Datenbanken und SQL

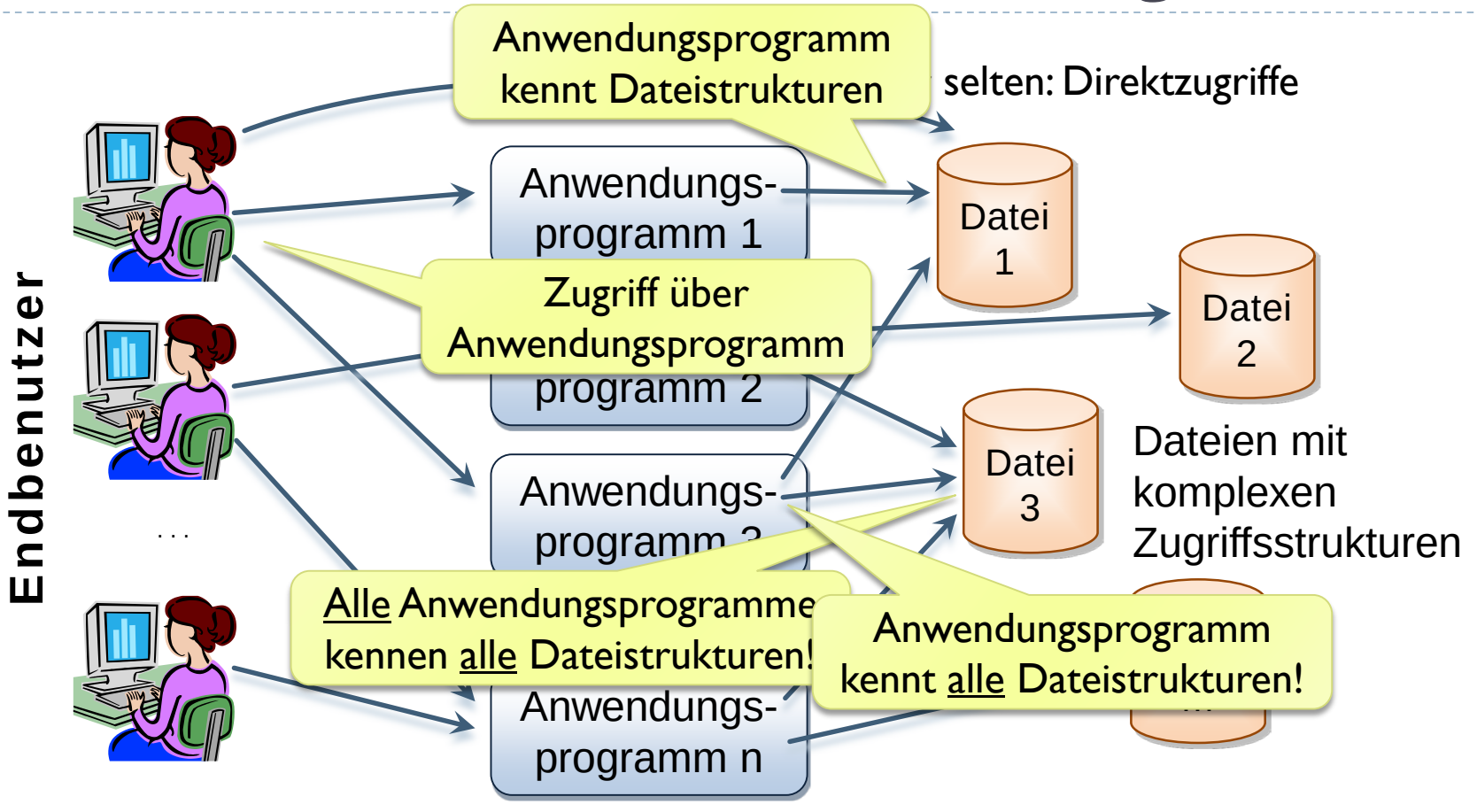
Kapitel 1

Übersicht über Datenbanken

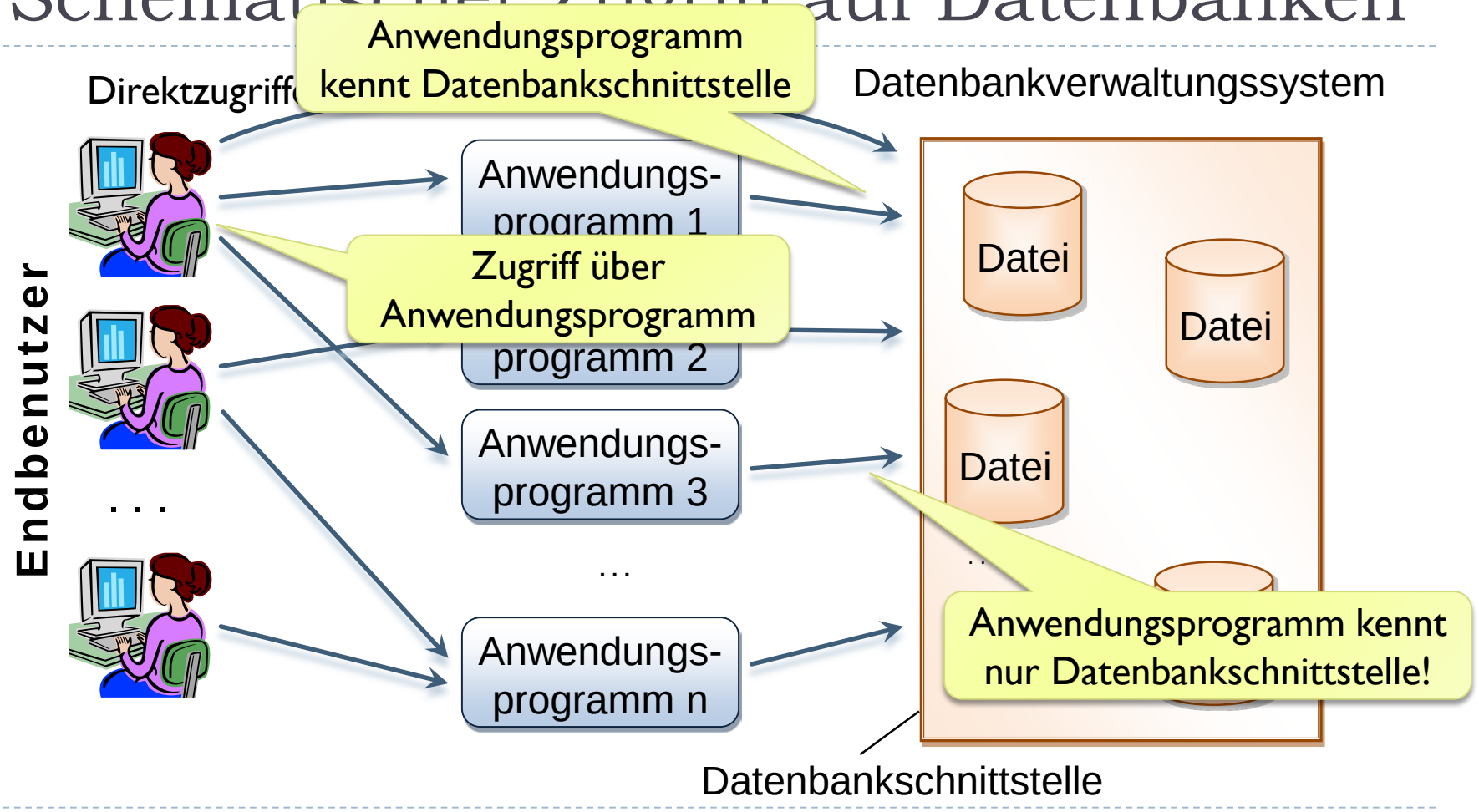
Übersicht über Datenbanken

- Vergleich: Datenorganisation versus Datenbank
- Definition einer Datenbank
- Bierdepot: Eine Mini-Beispiel-Datenbank
- Anforderungen an eine Datenbank
- Der Datenbankadministrator
- Relationale Datenbanken
- Nicht relationale Datenbanken
- Transaktionen

Mehrbenutzerbetrieb mit Datenorganisation



Schematischer Zugriff auf Datenbanken



Definition einer Datenbank

- **Definition (Datenbank):**

- Eine Datenbank ist eine Sammlung von Daten,
  - die untereinander in einer logischen Beziehung stehen und
  - die von einem eigenen Datenbankverwaltungssystem (DBMS) verwaltet werden.

Database Management System

Bierdepot

Nr	Sorte	Hersteller	Typ	Anzahl
1	Hell	Lammsbräu	Kasten	12
3	Roggen	Thurn und Taxis	Kasten	10
4	Pils	Löwenbräu	Kasten	22
8	Export	Löwenbräu	Fass	6
11	Weißbier	Paulaner	Kasten	7
16	Hell	Spaten	6er Pack	5
20	Hell	Spaten	Kasten	12
23	Hell	EKU	Fass	4
24	Starkbier	Paulaner	Kasten	4
26	Dunkel	Kneitinger	Kasten	8
28	Märzen	Hofbräu	Fass	3
33	Pils	Jever	6er Pack	6
36	Alkoholfreies Bier	Löwenbräu	6er Pack	5
39	Weißbier	Erdinger	Kasten	9
47	Alkoholfreies Pils	Clausthaler	Kasten	1

Lesezugriff auf das Bierdepot

```
SELECT    Sorte, Hersteller, Anzahl
FROM      Bierdepot
WHERE     Sorte = 'Weißbier' ;
```

Nr	Sorte	Hersteller	Typ	Anzahl
	Weißbier	Paulaner		7
	Weißbier	Erdinger		9
11	Weißbier	Paulaner	Kasten	7

Schreibzugriffe auf das Bierdepot

```
INSERT
INTO      Bierdepot
VALUES    (43, 'Dunkel', 'Kaltenberg', 'Kasten', 6) ;

UPDATE    Bierdepot
SET       Anzahl = Anzahl - 1
WHERE     Nr = 11 ;

DELETE
FROM      Bierdepot
WHERE     Nr = 47 ;
```

Fügt eine neue Zeile hinzu

Reduziert die Anzahl zu Artikel 11

Löscht die Zeile zu Artikel 47

Datenbankhersteller

- Angaben zu Marktanteilen streuen (je nach Quelle)

Hersteller	Marktanteil geschätzt
Oracle	33%-48%
DB2	20%-30%
Microsoft	16%-20%
SAP	3%-4%

- Open Source Datenbanken:

MySQL
PostgreSQL

Anforderungen an eine Datenbank (1)

- Sammlung logisch verbundener Daten
- Speicherung der Daten mit möglichst wenig Redundanz
  - Beispiel für Redundanz:
    - Einkauf ←→ Lagerhaltung ←→ Verkauf
    - da: Lager = Einkauf – Verkauf
- Abfragemöglichkeit und Änderbarkeit

Anforderungen an eine Datenbank (2)

- Logische Unabhängigkeit der Daten von der Speicherung
- Zugriffsschutz
- Integrität und Korrektheit
- Mehrfachzugriff (Concurrency)
- Zuverlässigkeit und Audit
- Ausfallsicherheit
- Funktionalität zur Kontrolle der Datenbank

Datenbankadministrator

- **Aufgaben des Administrators:**
  - Einrichten einer Datenbank, Zugriffsschutz
  - Betrieb und Kontrolle der Datenbank
- **Datenbank-Schnittstellen:**
  - **DDL** Data Description Language
  - Kontrollsprache (in DDL integriert)
  - **DML** Data Manipulation Language
- **Aufgabenteilung:**
  - Anwender: DML (Select, Insert, Update, Delete)
  - Administrator: DDL (Create Table, Create View, Grant, …)

Datenbankmodelle im Überblick

- Relationale Datenbanken
  - Oracle, DB2, MS SQL Server, MySQL, PostgreSQL, Sybase
- Objektorientierte und objektrelationale Datenbanken
  - Oracle, PostgreSQL
- Hierarchische Datenbanken
  - IMS
- Netzwerkartige Datenbanken
  - IDMS, UDS
- Neue Datenbanksysteme
  - NOSQL: MongoDB

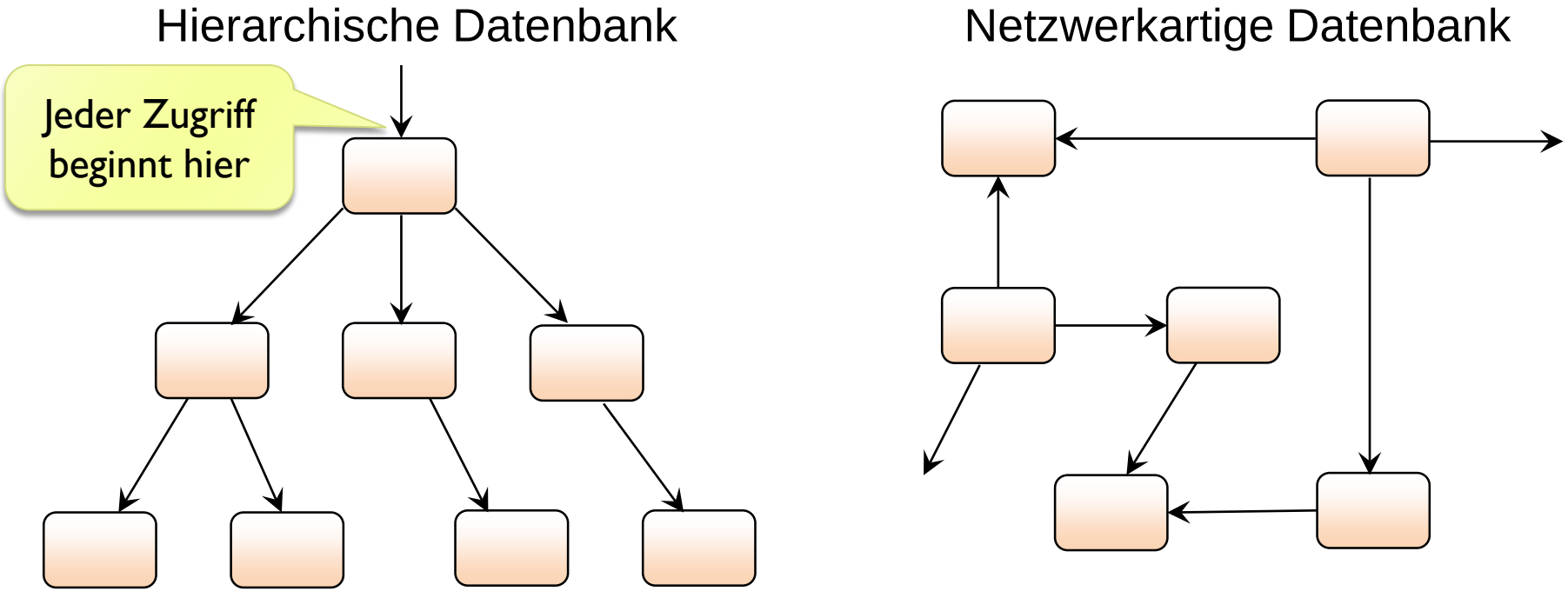
Relationale Datenbanken

	Relationale Datenbanken
Vorteile	Leichte Änderbarkeit des Datenbankaufbaus, mathematisch fundiert, leicht programmierbar und zu verwalten
Nachteile	Häufig viele Ein-/Ausgaben notwendig, erfordert hohe Rechnerleistung, erzeugt Redundanz

Objektorientierte Datenbanken

	Objektorientierte und objektrelationale Datenbanken
Vorteile	Objektorientierter Ansatz, universell einsetzbar, noch relativ einfach programmierbar und zu verwalten, (meist) aufwärtskompatibel zu relationalen Datenbanken
Nachteile	Relativ viele Ein-/Ausgaben notwendig, erfordert eine relativ hohe Rechnerleistung, teilweise recht komplexer Aufbau

Hierarchische und Netzwerk-Datenbanken



Hierarchische und Netzwerk-Datenbanken

	hierarchisch	netzwerkartig
Vorteile	sehr kurze Zugriffszeiten, minimale Redundanz	kurze Zugriffszeiten, geringe Redundanz
Nachteile	Strukturänderung kaum möglich, komplexe Programmierung	Strukturänderung nicht einfach, relativ komplexe Programmierung

NoSQL Datenbanken

- NoSQL = Not Only SQL
- NoSQL Datenbanken gliedern sich in
  - Key/Value und dokumentenbasierte Datenbanken
    - z.B. CouchDB, MongoDB
  - Spaltenorientierte Datenbanken
    - z.B. Google Big Table, Simple DB von Amazon
    - HBase, Cassandra
  - Graphenorientierte Datenbanken
    - z.B. Sones, Neo4j

NoSQL Datenbankmodelle

- **Key/Value und dokumentenbasierte Modelle**
  - Schemafreie Modelle, daher sehr flexibel
  - Seit 1979 im ersten Einsatz
  - Lotus Notes ist dokumentenbasiert
- **Spaltenorientierte Modell**
  - Die Daten werden spaltenweise gespeichert!
  - Bei Anfragen nach wenigen Eigenschaften extrem performant
- **Graphen Modelle**
  - In Navigationsgeräten
  - Wie finde ich den besten Weg von A nach B?

Transaktionen

- **Abfrage**
  - Lesezugriff: Select
  - Query
  - Retrieval
- **Mutation**
  - Schreibzugriff: Insert, Update, Delete
- **Transaktion**
  - Konsistenzerhaltende Operation
  - Atomare Operation

Konsistenz und Redundanz

- **Definition (Konsistenz):**
- Eine Datenbank heißt in sich **konsistent**, wenn alle gespeicherten Daten untereinander widerspruchsfrei sind.

- **Definition (Redundanz):**
- Daten heißen **redundant**, wenn sie mehr als einmal in einer Datenbank abgespeichert werden, also an sich überflüssig sind.

Beispiel zu Redundanz und Konsistenz

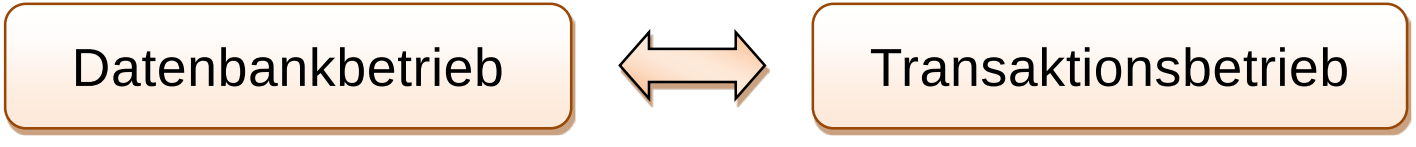
- **Es gilt:**
  - Warenbestand = Wareneingang – Warenausgang
- **In der Datenbank:**
  - Lagertabelle + Einkaufstabelle + Verkaufstabelle
- **Folgerung:**
  - Redundanz und Gefahr der Inkonsistenz
- **Frage:**
  - Welche der drei obigen Tabellen würden Sie entfernen?

Beispiel: Buchung

- **Bank speichert:**
  - Alle Kontostände S_i der n Kunden (i=1..n)
  - Summe der Kontostände S_{ges} aller Kunden (Redundanz!)
- **Szenario:**
  - Überweisung von 500 Euro von Kunde A nach Kunde B
  - 1. Schritt: Abbuchung von 500 Euro von Kunde A
  - 2. Schritt: Buchung von 500 Euro für Kunde B
  - Nach dem 1. Schritt: Absturz des Rechners
  - Datenbank ist nun inkonsistent ($\sum S_i \neq S_{ges}$) und fehlerhaft!
- **Folgerung: Transaktion muss atomar ablaufen!**

Transaktionen

- In betriebswirtschaftlichen Anwendungen und Buchungssystemen **zwingend** erforderlich, da
  - Inkonsistenzen nicht hinnehmbar sind
  - eine Buchung immer atomar ausgeführt werden muss
    - Atomare Ausführung heißt: Nichts oder alles wird ausgeführt
- Datenbanken garantieren auch im Fehlerfall die **atomare** Ausführung von Transaktionen
- Folgerung:
  - Datenbanksystem und Transaktionssystem sind Synonyme



A C I D

- Ein **Transaktionsbetrieb** muss folgende Bedingungen erfüllen:

- **A** **Atomarity** (Atomarität)
- **C** **Consistency** (Konsistenz)
- **I** **Isolation**
- **D** **Durability** (Dauerhaftigkeit)

A = Atomarität

- Eine Transaktion läuft immer **atomar** ab
- Eine noch laufende Transaktion kann **jederzeit**, insbesondere im Fehlerfall, zurückgesetzt werden

- In SQL:
  - **COMMIT**; Transaktion ist beendet, Daten sind gespeichert
  - **ROLLBACK**; Transaktion wird komplett zurückgesetzt

C = Konsistenz (Consistency)

- **Eine Transaktion ist konsistenzerhaltend**
- **Teiltransaktionen gibt es nicht:**
  - Eine Transaktion läuft komplett ab (Commit;) oder
  - Eine Transaktion wird nicht wirksam (Rollback;)

- **Folgerung:**
  - Eine Datenbank ist konsistent, wenn
    - alle Mutationen innerhalb von Transaktionen erfolgen
    - der Transaktionsmechanismus, insbesondere der Rollback, immer und jederzeit unterstützt wird (auch im Fehlerfall!)

I = Isolation

- Eine Transaktion läuft so ab, als sei sie allein im System
- Eine Transaktion ist vollständig isoliert von anderen parallel laufenden Transaktionen
- (Fast) gleichzeitige Zugriffe auf gleiche Daten müssen wegen Konsistenzverletzungen synchronisiert werden
- Beispiel zum Bierdepot bei 2 Verkaufsstellen:
  - 2 Kunden wollen das letzte Fass Pils von Bischofshof
  - Beide Kunden erfahren, dass noch Ware vorhanden ist
  - Aber: Nur ein Kunde bekommt das Fass

D = Dauerhaftigkeit

- **Die Daten werden dauerhaft gespeichert**
- **Ein Benutzer kann sich also auf Folgendes verlassen:**
  - Er erhält die Rückmeldung, dass seine Transaktion erfolgreich abgeschlossen wurde
  - Seine von dieser Transaktion manipulierten Daten sind daher dauerhaft und sicher gespeichert
- **Beispiel:**
  - Ein Kunde einer Versicherung verlässt sich darauf, dass seine vor 15 Jahren abgeschlossene Versicherung nicht verloren geht

Datenbanken und SQL

Kapitel 2

Das Relationenmodell

Das Relationenmodell

- Beispiel: Relation *Verkäufer-Produkt*
- Relationale Datenstrukturen
  - Begriffe
  - Definition: Relation, Relationale Datenbank
- Primärschlüssel
- Relationale Integritätsregeln
  - Regel 1: Entity Integritätsregel
  - Regel 2: Referenz Integritätsregel
- Relationale Algebra
  - Relationale Operatoren
  - Eigenschaften der relationalen Operatoren

Das Relationenmodell

- **Ziele des Relationenmodells sind**
  - Geringe Redundanz
  - Gute Handhabbarkeit
  - Einfache und schnelle Zugriffe
  - Sicherstellung von Konsistenz und Integrität

Keine doppelten Einträge

Einfache Befehle

Zugriffe über wenige Tabellen

Aufstellen von Regeln

- Folgerung:
  - Entsprechende Forderungen an Relationen

Beispiel: Relation VerkaeuferProdukt

VerkNr	VerkName	PLZ	VerkAdresse	Produktname	Umsatz
V1	Meier	80075	München	Waschmaschine	11000
V1	Meier	80075	München	Herd	5000
V1	Meier	80075	München	Kühlschrank	1000
V2	Schneider	70038	Stuttgart	Herd	4000
V2	Schneider	70038	Stuttgart	Kühlschrank	3000
V3	Müller	50083	Köln	Staubsauger	1000

Vorteil:

✓ Übersichtlich in einer Tabelle

Nachteil:

✓ Redundanz

✓ Schlechte Handhabbarkeit

Probleme mit VerkaeuferProdukt

- **Redundanz**
  - Je mehr ein Verkäufer verkauft, um so häufiger in Tabelle!
  - Ändert sich die Adresse eines Verkäufers, muss dies in **allen** entsprechenden Einträgen erfolgen. Sonst: **Inkonsistenz**!
- **Handhabung**
  - Soll Produkt Staubsauger aus dem Sortiment genommen werden, so ist auch Verkäufer Müller zu löschen!?
  - Verkäufer Schmidt kann erst eingetragen werden, wenn er etwas verkauft hat!?

Begriffe in relationalen Datenbanken

Formale relationale Bezeichner	Informelle Bezeichnung
Relation	Tabelle
Tupel	Zeile einer Tabelle
Kardinalität	Anzahl der Zeilen einer Tabelle
Attribut	Spalte einer Tabelle
Grad	Anzahl der Spalten einer Tabelle
Primärschlüssel	eindeutiger Bezeichner
Gebiet	Menge aller möglichen Werte

Relation

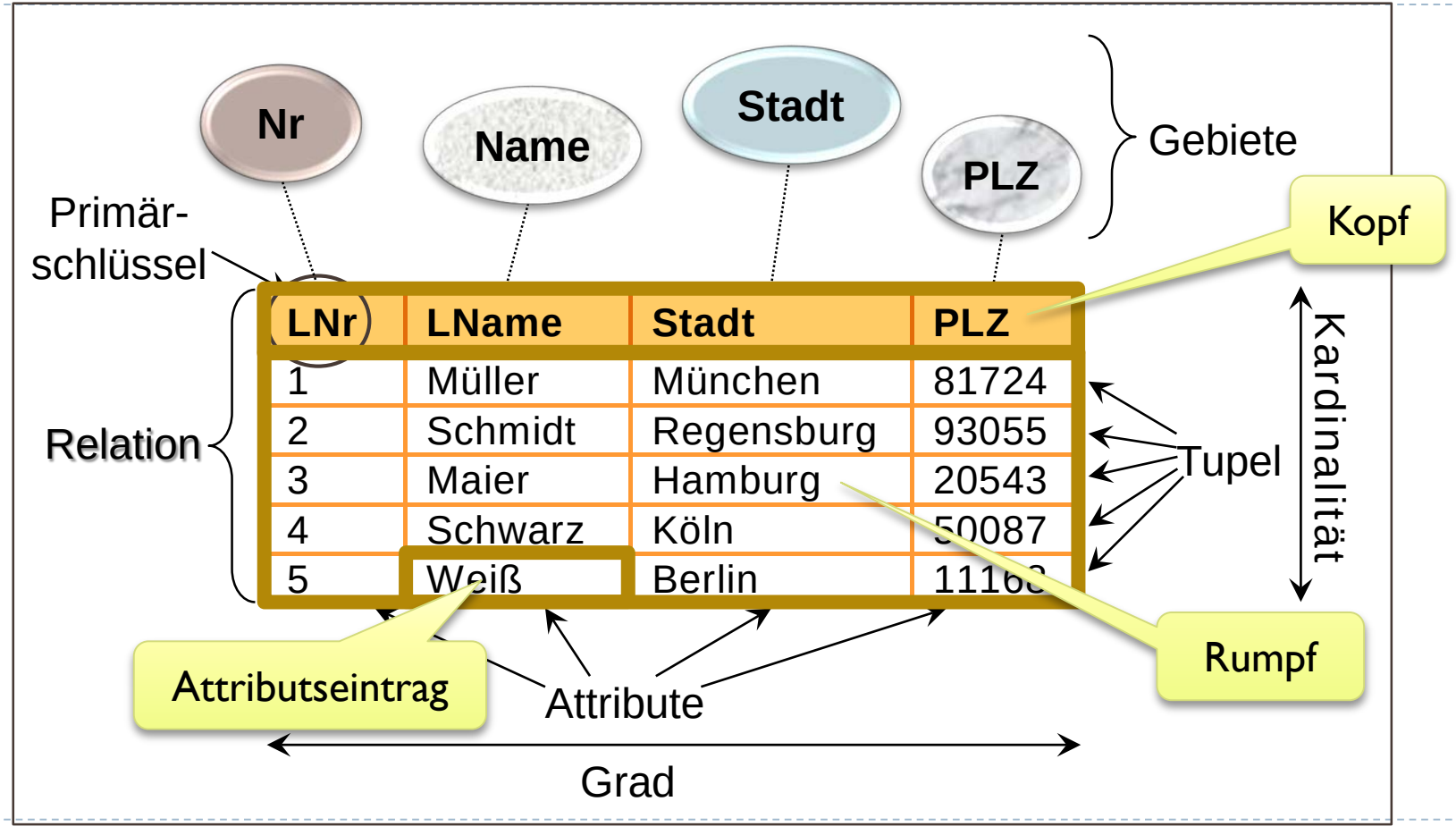
Tupel

Attribut

Kardinalität = 7

Grad = 2

Begriffe in relationalen Datenbanken (2)



Definition (Relation)

- Eine (normalisierte) **Relation** ist eine Tabelle, bestehend aus einem Kopf und einem Rumpf, mit folgenden vier Eigenschaften:

(1) Es gibt keine doppelten Tupel

(2) Tupel sind nicht geordnet (z.B. von oben nach unten)

(3) Attribute sind nicht geordnet (z.B. von links nach rechts)

(4) Alle Attribute sind atomar

Jede Zeile ist eindeutig

Die Reihenfolge spielt keine Rolle

Es gibt keine „erste“ oder „zweite“ Zeile

Die Reihenfolge spielt keine Rolle

Es gibt keine „erste“ oder „zweite“ Spalte

Es gibt nur Einzeleinträge, keine Aufzählungen oder Listen in einem Eintrag

Relation: (1) Keine doppelten Tupel

- Reduziert Redundanz ohne Informationsverlust
- Folgender doppelter Eintrag macht auch gar keinen Sinn:

VerkNr	VerkName	PLZ	VerkAdresse	Produktname	Umsatz
…	…	…	…	…	…
V1	Meier	80075	München	Kühlschrank	1000
V1	Meier	80075	München	Kühlschrank	1000
…	…	…	…	…	…

Relation: (2) Tupel sind nicht geordnet

- **Erleichtert das Einfügen neuer Zeilen**
  - Das DBMS entscheidet: Am Ende? In Lücke?
- **Erleichtert das Löschen von Zeilen**
  - Das DBMS entscheidet: Lücke? Nachrücken? Ersetzen?
- **Kein Informationsgewinn!**
  - Lieferant Maier steht an Position 17. Was bringt uns diese Info?
- **Aber: Performance**
  - Mit Sortierung könnte eventuell schneller zugegriffen werden

Relation: (3) Attribute sind nicht geordnet

- **Sehr seltene Änderungen bei Attributen**
  - Daher: kaum Nachteile oder Vorteile
- **Vermeidet Programmierschwäche**
  - Ausgabe der 7. Spalte ist nicht möglich
  - Zum Glück!
  - Beim Einfügen einer neuen Spalte wäre Programm falsch!
- **Konsequent**
  - Keine Tupelreihenfolge, also auch keine Attributreihenfolge

Relation: (4) Attribute sind atomar

- **Atomar heißt:**
  - Jeder Attributeintrag enthält nur einen Wert aus dem Definitionsgebiet
  - Aufzählungen sind nicht erlaubt
  - Listen sind nicht erlaubt
  - Atomar hat nichts damit zu tun, dass beispielsweise ein Wort aus einzelnen Buchstaben besteht!
- **Beispiel:** Attribut Stadt
  - In jeder Zeile steht eine Stadt (oder NULL)
  - Hat ein Mitarbeiter zwei Wohnsitze → Zwei Zeilen!

Nicht atomare Relation

- **Relation VerkaeuferProduktNF2**
  - Produkte zusammenfassen, Umsatz addieren

VerkNr	VerkName	PLZ	VerkAdresse	Produktname	Umsatz
V1	Meier	80075	München	Waschmaschine, Herd, Kühlschrank	17000
V2	Schneider	70038	Stuttgart	Herd, Kühlschrank	7000
V3	Müller	50083	Köln	Staubsauger	1000

- **Vorteile:** Kompakt und übersichtlich, keine Redundanz
- **Nachteile:** Komplexes Handling
  - Meier verkauft Staubsauger → nur Produktliste ergänzen
  - Schmidt verkauft Staubsauger → neue Zeile hinzufügen
  - Zeilen sind nicht mehr alle gleich lang!

Relationale Datenbank

- Eine relationale Datenbank ist eine Datenbank,

  die der Benutzer ausschließlich als eine Ansammlung von zeitlich variierenden, normalisierten Relationen passender Grade erkennt.

- Informell:
  - Eine relationale Datenbank ist eine Datenbank, die nur aus Relationen besteht.

Relationenarten

- **Basisrelationen**
  - Real existierende Relationen, persistenter Bestandteil der Datenbank
- **Sichten (Views)**
  - Virtuelle Relationen, abgeleitet aus Basisrelationen. Sie erscheinen dem Benutzer wie „normale“ Relationen.
- **Abfrageergebnisse**
  - Relationen, die temporär im Arbeitsspeicher während der Ausgabe existieren.
- **Temporäre Relationen**
  - Relationen, die nur temporär existieren. Sie werden bei bestimmten Ereignissen zerstört, etwa beim Beenden einer Transaktion.

Erzeugen von Relationen: SQL Befehle

- **CREATE TABLE** Tabellenname (. . .) ;
  - Erzeugen einer Basisrelation
- **CREATE VIEW** Sichtname AS . . . ;
  - Erzeugen einer Sicht
- **SELECT** Spalte **FROM** Tabelle . . . ;
  - Abfrage
- **CREATE TEMPORARY TABLE** Tabellenname (. . .) . . . ;
  - Erzeugen einer temporären Relation

Primärschlüssel (informell)

- **Der Primärschlüssel identifiziert jedes Tupel eindeutig**
- **Der Primärschlüssel besteht aus**
  - einem Attribut, z.B. LNr (Lieferantennummer)
  - mehreren Attributen (Primärschlüssel von VerkaeuferProdukt?)
- **Jede Relation besitzt einen Primärschlüssel**
  - Beweis:
    - Jedes Tupel ist eindeutig
    - Alle Attribute zusammen identifizieren daher jedes Tupel eindeutig
    - Alle Attribute zusammen könnten daher der Primärschlüssel sein
- **Was ist also der Primärschlüssel genau? → Klärung!**

Tabelle der chemischen Elemente

Protonen	Atomgewicht	Name	Symbol	Schmelzpkt.	Siedepkt.
1	1,008	Wasserstoff	H	-259	-253
2	4,003	Helium	He	-272	-269
3	6,941	Lithium	Li	180	1317
4	9,012	Berylium	Be	1278	2970
5	10,811	Bor	B	2300	2550
6	12,011	Kohlenstoff	C	3550	4827
7	14,007	Stickstoff	N	-210	-196
8	15,999	Sauerstoff	O	-218	-183
…	…	…	…		

- **Welches ist der eindeutige Identifikator (Primärschlüssel)?**
  - Protonenzahl, Atomgewicht, Name, Symbol?
  - Nicht geeignet: Schmelzpunkt, Siedepunkt (da nicht zwingend eindeutig)

Chemische Elemente: Primärschlüssel

- **Protonenzahl**
  - Jedes Element ist durch die Protonenzahl eindeutig identifiziert
  - Was in der Praxis gilt, sollte auch in Datenbank gelten → **Primärschlüssel**
- **Atomgewicht**
  - Nur zufällig haben alle Elemente unterschiedliches Atomgewicht
- **Name**
  - Jedes Element hat einen eindeutigen Namen, aber sprachabhängig
  - Wenn ein neues Element entdeckt wird, hat es noch keinen Namen!
- **Symbol**
  - Jedes chemische Element hat ein eindeutiges Symbol
  - Wenn ein neues Element entdeckt wird, hat es noch keinen Namen!

Definition (Superschlüssel)

- Ein eventuell aus mehreren einzelnen Attributen zusammen gesetztes Attribut heißt **Superschlüssel**, falls es eindeutig jedes Tupel identifiziert.

- **Superschlüssel in Tabelle der chemischen Elemente:**
  - (Protonen, Atomgewicht, Name, Symbol, Schmelzpunkt, Siedepunkt)
  - (Protonen, Atomgewicht)
  - (Name, Symbol)
  - Symbol
  - Protonen usw.

Definition (Schlüsselkandidat)

- Ein eventuell aus mehreren einzelnen Attributen zusammen gesetztes Attribut heißt **Schlüsselkandidat**, falls es
  - ein Superschlüssel ist und
  - minimal ist.

- **Schlüsselkandidaten in Tabelle der chemischen Elemente:**
  - Protonen
  - Name
  - Symbol

Definition (Primärschlüssel)

- Besitzt eine Relation mehrere Schlüsselkandidaten, so wird davon einer als **Primärschlüssel** ausgewählt.
- Alle anderen heißen **alternative Schlüssel**.

- Dies impliziert:
  - Gibt es nur einen Schlüsselkandidaten, so ist dieser Primärschlüssel.
- Primärschlüssel in Tabelle der chemischen Elemente:
  - Protonen

Relation Lagerbestand

Produktname	Produkttyp	Bestand	Preis
Staubsauger	T06	25	498
Staubsauger	T17	17	219
...	...	...	...
Küchenherd	T04	10	1598
Küchenherd	T06	7	1998

← Primärschlüssel →

Superschlüssel:
Produktname + Produkttyp + Bestand + Preis
Produktname + Produkttyp + Bestand
Produktname + Produkttyp + Preis
Produktname + Produkttyp

Minimal!!!!!

