,* will *Country* ohne Hauptstadt und deren Lage für db07ws00 les- und schreibbar machen
- View mit Lese- und Schreibrecht für db07ws00 :

```
CREATE VIEW pubCountry AS
  SELECT Name, Code, Population, Area
  FROM Country;

GRANT SELECT, INSERT, DELETE, UPDATE
ON pubCountry
TO db76ws00;
```

Beispiel: Konfiguration von Derby

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Während fast alle Konzepte der relationalen Datenbanken sich in allen Systemen stark ähneln, ist die eigentliche DB-Konfiguration extrem systemindividuell
- Derby kann zur Nutzungsverwaltung an LDAP angeschlossen werden, weitere DB nutzen oder lokalen, zur DB gehörenden Bereich
- Derby bietet dazu Konfigurationseinstellungen
  - für die gesamte DB-Installation (config-Datei)
  - für einzelne Datenbanken
- Änderungen von Eigenschaften
  - können direkt wirken
  - erst nach einem Neustart der DB wirksam sein
  - erst nach dem Neustart des DB-Servers wirksam sein