Einziger Schlüsselkandidat
also: Primärschlüssel

also: Schlüsselkandidat

Abfragen auf Schlüsselkandidaten

- Abfragen auf Schlüsselkandidaten liefern eindeutige Ergebnisse!

```
SELECT   Produktname, Preis
 FROM    Lagerbestand
 WHERE   Produktname = 'Staubsauger'
  AND    Produkttyp = 'T06' ;
```

Liefert eindeutiges Ergebnis!

Wir benötigen je eine Variable für Produktname und Preis

```
SELECT   Produktname, Preis
 FROM    Lagerbestand
 WHERE   Produktname = 'Küchenherd';
```

Liefert mehrdeutiges Ergebnis!

Wir benötigen ein unbekannt großes Feld für Produktname und Preis

Integrität in Datenbanken

- Integrität kommt von integer
- Eine integre Person ist eine Person, auf die ich mich verlassen kann
- Eine integre Datenbank ist eine Datenbank,
  - auf die ich mich verlassen will
  - deren Daten in sich konsistent sind
  - deren Daten korrekt sind und mit der realen Welt übereinstimmen
  - deren Daten vor fremden Blicken geschützt sind

Arten der Integrität

- **Physische Integrität**
  - Vollständigkeit der physischen Speicherstrukturen
  - verantwortlich: Datenbank, Betriebssystem
- **Ablaufintegrität**
  - Korrektheit der Programme, z.B. keine Endlosschleife
  - verantwortlich: Anwendungsprogrammierer, Datenbankdesigner
- **Zugriffsberechtigung**
  - Korrekte Zugriffsrechte
  - verantwortlich: Datenbank-Administrator
- **Semantische Integrität**
  - Übereinstimmung der Daten aus der nachzubildenden realen Welt mit den abgespeicherten Informationen
  - verantwortlich: Datenbankdesigner, Programmierer, Anwender

Wichtig für Administrator

Zu berücksichtigen im Programm

Zu berücksichtigen im Datenbankdesign

Folgerungen zur semantischen Integrität

- Gebiete D_j so weit wie möglich einschränken
- Attributwerte v_{ij} aus D_j auswählen (für alle i)
- Soweit möglich: Nummern automatisch vergeben
- Damit können Eingabefehler reduziert werden.
- Beispiel:
  - In einer Firma arbeiten Mitarbeiter zwischen 10 und 40 Stunden pro Woche
  - → $D_{Arbeitszeit} = [10..40]$ und nicht: $D_{Arbeitszeit}$ = Int-Wert

Entitäts-Integritätsregel

- **Erste Integritätsregel**
  - **Keine Komponente** des Primärschlüssels einer Basisrelation darf **nichts** enthalten

- **Wichtig**
  - Diese Regel gilt nur für Basisrelationen
  - Diese Regel gilt nicht für alternative Schlüssel
  - Kein Teilattribut eines Primärschlüssels darf leer sein
  - Es gibt einen eigenen „Nichts“-Wert; in SQL: **NULL**
  - Die Datenbank soll die 1. Integritätsregel immer überprüfen!

Definition (Fremdschlüssel)

- Ein Attribut einer Basisrelation heißt **Fremdschlüssel**,
  - falls das ganze **Attribut** nichts oder einen definierten Inhalt enthält,
  - eine Basisrelation existiert, so dass jeder definierte Wert des Fremdschlüssels einem Wert des Primärschlüssels jener Basisrelation entspricht.
- **Wichtig:**
  - Ein zusammengesetzter Fremdschlüssel darf nicht in einigen Teilattributen NULL-Werte besitzen und in anderen nicht!
  - Jeder Wert eines Fremdschlüssels bezieht sich auf einen existierenden Primärschlüsselwert!

Beispiel zu Fremdschlüsseln (1)

Relation Auftrag

Auftrnr	Datum	Kundnr	Persnr
1	04.01.2013	1	2
2	06.01.2013	3	5
3	07.01.2013	4	2
4	18.01.2013	6	5
5	03.02.2013	1	2

Derzeit nur Werte zwischen 1 und 9 erlaubt

Relation Personal (Auszug)

Persnr	Name	Ort	Vorgesetzt	Gehalt
1	Maria Forster	Regensburg	NULL	4800.00
2	Anna Kraus	Regensburg	1	2300.00
3	Ursula Rank	Frankfurt	6	2700.00
4	Heinz Rolle	Nürnberg	1	3300.00
5	Johanna Köster	Nürnberg	1	2100.00
6	Marianne Lambert	Landshut	NULL	4100.00
7	Thomas Noster	Regensburg	6	2500.00
8	Renate Wolters	Augsburg	1	3300.00
9	Ernst Pach	Stuttgart	6	800.00

Beispiel zu Fremdschlüsseln (2)

Auftrnr	Datum	Kundnr	Persnr
1	04.01.2013	1	2
2	06.01.2013	3	5
3	07.01.2013	4	2
4	18.01.2013	6	5
5	03.02.2013	1	2

Relation Auftrag

Derzeit nur Werte zwischen 1 und 6 erlaubt

Relation Kunde

Nr	Name	Strasse	PLZ	Ort
1	Fahrrad Shop	Obere Regenstr. 4	93059	Regensburg
2	Zweirad-Center Staller	Kirschweg 20	44276	Dortmund
3	Maier Ingrid	Universitätsstr. 33	93055	Regensburg
4	Rafa - Seger KG	Liebigstr. 10	10247	Berlin
5	Biker Ecke	Lessingstr. 37	22087	Hamburg
6	Fahrräder Hammerl	Schindlerplatz 7	81739	München

Referenz-Integritätsregel

- **Zweite Integritätsregel**
  - Eine relationale Datenbank enthält keinen Fremdschlüsselwert (ungleich *Null*), der im dazugehörigen Primärschlüssel nicht existiert.

- **Nichts Neues: Teil der Definition des Fremdschlüssels!**
- **Diese Regel ist extrem wichtig!**
- **Diese Regel muss immer eingehalten werden!**

Beispiele zur 2. Integritätsregel

- **1. Fall**
  - Ein Auftrag wird vergeben. Eine Kundennr. 13 wäre nicht erlaubt!
  - Hier werden selten Fehler gemacht, außer aus Versehen!
- **2. Fall**
  - Frau Köster mit Persnr 5 scheidet aus der Firma aus.
  - Wir löschen das Tupel mit Persnr 5
  - Jetzt enthält der Fremdschlüssel *Persnr* in Relation *Auftrag* einen nicht erlaubten Wert!
  - Aber: Wie verhindern wir einen solchen Fall?

Sicherstellen der 2. Integritätsregel

- Der Mensch macht Fehler!
- → Die Datenbank muss die 2. Integritätsregel garantieren

- Szenario:
  - Frau Köster scheidet aus, das Tupel soll gelöscht werden
  - Die Datenbank stellt fest, dass ein Fremdschlüssel dazu existiert
- Die Datenbank muss reagieren (aber wie?):
  - Sie verhindert das Löschen des Tupel mit Fehlermeldung
  - Oder: Sie entfernt auch die Einträge im Fremdschlüssel (NULL!)
  - Oder: Sie löscht auch die Tupel, die auf Frau Köster verweisen

SQL und die 2. Integritätsregel (1)

- **Fremdschlüsselbedingungen in SQL**
  - ON DELETE **NO ACTION**
  - ON DELETE **SET NULL**
  - ON DELETE **CASCADE**
- **Funktionsweise**
  - Wird ein Tupel gelöscht, auf den ein Fremdschlüssel mit obiger Bedingung verweist, dann
    - wird das Löschen dieses Tupel verhindert (Nichtstun, No Action)
    - wird der darauf verweisende Fremdschlüssel auf Null gesetzt (Set Null)
    - wird auch das den Fremdschlüssel enthaltene Tupel gelöscht (Cascade)

SQL und die 2. Integritätsregel (2)

- **Analog wird das Ändern eines Primärschlüssel behandelt:**
  - ON UPDATE **NO ACTION**
  - ON UPDATE **SET NULL**
  - ON UPDATE **CASCADE**
- **Funktionsweise**
  - Wird ein Primärschlüsselwert geändert, auf den ein Fremdschlüssel mit obiger Bedingung verweist, dann
    - wird das Ändern dieses Tupel verhindert (Nichtstun, No Action)
    - wird der darauf verweisende Fremdschlüssel auf Null gesetzt (Set Null)
    - wird der Fremdschlüsselwert mit geändert (Cascade)

Kaskadierendes Löschen

- **Auf ein Tupel mit einem Fremdschlüssel kann wiederum ein Fremdschlüssel verweisen, usw.**
- **Im Falle von ON DELETE CASCADE gilt:**
  - Ein Tupel soll gelöscht werden
  - Ein Fremdschlüssel verweist auf dieses Tupel, das Tupel mit diesem Fremdschlüssel wird mit gelöscht
  - Auf letztes Tupel verweist ein weiterer Fremdschlüssel, der entsprechende Eintrag wird dann auch gelöscht usw.
- Wir nennen dies: **Kaskadierendes Löschen**

Beispiel zum kaskadierenden Löschen

Auftrnr	Datum	Kundnr	Persnr
1	04.01.2013	1	2
2	06.01.2013	3	5
3	07.01.2013	4	2
4	18.01.2013	6	5
5	03.02.2013	1	2

- Für alle Fremdschlüssel gelte ON DELETE CASCADE
- Frau Forster soll gelöscht werden

Welche Tupel werden noch gelöscht?

Persnr	Name	Ort	Vorgesetzt	Gehalt
1	Maria Forster	Regensburg	NULL	4800.00
2	Anna Kraus	Regensburg	1	2300.00
3	Ursula Rank	Frankfurt	6	2700.00
4	Heinz Rolle	Nürnberg	1	3300.00
5	Johanna Köster	Nürnberg	1	2100.00
6	Marianne Lambert	Landshut	NULL	4100.00
7	Thomas Noster	Regensburg	6	2500.00
8	Renate Wolters	Augsburg	1	3300.00
9	Ernst Pach	Stuttgart	6	800.00

Kaskadierendes Löschen u. Transaktion

- **Merke:**
  - Ein kaskadierendes Löschen wird als **Transaktion** ausgeführt
    - Entweder wird das komplette kaskadierende Löschen ausgeführt
    - Oder es wird nichts gelöscht
- **Folgerung:**
  - Gibt es in der Kette ein CASCADE, so wird entsprechend weiter gelöscht
  - Gibt es in der Kette ein SET NULL, so endet dieses Glied
  - Gibt es in der Kette ein NO ACTION, so wird das komplette kaskadierende Löschen **rückgängig** gemacht (Transaktion)!

Fremdschlüssel und NULL-Werte

- **NULL-Werte können in einem Attribut explizit verboten werden. Wir setzen dazu im CREATE-TABLE-Befehl:**
  - NOT NULL
- **Es gilt:**
  - Ist ein Fremdschlüssel auch Primärschlüssel oder ist explizit NOT NULL gesetzt,
    - so sind NULL-Werte nicht erlaubt
    - so darf ON DELETE SET NULL nicht verwendet werden
    - so darf ON UPDATE SET NULL nicht verwendet werden

Begriffe zur Relationalen Algebra

- Menge
  - Behälter, der unterscheidbare Elemente enthält
- Operator
  - Vorschrift zur Überführung eines oder mehrerer Elemente in ein anderes Element
- Unärer Operator
  - Vorschrift zur Überführung eines Elements
- Binärer Operator
  - Vorschrift zur Überführung von zwei Elementen
- Relationale Algebra
  - Abfragesprache auf relationale Datenbanken, in der geeignete Operatoren definiert sind

$\mathfrak{R}$ = Menge aller Relationen

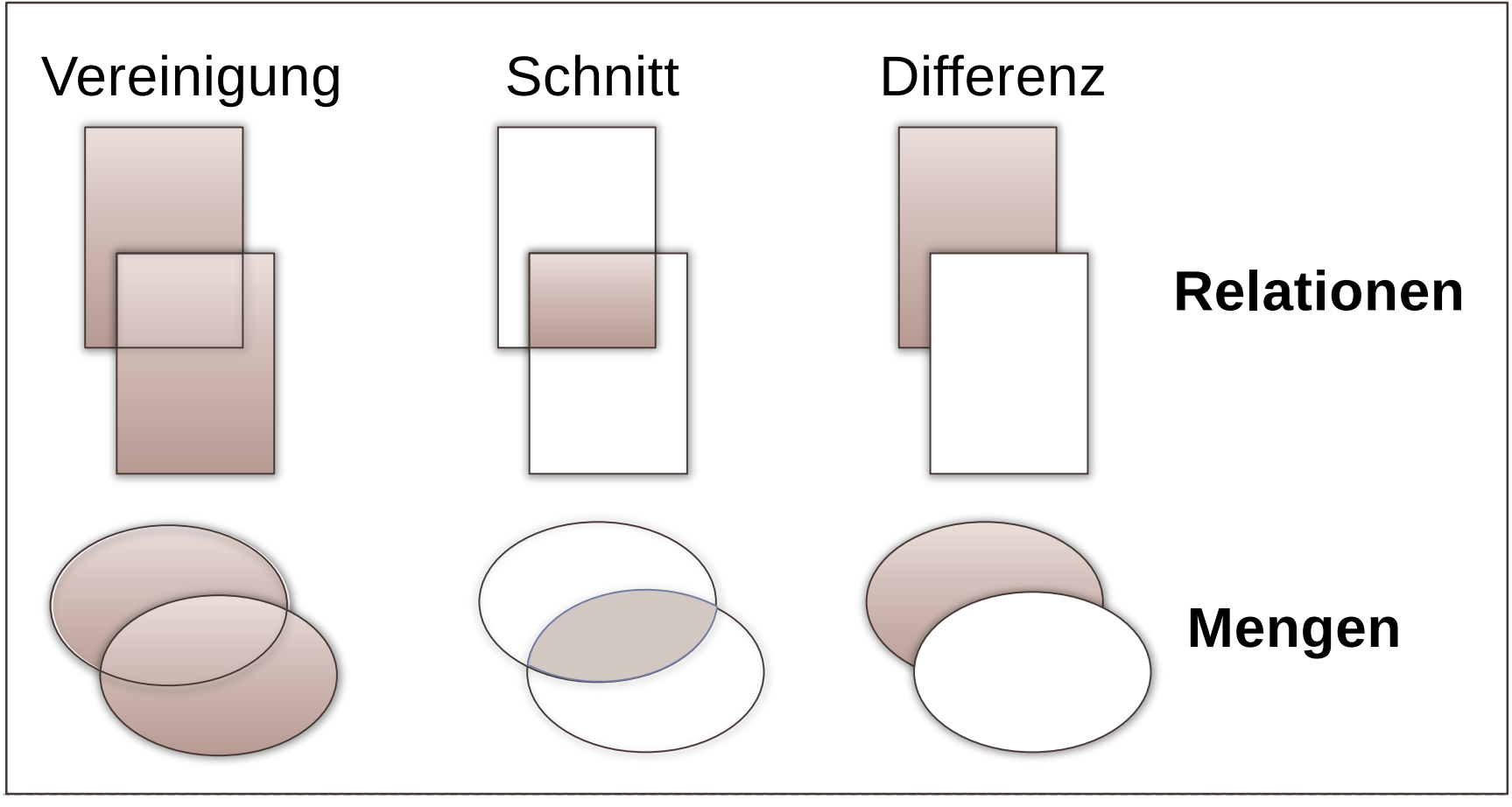
$op : \mathfrak{R} \rightarrow \mathfrak{R}$

$op : \mathfrak{R} \times \mathfrak{R} \rightarrow \mathfrak{R}$

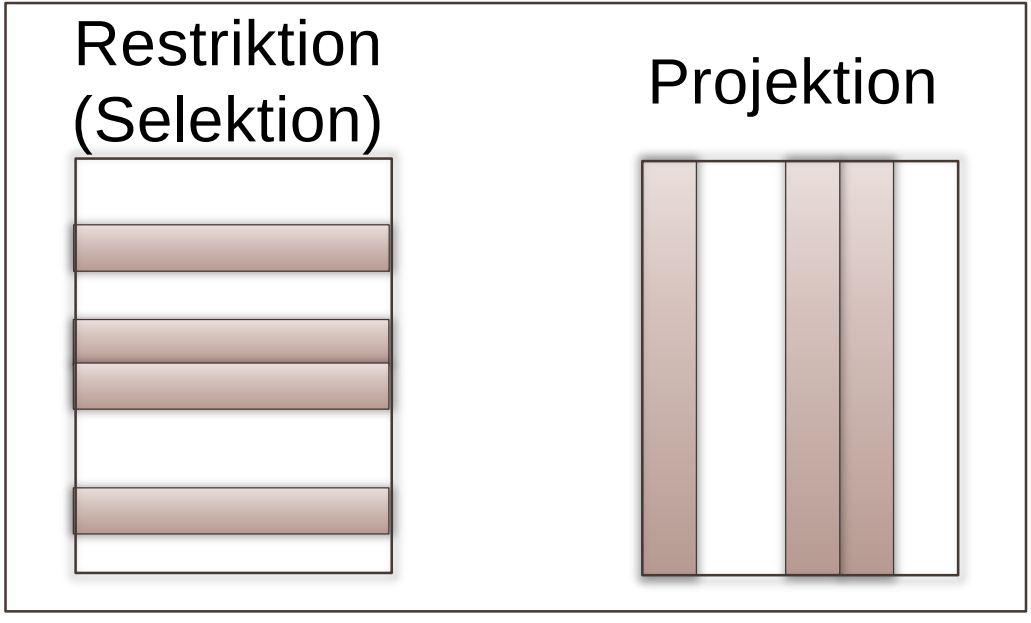
Alle neun relationalen Operatoren

	Operator	Beispiel
Vereinigung	$\cup$	$R1 \cup R2$
Schnitt	$\cap$	$R1 \cap R2$
Differenz	$\setminus$	$R1 \setminus R2$
Kreuzprodukt	$\times$	$R1 \times R2$
Restriktion	σ	$\sigma_{Bedingung}(R)$
Projektion	π	$\pi_{Auswahl}(R)$
Verbund	$\bowtie$	$R1 \bowtie R2$
Division	$\div$	$R1 \div R2$
Umbenennung	ρ	$\rho_{Rneu}(R)$

Vereinigung, Schnitt, Differenz

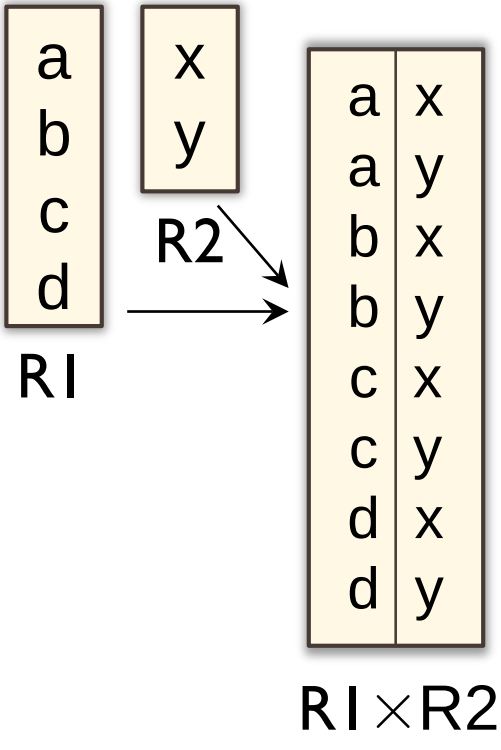


Projektion, Restriktion



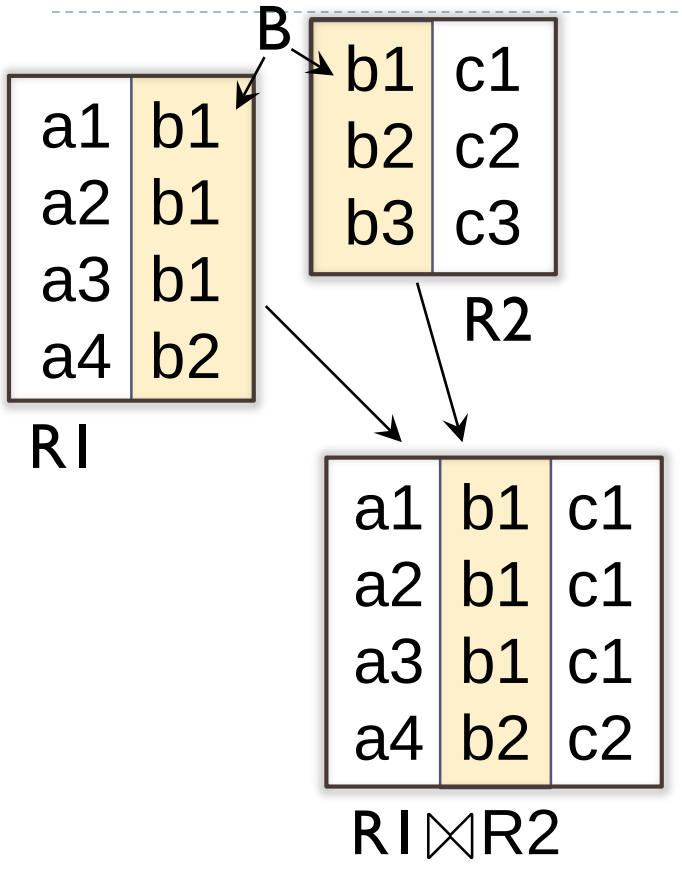
- Projektion:
  - Einschränkung auf weniger Attribute (Spalten)
- Restriktion:
  - Einschränkung auf weniger Tupel (Zeilen)

Kreuzprodukt



- **Jede Zeile der einen Tabelle wird mit jeder Zeile der anderen Tabelle verknüpft**
  - Sei n=Kardinalität(R1),
    sei m=Kardinalität(R2),
  - dann: **Kardinalität(R1×R2) = n · m**

(Natürlicher) Verbund



- Voraussetzung:
  - R1 und R2 besitzen Attribut mit gleichem Namen (hier: Spalte B)
- Verbund verbindet alle Tupel, die im Attribut B gleiche Einträge haben
- Verbund führt Attribut B nur einmal auf

Begriffe zum Verbund

- **Natürlicher Verbund (Natural Join):** $\bowtie$
  - Alle Attribute gleichen Namens dienen als Verbindung, Überprüfung auf Gleichheit
- **Equi-Join:** $\bowtie_{R1.B=R2.B}$
  - Die angegebenen Attribute R1.B und R2.B dienen als Verbindung, Überprüfung auf Gleichheit
- **Theta-Join:** $\bowtie_{R1.B\ op\ R2.B}$
  - Die angegebenen Attribute R1.B und R2.B dienen als Verbindung, Überprüfung mittels Operator op (z.B. <, <=)
- **Outer Join:** $⟕$, $⟖$, $⟗$
  - Erweiterung: Auch nicht betroffene Tupel werden aufgelistet

Verbund: Ein Beispiel

Auftrnr	Datum	Kundnr	Persnr
1	04.01.2013	1	2
2	06.01.2013	3	5
3	07.01.2013	4	2
4	18.01.2013	6	5
5	03.02.2013	1	2

- Verbund über Persnr
- Nur Persnr 2 und 5 bleiben übrig

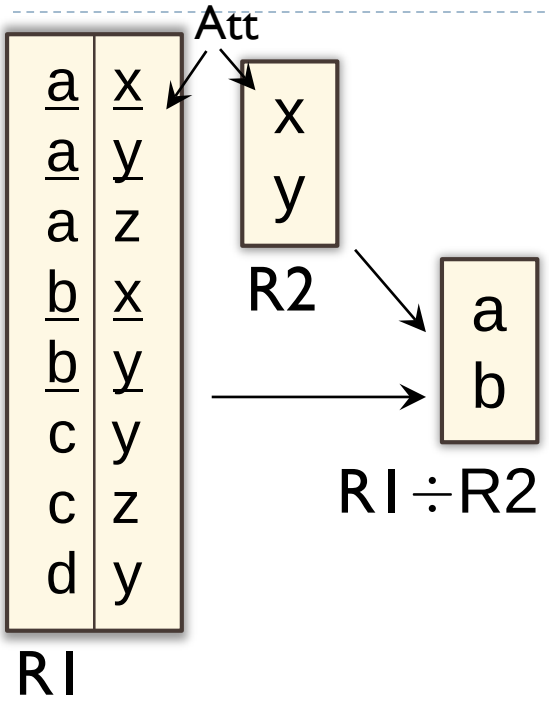
Persnr	Name	Ort	Vorgesetzt	Gehalt
1	Maria Forster	Regensburg	NULL	4800.00
2	Anna Kraus	Regensburg	1	2300.00
3	Ursula Rank	Frankfurt	6	2700.00
4	Heinz Rolle	Nürnberg	1	3300.00
5	Johanna Köster	Nürnberg	1	2100.00
6	Marianne Lambert	Landshut	NULL	4100.00
7	Thomas Noster	Regensburg	6	2500.00
8	Renate Wolters	Augsburg	1	3300.00
9	Ernst Pach	Stuttgart	6	800.00

Verbund: Das Ergebnis

AuftrNr	Datum	Kundnr	Persnr	Name	Vorgesetzt	Gehalt	Ort
1	04.01.13	1	2	Anna Kraus	1	3400.00	Regensburg
2	06.01.13	3	5	Joh. Köster	1	3200.00	Nürnberg
3	07.01.13	4	2	Anna Kraus	1	3400.00	Regensburg
4	18.01.13	6	5	Joh. Köster	1	3200.00	Nürnberg
5	06.02.13	1	2	Anna Kraus	1	3400.00	Regensburg

- Alle Attribute von Auftrag
- Alle Attribute von Personal
- Aber: Verknüpfendes Attribut Persnr nur einmal

Division



- Voraussetzung:
  - R1 enthält alle Attribute Att von R2
- Die Division liefert die restlichen Attribute von R1 (ohne Att)
- Die Division enthält alle Werte, die in R1 mit allen Attributen aus R2 verknüpft sind
  (im Beispiel unterstrichen)

Division: Beispiel (1)

Lieferung:

ANr	Liefnr	Lieferzeit	Nettopreis	Bestellt
500001	5	1	6.50	0
500002	2	4	71.30	10
500002	1	5	73.10	0
500003	3	6	5.60	0
500003	4	5	6.00	0
500003	2	4	5.70	0
500004	3	2	5.20	0
500004	4	3	5.40	0
500005	4	5	6.70	0
500006	1	1	31.00	0
500007	1	2	16.50	0

Artikelnr

Lieferantennr

- Lieferant 3 liefert Artikel 500003 und 500004.
- Gibt es weitere, die mindestens diese beiden Artikel liefern?

$R1 = \pi_{ANr,Liefnr}(Lieferung)$

$R2 = \sigma_{Liefnr=3}(R1)$

ANr	Liefnr
500003	3
500004	3

$R3 = \pi_{ANr}(R2)$

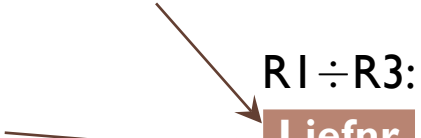
Division: Beispiel (2)

R1:

Liefnr	ANr
5	500001
2	500002
1	500002
3	500003
4	500003
3	500004
4	500004
4	500005
1	500006
1	500007

R3:

ANr
500003
500004



R1÷R3:

Liefnr
3
4

- Die Division liefert:
  - Lieferanten 3 und 4 liefern Artikel 500003 und 500004
  - Lieferant 3 war klar
- Ergebnis:
  - Lieferant 4 liefert alle Artikel, die auch Lieferant 3 liefert

Umbenennung

- Umbenennung ist Hilfsoperator, um Algebra zu vervollständigen
- Beispiel:
  - $R = \rho_{Nr->Kundnr}(Kunde) \bowtie Auftrag$
- Alternative (hier: Equi-Join):
  - $R = Kunde \bowtie_{Kunde.Nr=Auftrag.Kundnr} Auftrag$

Kommutativ- und Assoziativgesetze

- $A \cup B = B \cup A$
- $A \cap B = B \cap A$
- $A \times B = B \times A$
- $A \bowtie B = B \bowtie A$

- $A \cup (B \cup C) = (A \cup B) \cup C = A \cup B \cup C$
- $A \cap (B \cap C) = (A \cap B) \cap C = A \cap B \cap C$
- $A \times (B \times C) = (A \times B) \times C = A \times B \times C$
- $A \bowtie (B \bowtie C) = (A \bowtie B) \bowtie C = A \bowtie B \bowtie C$ Vorsicht!

Weitere Regeln (1)

- $\sigma_{BedingungA}(\sigma_{BedingungB}(R)) = \sigma_{BedingungB}(\sigma_{BedingungA}(R))$
  $= \sigma_{BedingungA\ AND\ BedingungB}(R)$
  $= \sigma_{BedingungA}(R) \cap \sigma_{BedingungB}(R)$

  Vertauschbarkeit von Bedingungen

- $\sigma_{BedingungA\ OR\ BedingungB}(R) = \sigma_{BedingungA}(R) \cup \sigma_{BedingungB}(R)$

  Bedingungen direkt verknüpfen

- $\sigma_{Bedingung}(R1 \cup R2) = \sigma_{Bedingung}(R1) \cup \sigma_{Bedingung}(R2)$

  Bedingungen auf einzelne Relationen einschränken

Weitere Regeln (2)

- $\sigma_{Bedingung}(R1 \bowtie R2) = \sigma_{Bedingung}(R1) \bowtie \sigma_{Bedingung}(R2)$

Bedingung direkt auf Relation anwenden.
Welche Seite ist performanter?

Spezialfall

- $\sigma_{Bedingung\_an\_R2}(R1 \bowtie R2) = R1 \bowtie \sigma_{Bedingung\_an\_R2}(R2)$

- $\pi_{Auswahl}(\sigma_{Bedingung}(R)) = \sigma_{Bedingung}(\pi_{Auswahl}(R))$

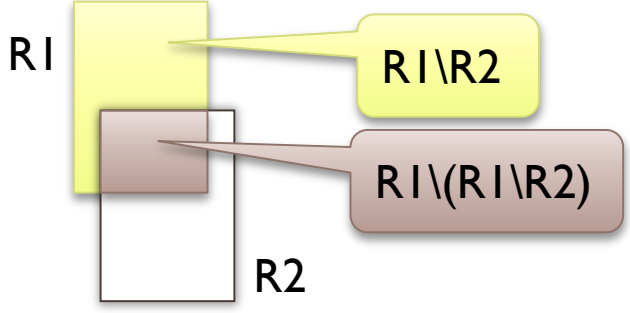
Der Name Division
ist berechtigt

Vertauschbarkeit von
Projektion und Restriktion

- $R1 = (R1 \times R2) \div R2$

Drei Operatoren sind „überflüssig“

- 3 Operatoren lassen sich aus den anderen 6 Operatoren ableiten!
- Dies sind
  - Schnitt: $R1 \cap R2 = R1 \setminus (R1 \setminus R2)$



  - Verbund: $R1 \bowtie R2 = \pi_{R1.X,R1.Y,R2.Z}(\sigma_{R1.Y=R2.Y}(R1 \times R2))$

gefolgt von Projektion (verbindendes Attribut nur einmal)

gefolgt von Restriktion

Kreuzprodukt

  - Division: $R1 \div R2 = \pi_{R1.X}(R1) \setminus (\pi_{R1.X}((\pi_{R1.X}(R1) \times R2) \setminus R1))$

Datenbanken und SQL

Kapitel 3

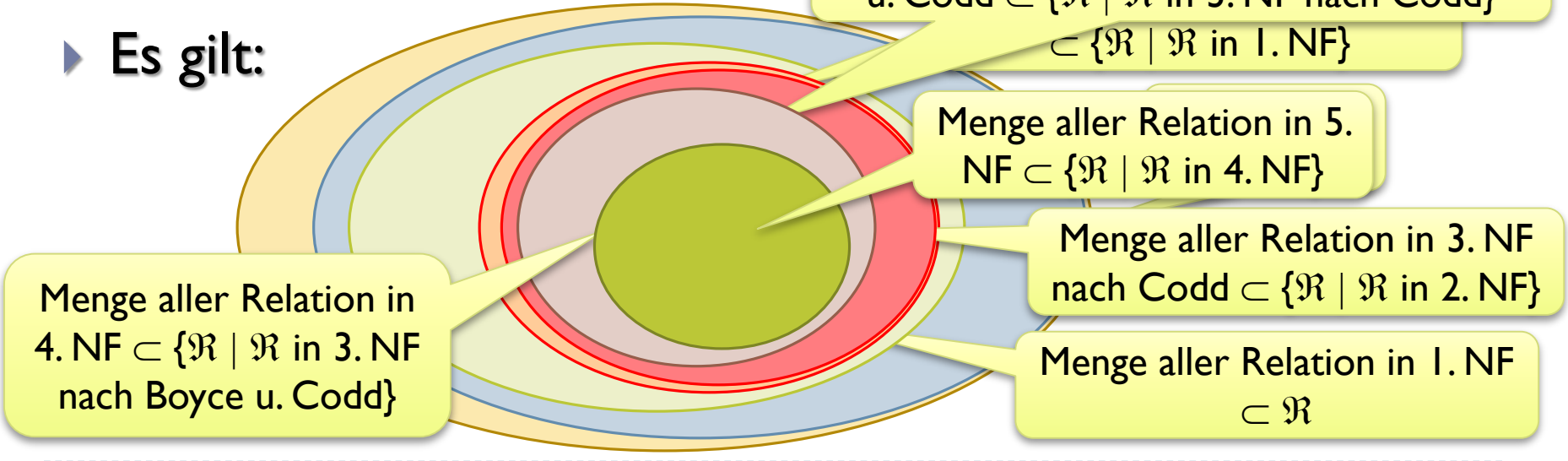
Datenbankdesign – Teil 1: Normalformen

Datenbankdesign

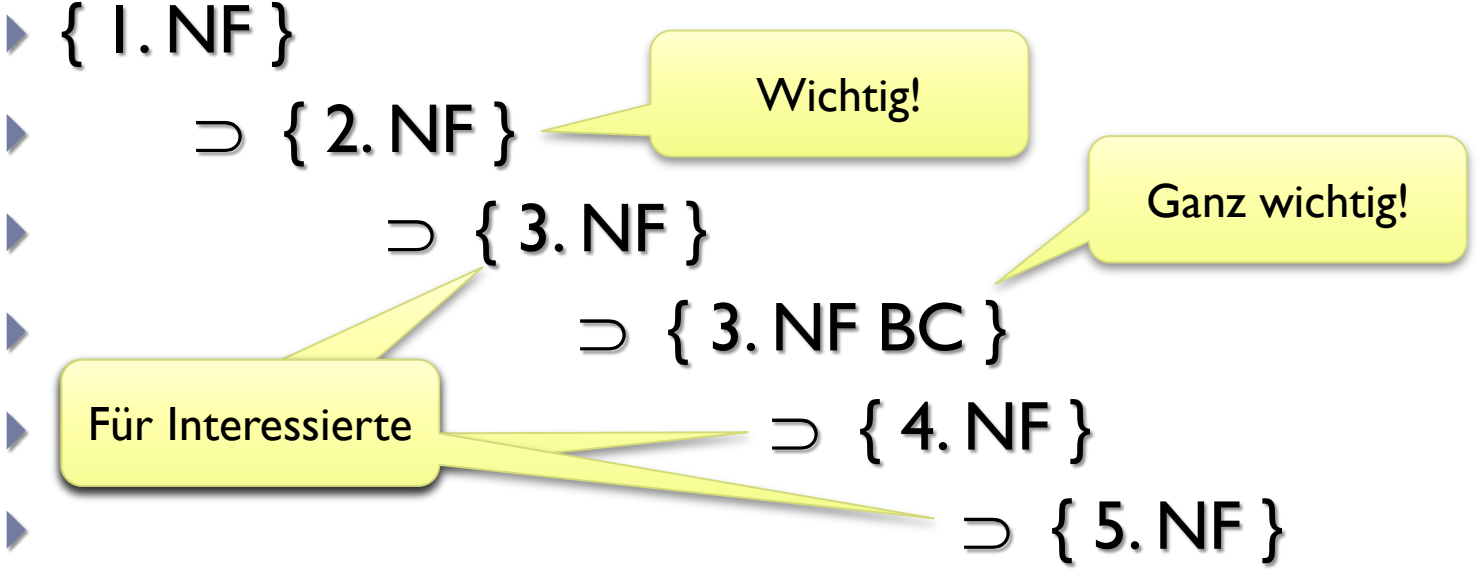
- Normalformen
  - *1. Normalform*
  - *Funktionale Abhängigkeit*
  - *2. Normalform*
  - *3. Normalform nach Boyce und Codd*
  - *3. Normalform nach Codd*
  - *Mehrwertige Abhängigkeit und 4. Normalform*
  - *Verbundabhängigkeit und 5. Normalform*

Die Normalformen im Überblick

- Es gibt 6 Definitionen von Normalformen
  - Die 1. Normalform (NF) schränkt am wenigsten ein
  - Es gibt 2 Definitionen für die 3. NF
  - Die 5. NF schränkt am stärksten
- Es gilt:



Für die Normalformen gilt:



- 3. NF = 3. NF nach Codd
  3. NF BC = 3. NF nach Boyce und Codd

Definition (1. Normalform)

- **Eine Relation ist in erster Normalform (1. NF), wenn alle zugrundeliegende Gebiete nur atomare Werte enthalten.**

- **Folgerung:**
  - Jede (normalisierte) Relation ist in 1. NF
- **Die 1. NF ist historisch bedingt:**
  - In der Originaldefinition von Relationen war die Atomarität nicht gefordert
- **NF^2 = NFNF = NonFirstNormalForm**
  - Relationen, die nicht in 1. NF sind
  - NF^2 ist Basis für objektrelationale Datenbanken

Beispiel: Relation VerkaeuferProdukt

VerkNr	VerkName	PLZ	VerkAdresse	Produktname	Umsatz
V1	Meier	80075	München	Waschmaschine	11000
V1	Meier	80075	München	Herd	5000
V1	Meier	80075	München	Kühlschrank	1000
V2	Schneider	70038	Stuttgart	Herd	4000
V2	Schneider	70038	Stuttgart	Kühlschrank	3000
V3	Müller	50083	Köln	Staubsauger	1000

- ✓ VerkaeuferProdukt enthält nur atomare Werte
- ✓ VerkaeuferProdukt ist eine Relation
- ✓ VerkaeuferProdukt ist in 1. NF

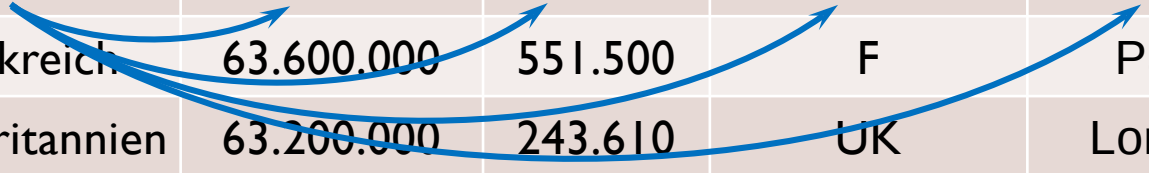
VerkaeuferProdukt (Wiederholung)

- **Redundanz**
  - Je mehr ein Verkäufer verkauft, um so häufiger in Tabelle!
  - Ändert sich die Adresse eines Verkäufers, muss dies in **allen** entsprechenden Einträgen erfolgen. Sonst: **Inkonsistenz**!
- **Handhabung**
  - Soll Produkt Staubsauger aus dem Sortiment genommen werden, so ist auch Verkäufer Müller zu löschen!?
  - Verkäufer Schmidt kann erst eingetragen werden, wenn er etwas verkauft hat!?

Warum verwenden wir Tabellen?

- Beispiel: Tabelle der Länder (Stand 2012)

Land	Einwohner	Fläche	Kennzeichen	Hauptstadt
Deutschland	81.800.000	357.121	D	Berlin
Frankreich	63.600.000	551.500	F	Paris
Großbritannien	63.200.000	243.610	UK	London
Italien	60.900.000	301.340	I	Rom
Niederlande	16.700.000	61.543	NL	Amsterdam
Polen	38.200.000	312.685	PL	Warschau
Spanien	46.200.000	505.370	E	Madrid



- Alle Einträge hängen eindeutig von der Spalte Land ab:
  - Aus dem Land folgen eindeutig alle anderen Spalten!

Definition (Funktionale Abhängigkeit)

- Ein Attribut Y einer Relation R heißt funktional abhängig vom Attribut X derselben Relation, wenn zu jedem X-Wert höchstens ein Y-Wert möglich ist.

- Informell: „Aus X folgt eindeutig Y“
- Wir schreiben: $X \rightarrow Y$

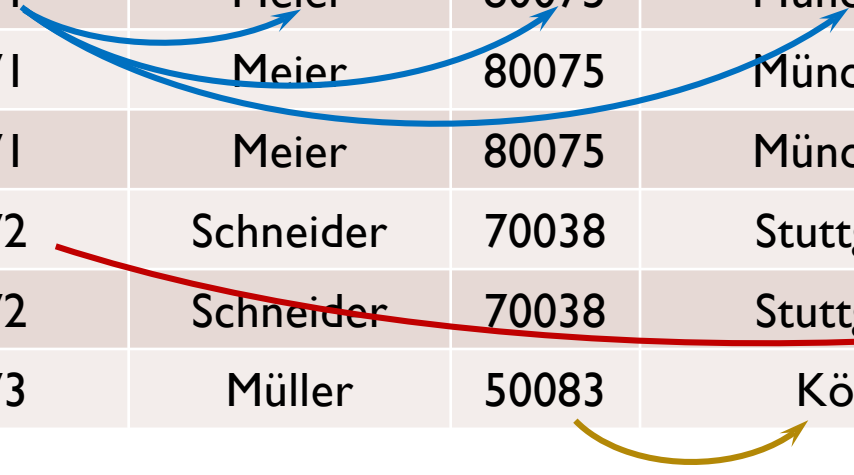
- Land → Einwohner
- Land → Kennzeichen
- Land → Fläche
- Land → Hauptstadt

Folgerungen zur funktionalen Abh.

- Primärschlüssel $\rightarrow$ alle anderen Attribute
  - (da Primärschlüssel eindeutig jedes Tupel identifizieren)
- Schlüsselkandidat $\rightarrow$ alle anderen Attribute
  - (da Schlüsselkandidaten eindeutig jedes Tupel identifizieren)
- Superschlüssel $\rightarrow$ alle anderen Attribute
  - (da Superschlüssel eindeutig jedes Tupel identifizieren)

VerkaeuferProdukt: Funktionale Abh.

VerkNr	VerkName	PLZ	VerkAdresse	Produktname	Umsatz
V1	Meier	80075	München	Waschmaschine	11000
V1	Meier	80075	München	Herd	5000
V1	Meier	80075	München	Kühlschrank	1000
V2	Schneider	70038	Stuttgart	Herd	4000
V2	Schneider	70038	Stuttgart	Kühlschrank	3000
V3	Müller	50083	Köln	Staubsauger	1000



- ✓ VerkNr → VerkName
- ✓ VerkNr → PLZ
- ✓ VerkNr → VerkAdresse

- ✓ (VerkNr, Produktname) → Umsatz
- ✓ PLZ → VerkAdresse ?

Problem: Abhängigkeit PLZ und Adresse

- **Folgt aus PLZ der Ort?**
  - Definition Ort: Alle selbstständigen Gemeinden Deutschlands
    - Antwort: NEIN, da einige kleine Gemeinden gleiche PLZ
  - Definition Ort: Alle Gemeinden mit mehr als 20000 Einwohner
    - Antwort: JA, da alle großen Gemeinden unterschiedliche PLZ
- **Folgt aus Adresse (Ort+Straße+Nr) die PLZ?**
  - Antwort: NEIN
  - Beispiel: Es gibt mehrere Neustadt mit Bahnhofstraße 1
  - Dies war mit ein Grund zur Einführung der PLZ!
- **Folgt aus der PLZ das Bundesland?**
  - Antwort: JA, da bei der Einführung der PLZ darauf Rücksicht genommen wurde

Problem: Fehlende Minimalität

- **Schlüsselkandidat → alle anderen Attribute**
  - Schlüsselkandidaten sind minimal
  - Jeder Wert kommt nur einmal vor
- **Superschlüssel → alle anderen Attribute**
  - Superschlüssel sind nicht notwendigerweise minimal! Problem!

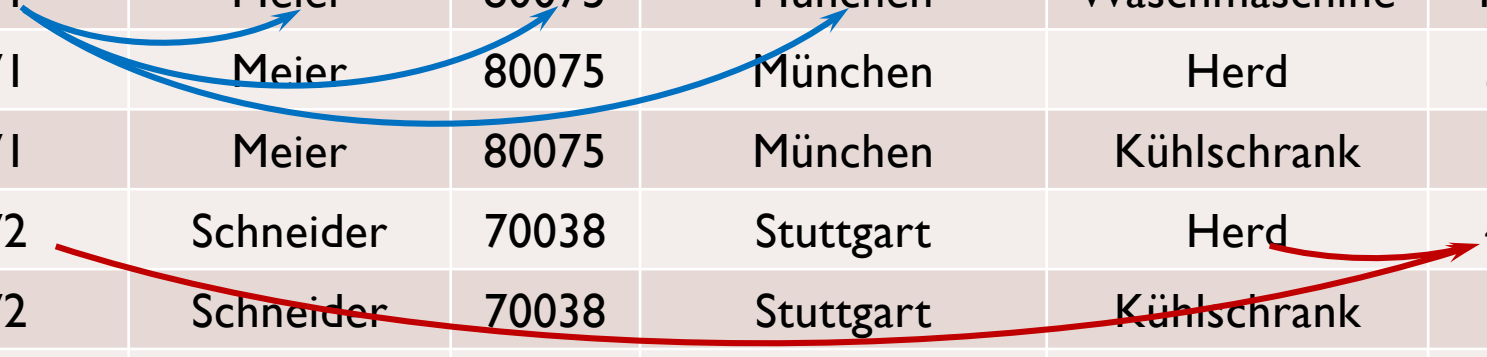
- **Beispiel:**
  - (Verknr, Produktname) → Umsatz — Primärschlüssel!
  - (Verknr, VerkName, Produktname) → Umsatz — Superschlüssel
  - (Verknr, PLZ, Produktname) → Umsatz — Superschlüssel, usw.
- **Folgerung:**
  - Invasion weiterer wertloser funktionaler Abhängigkeiten

Definition (Volle funktionale Abh.)

- **Ein Attribut Y einer Relation R heißt voll funktional abhängig vom Attribut X derselben Relation, wenn**
  - es funktional abhängig ist von X
  - es nicht funktional abhängig ist von beliebigen Teilattributen von X
- **Wir schreiben: $X \Rightarrow Y$**
- **Folgerung:**
  - Es gilt immer: Primärschlüssel $\rightarrow$ alle anderen Attribute
  - Es gilt <u>nicht</u> immer: Primärschlüssel $\Rightarrow$ alle anderen Attribute

VerkaeuferProdukt: Volle funkt. Abh.

VerkNr	VerkName	PLZ	VerkAdresse	Produktname	Umsatz
V1	Meier	80075	München	Waschmaschine	11000
V1	Meier	80075	München	Herd	5000
V1	Meier	80075	München	Kühlschrank	1000
V2	Schneider	70038	Stuttgart	Herd	4000
V2	Schneider	70038	Stuttgart	Kühlschrank	3000
V3	Müller	50083	Köln	Staubsauger	1000



- ✓ VerkNr ⇒ VerkName
- ✓ VerkNr ⇒ PLZ
- ✓ VerkNr ⇒ VerkAdresse
- ✓ (VerkNr, Produktname) ⇒ Umsatz

Es gibt keine weiteren vollen funktionalen Abhängigkeiten!

Na ja: Eventuell PLZ ⇒ VerkAdresse

Definition (Zweite Normalform)

- **Eine Relation ist in der zweiten Normalform (2. NF), wenn sie in der ersten Normalform ist, und jedes Nichtschlüsselattribut voll funktional vom Primärschlüssel abhängt.**

- **Bemerkungen:**
  - Die 2. NF bezieht sich nur auf Primärschlüssel, nicht auf alternative Schlüssel
  - Die Relation VerkaeuferProdukt ist nicht in der 2. NF

Wichtige Folgerungen zur 2. NF

- 1. NF:
  - Primärschlüssel → alle Nichtschlüsselattribute
- 2. NF:
  - Primärschlüssel ⇒ alle Nichtschlüsselattribute
- Primärschlüssel ist ein einzelnes Attribut:
  - Dann folgt: Relation ist in mindestens 2. NF
- Jede Relation in 1. NF lässt sich in die 2. NF überführen:
  - Hinzufügen eines Zählers als Primärschlüssel (einzelnes Attr.)

VerkaeuferProdukt2NF

Nr	VerkNr	VerkName	PLZ	VerkAdresse	Produktname	Umsatz
1	V1	Meier	80075	München	Waschmaschine	11000
2	V1	Meier	80075	München	Herd	5000
3	V1	Meier	80075	München	Kühlschrank	1000
4	V2	Schneider	70038	Stuttgart	Herd	4000
5	V2	Schneider	70038	Stuttgart	Kühlschrank	3000
6	V3	Müller	50083	Köln	NULL	1000

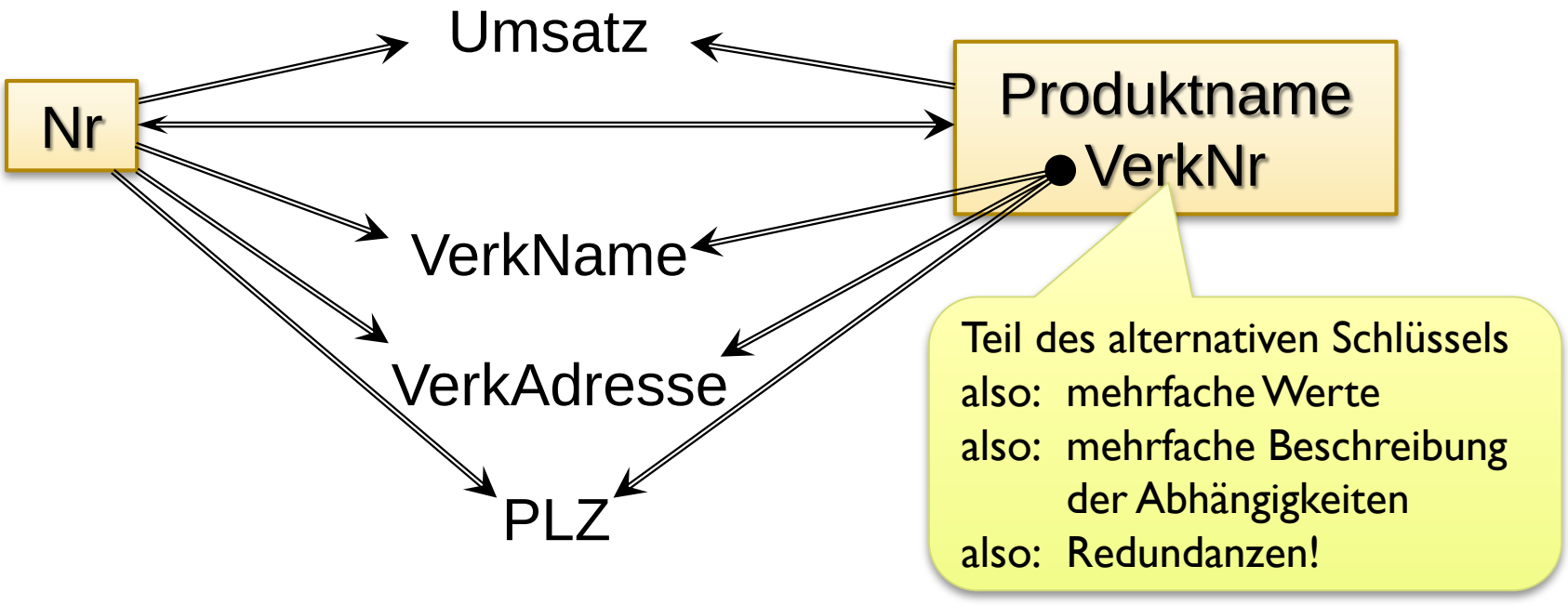
Neues Attribut

jetzt erlaubt

- Primärschlüssel: Nr also: 2. NF
- Noch mehr Redundanzen (Attribut Nr)
- Aber: Weniger Anomalien (Entfernen von Staubsauger!)

VerkaeuferProdukt2NF

- Volle funktionale Abhängigkeiten:



Definition (Determinante)

- **Eine Determinante ist ein (eventuell zusammengesetztes) Attribut, von dem ein anderes voll funktional abhängt.**

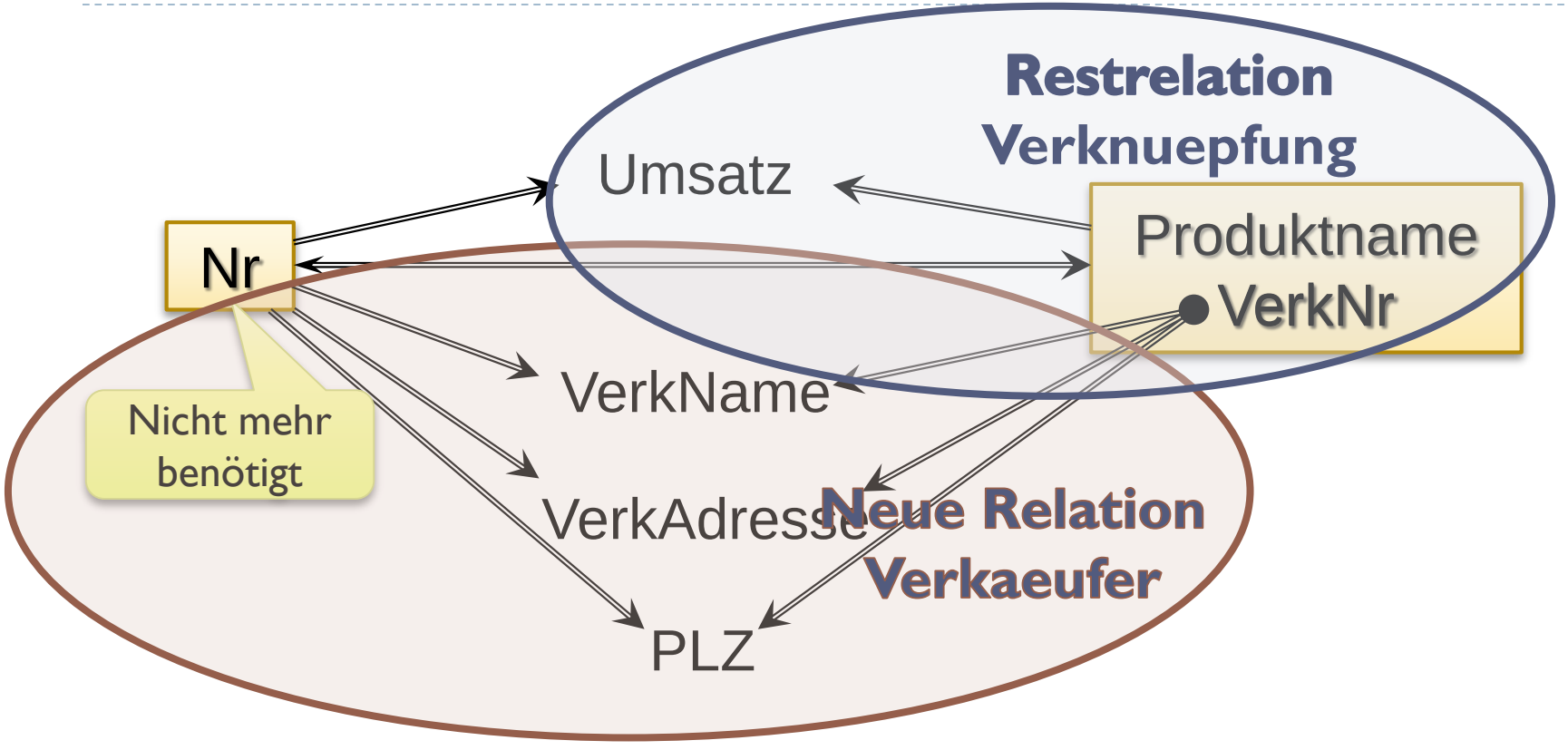
- **Bemerkungen:**
  - Ein wertvolles Hilfsmittel
  - Alle Attribute, von denen Doppelpfeile ausgehen, sind Determinanten
  - Determinanten in VerkaeuferProdukt2NF:
    - Nr, Verknr, (Verknr, Produktname)

Dritte Normalform nach Boyce u. Codd

- **Eine normalisierte Relation ist in der dritten Normalform, wenn jede Determinante dieser Relation ein Schlüssel-kandidat ist.**

- **Bemerkungen:**
  - Alle Abhängigkeiten von nicht eindeutigen Werten (Schlüssel-kandidaten) werden verboten!
  - Damit ist eine Relation in 3. NF redundanzfrei (außerhalb der Schlüsselkandidaten)
  - VerkaeuferProdukt2NF ist nicht in 3. NF
    - denn: VerkNr ist kein Schlüsselkandidat
  - Jede Relation in 2. NF lässt sich in Relationen der 3. NF überführen

Überführung in die 3. NF



Verkaeufer und Produkte (Schritt 1)

Restrelation **Verknuepfung**:

VerkNr	Produktname	Umsatz
V1	Waschmaschine	11000
V1	Herd	5000
V1	Kühlschrank	1000
V2	Herd	4000
V2	Kühlschrank	3000
V3	Staubsauger	1000

Neue Relation **Verkaeufer**:

VerkNr	VerkName	PLZ	VerkAdresse
V1	Meier	80075	München
V2	Schneider	70038	Stuttgart
V3	Müller	50083	Köln

Alles ist in Ordnung, da 3. NF; aber:

Wir haben den Verkäufer heraus-genommen, warum nicht auch das Produkt?

Verkaeufer und Produkte (Schritt 2)

Relation **Verkaeufer**:

VerkNr	VerkName	PLZ	VerkAdresse
V1	Meier	80075	München
V2	Schneider	70038	Stuttgart
V3	Müller	50083	Köln

Relation **Verknuepfung**:

VerkNr	ProdNr	Umsatz
V1	P1	11000
V1	P2	5000
V1	P3	1000
V2	P2	4000
V2	P3	3000
V3	P4	1000

Relation **Produkt**:

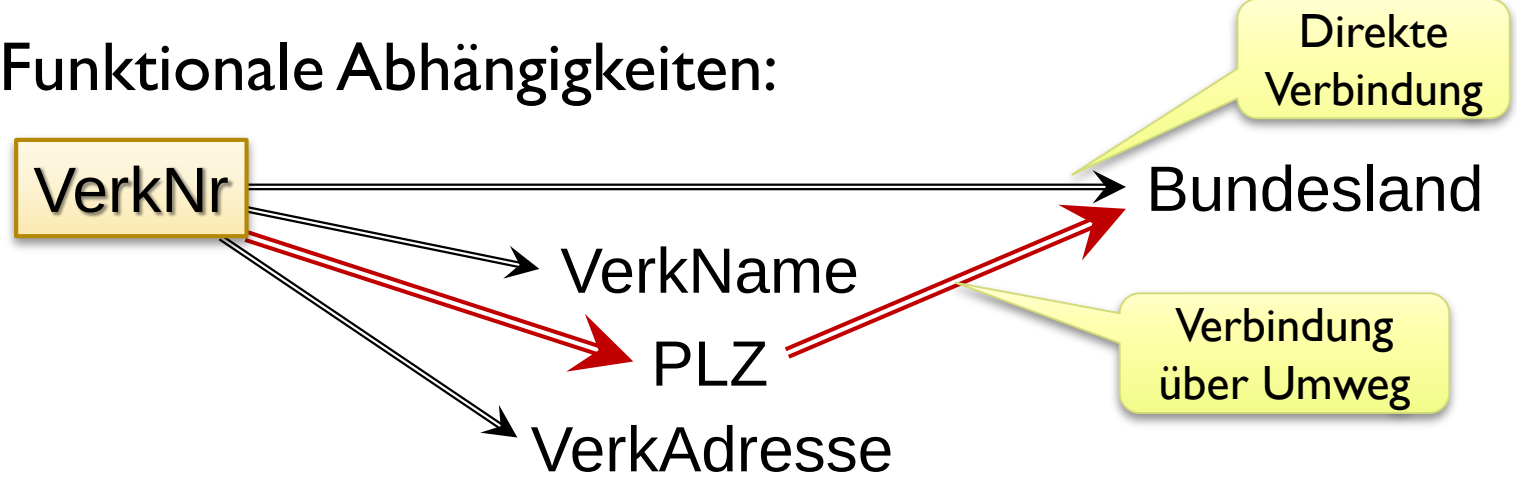
ProdNr	Produktname
P1	Waschmaschine
P2	Herd
P3	Kühlschrank
P4	Staubsauger

Weitere Produkteigen-schaften jetzt möglich!

VerkaeuferLand

VerkNr	VerkName	PLZ	VerkAdresse	Bundesland
V1	Meier	80075	München	Bayern
V2	Schneider	70038	Stuttgart	Baden-Württemberg
V3	Müller	50083	Köln	Nordrhein-Westfalen

- Funktionale Abhängigkeiten:



Transitive Abhängigkeit

- Ein Attribut Y einer Relation R heißt transitiv abhängig vom Attribut X derselben Relation, wenn ein Nichtschlüssel-Attribut Z existiert, so dass gilt:
  - Das Attribut Z hängt voll funktional vom Attribut X und das Attribut Y voll funktional vom Attribut Z ab.

- wenn also ein Z existiert mit: $X \Rightarrow Z \Rightarrow Y$

- VerkaeuferLand: VerkNr $\Rightarrow$ PLZ $\Rightarrow$ Bundesland
- Transitive Abhängigkeit des Bundeslands von der Verkäufernummer!

Dritte Normalform nach Codd

- Eine Relation ist in der dritten Normalform (nach Codd), wenn sie sich in der zweiten Normalform befindet und jedes Nichtschlüsselattribut nicht transitiv vom Primärschlüssel abhängt.

- Relation VerkaeuferLand ist nicht in der 3. NF nach Codd
- VerkaeuferLand ist nicht in der 3. NF nach Boyce u. Codd
  - denn: PLZ ist Determinante, aber kein Schlüsselkandidat
- Gibt es einen Unterschied zwischen den beiden 3. NFs?

Vergleich der dritten Normalformen

- **3. NF nach Boyce und Codd → 3. NF nach Codd**
  - Beweis:
    - Transitive Abhängigkeiten bedingen eine Determinante, die nicht Schlüsselkandidat ist.
    - 3. NF nach Boyce u. Codd → keine solchen Determinanten → keine transitiven Abhängigkeiten → 3. NF nach Codd
- **Das Umgekehrte gilt nicht!**
  - Es gibt Relationen in 3. NF nach Codd, die nicht in 3. NF nach Boyce u. Codd sind.
    - Übung! Bitte die nächste Folie beachten!

Hinweise zur dritten Normalform

- **Für Relationen R mit einfachen (nicht zusammen-gesetzten) Schlüsselkandidaten gilt:**
  - Beide Definitionen der 3. NF sind gleichwertig
- **Es gilt (ohne Beweis):**
  - Unterschiede kann es nur dann geben, wenn zwei zusammengesetzte Schlüsselkandidaten existieren, die ein gemeinsames Attribut besitzen
- **Die dritten Normalformen beseitigen alle Redundanzen und Anomalien außerhalb der Schlüsselkandidaten**
- **Also: Wir definieren nur sinnvolle Schlüsselkandidaten!**

VerkaeuferProdukt3NF

VerkName	PLZ	VerkAdresse	Produktname	Umsatz
Meier	80075	München	Waschmaschine	11000
Meier	80075	München	Herd	5000
Meier	80075	München	Kühlschrank	1000
Schneider	70038	Stuttgart	Herd	4000
Schneider	70038	Stuttgart	Kühlschrank	3000
Müller	50083	Köln	Staubsauger	1000

- **Voraussetzung:**
  - (VerkName, PLZ, VerkAdresse) identifiziert Verkäufer eindeutig
  - Gegebenenfalls Zusätze: komplette Adresse, Vorname, junior usw.
- **Nur ein Schlüsselkandidat und eine Determinante:**
  - (VerkName, PLZ, VerkAdresse, Produktname)

Problem der dritten Normalform

- **3. NF beseitigt keine Redundanzen innerhalb der Schlüsselkandidaten**
- **Lösung**
  - Wir erstellen Relationen mit vernünftigen Schlüsselkandidaten
- **Was ist vernünftig?**
  - Gesunden Menschenverstand anwenden, oder:
  - Weitere Normalformen studieren
- **Achtung:**
  - Relation **VerkaeuferProdukt3NF** ist in 4. NF (siehe später)!

Relation VerkaeuferProduktKFZ

VerkNr	Produktname	KFZNr
1	Waschmaschine	M-E 515
1	Waschmaschine	M-X 333
1	Herd	M-E 515
1	Herd	M-X 333
1	Kühlschrank	M-E 515
1	Kühlschrank	M-X 333
2	Herd	S-H 654
2	Herd	K-J 123
2	Kühlschrank	S-H 654
2	Kühlschrank	K-J 123
3	Staubsauger	K-J 123

- **Es gelte:**
  - Meier (Verknr 1) benutzt M-E 515 und M-X 333
  - Schneider (Nr 2) benutzt S-H 654 und K-J 123
  - Müller (Verknr 3) benutzt K-J 123
- **Keine funktionalen Abh.**
- **Primärschlüssel:**
  - (VerkNr, Produktname, KFZNr)

Probleme von VerkaeuferProduktKFZ

- **Relation ist in 3. NF**
- **Primärschlüssel enthält aber Redundanzen**
- **Anomalien treten auf:**
  - Die erste Zeile kann nicht gelöscht werden, ohne dass auch andere Zeile gelöscht werden müssen
  - Verkauft Verkäufer Schneider (Verknr 2) Staubsauger, so müssen 2 Zeilen eingefügt werden
- **Problem:**
  - Es wurden 2 funktionale Abhängigkeiten ineinander gemengt, die voneinander aber völlig unabhängig sind:
  - Verkäufer verkauft Produkte <u>und</u> Verkäufer fährt mit KFZ

Mehrwertige Abhängigkeit

- Ein Attribut Y einer Relation ist von einem Attribut X dieser Relation mehrwertig abhängig (X↠Y), wenn ein weiteres Attribut Z dieser Relation existiert mit den Eigenschaften:
  - Ein Y-Attributwert hängt vom dazugehörigen (X,Z)-Paar bereits allein eindeutig vom X-Wert ab und ist unabhängig vom Z-Attribut.
  - Das Attribut X ist minimal.
- Es gilt in VerkaeuferProduktKFZ
  - X=Verknr, Y=Produktname, Z=KFZNr,
  - also: **Verknr ↠ Produktname**

Infos zur mehrwertigen Abhängigkeit

- Aus $X \twoheadrightarrow Y$ (über Z) folgt $X \twoheadrightarrow Z$ (über Y)
  - Begründung: Symmetrie zwischen den beiden Abhängigkeiten
- Also:
  - Verknr $\twoheadrightarrow$ Produktname
  - Verknr $\twoheadrightarrow$ KFZNr
- Aus $X \Rightarrow Y$ folgt $X \twoheadrightarrow Y$
- Die mehrwertige Abhängigkeit ist eine Verallgemeinerung der (vollen) funktionalen Abhängigkeit
  - Beweis: Setzen wir in der Definition der mehrwertigen Abh. $Z=\emptyset$, so ist die funktionale Abh. gegeben

Überführung in NF²

- Verletzen wir die Atomarität, so lässt sich Relation VerkaeuferProduktKFZ einfach abbilden:

VerkNr	Produktname	KFZNr
1	Waschmaschine Herd Kühlschrank	M-E 515 M-X 333
2	Herd Kühlschrank	S-H 654 K-J 123
3	Staubsauger	K-J 123

- Hinweis:
  - Wir verbieten solche Relationen in der 4. NF
  - Die nicht atomare Relation zeigt, wie wir zerlegen können!

Vierte Normalform

- Eine normalisierte Relation ist in der vierten Normalform, wenn aus jeder mehrwertigen Abhängigkeit $X \twoheadrightarrow Y$ folgt, dass X ein Schlüsselkandidat ist.

- VerkaeuferProduktKFZ besitzt zwei mehrwertige Abhängigkeiten und VerkNr ist kein Schlüsselkandidat.
  - Also: Keine 4. NF
- Jede Relation mit mehrwertigen Abhängigkeiten lässt sich in seine Abhängigkeiten zerlegen, bei zwei mehrwertigen Abhängigkeiten also in zwei Relationen!

Zerlegung von VerkaeuferProduktKFZ

Relation VerkaeuferProduktname:

VerkNr	Produktname
1	Waschmaschine
1	Herd
1	Kühlschrank
2	Herd
2	Kühlschrank
3	Staubsauger

Relation VerkaeuferKFZ:

VerkNr	KFZNr
1	M-E 515
1	M-X 333
2	S-H 654
2	K-J 123
3	K-J 123

- Beide Relationen sind in 4. NF und optimal
- Bei vielen Verkäufen und Verwendung vieler KFZ haben beide Relationen wesentlich weniger Redundanz als die Originalrelation

Weitere Probleme mit 4. Normalform

- Gewünscht: Angabe der Kilometer, die Verkäufer mit KFZ gefahren ist, abhängig vom verkauften Produkt und Jahr:

VerkNr	Produktname	KFZNr	Jahr	KM
I	Waschmaschine	M-E 515	2011	622
I	Waschmaschine	M-E 515	2012	1105
I	Waschmaschine	M-X 333	2011	305
I	Waschmaschine	M-X 333	2012	0
I	Herd	M-E 515	2011	912
I	Herd	M-E 515	2012	1111
I	Herd	M-X 333	2011	0
I	Herd	M-X 333	2012	222
I	Kühlschrank	M-E 515	2011	333
…	…	…	…	…

Fragen zu dieser Relation

- Schlüsselkandidaten, Primärschlüssel?
- Volle funktionale Abhängigkeiten?
- Mehrwertige Abhängigkeiten?
- Normalform?
- Wenn nicht 4. NF, wie zerlegen wir diese Relation?

- Antwort:
  - Übung

VerkaeuferProdukt4NF

VerkNr	Produktname	KFZNr
1	Waschmaschine	M-E 515
1	Herd	M-E 515
1	Herd	M-X 333
1	Kühlschrank	M-E 515
2	Herd	S-H 654
2	Herd	K-J 123
2	Kühlschrank	S-H 654
3	Staubsauger	K-J 123

Tupel nicht einzeln löschbar

Löschen möglich!

- Annahme:
  - Spezialhalterung für Waschmaschinen und Kühlschränke gibt es nicht in:
  - M-X 333 und K-J 123

- Produktname und KFZNr hängen jetzt voneinander ab
- Es gibt keine mehrwertigen Abhängigkeiten, also: 4. NF
- Aber: Es treten immer noch Anomalien auf:
  - Beispiel: Nicht jeder Herd lässt sich einzeln löschen!!

VerkaeuferProdukt4NF

- **Es gibt auch Einfügeanomalien!**
- **Ein Zerlegen in 2 Relationen ist nicht möglich ohne Verlust an Information!**
  - Zerlegen in VerkaeuferProduktname und VerkaeuferKFZ bringt nichts, da
    VerkaeuferProduktname ⋈ VerkaeuferKFZ =
    VerkaeuferProduktKFZ !!!
  - Die Info über die fehlenden Halterungen in einigen KFZ geht verloren!
- **Aber: Zerlegen mit anschließendem Verbund ist ein guter Ansatz**

Definition (Verbundabhängigkeit)

- Eine Relation R besitzt eine **Verbundabhängigkeit**, wenn sie mittels Projektion nicht trivial in Teilrelationen zerlegt werden kann, so dass der Verbund dieser Teilrelationen wieder die Relation R ergibt.
- **Nicht trivial** heißt, dass die Teilrelationen jeweils unterschiedliche Primärschlüssel besitzen.
- Beispiel. Mit
  - $R = \pi_{Projektion1}(R) \bowtie \pi_{Projektion2}(R) \bowtie \pi_{Projektion3}(R)$
  - Projektion1 bis 3 sind nicht trivial
  - Dann besitzt R eine Verbundabhängigkeit

Verbundabhängigkeit

- Relation VerkaeuferProduktKFZ besitzt nicht nur eine mehrwertige Abhängigkeit, sondern auch eine Verbund-abhängigkeit wegen

$$\text{VerkaeuferProduktname} \bowtie \text{VerkaeuferKFZ} = \text{VerkaeuferProduktKFZ}$$
$$\text{VerkaeuferProduktname} = \pi_{\text{Verknr,Produktname}}(\text{VerkaeuferProduktKFZ})$$
$$\text{VerkaeuferKFZ} = \pi_{\text{Verknr,KFZNr}}(\text{VerkaeuferProduktKFZ})$$

- Dies gilt allgemein (ohne Beweis):
  - Aus funktionaler Abhängigkeit folgt mehrwertige Abhängigkeit
  - Aus mehrwertiger Abhängigkeit folgt Verbundabhängigkeit

Definition (Fünfte Normalform)

- **Eine Relation ist in der fünften Normalform, wenn sie in der vierten Normalform ist und keine Verbundabhängigkeiten besitzt.**

- **In der Praxis:**
  - Es ist extrem schwer, Verbundabhängigkeiten zu finden
  - Diese Abhängigkeiten müssen noch nicht vorhanden sein und könnten erst in Zukunft auftreten!
  - Die 5. NF spielt daher fast keine Rolle
- **Ist Relation VerkaeuferProdukt4NF in der 5. NF?**

Zerlegung von VerkaeuferProdukt4NF

VerkaeuferProduktname = $\pi_{VerkNr,Produktname}$(VerkaeuferProdukt4NF)
VerkaeuferKFZ = $\pi_{VerkNr,KFZNr}$(VerkaeuferProdukt4NF)
ProduktKFZ = $\pi_{Produktname,KFZNr}$(VerkaeuferProdukt4NF)

VerkaeuferProduktname

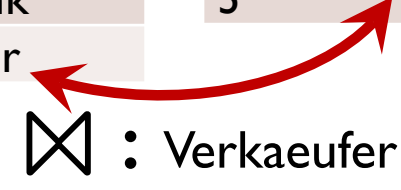
VerkNr	Produktname
1	Waschmaschine
1	Herd
1	Kühlschrank
2	Herd
2	Kühlschrank
3	Staubsauger

VerkaeuferKFZ

VerkNr	KFZNr
1	M-E 515
1	M-X 333
2	S-H 654
2	K-J 123
3	K-J 123

ProduktKFZ

Produktname	KFZNr
Waschmaschine	M-E 515
Herd	M-E 515
Herd	M-X 333
Herd	S-H 654
Herd	K-J 123
Kühlschrank	M-E 515
Kühlschrank	S-H 654
Staubsauger	K-J 123



⋈ : VerkaeuferProduktKFZ !

Verbund: VerkaeuferProdukt4NF

VerkaeuferProduktKFZ

VerkNr	Produktname	KFZNr
1	Waschmaschine	M-E 515
~~1~~	~~Waschmaschine~~	~~M-X 333~~
1	Herd	M-E 515
1	Herd	M-X 333
1	Kühlschrank	M-E 515
~~1~~	~~Kühlschrank~~	~~M-X 333~~
2	Herd	S-H 654
2	Herd	K-J 123
2	Kühlschrank	S-H 654
~~2~~	~~Kühlschrank~~	~~K-J 123~~
3	Staubsauger	K-J 123

ProduktKFZ

Produktname	KFZNr
Waschmaschine	M-E 515
Herd	M-E 515
Herd	M-X 333
Herd	S-H 654
Herd	K-J 123
Kühlschrank	M-E 515
Kühlschrank	S-H 654
Staubsauger	K-J 123

Folgerung:
Wir erhalten VerkaeuferProdukt4NF!
VerkaeuferProdukt4NF besitzt Verbund-abhängigkeit und ist nicht in 5. NF

Zusatzinfos zur 4. und 5. Normalform

- Besitzt eine Relation nur nicht zusammengesetzte Schlüsselkandidaten, dann gilt:
  - 3. NF (beide Versionen) = 4. NF = 5. NF
- Eine Relation in 4. NF benötigt mindestens 3 Relationen zum Zerlegen in Relationen der 5. NF
- Relationen in mindestens der 3. NF, aber nicht in 4. oder 5. NF besitzen einen Schlüsselkandidaten mit mindestens drei Attributen!

Zusammenfassung

- Ziel ist die dritte Normalform!
- Wir benötigen Wissen zu funktionaler Abhängigkeit!

- Wir erzeugen Relationen mit möglichst einfachen Schlüsselkandidaten
  - Somit benötigen wir kein Wissen zur 4. und 5. NF
  - Alle Relationen in 3. NF sind dann auch in 4. und 5. NF!

Datenbanken und SQL

Kapitel 3

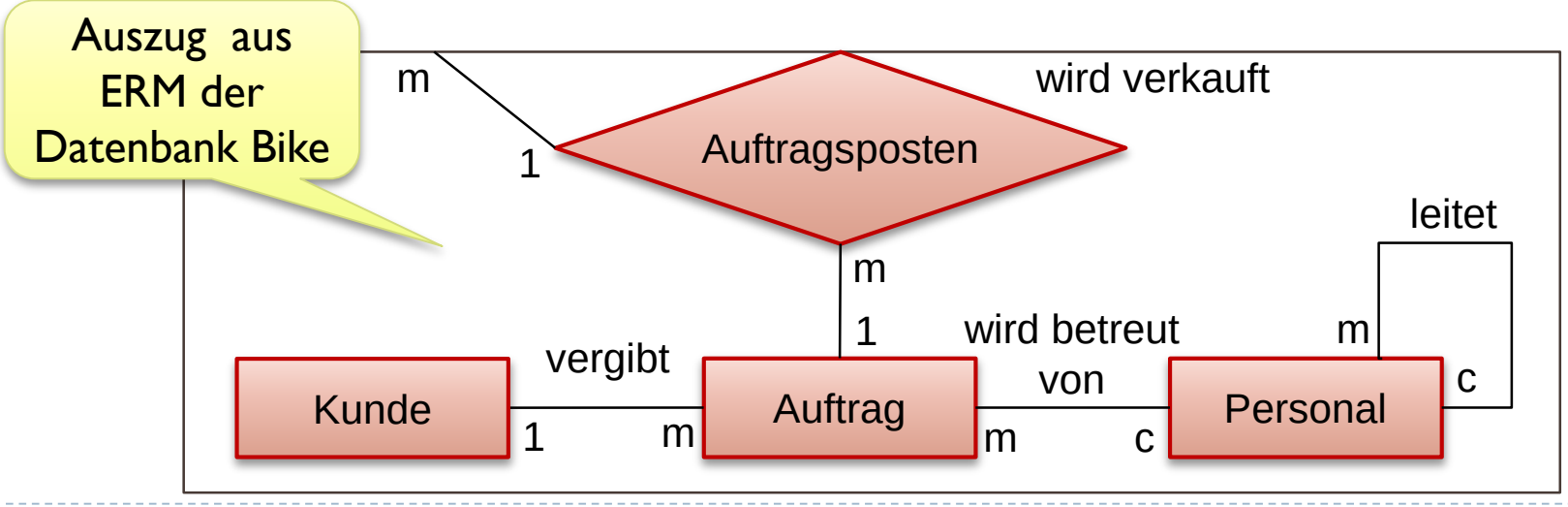
Datenbankdesign – Teil 2: Entity-Relationship-Modell

Datenbankdesign

- Entity-Relationship-Modell ERM
  - Entitäten und ihre Eigenschaften
  - Beziehungen zwischen den Entitäten
  - Überführung der Entitäten in Relationen
  - Überführung der Beziehungen in Fremdschlüssel
  - Fremdschlüsseleigenschaften
  - Schwache Entitäten
  - Subtypen

Entity-Relationship-Modell (ERM)

- Bisher:
  - Betrachten einzelner Relationen isoliert für sich
- Jetzt:
  - Betrachten des Zusammenspiels der Relationen



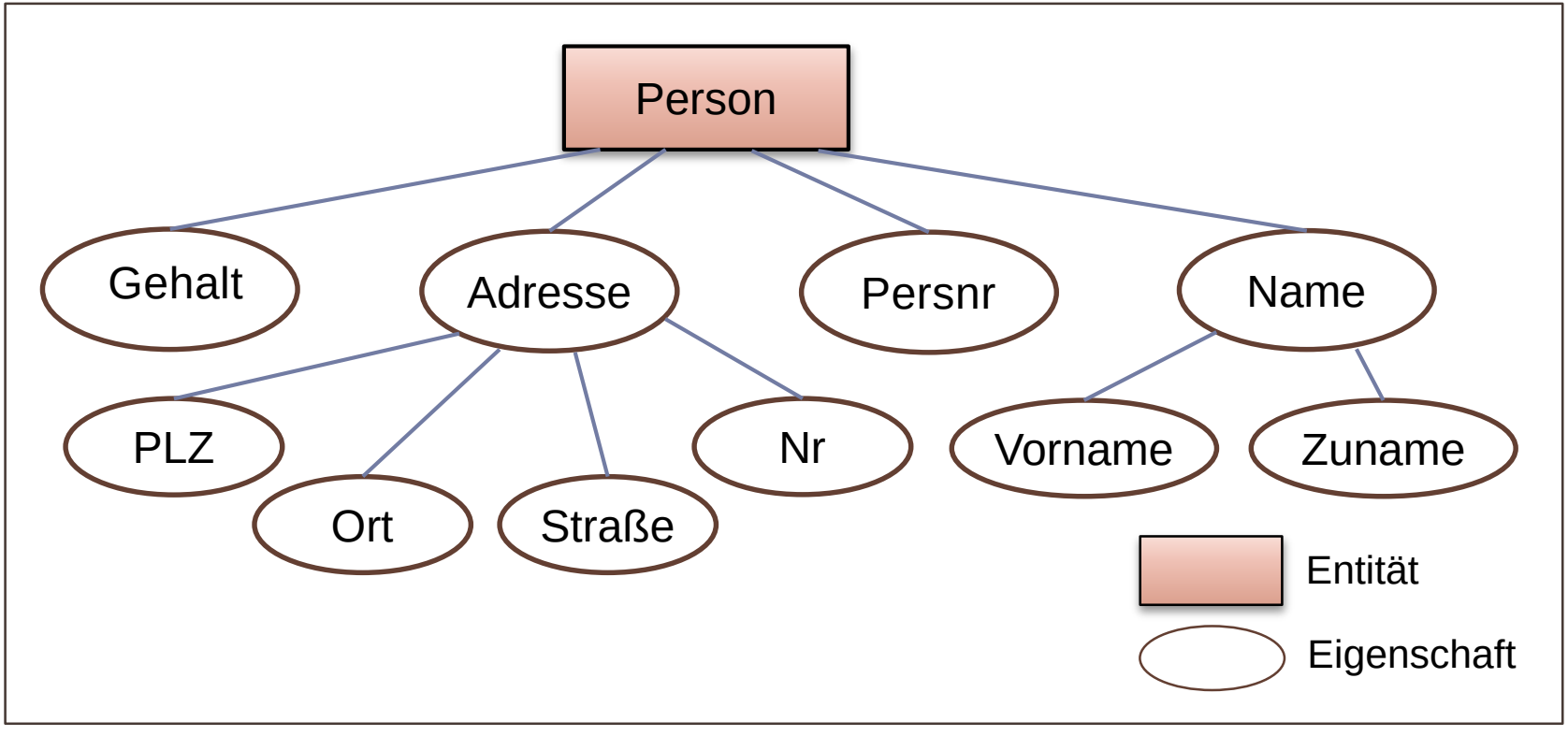
Begriffe im Entity-Relationship-Modell

Entität	Ein eindeutig unterscheidbares Objekt, ein unterscheidbares Element
Eigenschaft	Ein Teil einer Entität, der die Entität beschreibt
Beziehung	Eine Entität, die zwei oder mehr Entitäten miteinander verknüpft
Subtyp	Eine Entität, die ein Teil einer anderen, umfassenderen Entität ist
Supertyp	Eine Entität, die Subtypen enthält
Schwache Entität	Entität, die von einer anderen Entität vollständig abhängig ist

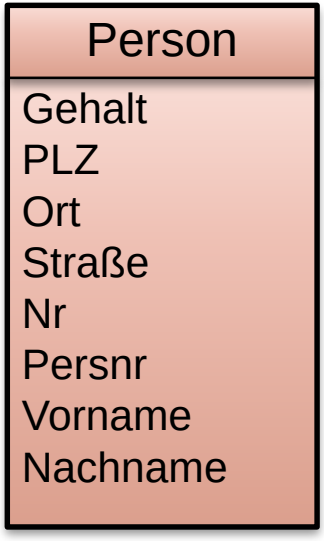
Beispiele zu den Begriffen

Begriff	Beispiele
Entität	Person, Werkzeug, Produkt, Rechnung
Eigenschaft	Name, Vorname, PLZ, Ort einer Person; Größe, Gewicht eines Werkzeugs; Preis eines Produkts, Rechnungsdatum
Beziehung	Die Entitäten Verkäufer und Produkt stehen miteinander in einer Beziehung: Der Verkäufer verkauft Produkte.
Subtyp	Die Entität Verkäufer ist ein Subtyp zur Entität Mitarbeiter
Supertyp	Die Entität Mitarbeiter ist ein Supertyp der Entität Verkäufer
Schwache Entität	Die Entität Arbeitszeit ist schwach gegenüber der Entität Mitarbeiter

Beispiel: Entität Person



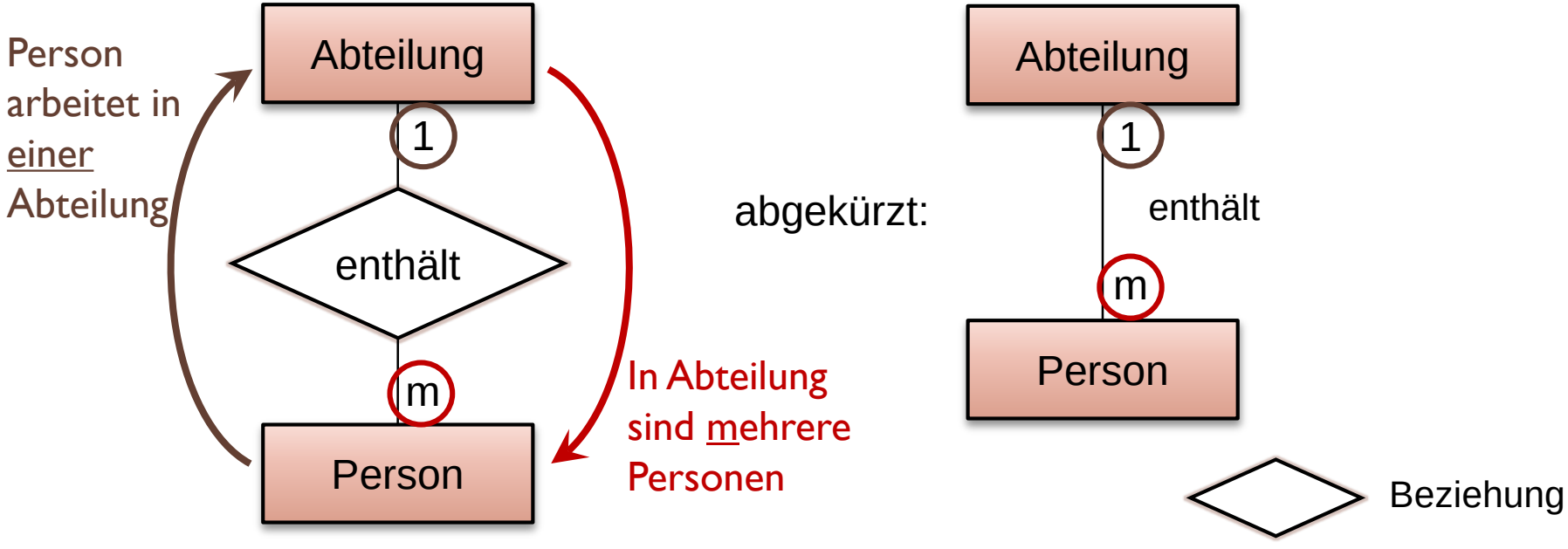
Entität Person in „UML“-Notation




Entität

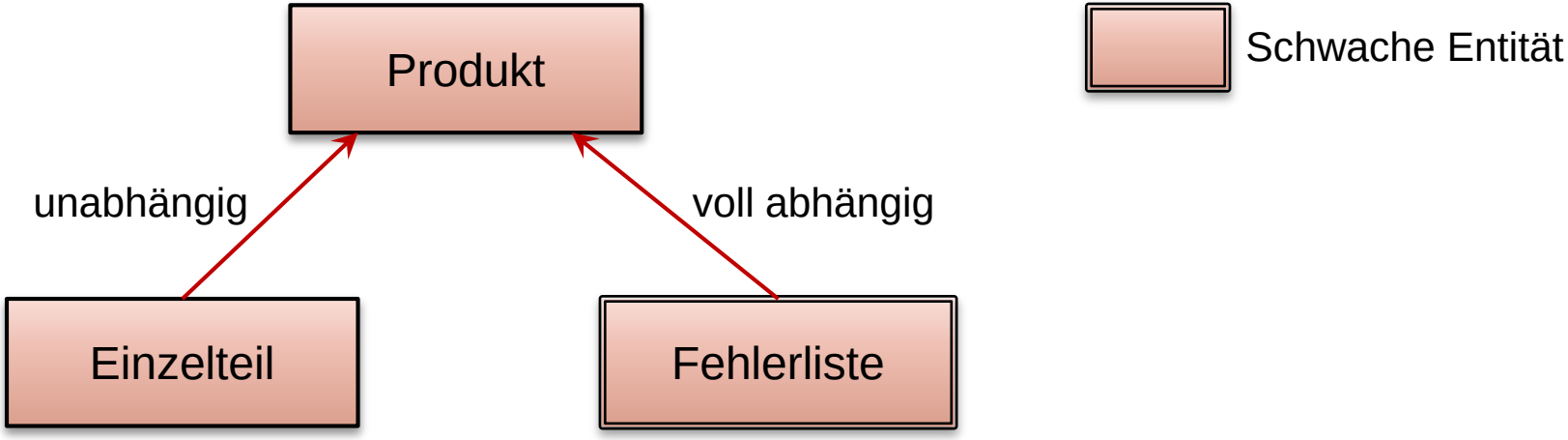
- In die Entität werden die Eigenschaften mit aufgenommen
- Manchmal werden auch Primärschlüssel, alternative Schlüssel und Fremdschlüssel gleich mit gekennzeichnet

Beispiel einer Beziehung



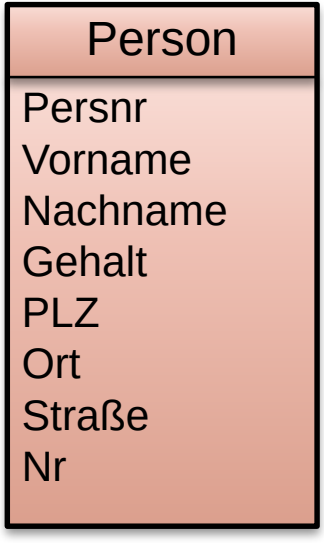
- In einer Abteilung arbeiten mehrere (m) Personen
- Eine Person arbeitet in genau einer (1) Abteilung

Schwache Entität



- Einzelteil:
  - Ist auf Lager, auch wenn Produkt nicht mehr ex.
  - Unabhängig vom Produkt
- Produkt-Fehlerliste:
  - Wertlos, wenn Produkt nicht mehr ex.
  - Komplett abhängig vom Produkt

Umsetzung der Entität Person in SQL




```
CREATE TABLE  Person
(     Persnr          INTEGER,
      PRIMARY KEY (Persnr),
      Vorname         CHARACTER(20),
      Nachname        CHARACTER(20)  NOT NULL,
      Gehalt          NUMERIC (10, 2),
      PLZ             CHARACTER(5),
      Ort             CHARACTER(25),
      Strasse         CHARACTER(25),
      Nr              CHARACTER(4)
) ;
```

Ganzzahl

Primärschlüssel

Zeichenkette der Länge 20

Nachname muss angegeben werden

Gleitpunktzahl: 10 Zeichen, davon 2 Nachkommastellen

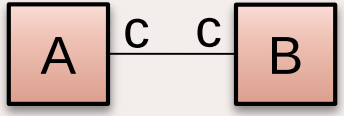
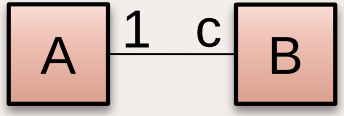
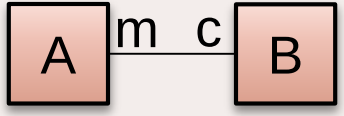
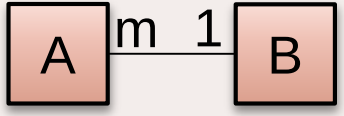
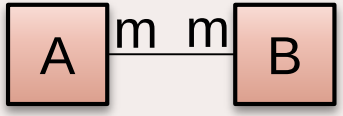
Beziehungen (grobe Einteilung)

- **1 zu 1 Beziehung**
  - Ein Auto hat einen Motor
  - Ein Motor ist in einem Auto
- **m zu 1 Beziehung**
  - In einer Abteilung arbeiten mehrere Personen
  - Eine Person ist einer Abteilung zugeordnet
- **m zu n Beziehungen**
  - Ein Verkäufer verkauft mehrere Produkte
  - Ein Produkt wird von mehreren Verkäufern angeboten

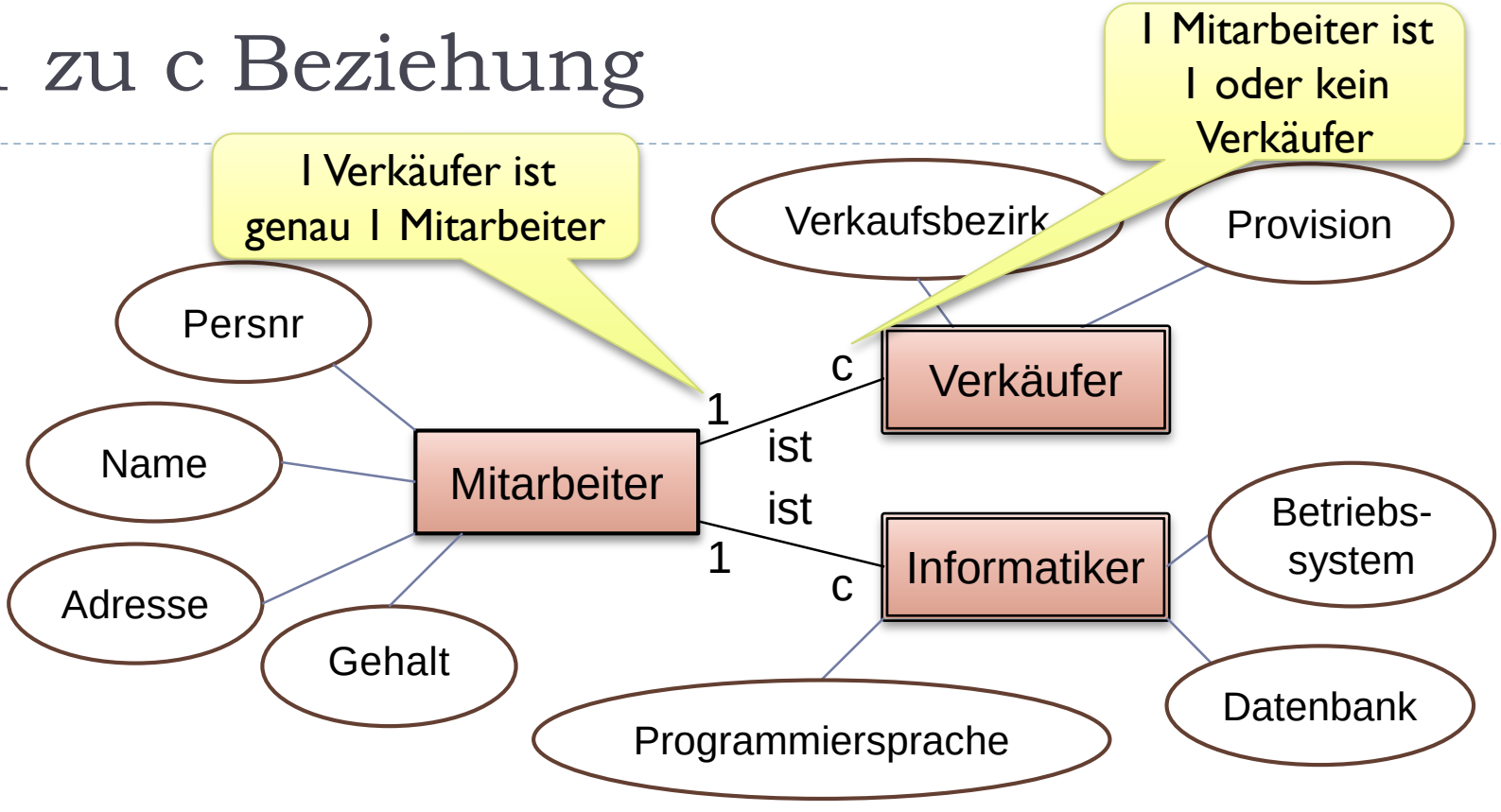
Beziehungen (Besonderheiten)

- **m:**
  - Mehrere kann sein: 0, 1, 2, … 13, … 103.517 usw.
  - m entspricht dem häufig verwendeten Sternsymbol (*)
- **n:**
  - Nur ein anderer Buchstabe für m
- **1:**
  - Eins kann sein: 0 oder 1 (Beispiel: Ein Motor ist nicht im KFZ!)
  - Unterscheidung ist wichtig in relationalen Datenbanken
  - Wir verwenden **c** für 0 oder 1, also **c** $\in$ {0, 1}
  - Wir verwenden 1, wenn der Wert 0 nicht vorkommen darf

Mögliche Beziehungen

Beziehungen	c (0..1)	1	m
c (0..1)		Symmetrie!	Symmetrie!
1		Nicht möglich	Symmetrie!
m			

1 zu c Beziehung

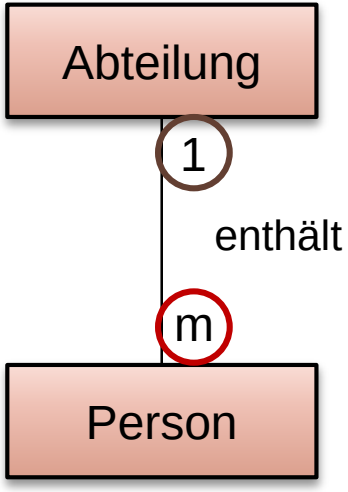


- Detailinfos: ausgelagert in Subtypen Verkäufer, Informatiker
- Reduziert Redundanzen

c zu c und 1 zu 1 Beziehungen

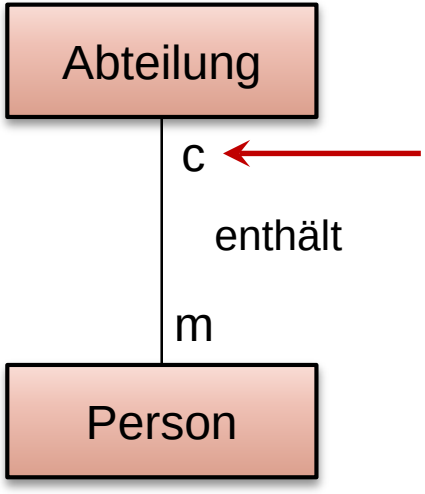
- **c zu c:**
  - Sehr selten, etwa: Auto --- Motor
  - Spezialfall von m zu c, zusätzlich: Fremdschlüssel ist eindeutig
- **1 zu 1:**
  - Erfordert, dass in beiden Entitäten immer je ein Eintrag existiert (ein Verweis muss ja gegenseitig existieren!)
  - In relationalen Datenbanken erfolgt erst ein Eintrag der einen, dann ein Eintrag der anderen, also: 1 zu c Bedingung
  - In relationalen Datenbanken treten diese Beziehungen also nicht auf (außer mittels komplexer Transaktionsmechanismen)

m zu 1 Beziehung



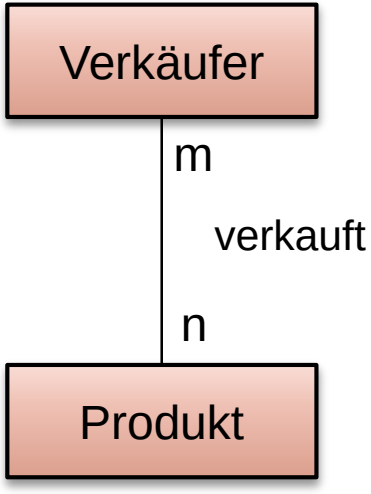
- In einer Abteilung arbeiten mehrere Personen
- Eine Person ist exakt einer Abteilung zugeordnet
- Wichtig:
  - Eine Person ist immer einer Abteilung zugewiesen
  - Aber: In einer Abteilung können vorübergehend auch keine Personen arbeiten
    - m lässt den Wert 0 zu!

m zu c Beziehungen



- In einer Abteilung arbeiten mehrere Personen
- Eine Person ist einer oder keiner Abteilung zugeordnet
  - Szenario:
  - Abteilung wird aufgelöst. Mitarbeiter gehören dann keiner Abteilung an und werden erst nach und nach anderen Abteilungen zugeordnet
  - Dies erfordert: m zu c !

m zu n Beziehungen



- Ein Verkäufer verkauft mehrere Produkte
- Ein Produkt wird von mehreren Verkäufern verkauft
- Wichtig:
  - m zu n schließt ein:
  - Ein neuer Verkäufer hat noch nichts verkauft
  - Ein neues Produkt wurde noch nicht verkauft

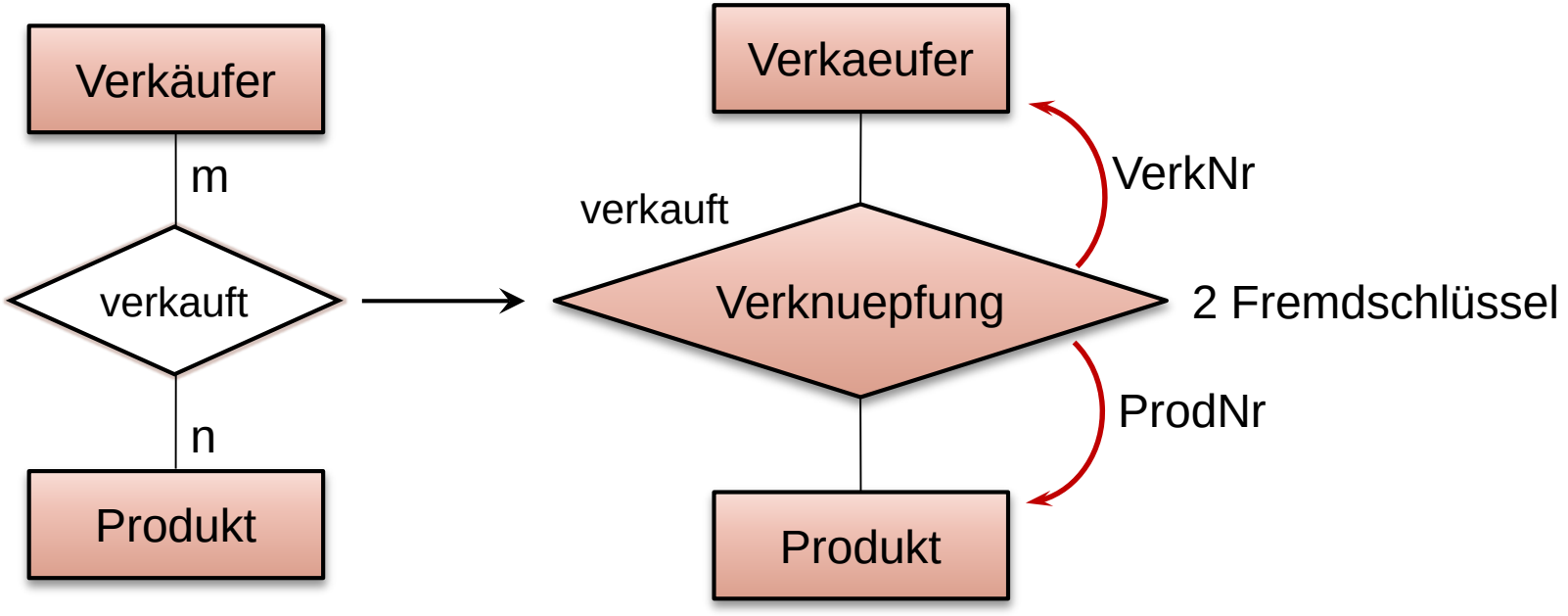
Beispiele (1)

Beziehung	Bemerkung
KFZ-Halter $\underset{c\ zu\ m}{\longleftrightarrow}$ KFZ	I Halter kann mehrere KFZ anmelden I KFZ ist auf maximal einen Halter zugelassen
Student $\underset{m\ zu\ n}{\longleftrightarrow}$ Vorlesung	I Student besucht mehrere Vorlesungen I Vorlesung belegen mehrere Studenten
Kunde $\underset{1\ zu\ m}{\longleftrightarrow}$ Bestellung	I Kunde gibt mehrere Bestellungen auf I Bestellung gehört zu genau einem Kunden
Bewohner $\underset{m\ zu\ 1}{\longleftrightarrow}$ Haus	I Bewohner wohnt in einem Haus In I Haus wohnen mehrere Bewohner

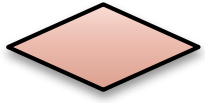
Beispiele (2)

Beziehung	Bemerkung
Park $\underset{1\ zu\ m}{\longleftrightarrow}$ Baum	In 1 Park wachsen mehrere Bäume 1 bestimmter Baum steht in einem Park
Park $\underset{m\ zu\ n}{\longleftrightarrow}$ Baumart	In 1 Park wachsen mehrere Baumarten 1 Baumart wächst in mehreren Parks
KFZ $\underset{c\ zu\ c}{\longleftrightarrow}$ Motor	1 KFZ besitzt maximal einen Verbrennungsmotor 1 Motor wird in maximal einem KFZ eingebaut
KFZ-Typ $\underset{m\ zu\ \mathrm{n}}{\longleftrightarrow}$ Motortyp	1 KFZ-Typ besitzt mehrere Motorvarianten 1 Motortyp wird in mehreren KFZ-Typen verbaut
Leiter $\underset{c\ zu\ 1}{\longleftrightarrow}$ Abteilung	1 Abteilungsleiter leitet genau eine Abteilung 1 Abteilung besitzt maximal einen Abteilungsleiter

m zu n Beziehungen: Realisierung (1)

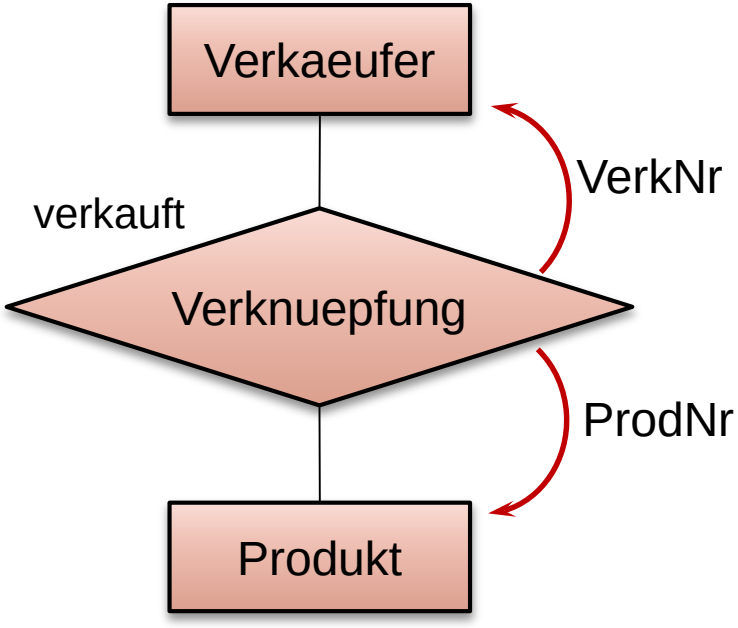


- Eigene Relation erforderlich
- Relation enthält 2 Fremdschlüssel
- Die 2 Fremdschlüssel sind Schlüsselkandidat



Beziehungsrelation

m zu n Beziehungen: Realisierung (2)




```
CREATE TABLE  Verknuepfung
(  VerkNr     CHARACTER(4)
              REFERENCES Verkaeufer,
   ProdNr     CHARACTER(4)
              REFERENCES Produkt,
   Umsatz     INTEGER,
   PRIMARY KEY  (VerkNr, ProdNr)
) ;
```

Fremdschlüssel

Fremdschlüssel

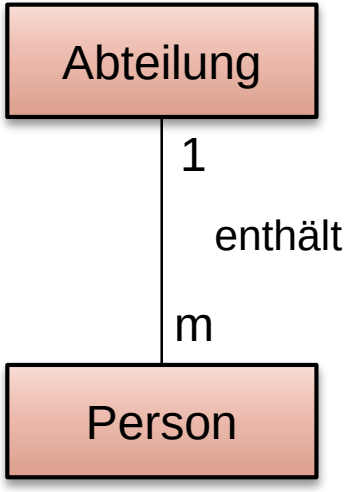
Primärschlüssel

Definition (Beziehungsrelation)

- Seien k Relationen mit k>1 gegeben. Eine Relation R heißt Beziehungsrelation, wenn sie diese k Relationen wie folgt miteinander verbindet:
  - R enthält k Fremdschlüssel mit k>1, die je auf genau eine der k gegebenen Relationen verweisen.
  - Die k Fremdschlüssel bilden zusammen einen Schlüsselkandidaten.

- Wichtig:
  - Jede Relation mit obigen Eigenschaften ist also eine Beziehungsrelation!

m zu 1 Beziehung: Realisierung



- In der m Beziehung wird ein Fremdschlüssel hinzugefügt

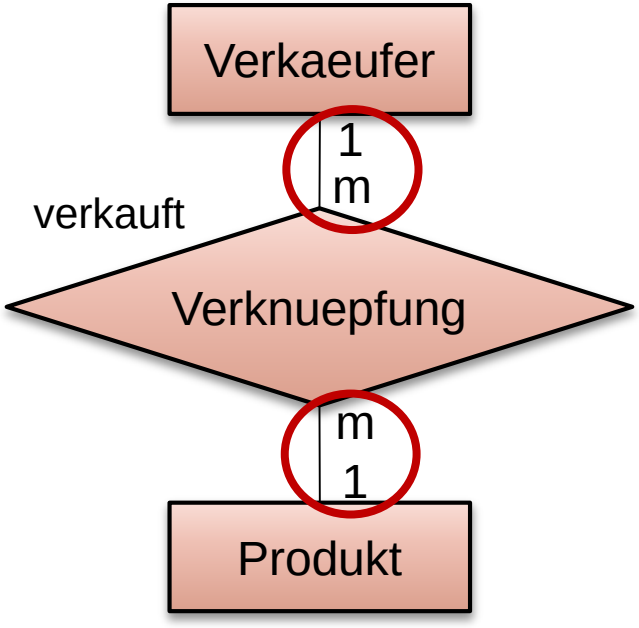
```
CREATE TABLE Person
(  PersNr        INTEGER,
   Name          CHARACTER (25),
   ...
   Abteilungsnr  INTEGER   NOT NULL
                 REFERENCES Abteilung,
   PRIMARY KEY (Persnr)
) ;
```

Wegen m zu 1

Fremdschlüssel

Primärschlüssel

Einschub: m zu n = Zwei m zu 1



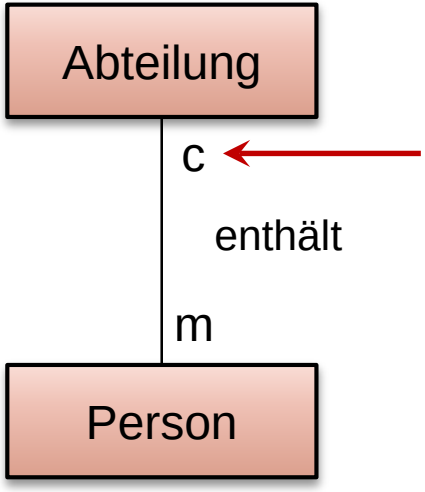

```
CREATE TABLE Verknuepfung
(  VerkNr    CHARACTER(4)
             REFERENCES Verkaeufer,
   ProdNr    CHARACTER(4)
             REFERENCES Produkt,
   Umsatz    INTEGER,
   PRIMARY KEY  (VerkNr, ProdNr)
) ;
```

Fremdschlüssel

Fremdschlüssel

Kein NOT NULL, da Teil des Primärschlüssels

m zu c Beziehung: Realisierung



- Wie m zu 1, allerdings sind Nullwerte erlaubt

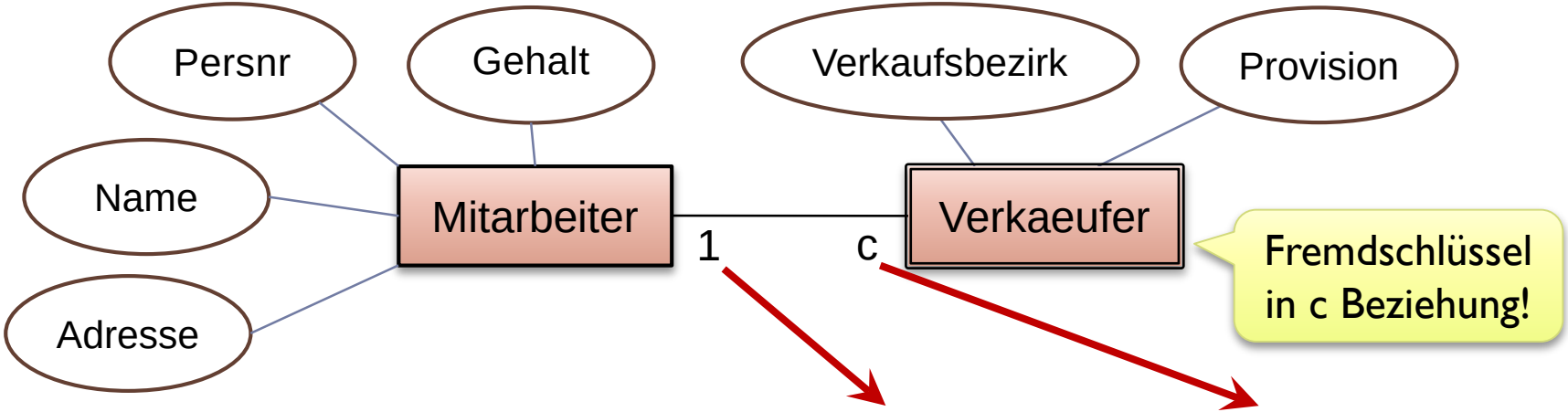
```
CREATE TABLE Person
(  PersNr          INTEGER,
   Name            CHARACTER (25),
   ...
   Abteilungsnr    INTEGER
                   REFERENCES Abteilung,
   PRIMARY KEY (Persnr)
) ;
```

Kein NOT NULL

Fremdschlüssel

Primärschlüssel

1 zu c Beziehung: Realisierung



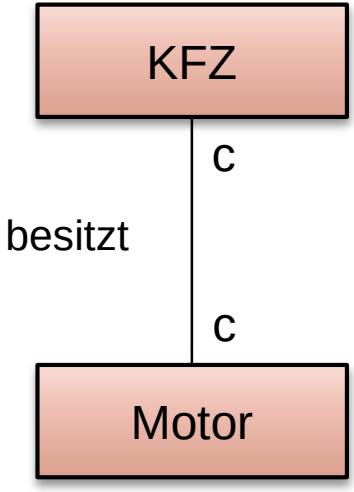
- Fremdschlüssel in Verkaeufer erfordert NOT NULL und Eindeutigkeit
- Der Primärschlüssel ist meist auch Fremdschlüssel (bei Subtypen!)

```
CREATE TABLE Verkaeufer
(  PersNr   INTEGER    REFERENCES Mitarbeiter,
   PRIMARY KEY (PersNr),
   . . .    ) ;
```

Primärschlüssel

Fremdschlüssel

c zu c Beziehung: Realisierung



- Wie m zu c Beziehung, allerdings ist Fremdschlüssel zusätzlich eindeutig
- Empfehlung: Die „umfassendere“ Entität enthält den Fremdschlüssel

```
CREATE TABLE KFZ
(  KFZNr      INTEGER,
   PRIMARY KEY (KFZNr),
   MotorNr    INTEGER  REFERENCES Motor,
   UNIQUE (MotorNr),
   ...
) ;
```

Eindeutig!

Fremdschlüssel

Zusammenfassung zur Realisierung

Beziehung	Überführung in Relationen und Fremdschlüssel
m zu n	Erfordert Beziehungsrelation mit zwei m zu 1 oder m zu c Beziehungen
m zu c	Hinzufügen eines Fremdschlüssels zur m Relation
m zu 1	Wie m zu c! Zusätzlich: Fremdschlüssel ist NOT NULL
c zu c	Wie m zu c! Zusätzlich: Fremdschlüssel ist UNIQUE
c zu 1	Wie c zu c! Zusätzlich: Fremdschlüssel ist NOT NULL Meist ist Fremdschlüssel der Primärschlüssel

Fremdschlüsseleigenschaften

(1) **Darf ein Fremdschlüsselwert leer bleiben, also Null-Werte enthalten?**

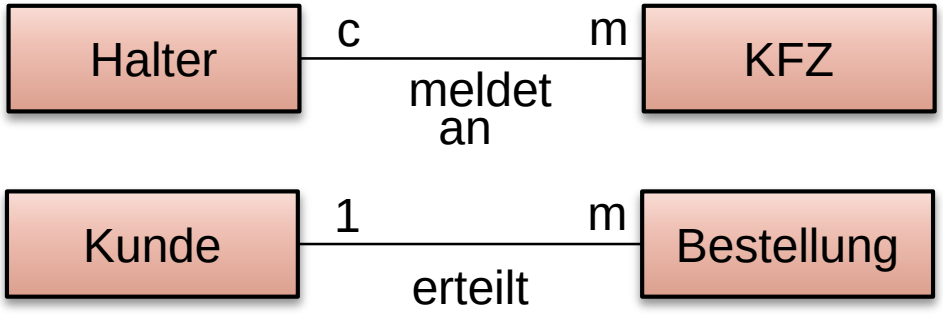
(2) **Darf ein Tupel gelöscht werden, auf den sich ein Fremdschlüssel bezieht?**

- Wie sollte die Datenbank reagieren?

(3) **Darf ein Tupel geändert werden, auf den sich ein Fremdschlüssel bezieht?**

- Wie sollte die Datenbank reagieren?

Frage 1: Nullwerte erlaubt?



- ERM gibt die Antwort vor!
  - Bei c Beziehung: Nullwerte sind immer zuzulassen
  - Bei 1 Beziehung: Nullwerte sind immer verboten
- Zu beachten:
  - Für Primärschlüssel gilt immer NOT NULL

Frage 2: Löschen eines Tupel

- **Fremdschlüsselbedingungen in SQL**
  - ON DELETE **NO ACTION**
  - ON DELETE **SET NULL**
  - ON DELETE **CASCADE**
- **Funktionsweise**
  - Wird ein Tupel gelöscht, auf den ein Fremdschlüssel mit obiger Bedingung verweist, dann
    - wird das Löschen dieses Tupel verhindert (Nichtstun, No Action)
    - wird der darauf verweisende Fremdschlüssel auf Null gesetzt (Set Null)
    - wird auch das den Fremdschlüssel enthaltene Tupel gelöscht (Cascade)

Frage 3: Ändern des Primärschlüssels

- **Analog wird das Ändern eines Primärschlüssel behandelt:**
  - ON UPDATE **NO ACTION**
  - ON UPDATE **SET NULL**
  - ON UPDATE **CASCADE**
- **Funktionsweise**
  - Wird ein Primärschlüsselwert geändert, auf den ein Fremdschlüssel mit obiger Bedingung verweist, dann
    - wird das Ändern dieses Tupel verhindert (Nichtstun, No Action)
    - wird der darauf verweisende Fremdschlüssel auf Null gesetzt (Set Null)
    - wird der Fremdschlüsselwert mit geändert (Cascade)

Hinweise zu Frage 2 und 3

- **m zu 1** und **c zu 1** Beziehungen:
  - ON DELETE **SET NULL** ist nicht erlaubt!
  - ON UPDATE **SET NULL** ist nicht erlaubt!
- Es gelten die Hinweise zum kaskadierenden Löschen aus Kapitel 2
- Wir fügen zu jedem Fremdschlüssel eine **ON DELETE** Eigenschaft hinzu
- Wir fügen zu jedem Fremdschlüssel eine **ON UPDATE** Eigenschaft hinzu
- **ON UPDATE CASCADE** wird grundsätzlich empfohlen

Relation Verknuepfung (vollständig)

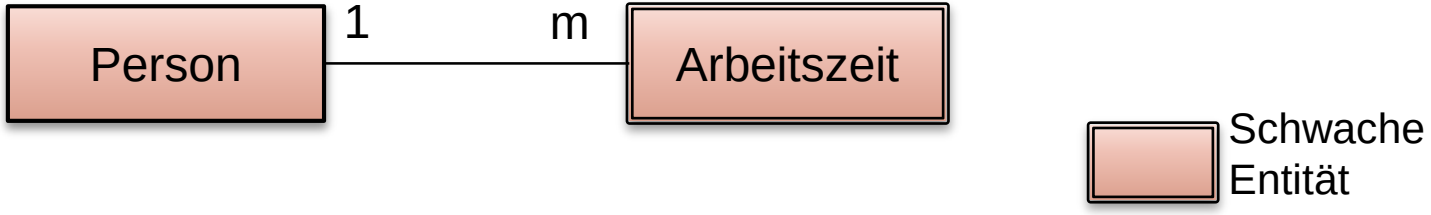
```
CREATE TABLE  Verknuepfung
(  VerkNr  CHARACTER(4)   REFERENCES  Verkaeufer
                          ON DELETE NO ACTION
                          ON UPDATE CASCADE,
   ProdNr  CHARACTER(4)   REFERENCES  Produkt
                          ON DELETE NO ACTION
                          ON UPDATE CASCADE,
   Umsatz  INTEGER,
   PRIMARY KEY  (VerkNr, ProdNr)
) ;
```

Relation KFZ (vollständig)

```
CREATE TABLE KFZ
(  KFZNr      INTEGER,
   MotorNr    INTEGER    REFERENCES Motor
                         ON DELETE SET NULL
                         ON UPDATE CASCADE,
   PRIMARY KEY (KFZNr),
   UNIQUE (MotorNr),
   . . .   ) ;
```

Falls Motor kaputt und entsorgt wird, so wird automatisch die MotorNr auf NULL gesetzt!

Schwache Entität



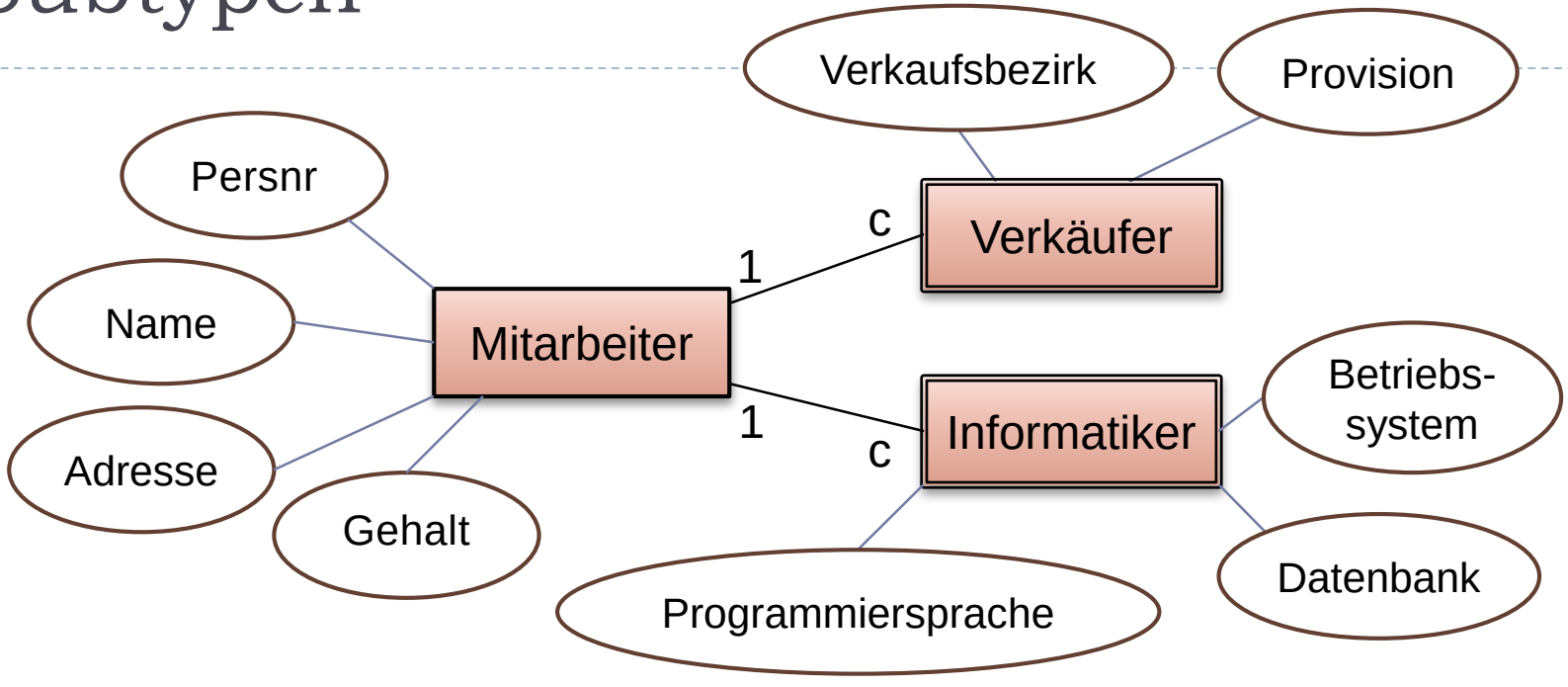
- Arbeitszeit ist vollständig abhängig von der Person
- Also: Arbeitszeit ist schwache Entität gegenüber Person
- Für schwache Entitäten gilt generell:
  - NOT NULL
  - ON DELETE CASCADE
  - ON UPDATE CASCADE

Definition (Schwache Entität)

- Eine Entität heißt schwach, wenn für die dazugehörige Relation R gilt:
  - R enthält genau einen Fremdschlüssel mit den drei Eigenschaften Not Null, On Delete Cascade und On Update Cascade.
  - Auf R verweist kein Fremdschlüssel.

- Eine Relation, auf die Fremdschlüssel verweisen, ist nicht schwach!

Subtypen



- Verkäufer und Informatiker sind **Subtypen** zu Mitarbeiter
- Subtypen sind **schwach** und enthalten Zusatzinformationen
- Es liegen **1 zu c Beziehungen** vor!

Subtyp Verkaeufer

```
CREATE TABLE Verkaeufer
(  PersNr    INTEGER    REFERENCES Mitarbeiter
                        ON DELETE CASCADE
                        ON UPDATE CASCADE,
   PRIMARY KEY (PersNr),
   . . . ) ;
```

Primärschlüssel ist auch Fremdschlüssel

NOT NULL
ON DELETE CASCADE
ON UPDATE CASCADE
Keine weiteren Fremdschlüssel
→ schwache Entität!

Zusammenfassung

- Bestimmen aller Entitäten mit ihren Eigenschaften.
- Ermittlung der Beziehungen zwischen den einzelnen Entitäten.
- Überführung der Entitäten in Relationen und der Eigenschaften in die Attribute dieser Relationen.
- Normalformen beachten!
- Überführung der m zu n Beziehungen in Beziehungsrelationen und aller anderen Beziehungen in Fremdschlüssel.
- Ermittlung der Eigenschaften der Fremdschlüssel. Hier helfen Stichworte wie *schwache Entität* und *Subtyp* weiter.

Datenbanken und SQL

Kapitel 4

Die Datenbankzugriffssprache SQL

Die Datenbankzugriffssprache SQL

- Der Select-Befehl
  - Der Hauptteil des Select-Befehls
    - From-, Where-, Select-Klausel
    - Group By und Having
    - Join
  - Union, Except, Intersect
  - Order By
  - Die Arbeitsweise des Select-Befehls
- Delete, Update, Insert
- Transaktionsbetrieb
- Relationale Algebra

Überblick (1)

- SELECT * FROM Personal ;

Persnr	Name	Ort	Vorgesetzt	Gehalt
1	Maria Forster	Regensburg	NULL	4800.00
2	Anna Kraus	Regensburg	1	2300.00
3	Ursula Rank	Frankfurt	6	2700.00
4	Heinz Rolle	Nürnberg	1	3300.00
5	Johanna Köster	Nürnberg	1	2100.00
6	Marianne Lambert	Landshut	NULL	4100.00
7	Thomas Noster	Regensburg	6	2500.00
8	Renate Wolters	Augsburg	1	3300.00
9	Ernst Pach	Stuttgart	6	800.00

Überblick (2)

```
SELECT  Name, Ort
FROM    Personal ;
```

Name	Ort
Maria Forster	Regensburg
Anna Kraus	Regensburg
Ursula Rank	Frankfurt
Heinz Rolle	Nürnberg
Johanna Köster	Nürnberg
Marianne Lambert	Landshut
Thomas Noster	Regensburg
Renate Wolters	Augsburg
Ernst Pach	Stuttgart

```
SeleCT name
, orT        fROM
                Personal ;
```

Die Syntax des SELECT Befehls

Select-Befehl:

Select-Hauptteil

[{ UNION [ALL] | EXCEPT | INTERSECT }

Select-Hauptteil

]

[…]

[ORDER BY Ordnungsliste]

SELECT Befehl: Hauptteil

Select-Hauptteil:

```
SELECT     [ ALL | DISTINCT ] Spaltenauswahlliste
 FROM      Tabellenliste
 [ WHERE           Bedingung ]
 [ GROUP BY        Spaltenliste
    [ HAVING       Bedingung ] ]
```

Hinweise zur Syntax

- GROSSBUCHSTABEN : Reservierte Bezeichner
- [xyz] : Wahlfrei
- { xx | yy | zz } : Auswahlliste
- [xx | yy | zz] : Wahlfreie Auswahlliste
- a2, a_a, aaa, ab___33 : Erlaubte Bezeichner
  Erstes Zeichen ist Buchstabe
  Weitere Zeichen: Buchstabe / Ziffer / ‚_‘
- … : Wiederholzeichen

Vergleich: SQL – Relationale Algebra

	Algebra	SQL
Vereinigung	$R1 \cup R2$	UNION
Schnitt	$R1 \cap R2$	INTERSECT
Differenz	$R1 \setminus R2$	EXCEPT
Kreuzprodukt	$R1 \times R2$	Tabellenliste
Restriktion	$\sigma_{Bedingung}(R)$	WHERE - Klausel
Projektion	$\pi_{Auswahl}(R)$	SELECT - Klausel
Verbund	$R1 \bowtie R2$	Tabellenliste
Division	$R1 \div R2$	---
Umbenennung	$\rho_{Rneu}(R)$	Spaltenauswahlliste, Tabellenliste

Die FROM Klausel

Tabellenliste:

Tabellenreferenz [, ...]

Tabellenreferenz:

Tabellenname [[AS] Aliasname]
| (Select-Hauptteil) [[AS] Aliasname]
| (Tabellenreferenz) [[AS] Aliasname]
| Joinausdruck [[AS] Aliasname]

Die FROM Klausel: Beispiele

- SELECT * FROM Personal AS P ;
- SELECT * FROM (SELECT * FROM Personal) AS P2 ;
- SELECT * FROM (Personal) ;
- SELECT * FROM Personal, Auftrag ;

Füllwort AS in Oracle nicht erlaubt

In SQL Server: Klammern hier nicht erlaubt

Aliasname hier in MySQL und SQL Server zwingend

Kreuzprodukt

Kreuzprodukt

SELECT * FROM Personal, Auftrag ;

Persnr	Name	Ort	Vorg.	Gehalt	A.Nr	Datum	K.nr	Persnr
1	Maria Forster	Regensburg	NULL	4800.00	1	04.01.13	1	2
1	Maria Forster	Regensburg	NULL	4800.00	2	06.01.13	3	5
1	Maria Forster	Regensburg	NULL	4800.00	3	07.01.13	4	2
1	Maria Forster	Regensburg	NULL	4800.00	4	18.01.13	6	5
1	Maria Forster	Regensburg	NULL	4800.00	5	06.02.13	1	2
2	Anna Kraus	Regensburg	1	2300.00	1	04.01.13	1	2
2	Anna Kraus	Regensburg	1	2300.00	2	06.01.13	3	5
2	Anna Kraus	Regensburg	1	2300.00	3	07.01.13	4	2
2	Anna Kraus	Regensburg	1	2300.00	4	18.01.13	6	5
2	Anna Kraus	Regensburg	1	2300.00	5	06.02.13	1	2
3	Ursula Rank	Frankfurt	6	2700.00	1	04.01.13	1	2
3	Ursula Rank	Frankfurt	6	2700.00	2	06.01.13	3	5
3	Ursula Rank	Frankfurt	6	2700.00	3	07.01.13	4	2
…	…	…	…	…	…	…	…	…

Die SELECT Klausel

SELECT Klausel:

SELECT [ALL | DISTINCT] Spaltenauswahlliste

Spaltenauswahlliste:

Spaltenausdruck [[AS] Aliasname] [, ...]

Füllwort AS hier auch in Oracle erlaubt

```
SELECT Name, 12 * Gehalt AS Jahresgehalt
 FROM  Personal ;
```

$\rho_{12*Gehalt->Jahresgehalt}(\pi_{Name,12*Gehalt}(Personal))$

Vergleich: SQL – Rel. Algebra

Qualifizieren von Attributen

```
SELECT  *
 FROM   Personal, Auftrag ;
```

Alle Attribute, Reihenfolge abhängig von From-Klausel

```
SELECT  Personal.*, Auftrag.*
 FROM   Personal, Auftrag ;
```

Erst alle Personalattribute, dann alle Auftragsattribute

```
SELECT  Personal.Persnr, Name, Ort,  Vorgesetzt, Gehalt,
          AuftrNr, Datum, Kundnr, Auftrag.Persnr
 FROM   Personal, Auftrag ;
```

Reihenfolge der Attribute angegeben

Skalare Funktionen (Auswahl)

UPPER	Wandelt Kleinbuchstaben in Großbuchstaben um. Andere Zeichen bleiben unverändert.
LOWER	Wandelt Großbuchstaben in Kleinbuchstaben um. Andere Zeichen bleiben unverändert.
TRIM	Führende und schließende Leerzeichen werden entfernt
RTRIM	Schließende Leerzeichen werden entfernt
SUBSTRING	Aus einer Zeichenkette wird eine Teilzeichenkette extrahiert

Syntax: UPPER(String) analog LOWER…

SUBSTRING(String FROM Pos FOR Anzahl)

In SQL Server: String, Pos, Anzahl

In Oracle: SUBSTR(String, Pos, Anzahl)

Skalare Funktionen (Beispiele)

```
SELECT UPPER(Name), LOWER(Name), Name
FROM Personal ;
```

Ausgabe des Namens in Klein- und Großbuchstaben

```
SELECT Name || '.' , TRIM(Name) || '.'
FROM Personal ;
```

In ANSI SQL: Konstante Zeichenketten in Hochkommata

In Oracle und ANSI SQL: Konkatenierungsoperator ||

Ausgabe: Leerzeichen zwischen Name und Punkt beachten!

```
SELECT SUBSTRING(TRIM(Name) FROM 1 FOR
        POSITION(' ' IN TRIM(Name)))
FROM Personal ;
```

In Oracle: SUBSTR(…,…,…)

Position des ersten Vorkommens eines Zeichens in einer Zeichenkette, in Oracle: InStr(Trim(Name),' ')

Ausgabe: Nur Vornamen!

Aggregatfunktionen in SQL

AVG	Average	Mittelwert, ermittelt über alle Zeilen
COUNT	Count	Anzahl aller Zeilen
MAX	Maximum	Maximalwert aller Zeilen
MIN	Minimum	Minimalwert aller Zeilen
SUM	Sum	Summenwert, summiert über alle Zeilen

```
SELECT  Persnr, Name, 12*Gehalt + 1000 *(6 - Beurteilung) AS Jahresgehalt
 FROM   Personal ;
```

```
SELECT  SUM (12*Gehalt +1000 *(6- Beurteilung)) AS Jahrespersonalkosten
 FROM   Personal ;
```

In MySQL: hier kein Leerzeichen

Der Bezeichner DISTINCT

```
SELECT Ort FROM Personal ;
SELECT DISTINCT Ort FROM Personal ;
SELECT COUNT ( Ort ) FROM Personal ;
SELECT COUNT ( DISTINCT Ort ) FROM Personal ;

SELECT COUNT(*) FROM Personal ;
SELECT COUNT( Vorgesetzt ) FROM Personal ;
SELECT COUNT( DISTINCT Vorgesetzt ) FROM Personal ;
```

9 Orte

6 unterschiedliche Orte

9

6

9, da 9 Tupel

7, da 2 Nullwerte

2, da nur 2 unterschiedliche Werte

Die WHERE Klausel

```
SELECT   MIN( Gehalt )
 FROM    Personal
 WHERE   Gehalt > 3000 ;
```

Kleinstes Gehalt größer 3000

nicht: !=

Boolesche Operatoren	NOT , AND , OR
Vergleichsoperatoren	< , <= , > , >= , = , <>
Intervalloperator	[NOT] BETWEEN ... AND
Enthaltenoperator	[NOT] IN
Ähnlichkeitsoperator	[NOT] LIKE
Nulloperator	IS [NOT] NULL
Auswahloperatoren	ALL , ANY , SOME
Existenzoperator	EXISTS

Intervalloperator

A BETWEEN B AND C  A >= B AND A <= C

A NOT BETWEEN B AND C  NOT A BETWEEN B AND C

- **Beispiel:**

```
SELECT   *
 FROM    Personal
 WHERE   Gehalt BETWEEN 2000 AND 3000 ;
```

Enthaltenoperator

A IN (B_1, B_2, …, B_n) ⟺ A=B_1 OR A=B_2 OR … OR A=B_n

A NOT IN (B_1, B_2, …, B_n) ⟺ NOT A IN (B_1, B_2, …, B_n)

- **Beispiel:**

```
SELECT   *
 FROM    Personal
WHERE    Ort IN ('Regensburg', 'Nürnberg', 'Passau') ;
```

Ähnlichkeitsoperator LIKE

Wildcardsymbole	in SQL	in Windows in Unix
Beliebig viele Zeichen (0..n)	%	*
Genau ein Zeichen (1..1)	_	?

- Beispiele:

```
SELECT    *
 FROM     Personal
 WHERE    Name LIKE '%Heinz%' ;

SELECT     *
 FROM      Personal
 WHERE     Upper(Name) LIKE  '%A_E%'  ;
```

Nulloperator

A IS NULL — true, falls A gleich Null ist

A IS NOT NULL  NOT A IS NULL

- Beispiel:

```
SELECT   *
 FROM    Personal
WHERE   Vorgesetzt IS NULL ;
```

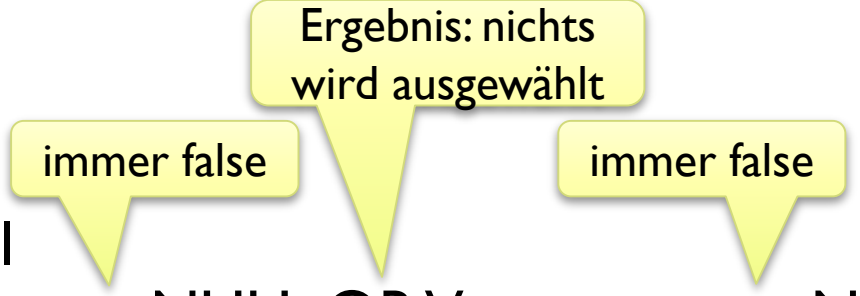
- nicht: WHERE Vorgesetzt = NULL

Einschub: NULL

- **Wichtig:** Alle Nullwerte unterscheiden sich voneinander
- Informell gilt also:

  NULL ist ungleich NULL

- Jeder Vergleich mit Nullwerten liefert FALSE zurück!
- Folgerung: Wir verwenden den Nulloperator **Is Null**
- **Beispiel:**

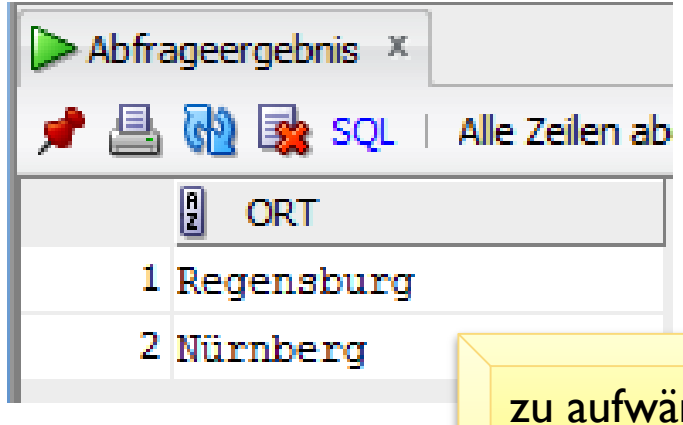



```
SELECT   *
FROM     Personal
WHERE    Vorgesetzt = NULL OR Vorgesetzt <> NULL
```

Unterabfragen (1)

- **Anfrage:** Gesucht sind alle Mitarbeiter, die in den gleichen Orten wie Mitarbeiterinnen 2 und 5 wohnen.
- **Lösung 1:**

```
SELECT  Ort
 FROM   Personal
 WHERE Persnr IN (2, 5) ;

SELECT  *
 FROM   Personal
 WHERE Ort IN ( 'Nürnberg', 'Regensburg' );
```



zu aufwändig bei großen Zwischen-ergebnissen

Unterabfragen (2)

- **Anfrage:** Gesucht sind alle Mitarbeiter, die in den gleichen Orten wie Mitarbeiterinnen 2 und 5 wohnen.
- **Lösung 2:**

```
SELECT  *
 FROM   Personal
 WHERE Ort IN ( SELECT  Ort
                FROM    Personal
                WHERE  Persnr IN ( 2, 5 ) ) ;
```

Unterabfrage: liefert alle Orte von Mitarbeiter 2 und 5

Unterabfrage steht in Klammern

Globaler Attributname bezieht sich auf globale Relation

Lokale Attributnamen beziehen sich auf lokale Relation

Auswahloperatoren

Ausdruck op ANY (Unterabfrage)
Ausdruck op SOME (Unterabfrage)
Ausdruck op ALL (Unterabfrage)

Any ≙ Some:
True, falls mindestens eine Übereinstimmung

All:
True, falls alle übereinstimmen

Vergleichsoperator:
<, <=, >, >=, =, <>

- Beispiel:

```
SELECT  *
 FROM   Personal
 WHERE Ort = ANY ( SELECT  Ort
                   FROM    Personal
                   WHERE  Persnr IN ( 2, 5 ) ) ;
```

„=ANY“ ≙ „IN“

Auswahloperatoren: Beispiele

```
SELECT  *
 FROM   Personal
WHERE   Gehalt >= ALL ( SELECT Gehalt
                        FROM Personal  ) ;
```

Ergebnis: Mitarbeiter mit maximalem Gehalt

```
SELECT  *
 FROM   Personal
WHERE   Gehalt = MAX( Gehalt ) ;
```

Fehler: Aggregatfunktion direkt in Where Klausel nicht möglich!

```
SELECT  *
 FROM   Personal
WHERE   Gehalt IN ( SELECT MAX(Gehalt)
                    FROM Personal  ) ;
```

Hier ist auch „=“ erlaubt

Korrekt: Aggregatfunktion in Unterabfrage

Unterabfragen oder Kreuzprodukt?

- **Gesucht:** Alle Mitarbeiter, die weniger als Mitarbeiter 3 verdienen

```
SELECT  *
 FROM   Personal
WHERE   Gehalt < ( SELECT Gehalt
                   FROM Personal
                   WHERE Persnr = 3 ) ;
```

Hier erlaubt: Unterabfrage liefert nur einen Wert

Gehalt von Mitarbeiter 3

```
SELECT  P1.*
 FROM   Personal AS P1, Personal AS P2
WHERE   P1.Gehalt < P2.Gehalt
  AND   P2.Persnr = 3 ;
```

Kreuzprodukt enthält 9 Tupel

P2.Gehalt: Gehalt von Mitarbeiter 3

Restriktion: P2 enthält nur ein Tupel!

Existenzoperator

- **EXISTS** liefert True, wenn die folgende Unterabfrage mindestens ein Ergebnis liefert.
- **Beispiel** (siehe vorherige Folie):

```
SELECT   *
 FROM    Personal  AS P1
 WHERE  EXISTS (  SELECT  *
                  FROM    Personal
                  WHERE   Persnr = 3
                    AND   Gehalt > P1.Gehalt ) ;
```

Wenn P1.Gehalt < 2700, dann liefert Unterabfrage Ergebnis

Ändert sich von Zeile zu Zeile: 4800, 2300, 2700, …

Nur Mitarbeiter 3

Also: Gehalt von Mitarbeiter 3: 2700

Die GROUP BY Klausel

- GROUP BY Spaltenliste
- Spaltenliste:
  - Liste von Spaltennamen (Ausdrücke sind nicht erlaubt!)
- Beispiel:
  - Gesucht: Alle Wohnorte der Mitarbeiter, jeder Wohnort soll nur einmal aufgeführt werden

```
SELECT DISTINCT Ort
FROM   Personal ;
```

```
SELECT  Ort
FROM    Personal
GROUP BY Ort ;
```

Aggregatfunktionen im GROUP BY

```
SELECT      Ort, COUNT (*) AS Anzahl
 FROM       Personal
GROUP  BY   Ort ;
```

Ort	Anzahl
Regensburg	3
Frankfurt	1
Nürnberg	2
Landshut	1
Augsburg	1
Stuttgart	1

Zu beachten im GROUP BY

- In Select-Klausel gleichzeitig erlaubt:
  - Attributsnamen und Aggregatfunktionen
- Die Aggregatfunktionen wirken auf die gruppierten Tupel
- Group-By-Klausel enthält mindestens:
  - alle Attribute, die in Select-Klausel außerhalb der Aggregat-funktionen vorkommen.
- Fehler:

```
SELECT Name, Ort, Count(*) AS Anzahl
FROM Personal
GROUP BY Ort ;
```

Ein Ort kann viele Mitarbeiter haben.
Welcher wird hier angegeben???

Die HAVING Klausel

- **Where Klausel:** Restriktion vor der Gruppierung
- **Having Klausel:** Restriktion nach der Gruppierung
- **Beispiel:** Alle Wohnorte mit mehreren Mitarbeitern

```
SELECT    Ort, COUNT (*) As Anzahl,
          SUM(12*Gehalt) AS Jahresgehalt,
          12 * MAX(Gehalt) AS MaxJahresgehalt
FROM      Personal
GROUP BY  Ort
HAVING    COUNT(*) > 1 ;
```

SQL | Alle Zeilen abgerufen: 2 in 0 Sekunden

ORT	ANZAHL	JAHRESGEHALT	MAXJAHRESGEHALT
Nürnberg	2	64800	39600
Regensburg	3	115200	57600

HAVING ist Restriktion nach Group By

- **Alternative Lösung zum letzten Beispiel:**

```
SELECT   *
 FROM    ( SELECT  Ort, COUNT (*) As Anzahl,
                   SUM (12*Gehalt) As Jahresgehalt,
                   12 * MAX (Gehalt) As MaxJahresgehalt
             FROM   Personal
             GROUP BY Ort        )     AS Zwischentabelle
WHERE Anzahl > 1 ;
```

Restriktion nach Group By: Entspricht Having

In SQL Server, MySQL: Aliasname ist zwingend

Zwischentabelle liefert alle Orte

Beispiel mit SELECT in FROM Klausel

- **Gesucht:**
  - Mittleres Auftragsvolumen über alle Aufträge

Hauptabfrage bildet daraus den Mittelwert

```
SELECT   'Mittleres Auftragsvolumen: ', AVG( Auftragsvolumen )
 FROM    ( SELECT     SUM( Gesamtpreis ) As Auftragsvolumen
           FROM       Auftragsposten
           GROUP BY Auftrnr              )   AS Auftragspreis;
```

Unterabfrage berechnet das Auftragsvolumen jedes einzelnen Auftrags aus der Summe der Auftragspositionen

Der UNION Operator

- SELECT-Hauptteil UNION SELECT-Hauptteil
- R1 $\cup$ R2
- Beide Hauptteile müssen zueinander kompatibel sein:
  - Gleiche Anzahl der Attribute
  - Die einzelnen Attribute müssen typverträglich sein
- Beispiel (Alle Wohnorte von Kunden und Mitarbeiter):

```
SELECT Ort  FROM Personal
UNION
SELECT Ort  FROM Kunde;
```

$\pi_{Ort}(Personal) \cup \pi_{Ort}(Kunde)$

UNION ALL

- Union liefert eindeutige Ergebnisse:

```
SELECT Ort  FROM Personal
UNION
SELECT Ort  FROM Kunde;
```

9 Mitarbeter

10 Tupel, da doppelte Einträge

6 Kunden

- Union All fasst doppelte Tupel nicht zusammen:

```
SELECT Ort  FROM Personal
UNION ALL
SELECT Ort  FROM Kunde;
```

9+6=15 Tupel

Der Verbund (Join)

Joinausdruck:

Tabellenreferenz
{ [NATURAL] [INNER]
| [NATURAL] { LEFT | RIGHT | FULL } [OUTER]
} JOIN Tabellenreferenz
[ON Bedingung | USING (Spaltenliste)]

- Genau eine der folgenden 3 Angaben muss vorkommen:
  NATURAL | ON | USING
- **Einfaches Beispiel:** R1 NATURAL JOIN R2

Der JOIN Operator

- R1 $\bowtie$ R2 :

  R1 NATURAL INNER JOIN R2

- R1 $\bowtie_{\text{Bedingung}}$ R2 :

  R1 INNER JOIN R2 ON Bedingung

- Weitere Möglichkeit:
- R1 INNER JOIN R2 USING (Spaltenliste)

Natürlicher Verbund: Ein Beispiel

Auftrnr	Datum	Kundnr	Persnr
1	04.01.2013	1	2
2	06.01.2013	3	5
3	07.01.2013	4	2
4	18.01.2013	6	5
5	03.02.2013	1	2

- Verbund über Persnr
- Nur Persnr 2 und 5 bleiben übrig

Persnr	Name	Ort	Vorgesetzt	Gehalt
1	Maria Forster	Regensburg	NULL	4800.00
2	Anna Kraus	Regensburg	1	2300.00
3	Ursula Rank	Frankfurt	6	2700.00
4	Heinz Rolle	Nürnberg	1	3300.00
5	Johanna Köster	Nürnberg	1	2100.00
6	Marianne Lambert	Landshut	NULL	4100.00
7	Thomas Noster	Regensburg	6	2500.00
8	Renate Wolters	Augsburg	1	3300.00
9	Ernst Pach	Stuttgart	6	800.00

Natürlicher Verbund (Auftrag⋈Personal)

```
SELECT  *
 FROM   Auftrag NATURAL INNER JOIN Personal ;
```

AuftrNr	Datum	Kundnr	Persnr	Name	Vorgesetzt	Gehalt	Ort
1	04.01.13	1	2	Anna Kraus	1	3400.00	Regensburg
2	06.01.13	3	5	Joh. Köster	1	3200.00	Nürnberg
3	07.01.13	4	2	Anna Kraus	1	3400.00	Regensburg
4	18.01.13	6	5	Joh. Köster	1	3200.00	Nürnberg
5	06.02.13	1	2	Anna Kraus	1	3400.00	Regensburg

Vergleich der verschiedenen Joins

```
SELECT  *
 FROM   Auftrag NATURAL INNER JOIN Personal ;
```

kein Qualifizieren möglich!
z.B. Auftrag.Datum ist Fehler

```
SELECT  *
 FROM   Auftrag INNER JOIN Personal USING ( Persnr ) ;
```

Nur Qualifizieren der Persnr ist verboten

```
SELECT  Auftrnr, Datum, Kundnr, Personal.*
FROM    Auftrag INNER JOIN Personal
                ON Auftrag.Persnr = Personal.Persnr ;
```

Qualifizieren ist beliebig möglich

Alle drei Befehle liefern die gleiche Ausgabe

Nachbilden des Verbunds

- **Aus der relationalen Algebra:**
- $R1 \bowtie R2 = \pi_{R1.X,R1.Y,R2.Z}(\sigma_{R1.Y=R2.Y}(R1 \times R2))$
  - mit $R1 = R1(X,Y)$, $R2 = R2(Y,Z)$
  - und Y ist gemeinsames Attribut
- $Auftrag \bowtie Personal =$

$$\pi_{\cancel{Auftrag.Persnr}}(\sigma_{Auftrag.Persnr=Personal.Persnr}(Auftrag \times Personal))$$

```
SELECT   Auftrnr, Datum, Kundnr, Personal.*
 FROM    Personal, Auftrag
 WHERE   Personal.Persnr = Auftrag.Persnr ;
```

Gleiche Ausgabe wie die drei vorherigen Befehle

Beispiel zum Verbund (1)

- Gesucht: Alle Mitarbeiter und die Anzahl der Verkäufe
- 1. Versuch:

```
SELECT     Persnr, Name, COUNT (*) As AnzahlAuftrag
 FROM      Personal NATURAL INNER JOIN Auftrag
 GROUP BY  Persnr, Name ;
```

Persnr	Name	AnzahlAuftrag
2	Anna Kraus	3
5	Johanna Köster	2

- Aber: 7 Mitarbeiter fehlen!

Beispiel zum Verbund (2)

- **Versuch 2:**

```
SELECT    Persnr, Name, COUNT(AuftrNr) AS AnzahlAuftrag
 FROM     Personal NATURAL LEFT OUTER JOIN Auftrag
 GROUP BY Persnr, Name ;
```

Count(*) wäre falsch

Persnr	Name	AnzahlAuftrag
2	Anna Kraus	3
5	Johanna Köster	2
1	Maria Forster	0
3	Ursula Rank	0
4	Heinz Rolle	0
6	Marianne Lambert	0
7	Thomas Noster	0
8	Renate Wolters	0
9	Ernst Pach	0

Bei Count(*): 1

Beispiel zum Verbund (3)

- **Lösung ohne Outer Join:**

Count(*) korrekt

```
SELECT      Persnr, Name, COUNT (*) AS AnzahlAuftrag
 FROM       Personal NATURAL INNER JOIN Auftrag
 GROUP BY   Persnr, Name
UNION
SELECT      Persnr, Name, 0
 FROM       Personal
 WHERE      Persnr NOT IN ( SELECT  Persnr
                            FROM    Auftrag  ) ;
```

Alle anderen haben nichts verkauft

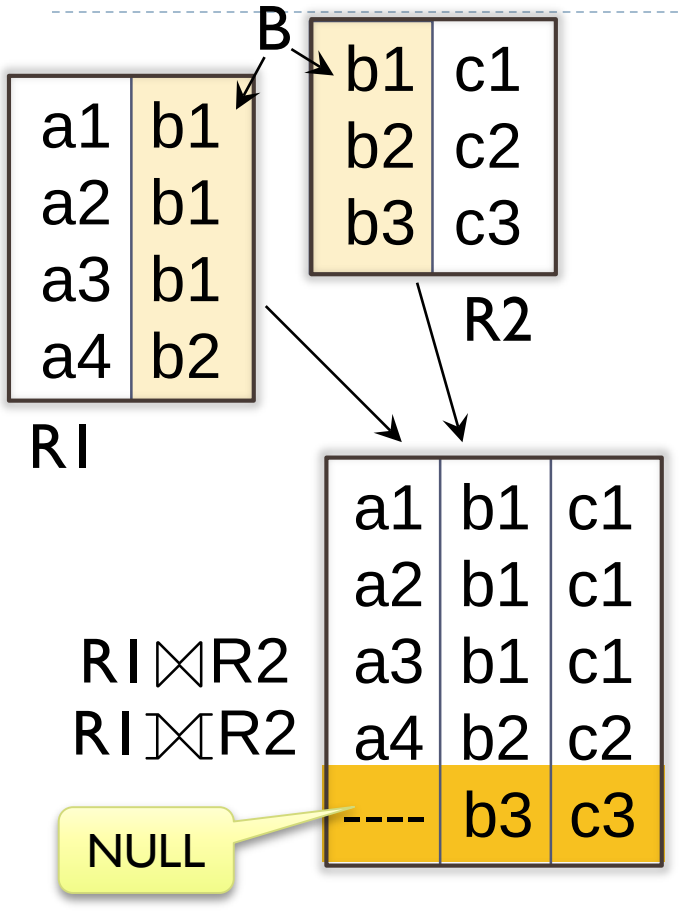
Beispiel zum Verbund (4)

- In der Praxis:
- Der Bezeichner NATURAL sollte vermieden werden, da
  - Fehler im Optimizer von Oracle
  - nicht unterstützt in SQL Server
  - das spätere Hinzufügen von Attributen falsche Ergebnisse liefern kann
- Lösung ohne Bezeichner NATURAL:

```
SELECT    P.Persnr, Name, COUNT(AuftrNr) AS AnzahlAuftrag
FROM      Personal AS P LEFT OUTER JOIN Auftrag As A
                        ON P.Persnr = A.Persnr
GROUP BY P.Persnr, Name ;
```

ohne Natural: Aliasname möglich

Der äußere Verbund



- Der innere Verbund:
  - Enthält alle Elemente, deren Elemente aus B in beiden Relationen vorkommen (Schnitt)
- Der äußere Verbund:
  - Enthält alle weiteren Elemente
  - R1 Left Outer Join R2: + R1
  - R1 Right Outer Join R2: + R2
  - R1 Full Outer Join R2: + R1 + R2
- Nicht vorhandene Werte mit Null auffüllen

Der äußere Verbund: OUTER JOIN

- R1 Left Outer Join R2 $R1 ⟕ R2$
  - R1 Inner Join R2 vereinigt mit allen weiteren Tupel von R1
- R1 Right Outer Join R2 $R1 ⟖ R2$
  - R1 Inner Join R2 vereinigt mit allen weiteren Tupel von R2
- R1 Full Outer Join R2 $R1 ⟗ R2$
  - R1 Left Outer Join R2 Union R1 Right Outer Join R2

- Nicht vorhandene Werte werden mit NULL aufgefüllt.

Die ORDER BY Klausel

- **Sortieren der Tupel eines Select-Befehls**

```
SELECT    Ort, COUNT (*) AS Anzahl
FROM      Personal
GROUP BY  Ort
ORDER BY  Anzahl DESC, Ort ;
```

Absteigendes Sortieren

Nur Namen sind erlaubt …

- **oder:**

```
SELECT    Ort, COUNT (*) AS Anzahl
FROM      Personal
GROUP BY  Ort
ORDER BY  2 DESC, 1 ;
```

… oder Zahlen

Nullwerte sind unterschiedlich

- Ein Vergleich mit Nullwerten liefert immer FALSE !
- Dies ist auch vernünftig. Beispiel:
  - Gesucht: Alle Mitarbeiter, die den gleichen Vorgesetzten haben wie Mitarbeiter 1 (oder 2..9)

```
SELECT   Name, Gehalt, Vorgesetzt
FROM     Personal
WHERE    Vorgesetzt = ( SELECT Vorgesetzt FROM Personal
                        WHERE Persnr = 1 ) ;
```

Unterabfrage liefert NULL

Vergleich liefert false:
Kein Mitarbeiter (auch nicht 6) hat den gleichen Vorgesetzten, und das ist korrekt!

Befehl liefert für 1..9 immer korrektes Ergebnis!

Funktion Coalesce

- Nullwert in einem Ausdruck liefert als Ergebnis NULL
- Nullwerte werden in Aggregatfunktionen nicht mitgezählt
- Falsch:

```
SELECT  SUM (12 * Gehalt +1000 * ( 6- Beurteilung))
                                        AS Jahrespersonalkosten
 FROM   Personal ;
```

Mitarbeiter 9 hat Beurteilung NULL

Also wird auch sein Gehalt verworfen

- Korrekt:

```
SELECT  SUM(12*Gehalt+COALESCE(1000 *( 6- Beurteilung),
                                1000) ) AS Jahrespersonalkosten
 FROM   Personal ;
```

falls NULL …

… dann Ersatzwert

Funktion Coalesce: Komplexes Beispiel

- Beim äußeren Verbund ist COALESCE sehr wichtig!
- Gesucht: Alle Mitarbeiter mit Verkaufsumsatz
- Lösung:

```
SELECT      Persnr, Name,
              COALESCE( SUM(Gesamtpreis), 0) AS Summe
 FROM       Personal NATURAL LEFT OUTER JOIN
              (Auftrag NATURAL INNER JOIN Auftragsposten)
GROUP BY    Persnr, Name ;
```

Vermeidet Null-Ausgaben

… dann Outer Join

Inner Join über Auftrnr …

Arbeitsweise des SELECT Befehls

1	Kreuzprodukt, Verbund: Alle in der Tabellenliste angegebenen Relationen werden miteinander verknüpft.
2	Restriktion: Aus dieser verknüpften Relation werden die Tupel ausgewählt, die die angegebene WHERE-Bedingung erfüllen.
3	Projektion: Mittels der Spaltenauswahlliste werden die angegebenen Spalten ausgewählt.
4	Gruppierung: Jeweils gleiche Tupel werden zusammengefasst. Angegebene Aggregierungen werden durchgeführt.
5	2. Restriktion: Nach der Gruppierung werden die Tupel gewählt, die die angegebene HAVING-Bedingung erfüllen.
6	Vereinigung, Schnitt, Differenz: Alle in Schritt 1 bis 5 erstellten Hauptteile des Select-Befehls werden miteinander verknüpft.
7	Sortierung: Die Ergebnisrelation wird gemäß der Order-By-Klausel sortiert.

Mutationsbefehle in SQL

UPDATE	Ändert bestehende Einträge
INSERT	Fügt neue Tupel (Zeilen) ein
DELETE	Löscht bestehende Tupel (Zeilen)

Der DELETE Befehl

- Syntax:

DELETE FROM Tabellenname [[AS] Aliasname]
[WHERE Bedingung]

- Beispiele:

```
DELETE FROM  Personal
WHERE        Name = 'Ursula Rank' ;
```

Entfernt Mitarbeiterin Ursula Rank

```
DELETE FROM  Personal ;
```

Entfernt alle Mitarbeiter!

Der UPDATE Befehl

- Syntax:

```
UPDATE   Tabellenname  [ [ AS ]  Aliasname ]
 SET          Spalte  =  Spaltenausdruck  [ , ... ]
 [ WHERE  Bedingung ]
```

- Beispiel:

```
UPDATE  Personal
 SET        Gehalt = 1.05 * Gehalt
 WHERE   Gehalt < 3000 ;
```

Zuweisung erfolgt immer zuletzt!

Alle Mitarbeiter mit weniger als 3000 Euro Gehalt erhalten 5% Gehaltserhöhung

Bisheriges Gehalt

Der INSERT Befehl

- **Syntax:**

```
INSERT INTO Tabellenname [ ( Spaltenliste ) ]
  { VALUES ( Auswahlliste ) [, … ]
  | Select-Befehl }
```

- **Beispiel:**

In Oracle: nur eine Auswahlliste möglich

Alle anderen Attribute werden mit NULL gefüllt

```
INSERT INTO Personal (Persnr, Name, GebDatum, Ort)
 VALUES  (10, 'Lars Anger', DATE '1980-01-13', 'Augsburg') ,
         (11, 'Karl Meier', DATE '1983-05-15', 'Darmstadt') ;
```

Cast-Operator DATE

Der INSERT Befehl: SELECT Auswahl

- **Beispiel:**

```
INSERT INTO  Personal (Persnr, Name, GebDatum,
                                        Ort, Vorgesetzt, Gehalt)
 SELECT  10, 'Lars Anger', DATE '1980-01-13',
                                        'Augsburg', Vorgesetzt, Gehalt
 FROM     Personal
WHERE   Persnr = 7 ;
```

In SELECT Klausel sind auch konstante Werte erlaubt

- **Empfehlung:** Select-Auswahl dann, wenn mindestens ein Wert aus Datenbank gelesen werden soll

Transaktionsbetrieb (1)

- Transaktionen mit Mutationsbefehlen enorm wichtig
- Ablauf:

[BEGIN TRANSACTION ;]

Nur in SQL Server
Sonst: automatischer Transaktionsstart

Innerhalb einer Transaktion

COMMIT ; / ROLLBACK ;

Ende einer Transaktion

[BEGIN TRANSACTION ;]

Nur in SQL Server

Innerhalb einer Transaktion

COMMIT ; / ROLLBACK ;

Alle geänderten Daten werden zurückgesetzt

…

Alle geänderten Daten werden dauerhaft gespeichert

Transaktionsbetrieb (2)

- Transaktionsbetrieb ist wichtig für Konsistenz der Daten
- Beispiel: Überweisung von 1000 € von Konto A nach B

```
UPDATE Bank  SET Saldo = Saldo – 1000  WHERE Konto = 'A' ;

UPDATE Bank  SET Saldo = Saldo + 1000  WHERE Konto = 'B' ;

COMMIT ;
```

Hier: inkonsistenter Datenstand

Hier: konsistenter Datenstand

Commit nur, wenn Datenstand konsistent

Vergleich: Relationale Algebra – SQL

Operator	Algebra	SQL-2
Vereinigung	$R1 \cup R2$	SELECT * FROM R1 UNION SELECT * FROM R2
Kreuzprodukt	$R1 \times R2$	SELECT * FROM R1, R2
Restriktion	$\sigma_p(R1)$	SELECT * FROM R1 WHERE p
Projektion	$\pi_{x1,\ldots xn}(R1)$	SELECT x1, ... , xn FROM R1
Differenz	$R1 \setminus R2$	SELECT * FROM R1 EXCEPT SELECT * FROM R2
Verbund	$R1 \bowtie R2$	SELECT * FROM R1 NATURAL INNER JOIN R2
Schnitt	$R1 \cap R2$	SELECT * FROM R1 INTERSECT SELECT * FROM R2

Die Division in SQL

- Gesucht: Alle Lieferanten, die mindestens die gleichen Artikel wie Lieferant 3 liefern.
- Lösung 1: Nachbilden der Division (siehe rel. Algebra)
- Lösung 2: Idee
  - Bestimme alle Artikel des Lieferanten 3: Artikelliste3
  - Bestimme zu jedem Lieferanten alle Artikel aus Artikelliste3
  - Gib alle Lieferanten aus, deren Artikel aus Artikelliste3 genau so groß ist wie die Artikelliste3

Die Division in SQL: Lösung

```
SELECT    Liefnr
 FROM     Lieferung
 WHERE    Anr IN ( SELECT   Anr
                   FROM     Lieferung
                   WHERE    Liefnr = 3  )
 GROUP BY  Liefnr
 HAVING    COUNT(*) = ( SELECT   COUNT(*)
                        FROM     Lieferung
                        WHERE    Liefnr = 3  ) ;
```

Alle Tupel mit gleicher Artikelnr wie Lieferant 3

Nach Lieferanten gruppiert, also Artikel zusammengefasst

Nur die Lieferanten ausgewählt, die gleich viele gleiche Artikel haben

Oracle, SQL Server, MySQL

	Abweichungen zur SQL-Norm
Oracle	Bezeichner As ist in From-Klausel nicht erlaubt Fehlerhafte Ausgabe bei geschachteltem Natural Join Der Operator Except ist durch Minus zu ersetzen Eigene Syntax für die Funktion Substring: Substr(str,pos,anz) Im Insert-Befehl ist nur eine Values-Angabe erlaubt
SQL Server	Ein Select-Befehl in der From-Klausel erfordert einen Aliasnamen Der Natural Join und Join…Using werden nicht unterstützt Eigene Syntax für die Funktion Substring: Substring(str,pos,anz) Der Cast-Operator Date ist nicht erlaubt Eine Transaktion muss mit Begin Transaction eingeleitet werden
MySQL	Ein Select-Befehl in der From-Klausel erfordert einen Aliasnamen Den Aggregatfunktionsnamen darf kein Leerzeichen folgen Beim Natural Inner Join ist der Bezeichner Inner nicht erlaubt Der Full Outer Join wird nicht unterstützt Die Operatoren Except und Intersect werden nicht unterstützt Der Transaktionsbetrieb wird nur in der Engine InnoDB unterstützt

Datenbanken und SQL

Kapitel 5

Die Datenbankbeschreibungssprache SQL

Die Datenbankbeschreibungssprache SQL

- Relationen erzeugen, ändern und löschen
- Temporäre Relationen
- Sichten erzeugen und löschen
- Zusicherungen, Gebiete
- Trigger
- Sequenzen
- Zugriffsrechte und Zugriffsschutz
- Integrität
- Aufbau einer Datenbank
- Einrichten einer Datenbank

Befehl CREATE TABLE

```
CREATE TABLE Tabellenname
(
 {  Spalte { Datentyp | Gebietsname } [ Spaltenbedingung [ … ] ]
 |  Tabellenbedingung                }
 [ , … ]
)
[ Herstellerspezifische Optionen ]
```

Spaltenname

gefolgt von Datentyp

wahlfrei: gefolgt von Spaltenbedingungen

alternativ: Tabellenbedingung

Beliebig viele Spalten, durch Kommata getrennt

- Es wird eine neue Basisrelation erzeugt

Beispiel zu CREATE TABLE

```
CREATE TABLE Personal
( Persnr        INT             PRIMARY KEY ,
  Name          CHAR (25)       NOT NULL ,
  Ort           CHAR (15) ,
  Vorgesetzt    INTEGER         REFERENCES Personal
                                ON DELETE SET NULL
                                ON UPDATE CASCADE ,
  Gehalt        NUMERIC(8,2)    CHECK(Gehalt BETWEEN 800 AND 9000),
  Beurteilung CHAR ,
  CONSTRAINT MinVerdienst
  CHECK( Gehalt >= Coalesce((6-Beurteilung)*400, 800) )
) ;
```

Komma

Spaltenbedingung

Spaltenbedingung

Spaltenname

Datentyp

Spaltenbedingung

Tabellenbedingung

Wichtige Datentypen in SQL

INTEGER **INT**	Ganzzahl
SMALLINT	Ganzzahl
NUMERIC(x,y)	x stellige Dezimalzahl mit y Nachkommastellen
DECIMAL(x,y)	x stellige Dezimalzahl mit y Nachkommastellen
FLOAT(x)	Gleitpunktzahl mit x Nachkommastellen
CHARACTER(n) **CHAR(n)**	Zeichenkette der festen Länge n
CHARACTER VARYING(n) **VARCHAR(n)**	Variable Zeichenkette mit bis zu n Zeichen
BIT(n)	Bitleiste der festen Länge n
DATE	Datum (Jahr, Monat, Tag)
TIME	Uhrzeit (Stunde, Minute, Sekunde)
DATETIME	Kombination aus DATE und TIME

Datentyp DATE

- Arithmetik:

  Datum1 – Datum2 = Anzahl der Tage dazwischen

  Datum + Zahl = Datum addiert um Zahl der Tage

  Datum1 + Datum2 FEHLER

- Aktuelles Datum (heute):
  - CURRENT_DATE
    - In Oracle auch: SYSDATE
    - In SQL Server: GETDATE( )
- Beispiel:
  - Gestern: CURRENT_DATE – 1

Umwandlung: String in DATE

- CAST-Funktion:
  - Beispiel: CAST ('2012-12-24' AS DATE)
- DATE Operator:
  - Format weltweit genormt: JJJJ-MM-TT
  - Beispiel: DATE '2012-12-24'
  - In SQL Server nicht implementiert, Operator DATE weglassen!
- Herstellerspezifisch:
  - In Oracle: to_date('24.12.2012', 'DD.MM.YYYY')
  - In SQL-Server: convert(date, '24.12.2012')
  - In MySQL: str_to_date('24.12.2012', '%d.%m.%y')

Zeichenketten CHAR und VARCHAR

- Char(n)
  - Zeichenkette der Länge n
  - Es wird ein Speicherplatz von n Byte benötigt
  - Beim Einfügen: Auffüllen mit Leerzeichen
- Varchar(n)
  - Variable Zeichen
  - Längenfeld enthält die Länge der Zeichenkette
  - Beim Einfügen: Kein Auffüllen mit Leerzeichen
  - Statische oder dynamische Speichertechnik, abhängig von der Länge n und vom Hersteller

Vergleich: CHAR / VARCHAR

- Bei kleinen Zeichenketten (n < 100):
  - Meist gleicher statischer Speicherbedarf
  - Werden wenige Zeichen eingefügt, so muss bei Varchar nicht aufgefüllt werden
  - Bei Char ist kein Längenfeld erforderlich
- Bei großen Zeichenketten:
  - In Relation existiert bei Varchar nur ein Link auf den dynamischen Bereich. Dies reduziert den Speicherverbrauch

Verhalten: CHAR / VARCHAR

SELECT * FROM Personal WHERE Name LIKE '%e_';

- Ausgabe, falls Name vom Typ VARCHAR(n) ist:
  - Alle Namen, deren vorletzter Buchstabe ein e ist
- Ausgabe, falls Name vom Typ CHAR(n) ist:
  - Vermutlich nichts, da die auffüllenden Leerzeichen mitzählen (außer in MySQL, eigenartig in SQL Server)
- Dringender Rat: TRIM oder RTRIM verwenden:

  SELECT * FROM Personal WHERE Trim(Name) LIKE '%e_';

Spaltenbedingungen

- NOT NULL

Spalte darf keine Nullwerte enthalten

- { PRIMARY KEY | UNIQUE }

Kennzeichnung des Primärschlüssels und der alternativen Schlüssel

- REFERENCES Tabellenname [(Spalte [, ...])]
  [ON DELETE { NO ACTION | CASCADE | SET NULL }]
  [ON UPDATE { NO ACTION | CASCADE | SET NULL }]

Kennzeichnung der Fremdschlüssel

- CHECK (Bedingung)

Zusätzliche Einschränkungen

Vor jeder Spaltenbedingung kann wahlfrei ein Bedingungsname angegeben werden:

- [CONSTRAINT Bedingungsname]

Hinweise zu Spaltenbedingungen

- Spaltenbedingungen beziehen sich auf eine Spalte
  - Auch die Check-Bedingung bezieht sich nur auf „ihre“ Spalte
- Eine Relation enthält nur eine Angabe PRIMARY KEY
- Vorgaben sind, falls nicht angegeben:
  ON DELETE NO ACTION
  ON UPDATE NO ACTION
- Problem:
  - Wie werden Schlüssel definiert, die sich auf mehrere Spalten beziehen?
  - Wie werden Check-Bedingungen definiert, die sich auf mehrere Spalten beziehen?

Tabellenbedingungen

- { PRIMARY KEY | UNIQUE } (Spalte [, ...])
- FOREIGN KEY (Spalte [, ...])
  REFERENCES Tabellenname [(Spalte [, ...])]
  [ON DELETE { NO ACTION | CASCADE | SET NULL }]
  [ON UPDATE { NO ACTION | CASCADE | SET NULL }]
- CHECK (Bedingung)

Spaltenangaben

Kann sich auf alle Attribute der Relation beziehen

Vor jeder Spaltenbedingung kann wahlfrei ein Bedingungsname angegeben werden:

- [CONSTRAINT Bedingungsname]

Beispiel zu CREATE TABLE

```
CREATE TABLE Personal
( Persnr        INT              PRIMARY KEY ,
  Name          CHAR (25)        NOT NULL ,
  Ort           CHAR (15) ,
  Vorgesetzt    INTEGER          REFERENCES Personal
                                 ON DELETE SET NULL
                                 ON UPDATE CASCADE ,
  Gehalt        NUMERIC(8,2)     CHECK(Gehalt BETWEEN 800 AND 9000),
  Beurteilung CHAR ,
  CONSTRAINT MinVerdienst
  CHECK( Gehalt >= Coalesce((6-Beurteilung)*400, 800) )
) ;
```

Persnr ist Primärschlüssel

Name darf nicht Null sein

Vorgesetzt ist Fremdschlüssel

Spaltenbedingung: Gehalt erfüllt Bedingung

Tabellenbedingung! Gehalt und Beurteilung involviert!

Create Table: Oracle, SQL Server, MySQL

- Oracle:
  - On Delete No Action: Angabe nicht erlaubt, standardmäßig gesetzt
  - On Update: Angabe grundsätzlich nicht erlaubt. Vorgabe: No Action
- SQL Server:
  - Kein Operator DATE. Umwandlung geschieht automatisch
- MySQL:
  - Keine Bedingungsnamen bei Spaltenbedingungen
  - Fremdschlüsselangaben nur als Tabellenbedingungen (InnoDB)
  - Spaltennamen müssen bei Fremdschlüsseln angegeben werden

Befehl ALTER TABLE

- Ändern einer bestehenden Basisrelation
- Eine Änderung kann pro Befehl vorgenommen werden

```
ALTER TABLE  Tabellenname
{  ADD [ COLUMN ]      Spalte   { Datentyp | Gebietsname }
                            [ Spaltenbedingung  [ … ]  ]
|  DROP [ COLUMN ]     Spalte   { RESTRICT | CASCADE }
|  ADD                 Tabellenbedingung
|  DROP CONSTRAINT     Bedingungsname
                       { RESTRICT | CASCADE }
```

Entweder: neue Spalte

Oder: Spalte löschen

Oder: neue Tabellenbedingung

Oder: Tabellenbedingung löschen

Alter Table: Oracle, SQL Server, MySQL

- Oracle:
  - ADD: Bezeichner COLUMN ist nicht erlaubt
  - DROP: Bezeichner COLUMN zwingend, oder Spalte geklammert
  - Bezeichner RESTRICT nicht erlaubt, gilt standardmäßig
  - CASCADE CONSTRAINT statt CASCADE
- SQL Server:
  - Bezeichner COLUMN, RESTRICT, CASCADE nicht erlaubt. Standardmäßig gilt RESTRICT
- MySQL:
  - DROP CONSTRAINT: Eigene Syntax (DROP INDEX, DROP KEY)

Befehl DROP TABLE

- Entfernen einer Relation

DROP TABLE Tabellenname { RESTRICT | CASCADE }

- Beispiel:

  DROP TABLE Personal CASCADE ;

- Oracle, SQL Server, MySQL:
  - RESTRICT, CASCADE nicht erlaubt bzw. ignoriert
  - Standard: RESTRICT
  - In Oracle CASCADE CONSTRAINT möglich

Temporäre Relationen

```
CREATE { LOCAL | GLOBAL } TEMPORARY TABLE Tabellenname
(  {  Spalte { Datentyp | Gebietsname }  [ Spaltenbedingung [ … ] ]
   |  Tabellenbedingung                 }
      [ , … ]
)  [ ON COMMIT  { PRESERVE | DELETE }  ROWS ]
```

Analog zu Create Table

Inhalt löschen bei Transaktionsende

- Anlegen von Spalten und Tabellenbedingungen wie bisher
- Sichtbarkeit lokal oder auch für andere Benutzer (global)
- Automatisches Löschen dieser Relation am Sessionende
- Falls gewünscht: Löschen des Inhalts bei Transaktionsende

Temporäre Tabelle: Oracle, SQL Server, MySQL

- Oracle:
  - Angabe LOCAL nicht möglich
- SQL Server:
  - CREATE TEMPORARY TABLE nicht implementiert. Stattdessen:
  - Lokal temporär: #…
  - Global temporär: ##…
  - ON COMMIT nicht implementiert, kein Löschen bei TA-Ende
- MySQL:
  - LOCAL und GLOBAL nicht möglich. LOCAL ist Standard
  - ON COMMIT nicht implementiert, kein Löschen bei TA-Ende

Sichten (Views)

- Eine Sicht ist eine virtuelle Tabelle, die über einen Select-Befehl implementiert ist

```
CREATE VIEW Sichtname [ ( Spalte [ , ... ] ) ]
  AS  Select-Befehl
 [ WITH CHECK OPTION ]
```

```
DROP VIEW  Sichtname { RESTRICT | CASCADE }
```

- Oracle, SQL Server, MySQL:
  - RESTRICT, CASCADE wie bei DROP TABLE

Sichten und Zugriffsschutz

- **Szenario:**
  - Benutzer darf nur auf einige Attribute von Personal zugreifen
  - Also: Zugriff auf Sicht Personal1 erlauben, nicht auf Personal!

```
CREATE VIEW Personal1
AS SELECT Persnr, Name, Ort AS Wohnort, Vorgesetzt AS Chef
FROM Personal ;
```

Personal1 ist eine Relation!

Personal1 besitzt 4 Attribute: Persnr, Name, Wohnort, Chef

- **Beispielhafter Zugriff:**

```
SELECT * FROM Personal1 ;
```

Relation Personal1

- SELECT * FROM Personal1 ;

Persnr	Name	Wohnort	Chef
1	Maria Forster	Regensburg	NULL
2	Anna Kraus	Regensburg	1
3	Ursula Rank	Frankfurt	6
4	Heinz Rolle	Nürnberg	1
5	Johanna Köster	Nürnberg	1
6	Marianne Lambert	Landshut	NULL
7	Thomas Noster	Regensburg	6
8	Renate Wolters	Augsburg	1
9	Ernst Pach	Stuttgart	6

Sichten und Lesbarkeit

- Szenario:
  - Auftrag soll „lesbar“ werden, z.B. Kundenname statt Kundnr

```
CREATE VIEW
 VAuftrag (AuftrNr, Datum, Kundname, Persname, Summe) AS
SELECT   AuftrNr, Datum, Kunde.Name,
                          Personal.Name, SUM(Gesamtpreis)
FROM     Auftrag  JOIN Kunde  ON Kunde.Nr = Auftrag.Kundnr
                  JOIN Personal  USING (Persnr)
                  JOIN Auftragsposten  USING (Auftrnr)
GROUP BY Auftrnr, Datum, Kunde.Name, Personal.Name ;
```

Auftragssumme (Gruppierung)

Join über 4 Tabellen

Sicht VAuftrag

AuftrNr	Datum	Kundname	Persname	Summe
1	04.01.2013	Fahrrad Shop	Anna Kraus	800
2	06.01.2013	Maier Ingrid	Johanna Köster	2350
3	07.01.2013	Rafa – Seger KG	Anna Kraus	800
4	18.01.2013	Fahrräder Hammerl	Johanna Köster	824
5	06.02.2013	Fahrrad Shop	Anna Kraus	70

- **Beispielhafter Zugriff:**

```
SELECT * FROM VAuftrag
WHERE Persname LIKE '%Köster%' ;
```

Zugriff: Intern generierter Befehl

VAuftrag:
Select-Befehl übernehmen

```
SELECT  * FROM
(  SELECT AuftrNr, Datum, Kunde.Name AS Kundname,
      Personal.Name AS Persname, SUM(Gesamtpreis) AS Summe
   FROM Auftrag  JOIN Kunde  ON Kunde.Nr = Auftrag.Kundnr
                 JOIN Personal  USING (Persnr)
                 JOIN Auftragsposten  USING (Auftrnr)
   GROUP BY AuftrNr, Datum, Kunde.Name, Personal.Name
)
WHERE  Persname LIKE '%Köster%' ;
```

Änderbare Sichten

- Eine Sicht ist änderbar, wenn gilt:
  - Die From-Klausel enthält nur eine Relation
  - Eine Group-By-Klausel ist nicht vorhanden
  - Die Select-Klausel enthält keine Distinct-Angabe
  - Die Spaltenliste besteht nur aus einzelnen Spaltennamen
  - Die Operatoren Union, Intersect, Except kommen nicht vor

- Sicht Personal1 ist änderbar
- Sicht VAuftrag ist nicht änderbar

Beispiel einer änderbaren Sicht

```
UPDATE Personal1
SET    Wohnort = 'Hannover'
WHERE  Persnr = 2 ;
```

- Ergebnis (Auszug aus Relation Personal):

Persnr	Name	Strasse	PLZ	Ort
1	Maria Forster	Ebertstr. 28	93051	Regensburg
2	Anna Kraus	Kramgasse 5	93047	Hannover
3	Ursula Rank	Dreieichstr. 12	60594	Frankfurt
4	Heinz Rolle	In der Au 5	90455	Nürnberg
…	…	…	…	…

Beispiel: Sicht Jugend

- **Gegeben: Relation Vereinsmitglieder und Sicht Jugend:**

```
CREATE VIEW Jugend AS
  SELECT * FROM  Vereinsmitglieder WHERE  Alter < 21 ;
```

- **Szenario: Mitglied 227 wird 21 Jahre alt:**

```
UPDATE  Jugend
SET         Alter = Alter + 1
WHERE   Mitgliednr = 227 ;
```

- **Problem: Mitglied 227 ist verschwunden !?**

Problem der Sicht Jugend

- Ein Update wirkt sich aus wie ein Delete!
  - Ein Mitglied verschwindet mit 21 Jahren aus der Sicht
  - Aber: Mitglied existiert noch (in Relation Vereinsmitglieder)
- Lösung: Wir verbieten solche Änderungen!
- Option: WITH CHECK OPTION

```
CREATE VIEW Jugend AS
  SELECT * FROM Vereinsmitglieder WHERE Alter < 21
WITH CHECK OPTION ;
```

Jetzt muss das Alter kleiner 21 bleiben

Ansonsten: Fehlermeldung

Zusicherungen (Assertions)

- Zusicherung: Datenbankweite Bedingung
- Zusicherung erzeugen:

CREATE ASSERTION Bedingungsname

CHECK (Bedingung)

- Zusicherung entfernen:

DROP ASSERTION Bedingungsname

Beispiel zu Zusicherung

- Angebotspreis soll nicht über Listenpreis liegen
- Gelöst mit CHECK-Bedingung:

```
ALTER TABLE Auftragsposten ADD
  CONSTRAINT Auftragspreis
  CHECK ( Gesamtpreis <= (SELECT  Anzahl*Preis
                          FROM    Artikel
                          WHERE ANr = Artnr )  ) ;
```

In fast allen Datenbanken: Select-Befehl ist hier nicht erlaubt!

- Aber:
  - Preisänderung in Relation Artikel bleibt unberücksichtigt!

Lösung mit ASSERTION

```
CREATE ASSERTION AssertionPreis
 CHECK ( NOT EXISTS
     ( SELECT *
       FROM Auftragsposten INNER JOIN Artikel ON ANr = Artnr
       WHERE Gesamtpreis > Anzahl * Preis
     )       ) ;
```

- Jetzt Überprüfung in Auftragsposten und Artikel!
- Aber:
  - In Oracle, SQL Server und MySQL nicht implementiert

Lösung mit WITH CHECK OPTION

- **Besser als nichts: WITH CHECK OPTION verwenden**

```
CREATE VIEW VAuftragsposten AS
   SELECT  * FROM Auftragsposten
   WHERE  Gesamtpreis <= ( SELECT   Anzahl * Preis
                           FROM     Artikel
                           WHERE  Artnr = Anr  )
    WITH CHECK OPTION ;
```

Lösung funktioniert, wenn Benutzer nur über diese Sicht zugreift

- **Noch besser: TRIGGER (siehe weiter unten)**

Gebiete (Domains)

- Verfeinerung von Datentypen
  - Beispiel Aufzählungen: Städte, Familienstand
- Gebiet erzeugen:

```
CREATE DOMAIN Gebietsname [ AS ] Datentyp
 [ [ CONSTRAINT Bedingungsname ] CHECK (Bedingung) ] [ ... ]
```

- Gebiet entfernen:

```
DROP DOMAIN Gebietsname { RESTRICT | CASCADE }
```

Beispiel zu Gebieten

- **Definition einiger wichtiger EURO-Hauptstädte:**

```
CREATE DOMAIN EURO_Hauptstadt AS CHARACTER (15)
  CHECK (VALUE IN ( 'Berlin', 'Paris', 'Rom', 'Madrid', 'Lissabon',
                    'Amsterdam', 'Dublin', 'Brüssel', 'Athen',
                    'Luxemburg', 'Wien', 'Helsinki' )    ) ;
```

- Aber:
  - In Oracle, SQL Server und MySQL nicht implementiert
- Alternative:
  - CHECK-Bedingungen oder Trigger; aber nicht so elegant!

Trigger

- Trigger sind ereignisgesteuerte Aktionen
- Einsatz von Triggern:
  - Überprüfen, ob Eingaben im erlaubten Bereich
  - Zusätzliches Ablegen von Informationen (z.B. Protokollierung)
  - Ausgaben von Infos und Warnungen
- Mögliche auslösende Ereignisse:
  - Vor oder nach dem Einfügen, Löschen, Ändern von Relationen
- Aktionen:
  - SQL-Befehle, (kleinere) Prozeduren

Trigger (Erzeugen und Entfernen)

```
CREATE TRIGGER Triggername
  { BEFORE | AFTER } { DELETE | INSERT | UPDATE } [ OR … ]
  ON Tabellenname
  [ REFERENCING [ OLD AS NameAlt ] [ NEW AS NameNeu ] ]
  [ FOR EACH STATEMENT | FOR EACH ROW ]
  [ WHEN Bedingung ]
    Anweisungen

DROP TRIGGER Triggername
```

vor oder nach …

… Änderungen …

… in Relation

Setzen von Aliasnamen für neue/alte Daten

Ganze Relation bearbeiten (Vorgabe) oder jedes Tupel einzeln

Anweisungen ausführen, wenn Bedingung wahr

Beispiel 1 zu Trigger

- Wunsch: Bei jedem neuen Auftrag wird automatisch das heutige Datum (CURRENT_DATE) gesetzt
- Lösungsversuch:

```
CREATE TRIGGER Auftragsdatum1_Trigger
  AFTER INSERT ON Auftrag
BEGIN
    UPDATE AUFTRAG
    SET Datum = CURRENT_DATE ;
END ;
```

Nach dem Einfügen in Auftrag …

In Standard-SQL: BEGIN ATOMIC

… erfolgt dieser Update

- Aber:
  - Es werden alle Aufträge auf das heutige Datum gesetzt!

Beispiel 2: Korrektur von Beispiel 1

- Jetzt: Tupelweises Ausführen des Triggers
- Lösung für Oracle

```
CREATE TRIGGER Auftragsdatum2_Trigger
  BEFORE INSERT ON Auftrag
    REFERENCING NEW AS neu FOR EACH ROW
  BEGIN
    :neu.Datum := CURRENT_DATE ;
  END;
/
```

Vor dem Einfügen in Auftrag …

In Standard-SQL: BEGIN ATOMIC

… wird zeilenweise überprüft …

In Standard-SQL: keine Doppelpunkte

… und zu änderndes Datum wird vor dem Einfügen durch neuen (:neu) Eintrag überschrieben

Beispiel 3

- Auftragspreis ist nicht größer als Listenpreis
- Lösung mit Trigger:
  - Wenn doch, dann Listenpreis setzen und Meldung ausgeben
  - Gelöst mit Oracle PL/SQL

```
CREATE TRIGGER Auftragspreis_Trigger
    BEFORE INSERT OR UPDATE ON Auftragsposten
    REFERENCING NEW AS neu FOR EACH ROW
  DECLARE
    listenpreis NUMERIC(8,2) ;
```

Vor Einfügen oder Ändern in Auftrag zeilenweise ausführen

Deklarationsteil in PL/SQL

Beispiel 3: Anweisungen

```
BEGIN
   SELECT :neu.Anzahl*Preis          -- Berechnung des Listenpreises
    INTO listenpreis
    FROM Artikel
    WHERE Anr = :neu.Artnr ;
   IF (:neu.Gesamtpreis > listenpreis)
   THEN
     :neu.Gesamtpreis := listenpreis ;
     dbms_output.put_line ('Preis in Posnr ' || :neu.posnr || 'geändert');
   END  IF;
  END ;
/
```

Speichert Ergebnis in Variable listenpreis

Abfrage, ob neuer Gesamtpreis zu hoch

Wenn ja, dann ersetzen und ausgeben

Ausgabefunktion in Oracle

Konkatenierung in Oracle

in Oracle wichtig!

Sequenzen

- Sequenzen sind Generatoren zum Erzeugen von automatischen Nummerierungen
- Sequenz erzeugen:

```
CREATE SEQUENCE Sequenzname [ AS Datentyp ]
  [ START WITH Konstante ]  [ INCREMENT BY Konstante ]
```

- Sequenz entfernen:

```
DROP SEQUENCE Sequenzname
```

Beispiel

- Aufträge mit automatischer Auftragsnummer:
  - Nummern fortlaufend ab 1000

```
CREATE SEQUENCE Auftragssequenz
START WITH 1000;
```

- Verwendung:

```
INSERT INTO Auftrag(Auftrnr, Datum, Kundnr, Persnr) VALUES
(NEXT VALUE FOR Auftragssequenz, CURRENT_DATE, 3, 5);
```

- In Oracle:

```
… VALUES (Auftragssequenz.NEXTVAL, CURRENT_DATE, 3, 5);
```

Zugriffsrechte

- Zugriffsrechte auf Relationen
  - Beliebiger Zugriff für den Eigentümer
    - Eigentümer: Benutzer, der die Relation erzeugte
  - Zugriff für andere nur, wenn Rechte eingeräumt werden

- Gewähren und Entziehen von Zugriffsrechten
  - Eigentümer kann Zugriffsrechte an Dritte vergeben
  - Diese Rechte können jederzeit widerrufen werden
  - Eigentümer behält immer alle Rechte

Zugriffsrechte gewähren (Grant)

```
GRANT      Zugriffsrecht  [ , ... ]
ON [ TABLE ]   { Tabellenname | Sichtname }
TO         Benutzer  [ , ... ]
[ WITH GRANT OPTION ]
```

ev. mehrere Rechte

nur eine Relation!

ev. mehrere Benutzer

Vererben eines Zugriffsrechts, siehe weiter unten

- Grant-Befehl gibt an,
  - welche(r) Benutzer
  - auf welche Relation
  - welche(s) Zugriffsrecht(e) erhält / erhalten

Zugriffsrechte

Zugriffsrecht	erlaubt ...
Select	den lesenden Zugriff auf eine Relation
Update	das Ändern von Inhalten einer Relation
Update (x1, ...)	das Ändern der Attributwerte x1,... einer Relation
Delete	das Löschen von Tupeln einer Relation
Insert	das Einfügen neuer Tupel in eine Relation
Insert (x1, ...)	das Einfügen der Attributwerte x1,... in eine Relation
References (x1, ...)	das Referenzieren der Attribute x1,... einer Relation
Usage	das Verwenden eines Gebietes

Besonderheiten im Grant Befehl

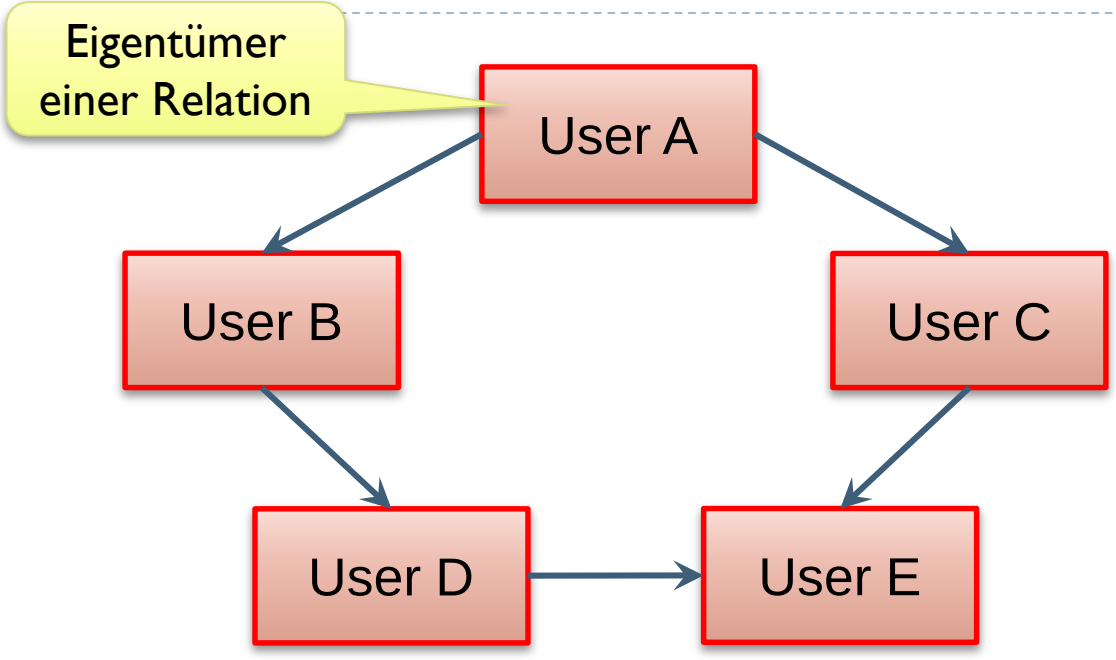
- Alle Rechte vergeben:
  - ALL [PRIVILEGES]
- Rechte an alle Benutzer vergeben:
  - Benutzername PUBLIC verwenden
- OPTION: WITH GRANT OPTION
  - Benutzer erhält mit dem Zugriffsrecht auch das Recht der Weitergabe dieses Rechts an Dritte
- Oracle, SQL Server, MySQL:
  - Komplett implementiert, mit Erweiterungen (z.B. mehrere Relationen, Wildcart-Syntax); teilweise: Füllwort TABLE verboten

Zugriffsrechte entziehen (Revoke)

```
REVOKE [ GRANT OPTION FOR ]
    { Zugriffsrecht [ , … ] | ALL PRIVILEGES }
    ON [ TABLE ] { Tabellenname | Sichtname }
    FROM Benutzer [ , … ]
    { RESTRICT | CASCADE }
```

- Oracle, SQL Server, MySQL:
  - GRANT OPTION FOR, RESTRICT, CASCADE, TABLE teilweise nicht unterstützt
  - In MySQL gibt es kein Reference-Recht

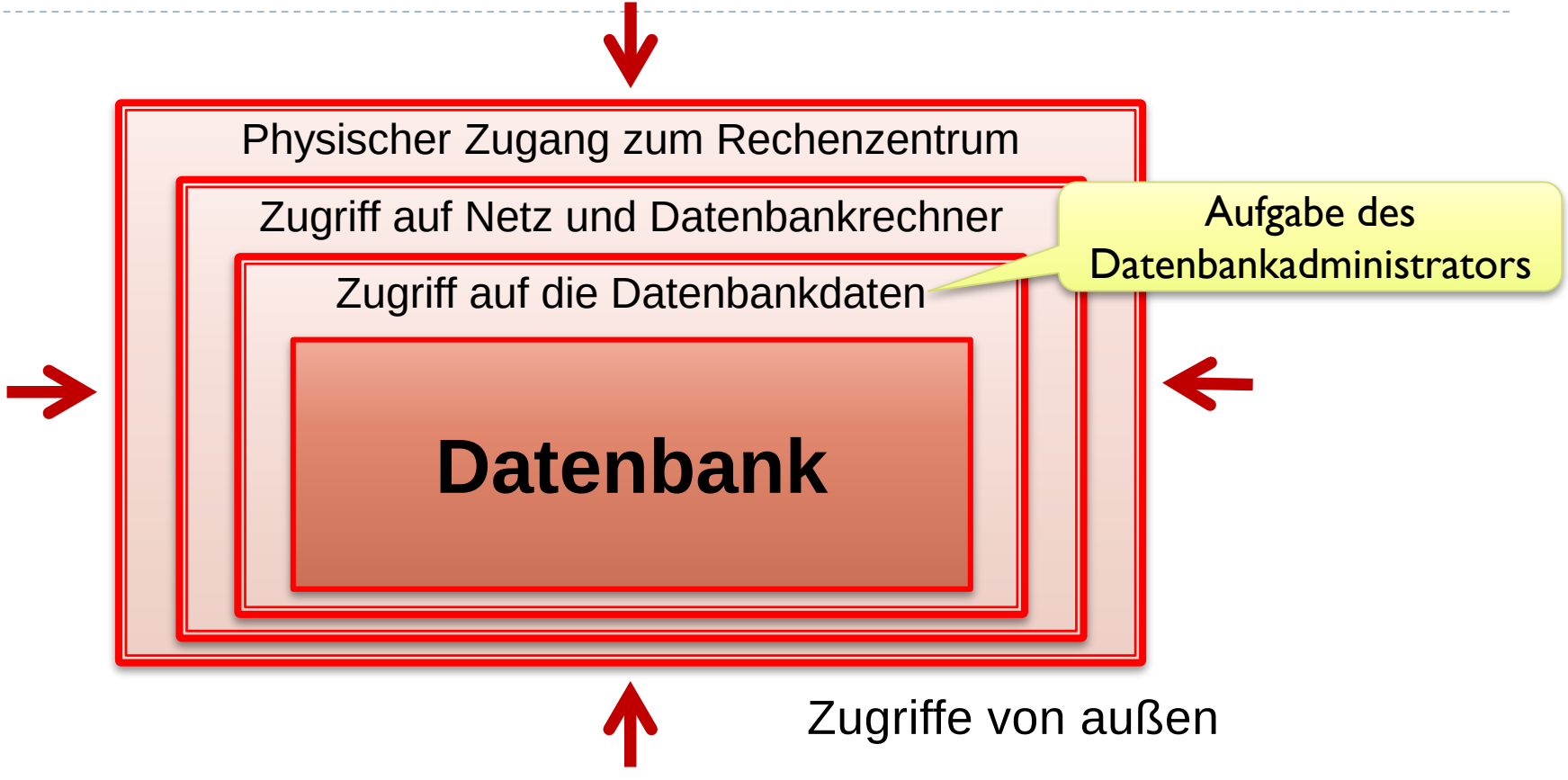
Kaskadierendes Entziehen von Rechten



- Rechtevergabe mit **Grant Option**:
  - A → B
  - A → C
  - B → D
  - D → E
  - C → E
- Rechteentzug kaskadierend:
  - A entzieht B
    - Automatisch auch:
    - B entzieht D
    - D entzieht E

Beim kaskadierenden Revoke wird die ganze betroffene Rechtekette gelöscht

Zugriffsschutz



Zugriffsschutz auf Relationen (1)

- In SQL: Select-Recht gibt es nur auf gesamte Relation
- Lösung: Sichten!
- Beispiel: Relation Personal; Erzeugen einer Sicht VPersonal
  - Projektion: Zugriff auf Persnr, Name, Ort, Vorgesetzt, Aufgabe
  - Restriktion: Kein Zugriff auf Daten von Vorgesetzten

```
CREATE VIEW VPersonal AS
    SELECT   Persnr, Name, Ort, Vorgesetzt, Aufgabe
    FROM     Personal
    WHERE  Vorgesetzt IS NOT NULL ;
```

Projektion

Restriktion

Zugriffsschutz auf Relationen (2)

- **Einstellen des Zugriffs für Benutzer *Mitarbeiter*:**

```
REVOKE ALL PRIVILEGES
  ON Personal
  FROM Mitarbeiter ;

GRANT SELECT
  ON VPersonal
  TO Mitarbeiter ;
```

Entziehen der Rechte auf Personal

Gewähren der Rechte auf VPersonal

Zugriffsschutz auf Relationen (3)

- Zugriffe, abhängig von Benutzergruppen

Benutzer	Rechte
Alle Benutzer	Select-Recht auf VPersonal
Abteilungsleiter	Select- und Update-Recht auf VPersonal Select-Recht auf Personal
Personalabteilung	Select-Recht auf Personal Änderungsrechte auf einzelne Attribute
Personalchef	Alle Rechte auf Personal

Semantische Integrität (Maßnahmen)

- Benutzer greift über Masken und Eingabefelder zu
- Eingabefelder sind eingabespezifisch:
  - Beispiel: Bei Zahleneingaben nur Ziffern zulassen.
- Bei Eingabefeldern möglichst Auswahlfelder verwenden
- Nummern automatisch generieren
  - Beispiel: Auftragsnummer, Personalnummer, Kundennummer
- Überprüfen auf Plausibilität und Korrektheit

GUI

GUI

GUI+Programmierung

Sequenzen

Check-Bedingung, Assertion, Domain, With Check Option, Trigger

Beispiel: Auswahlfeld

- **Auswahlfeld, programmiert mit PHP:**

Unterstützende Werkzeuge, Beispiele

Befehl	Info
Create Table	Check-Bedingung
Create View	With-Check-Option
Create Assertion	Datenbankweite Zusicherung
Create Domain	Datenbankeinheitliche Gebiete
Create Trigger	Eingabetrigger (Insert, Update, Delete)

```
ALTER TABLE Personal ADD Arbeitszeit INTEGER NOT NULL
        CHECK( Arbeitszeit BETWEEN 15 AND 40 ) ;

ALTER TABLE  Artikel ADD CONSTRAINT Preischeck
        CHECK( Preis = Netto + Steuer ) ;
```

Gefährlich: Rundungsfehler!

Schema

- Datenbank besteht aus mehreren Schemata
- Schema enthält Relationen, Zugriffsrechte, ev. Trigger
- Schema erzeugen:

```
CREATE SCHEMA Schemaname
 [ AUTHORIZATION Benutzername ]  [ Schemaelement  [ ... ] ]
```

- Schema entfernen:

```
DROP SCHEMA  Schemaname  { CASCADE | RESTRICT }
```

Beispiel zu Schema: Schema Bike

- Datenbank Bike als eigenes Schema definieren

```
CREATE SCHEMA Bike
        CREATE TABLE Personal     (     …     )
        CREATE TABLE Kunde        (     …     )
        CREATE TABLE Auftrag      (     …     )
        CREATE VIEW VAuftrag      (     …     )
        GRANT …
        …     ;
```

- Externer Zugriff mittels: Bike.Personal, Bike.Kunde usw.

Information Schema

Relation	enthält
SCHEMATA	alle Schemata
DOMAINS	alle Gebiete
TABLES	alle Basisrelationen
VIEWS	alle Sichten
VIEW_TABLE_USAGE	alle Abhängigkeiten der Sichten von Relationen
VIEW_COLUMN_USAGE	alle Abhängigkeiten der Sichten von Spalten
COLUMNS	alle Spaltennamen aller Basisrelationen
TABLE_PRIVILEGES	alle Zugriffsrechte auf Relationen
COLUMN_PRIVILEGES	alle Zugriffsrechte auf Spalten aller Relationen
DOMAIN_CONSTRAINTS	alle Gebietsbedingungen für alle Gebiete
TABLE_CONSTRAINTS	alle Tabellenbedingungen aller Relationen
REFERENTIAL_CONSTRAINTS	alle referentiellen Bedingungen
CHECK_CONSTRAINTS	alle Check-Bedingungen aller Relationen
TRIGGERS	alle Trigger
TRIGGER_TABLE_USAGE	alle Abhängigkeiten der Trigger von Relationen
TRIGGER_COLUMN_USAGE	alle Abhängigkeiten der Trigger von Spalten
ASSERTIONS	alle Zusicherungen
DOMAINS	alle Gebiete

Beispielhafter Zugriff:

```
Select *
From Information_Schema.Tables;
```

Datenbanken und Oracle

- SCHEMA = USER
  - Zu jedem Benutzer wird ein Schema gleichen Namens angelegt
  - Dies geschieht automatisch mit CREATE USER
- Rechte auf ein Schema können vergeben werden:
  CREATE SCHEMA AUTHORIZATION Benutzername …
- Es gibt kein INFORMATION_SCHEMA

Systemtabellen in Oracle (Auszug)

Relation	Enthält
DICTIONARY	Zusammenfassung zu allen Systemtabellen
USER_TABLES	alle Relationen des Benutzers
USER_TAB_COLUMNS	alle Attribute aller Relationen des Benutzers
USER_VIEWS	alle Sichten des Benutzers
USER_CONSTRAINTS	alle Spalten- und Tabellenbedingungen
USER_CONS_COLUMNS	alle Attribute mit Spalten- und Tabellenbedingungen
USER_INDEXES	alle Indexe in Relationen des Benutzers
USER_IND_COLUMNS	alle Attribute, die Indexe besitzen
USER_TAB_PRIVS	alle Privilegien in Bezug auf Relationen
USER_COL_PRIVS	alle Privilegien in Bezug auf Attribute
USER_TRIGGERS	alle Trigger des Benutzers
USER_TRIGGER_COLS	alle Attribute, auf die sich Trigger beziehen
USER_TABLESPACES	alle Tablespaces des Benutzers

Datenbanken und SQL Server

- CREATE USER:
  - Legt neuen Benutzer und sein Schema fest
  - Standardmäßig Schema dbo,
  - Oder explizit festlegen (WITH DEFAULT_SCHEMA =)
- ALTER USER:
  - Ändern der Zuordnung zu Schema
- CREATE SCHEMA:
  - Zuordnung zu Benutzer möglich
- INFORMATION_SCHEMA wird voll unterstützt

Datenbanken und MySQL

- SCHEMA = DATENBANK
  - CREATE SCHEMA = CREATE DATABASE
- Schema ist keinem Benutzer zugeordnet
  - Kein Parameter: AUTHORIZATION Benutzername
- USE Schemaname:
  - Zuordnung eines Benutzers während der Laufzeit zu Schema
- INFORMATION_SCHEMA wird voll unterstützt

Einloggen in Datenbank

- Einloggen, Start der Session und der ersten Transaktion:

CONNECT TO {DEFAULT|Servername} [AS Verbindungsname]
[USER Benutzername]

- Beenden der Session:

DISCONNECT { DEFAULT | CURRENT | SQL-Servername }

- In Standard-SQL:
  - Kein CREATE DATABASE, kein CREATE USER

Datenbank verwalten in Oracle

- Je Server mehrere Datenbanken möglich
  - CREATE DATABASE
  - ALTER DATABASE
  - DROP DATABASE
- Benutzer einrichten:
  - CREATE USER Benutzername IDENTIFIED BY Kennwort
  - Gewähren von Verbindungsrechten mit GRANT
- Beispiel:

  CREATE USER gast IDENTIFIED BY neu ;

  GRANT Connect, Resource TO gast ;

Connect: Erlaubt Verbinden mit DB
Resource: Erlaubt CREATE-Befehle

Verbinden mit Oracle, Befehle

- CONNECT gast/neu@xe ;

Verbindung mit Kennung gast und Passwort neu an Datenbank xe

CREATE DATABASE Datenbankname ...	Legt eine neue Datenbank inklusive Logdateien an
ALTER DATABASE Datenbankname ...	Ändert Datenbankeinstellungen
CREATE CLUSTER Clustername ...	Erzeugt einen neuen Cluster
CREATE USER Benutzername	Legt einen neuen Benutzer an
ALTER USER Benutzername	Ändert Benutzereinstellungen
CONNECT Benutzer/Passwort@DB	Einloggen in Datenbank
CREATE TABLESPACE Tablespacename ...	Erzeugt einen physischen Bereich zum Speichern der Basisrelationen und Indexe

Datenbank verwalten mit SQL Server

- Pro Server kann eine Datenbank erstellt werden
  - CREATE DATABASE
  - ALTER DATABASE
  - DROP DATABASE
- Benutzer einrichten:
  - CREATE USER Benutzername WITH DEFAULT_SCHEMA =
- Es gibt keinen Connect-Befehl
  - Einloggen mittels SSMS oder Programmierschnittstelle
- Beispiel:

  CREATE USER gast WITH DEFAULT_SCHEMA = Bike ;

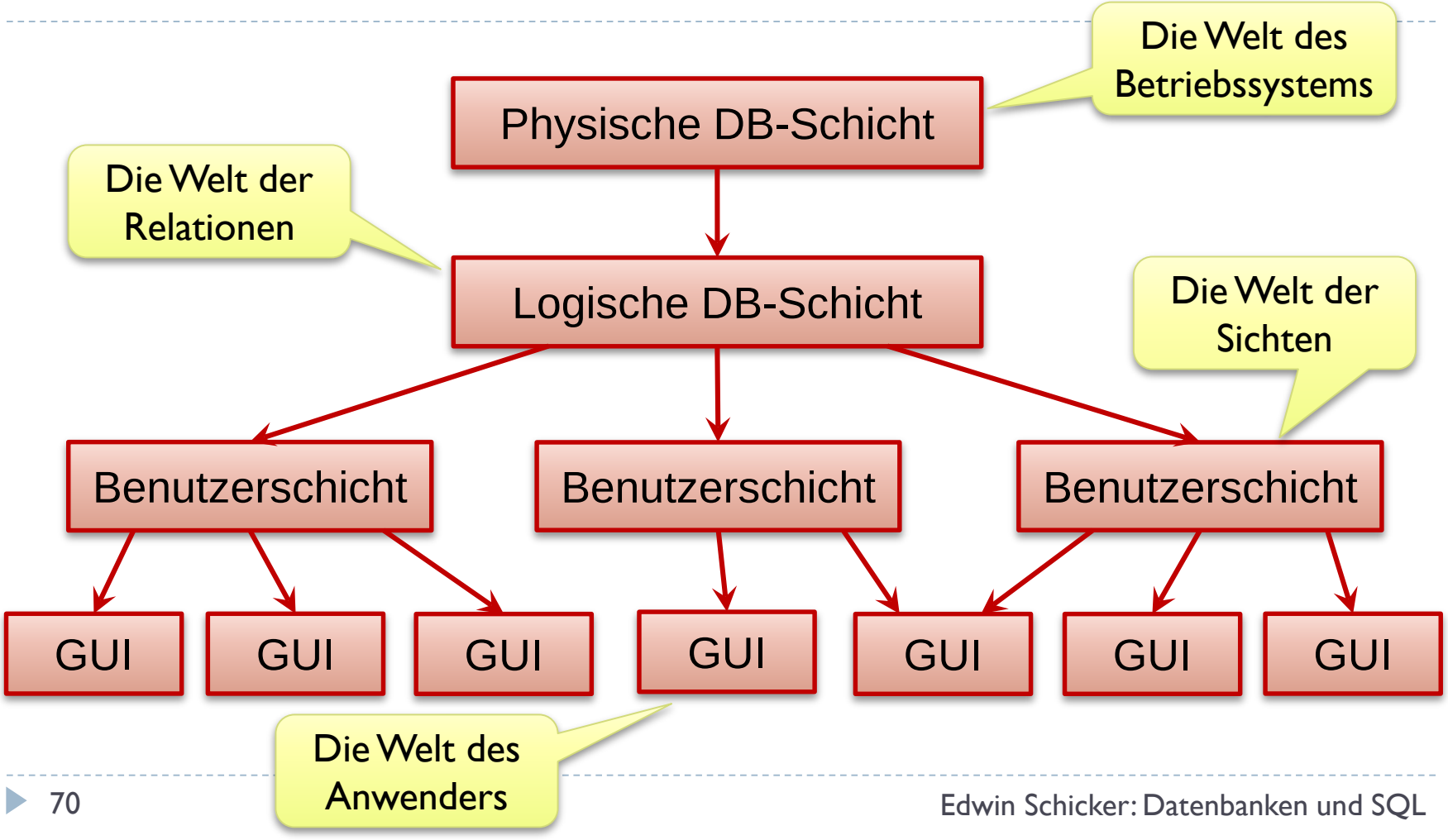
Datenbank verwalten mit MySQL

- Vorzugsweise: Arbeiten mit MySQL Workbench
- Mit der Konsole:
  - mysql –u root
    - use bike
- CREATE USER (Unterscheidung zwischen global / lokal):

```
CREATE USER gast IDENTIFIED BY 'neu' ;
CREATE USER gast@localhost IDENTIFIED BY 'neu' ;
GRANT SELECT, INSERT, DELETE, UPDATE ON Bike.* TO gast ;
GRANT SELECT, INSERT, DELETE, UPDATE ON Bike.*
                                    TO gast@localhost ;
```

In MySQL Wildcarts erlaubt

Aufbau einer Datenbank



Zusammenfassung

- DDL hat viele Möglichkeiten zur Gestaltung einer DB:
  - CREATE TABLE, ALTER TABLE, DROP TABLE
  - CREATE VIEW, DROP VIEW
  - CREATE ASSERTION, DROP ASSERTION
  - CREATE DOMAIN, ALTER DOMAIN
  - CREATE TRIGGER, ALTER TRIGGER, DROP TRIGGER
  - CREATE SEQUENCE, ALTER SEQUENCE, DROP SEQUENCE
  - CREATE SCHEMA, ALTER SCHEMA, DROP SCHEMA
  - GRANT, REVOKE
  - CREATE DATABASE, CREATE USER, CREATE TABLE_SPACE, …

Datenbanken und SQL

Kapitel 6

Datenbankprogrammierung mit PHP

Datenbankprogrammierung mit PHP

- Zusammenspiel: Browser, Webserver und Datenbank
- Kurzeinführung in HTML und PHP
- Einfache Datenbankzugriffe
- Fehlerbehandlung
- Verwendung von Sessionvariablen
- Umfangreiche Lese- und Schreibzugriffe auf Datenbanken
- Transaktionsbetrieb
- Arbeiten mit binären Daten (Bilder) in Datenbanken

Browser und Webserver (1)

- Browser
  - Beispiele: Internet Explorer, Firefox, Chrome, Safari, usw.
  - Browser sind lokal installiert
  - Browser lesen HTML-Seiten und geben diese formatiert aus
  - Browser kennen Javascript und Flash als Programmiersprachen
- Webserver
  - Beispiele: IIS unter Windows, Apache unter Unix und Windows
  - Webserver bieten Dienste an
  - Webdienste können statisch sein (nur HTML)
  - Webdienste können dynamisch sein (JSP, PHP, Perl, ASP, …)

Browser und Webserver (2)

Ergebnis vom Browser dargestellt

über Telefonleitung, Glasfaser oder drahtlos (LAN, WLAN)

statisch (nur HTML) dynamisch (PHP, …)

Anfrage

Ergebnis

Im Browser:
http://www.google.de

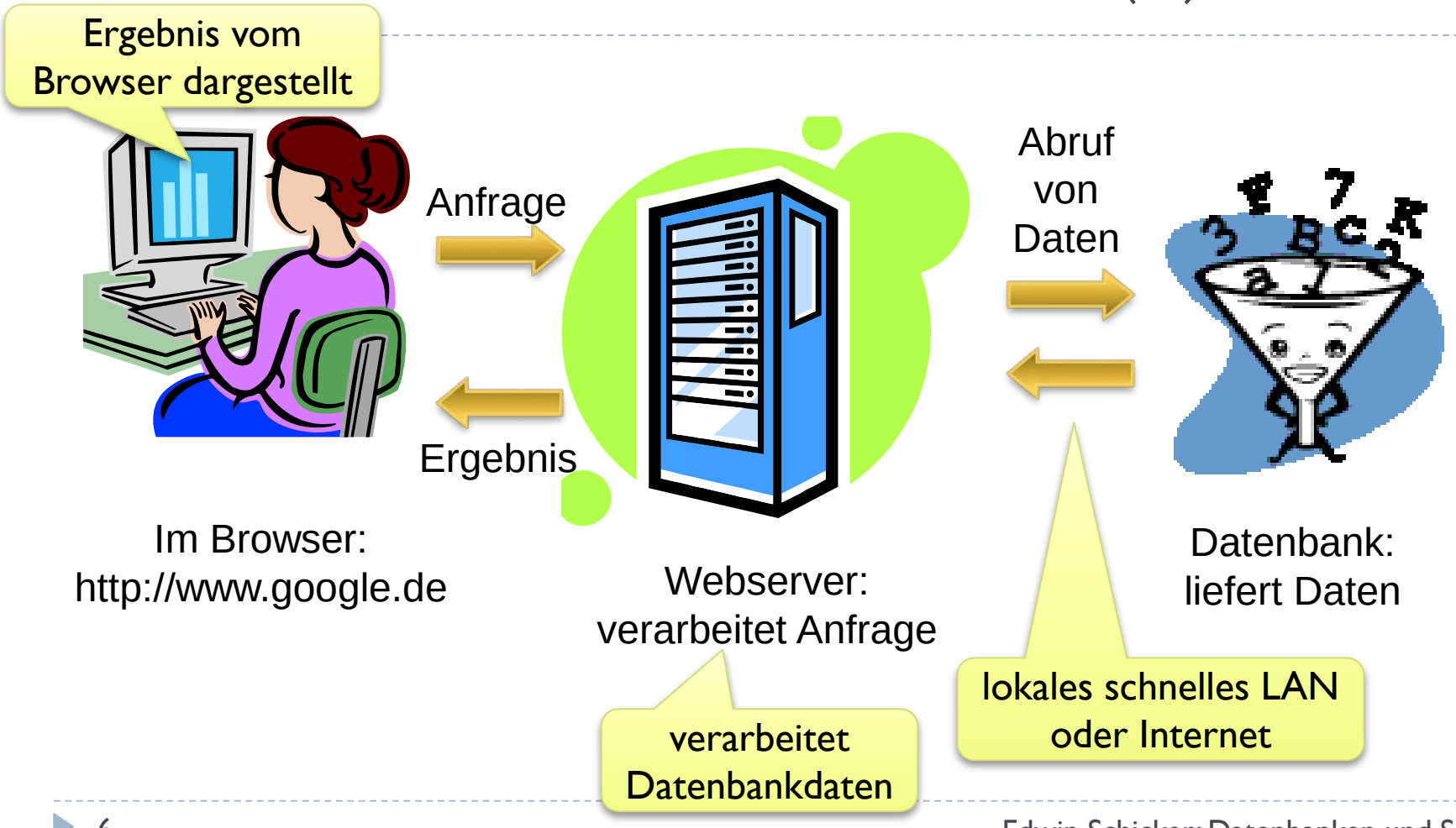
vom Name-Server umgesetzt in z.B.: 194.205.10.255

Webserver:
verarbeitet Anfrage

Webserver und Datenbanken (1)

- Webserver
  - Dynamische Dienste benötigen meist viele Daten
  - Daten werden in Dateien gespeichert oder in Datenbanken
  - Webserver greift über Datenbankschnittstellen zu
- Datenbank
  - Datenbank steht im Netz (Internet, Cloud)
  - Datenbank ist geschützt
  - Datenbank kann vom Webserver aufgerufen werden
  - Datenbank speichert neue Daten und gibt Daten zurück

Webserver und Datenbanken (2)



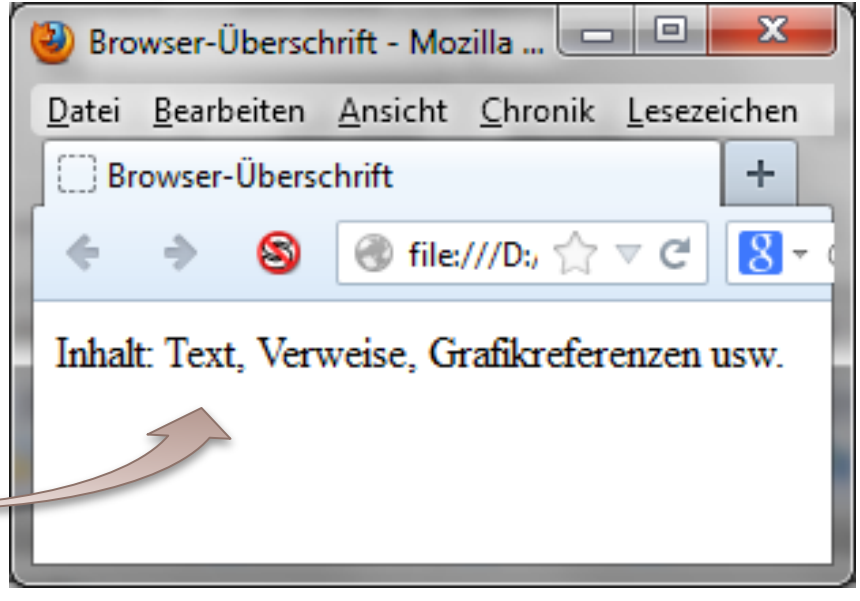
Aufbau einer HTML-Seite

- **Empfehlung:** http://de.selfhtml.org
- **Aufbau:**

```
<html>
  <head>
    <title>Browser-Überschrift</title>
  </head>
  <body>
    Inhalt: Text, Verweise, Grafikreferenzen usw.
  </body>
</html>
```

Tag

Ende-Tag



Einfache Textformatierung

- Der Browser bricht den Text automatisch um
- Wichtig sind also nur Formatangaben:
  - Absatzformatierungen, Überschriften, Fettdruck, …
- Beispiel:

```
<h1>Überschrift der Ordnung 1</h1>
<p>Text innerhalb eines Absatzes.</p>
<p> Beispiele für Texthervorhebung:
    <b>Fettdruck</b><br/>
    Normaltext <i>Kursivdruck</i><br/>
    Normal <u>Unterstrichener Text</u></p>
```

h: Hierarchy

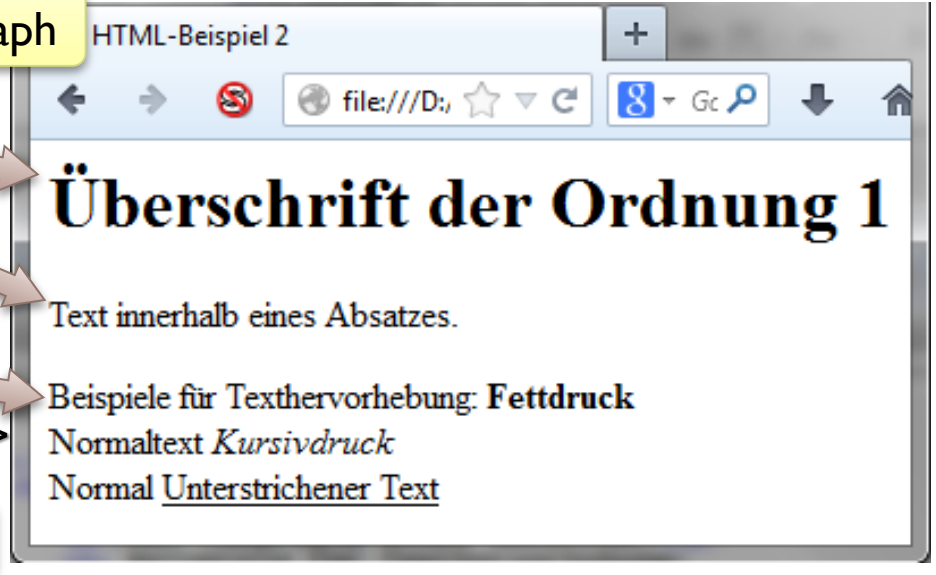
p: Paragraph

b: Bold

u: Underline

i: Italic

br: Break



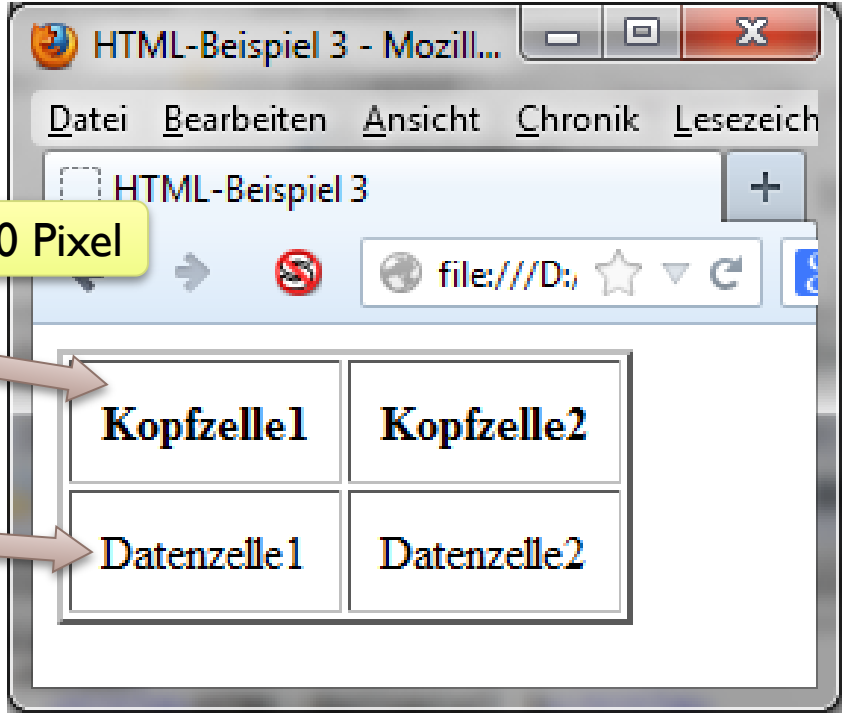
Tabellen in HTML

- Tabellen sind in HTML 4.0 ein Formatierungswerkzeug!
- Beispiel:

Randstärke: 2 Pixel

Textabstand: 10 Pixel

```
<table border="2" cellpadding="10">
 <tr>
  <th>Kopfzelle1</th>
  <th>Kopfzelle2</th>
 </tr>
 <tr>
  <td>Datenzelle1</td>
  <td>Datenzelle2</td>
 </tr>
</table>
```



Weitere Möglichkeiten

- **Links:**
  - Link auf andere Webseite:

    ```
    <a href="http://www.google.de">Google-Startseite</a>
    ```

    Linkadresse

    Hinweis zu Adresse
  - Link auf interne Seite *beispiel.html* im Unterverzeichnis *test*

    ```
    <a href="test/beispiel.html">Beispielseite</a>
    ```
- **Bild anzeigen:**
  - Bild *img.jpg* aus Verzeichnis *image* anzeigen:

    ```
    <img src="image/img.jpg" height="200" width="300"/>
    ```

    Link auf Bild

    Max. Höhe und Breite

    „/“ !
- **Manche Tags besitzen keine Endetags, z.B.
, <img/>**

  „/“ beachten!

Formulare in HTML

- Zu Formularen gibt es ein Form-Tag: <form … >
- Es gibt beispielsweise folgende Formularfelder:
  - Textfeld: <input type="text" … />
  - Radiobutton: <input type="radio" … />
  - Checkbox: <input type="checkbox" … />
  - Auswahlfeld: <select size="20" … > <option … > …
  - Combobox: <select size="1" … > <option … > …
  - Button: <input type="submit" … />
  - Reset-Button: <input type="reset" … />

falls >1

falls 1

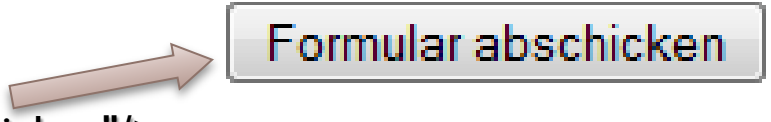
Funktionsweise von Formularen

```
<form action="start.php" method="post">

  <input …                              />

  …

  …

  …

  …

  <input type="submit" value="Formular abschicken"/>
</form>
```

Methode **post**: Interne Übergabe der Formularinhalte
Methode **get**: Übergabe der Formularinhalte in Browserzeile

… hier die Datei „start.php“ aufgerufen.

Beim Klick auf den Submit-Button wird …

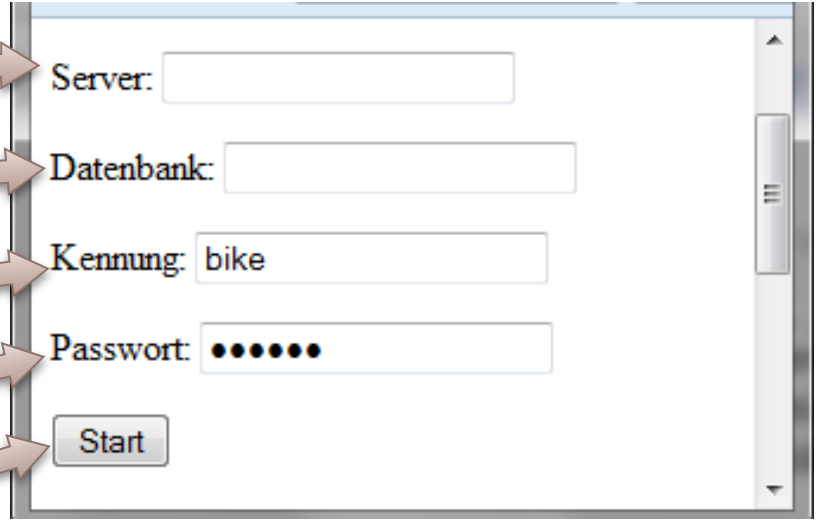


Beispiel zu Formularen (1)

Linkadresse

Methode post

```
<form action="start.php" method="post">
  <p>Server: <input type="text" size="20"
                    name="Server"/></p>
  <p>Datenbank: <input type="text" size="20"
                    name="Datenbank"/></p>
  <p>Kennung: <input type="text" size="20"
            name="Kennung" value="bike"/></p>
  <p>Passwort: <input type="password"
            size="20" name="Passwort"/></p>
  <p><input type="submit" value="Start"/></p>
</form>
```



Die Formatierung muss selbst gesetzt werden!

Beispiele zu Formularen (2)

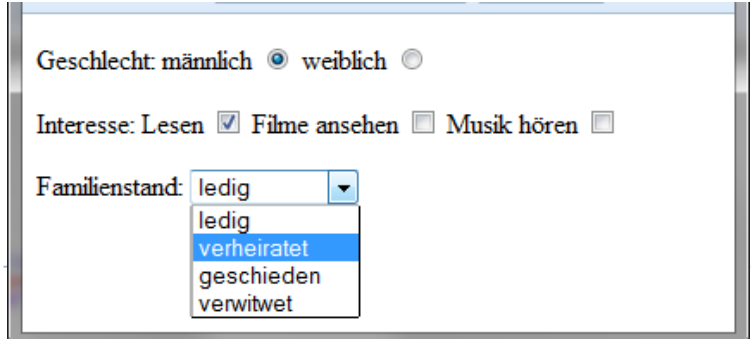
```
<p>Geschlecht:
   männlich <input type="radio" name="Anrede" value="Herr" checked />
   weiblich <input type="radio" name="Anrede" value="Frau"/></p>
<p>Interesse:
   Lesen <input type="checkbox" name="Interesse1" value="Buch" checked />
   Filme ansehen <input type="checkbox" name="Interesse2" value="Video"/>
   Musik hören <input type="checkbox" name="Interesse3" value="Audio"/> </p>
<p>Familienstand:
   <select name="Familienstand" size="1">
     <option> ledig </option>          <option> verheiratet </option>
     <option> geschieden </option>     <option> verwitwet </option>
   </select></p>
```

Vorab gesetzt

Vorab gesetzt

Combobox

Inhalt der Combobox



PHP

- PHP = PHP Hypertext Preprocessor
- 1995: von Rasmus Lerdorf vorgestellt
- PHP wird in HTML eingebettet
- PHP-Dateien besitzen die Endung „.php“
- PHP wird auf einem Webserver ausgeführt
- ~70% aller Serverprogramme sind in PHP geschrieben
- PHP ist stark an Perl, C und Java angelehnt

Gemeinsamkeiten: PHP und Java/C

- Blockstruktur, Anweisungen, Funktionen:
  - { … } Strichpunkt am Ende einer Anweisung
- Kontrollstrukturen:
  - if, switch
  - for, while, do … while

Identische Syntax!

- Operatoren:
  - nur minimale Unterschiede, zusätzliche Operatoren in PHP
  - Konkatenierungsoperator bei Strings: Punkt („.“)
- Groß- und Kleinschreibung wird unterschieden

Unterschied: PHP zu Java/C

	in PHP
Datentypen	Sechs Datentypen: int, bool, double, string, array, object
Variablen	Erstes Zeichen ist immer das Dollarzeichen (‚$‘)
Operatoren	Zusätzlich: ===, !==; Operator -> für Objektmethoden
Deklaration von Variablen	Deklaration nicht erforderlich; Variablen ändern den Datentyp dynamisch
Zeichenketten	Eigene Funktionalität, ähnlich zu Java
Funktionen	Funktionsnamen sind case-insensitiv

Wichtige Funktionen in PHP

trim(str)	entfernt Leerzeichen am Anfang und Ende von str
strlen(str)	gibt die Länge der Zeichenkette str zurück
strpos(str1,str2)	gibt das erste Vorkommen von str2 in der Zeichenkette str1 zurück; bzw. false, falls nicht enthalten
strcmp(str1,str2)	vergleicht die Zeichenketten str1 und str2
strcasecmp(str1,str2)	vergleicht die Zeichenketten str1 und str2 unabhängig von Groß- und Kleinschreibung
substring(str,pos,len)	gibt einen Teilstring von str der Länge len ab Position pos zurück
implode(str,feld)	liefert Zeichenkette mit allen Feldelementen zurück, die mittels der Zeichenkette str verknüpft werden
stripslashes(str)	entfernt Entwertungszeichen ‚\' in der Zeichenkette str
echo param1,…	gibt die Parameterliste auf HTML aus
isset(var)	liefert true, wenn Variable var existiert, sonst false
unset(var)	setzt die Variable var zurück, var existiert nicht mehr

Zeichenketten in PHP

- Zeichenketten entweder: "Dies ist ein String."
- Zeichenketten oder: 'Dies ist ein String.'
- Unterschied:
  - In ""-Strings werden Variablen substituiert
  - Beispiel:

```
$zahl = 100 ;
echo "<p>Die Zahl $zahl ist größer als Null</p>";
echo '<p>Die Zahl $zahl ist größer als Null</p>';
```

Hier wird $zahl durch Inhalt ersetzt

Gänsefüßchen

Hochkomma

hier nicht

  - Ausgabe:

```
Die Zahl 100 ist größer als Null
Die Zahl $zahl ist größer als Null
```

Formulare und PHP

- Alle Formularfelder besitzen den Parameter name
- Die jeweiligen Formularinhalte werden mit diesem Parameter identifiziert.
- Beispiel:

```
<form action="start.php" method="post">
<p>Server: <input type="text" size="20" name="Server"/></p>
```

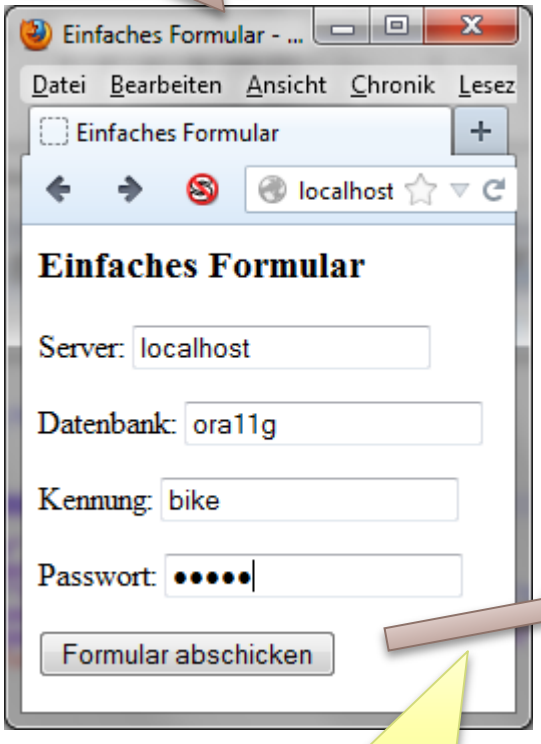
- In der Datei start.php ist der Inhalt verfügbar unter:

```
$_POST[ 'Server' ]
```

  - Analoges gilt für Methode get: $_GET['Server']

Formulare mit PHP auslesen

start.html



per POST übergeben

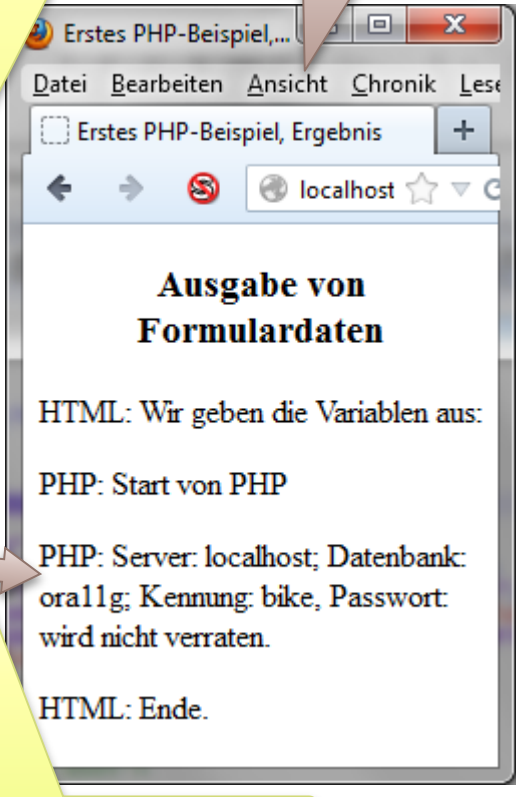
```
<h3>Ausgabe von Formulardaten </h3>
HTML: Wir geben die Variablen aus:
<?php
  echo "<p>PHP: Start von PHP</p>";
  $server = $_POST['Server'];
  $datenbank = $_POST['Datenbank'];
  $benutzer = $_POST['Kennung'];
  $passwort = $_POST['Passwort'];
  echo "<p>PHP: Server: $server;
     Datenbank: $datenbank;
     Kennung: $benutzer,
     Passwort: wird nicht verraten.</p>";
?>
<p>HTML: Ende.</p>
```

PHP-Start

PHP-Ende

POST-Variablen

start.php



Variablen ausgeben

Felder in PHP (1)

- **Felder in PHP flexibel: Felder, Listen, Aufzählungen!**
- **Beispiele:**

```
$feld = array( 0, 2, 4, 6, 8 );
$feld[ ] = 10;
$feld[10] = 20;
$feld["test"] = 100;
$feld["wert"] = "Ende";
echo "<p>Anzahl der Feldelemente: " . count($feld) . "</p>
        <p>Inhalt:</p>";
foreach ($feld as $inhalt)
   echo "<p>$inhalt</p>";
```

Feld mit 5 Elementen: $feld[0]=0, $feld[1]=2 usw.

Nächstes Feld: $feld[5]=10

$feld[6] existiert nicht, ebenso nicht 7, 8 und 9!

Assoziative Felder: Indexe sind Strings

Konkatenierung

Anzahl der Feldelemente: 9

Ausgabe aller Werte mit foreach

Felder in PHP (2)

- Felder besitzen einen Index (Schlüssel, key) und einen Inhalt (context). Es gibt den Zuordnungsoperator „=>“
- Also: key => context (z.B. 4=>8)
- Beispiele:

Beginnend bei 0, dann 1 usw.

feld[10]=20

feld["test"]=100

```
$feld = array( 0=>0, 2, 4, 6, 8, 10, 10=>20, "test"=>100, "wert"=>"Ende" );
echo "<p>Anzahl der Feldelemente: " . count($feld) . "</p>
        <p>Inhalt:</p>";
foreach ($feld as $key => $content)
  echo "<p>feld[$key] = $content</p>";
```

Ausgabe in Schleife:
feld[0]=0
feld[1]=2
feld[2]=4
usw.

PHP auf Datenbankzugriff vorbereiten

- Einstellungen in php.ini (je nach Hersteller):
  - extension=php_pdo_oci.dll
  - extension=php_pdo_mysql.dll
  - extension=php_pdo_sqlsrv_54_nts.dll
- Microsoft liefert eigene Treiber (z.B. für PHP 5.4)
- Im Unterverzeichnis ext von PHP müssen diese Treiber installiert sein!
- Die SQL Server Treiber werden von PHP nicht mit geliefert. Download von Microsoft!

Schnittstelle: PHP ←→ Datenbank (1)

- Datenbankunabhängige Programmierung mit PDO
  - PDO = PHP Data Objects
- Konstruktor der Klasse PDO stellt Verbindung her:

```
$conn = new PDO("oci:dbname=$datenbank",
                $kennung, $passwort);
$conn = new PDO("sqlsrv:server=$server; dbname=$datenbank",
                $kennung, $passwort);
$conn = new PDO("mysql:host=$server;dbname=$datenbank",
                $kennung, $passwort);
```

3. Parameter: Passwort

2. Parameter: Kennung

1. Parameter: Verbindung

Variable SERVER/HOST

Variable DBNAME

Schnittstelle: PHP ←→ Datenbank (2)

- Variable SERVER/HOST: Internetadresse der Datenbank
- Variable DBNAME: Name der Datenbank
- Variable KENNUNG und PASSWORT: Identifikation

	Variable Server/Host	Variable Dbname
Oracle lokal	(nicht verwendet)	Datenbank
Oracle entfernt	(nicht verwendet)	//Serveradresse/Datenbank
SQL Server lokal	localhost\sqlexpress	Datenbank
SQL Server entfernt	Serveradresse\sqlexpress	Datenbank
MySQL lokal	localhost	Datenbank
MySQL entfernt	Serveradresse	Datenbank

tatsächlichen Installationsnamen verwenden

Erster Zugriff auf Datenbank

- **Einloggen und Transaktionsstart:**

```
$conn = new PDO("oci:dbname=$_POST[ Datenbank ]",
                                    $_POST['Kennung'], $_POST['Passwort']);
echo "<p>Die Verbindung zur Datenbank wurde hergestellt.</p>";

$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$conn->setAttribute(PDO::ATTR_CASE, PDO::CASE_UPPER);

$conn->beginTransaction();
```

Kein Hochkomma!

Einloggen am Beispiel Oracle

Bei Fehler: Ausnahme werfen

Spaltennamen in Großbuchstaben ausgeben

Aus Kompaktibilitätsgründen (Oracle)

Objekt -> Methode

1. Parameter: Attribut
2. Parameter: Wert

Wichtig!
Startet Transaktion in PDO

Ausführen eines Select Befehls

- **Voraussetzung:**
  - Verbindung mit Datenbank ist hergestellt (siehe letzte Folie)
- **Beispiel:**
  - Lesen des Namens und Wohnorts von Mitarbeiter 2

```
$sql = " Select Name, Ort From Personal Where Persnr = 2 " ;

$stmt = $conn->query($sql);
```

Oracle mag keinen Strichpunkt!!

Abspeichern in Objekt vom Typ PDOStatement

SQL-Befehl ausführen

Auslesen des Ergebnisses

- Voraussetzung:
  - Ergebnis steht in Objekt ($stmt) vom Typ PDOStatement
- Beispiel:
  - Ausgabe des Namens und Wohnorts von Mitarbeiter 2

```
if ($row = $stmt->fetch())
{
    echo "<p>Der Mitarbeiter mit der Persnr 2 heißt
$row[ NAME ] und wohnt in $row[ ORT ]. </p>";
}
else
{
    echo "<p>Der Mitarbeiter mit dieser Nummer 2 existiert nicht! </p>";
}
```

Lesen der 1. Zeile aus $stmt
Abspeichern im assoziativen Feld $row

kein Hochkomma

Strings dürfen mehrzeilig sein!

falls keine 1. Zeile existiert, liefert fetch false zurück

Assoziativer Wert NAME (Großbuchstaben!)

Analog: ORT

Hinweise zum Auslesen

- Assoziative Namen beginnen mit einem Buchstaben
- Assoziative Namen enthalten keine Sonderzeichen
  - Also: Im Select-Befehl gegebenenfalls Aliasnamen verwenden!
- Die Methode fetch liefert assoziative und indizierte Werte
- In unserem Beispiel erhalten wir:

```
$row[ 'NAME' ]      // da Select Name, …
$row[ 'ORT' ]       // da Select …, Ort, …
$row[ 0 ]           // erste Spalte, entspricht $row[ 'NAME' ]
$row[ 1 ]           // zweite Spalte, entspricht $row[ 'ORT' ]
```

anfällig gegen spätere Änderungen

Beenden einer Transaktion / Sitzung

- Eine Transaktion wird beendet mit den Methoden:
  - commit // übernimmt alle Daten dauerhaft
  - rollback // setzt alle Änderungen zurück
- Eine Sitzung wird beendet
  - durch Rücksetzen der Sitzungsvariable (z.B. $conn = false;)
- Am Ende einer PHP-Datei erfolgen folgende Aktionen:
  - Alle noch offenen Transaktionen werden zurückgesetzt (Rollback)
  - Alle noch offenen Verbindungen werden beendet
- Unser Beispiel:

Führt Commit durch

Beendet die Verbindung/Sitzung

```
$conn->commit();
$conn = false;
echo "<p>Die Verbindung zur Datenbank wurde geschlossen. </p>";
```

Einführung in die Fehlerbehandlung

- In PDO: Fehlerklasse PDOException
- Methoden der Fehlerklassen Exception und PDOException:
  - getMessage — gibt einen Fehlertext aus
  - getCode — gibt den Fehlercode aus
  - getLine — gibt Fehlerzeile aus
- PDO wirft Fehler vom Typ PDOException, falls:
  - PDO::ATTR_ERRMODE ist PDO::ERRMODE_EXCEPTION
  - Wird nach Verbindungsaufbau mit Methode setAttribute gesetzt
- Trennung von Code und Fehlercode:
  - Try-Block enthält den Code
  - Catch-Blöcke fangen gegebenenfalls geworfene Fehler ab

Fehlerbehandlung am Beispiel

```
try {
    $conn = new PDO("oci:dbname=$_POST[Datenbank]", $_POST['Kennung'],
                                                    $_POST['Passwort']);
      // ...     Zugriffe
   $conn->commit();
   echo "<p>Die Verbindung zur Datenbank wird geschlossen. </p>";
}
catch (PDOException $e) {
   echo "<p>PDO-Fehler in Zeile ", $e->getLine(), "mit Code ", $e->getCode(),
                                  "</p><p>Fehlertext: ", $e->getMessage(), "</p>";
}
catch (Exception $e) {
   echo "<p>Fehler in Zeile ", $e->getLine(), "mit Code ", $e->getCode(),
                                  "</p><p>Fehlertext: ", $e->getMessage(), "</p>";
}
```

Normaler Programmcode, nicht belastet durch Fehlerbehandlungen

Fängt zunächst alle Datenbankfehler ab

Von Exception abgeleitete Klasse

Fängt noch alle anderen PHP-Fehler ab

Standardfehlerklasse in PHP

Hinweise zur Fehlerbehandlung

- Im Fehlerfall wird in den ersten zutreffenden Catch-Block gesprungen
- Alle im dazugehörigen Try-Block definierten Variablen sind dann nicht mehr gültig. Im Beispiel:
  - $conn existiert im Catch-Block nicht mehr
  - Die Verbindung wurde also aufgelöst!
  - Die Transaktion wurde also zurückgesetzt!
  - Ein explizites „$conn->rollback()“ führt im Catch-Block zu einem Fehler und damit zum Absturz des Programms
- Wir verwenden im Catch-Block ausschließlich die Methoden der Klassen Exception und PDOException

Auslesen mehrerer Datenzeilen

- Programmidee:
  - Eingabe eines Suchstrings
  - Ausgabe aller Mitarbeiter, die Suchstring im Namen enthalten
- Realisierung:
  - Komfort: Unterstützung der wiederholten Eingabe eines Suchstrings ohne erneute Eingabe der Anmeldedaten
  - Ein- und Ausgabe im gleichen PHP-Programm
- Problem:
  - Beim ersten Aufruf des PHP-Programms ist ein Einloggen noch nicht möglich, da die Anmeldedaten erst einzugeben sind!
- Lösung:
  - Funktion isset
  - Funktion überprüft, ob Variable schon existiert

Anwendung der Funktion isset

sich selbst aufrufen:

```
<form action="mitarbeiter.php" method="post">
  <p>Bitte Daten zum Einloggen eingeben</p>
  …          <!-- Server- und Datenbankdaten -->
  <p>Kennung eingeben:</p>
  <input type="Text" name="Kennung" size="20" value=
        <?php echo isset($_POST['Kennung']) ? $_POST['Kennung'] : "bike" ; ?> />
  <p>Passwort eingeben:</p>
  <input type="Password" name="Passwort" size="20" value=
        <?php echo isset($_POST['Passwort']) ? $_POST['Passwort'] : "" ; ?> />
  <p>Suchzeichen zur Mitarbeitersuche:</p>
  <input type="Text" name="Suchstring" size="30" value=
        <?php echo isset($_POST['Suchstring']) ? $_POST['Suchstring'] : "" ; ?> />
  <input type="Submit" value="Weiter" />
</form>
```

Beim ersten Aufruf gibt es diese POST-Variable noch nicht

1. Aufruf: Vorgabe „bike“

Ab 2. Aufruf: Letzter Wert

Problem: Passwort ist sichtbar!

Mit Suchstring: Suche von Mitarbeitern

```
if (isset($_POST['Kennung']) {
   $suche = trim($_POST['Suchstring']);  // ev. Leerzeichen entfernen
   if (strlen($suche) == 0) {
     echo "<p>Es wurde kein Suchstring eingegeben. </p>";
   } else {
     try {
       echo "Suche nach Mitarbeitern, die im Namen den String $suche enthalten.";


       $sql = "Select Persnr, Name, Ort, GebDatum, Gehalt, Vorgesetzt
                From Personal
               Where Upper( Name )  Like  Upper( '%$suche%' ) " ;
       $stmt = $conn->query( $sql );
```

Code wird erst ab Zweitaufruf durchlaufen

eventuelle Leerzeichen entfernen

Bei leerer Eingabe reagieren

hier: Einloggen, setAttribute, beginTransaction

Upper: Unabhängig von Groß- und Kleinschreibung

Suche mit Like und Wildcart

Ausgabe in eine Tabelle (1)

```
if ( !($row = $stmt->fetch()) ) {
   echo "Mitarbeiter mit dem Teilstring im Namen existiert nicht!";
}
else {
?>                          <!-- Tabelle aufbauen -->
   <table border cellpadding=10>
     <tr>  <!-- Erste Tabellenzeile -->
       <th>Persnr </th>
       <th>Name </th>
       <th>Ort </th>
       <th>GebDatum </th>
       <th>Gehalt </th>
       <th>Vorgesetzter? </th>
     </tr>
```

Überprüfen, ob Ergebnisse vorliegen

wenn nein: darauf hinweisen

wenn ja: Tabelle aufbauen

Erste Zeile (Überschriftenzeile) ausgeben

Ausgeben in eine Tabelle (2)

```
<?php
    do {
?>  <tr>
      <td> <?php echo $row["PERSNR"] ?> </td>
      <td> <?php echo $row["NAME"] ?> </td>
      <td> <?php echo $row["ORT"] ?> </td>
      <td> <?php echo $row["GEBDATUM"] ?> </td>
      <td> <?php echo $row["GEHALT"] ?> </td>
      <td> <?php echo ($row["VORGESETZT"] == null) ? "Ja": "Nein"; ?></td>
    </tr>
<?php
    } while ($row = $stmt->fetch());
?></table>
```

Mit do-while-Schleife Daten auslesen

Neue Zeile

Alle Spalten ausgeben

Überprüfen auf NULL

solange auslesen, solange Daten vorhanden sind

Beenden mit Commit

Typisches Auslesen von Daten (1)

- **Variante 1:**
  - Mittels Fetch in einer While-Schleife
  - Keine Überprüfung der Daten vor der Schleife

```
while ( $row = $stmt->fetch() )
{
        /* Ausgabe der Attribute der aktuellen Zeile
                mit Hilfe des assoziativen Feldes $row    */
}
```

Typisches Auslesen von Daten (2)

- **Variante 2:**
  - Mittels Fetch, If-Anweisung und Do-While-Schleife
  - Überprüfen der Daten vor der Schleife

```
if ( !($row = $stmt->fetch()) )
    echo "<p>Keine Daten</p>";
else
do {
   // Ausgabe der Attribute ab der ersten Zeile mit $row
} while ( $row = $stmt->fetch() ) ;
```

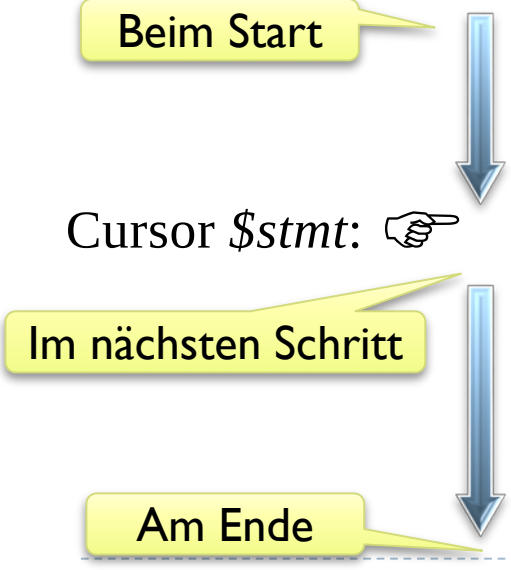
Vorab wird erste Zeile überprüft

Jetzt muss erst die erste Zeile ausgegeben werden, bevor weitere Daten gelesen werden

also: Do-While

Auslesen mit Fetch: CURSOR

- Vor dem ersten Fetch-Aufruf zeigt Cursor vor die 1. Zeile
- Bei jedem Aufruf wird betreffende Zeile gelesen, und der Cursor geht zur nächsten Zeile
- Zeigt Cursor hinter die letzte Zeile, liefert der Folgebefehl false



1	Maria Forster	Regensburg	05.07.79	JA
2	Anna Kraus	Regensburg	09.07.75	NEIN
3	Ursula Rank	Frankfurt	04.09.67	NEIN
4	Heinz Rolle	Nürnberg	12.10.57	NEIN
5	Johanna Köster	Nürnberg	07.02.84	NEIN
6	Marianne Lambert	Landshut	22.05.74	JA
7	Thomas Noster	Regensburg	17.09.72	NEIN
8	Renate Wolters	Augsburg	14.07.79	NEIN
9	Ernst Pach	Stuttgart	29.03.92	NEIN

Sessionvariable

- Sessionvariable gelten über PHP-Seiten hinweg
- Damit können Daten eingelesen werden und über viele Seiten verwendet werden
- Feld mit dem Namen $_SESSION:
  - $_SESSION['Name der Variable']
  - Analoges Handling wie $_POST, $_GET
- Voraussetzung:
  - PHP-Seite muss in Zeile 1 als Sessionseite definiert werden mit:

```
<?php
        session_start();
?>
```

Beispiel mit Sessionvariablen

```
$_SESSION[ 'Kennung' ] = "abc12345";
$_SESSION[ 'Nr' ] = 17;

if (isset($_SESSION[ 'Kennung' ]))
{
   echo "Variable Kennung existiert, wird nun wieder entfernt.";
   unset($_SESSION[ 'Kennung' ]);
}
echo "<p>Die Sessionvariable Nr hat den Wert:
                                    $_SESSION[ Nr ].</p>";
```

Zwei Sessionvariablen erzeugt

Überprüfen, ob Sessionvariable Kennung existiert

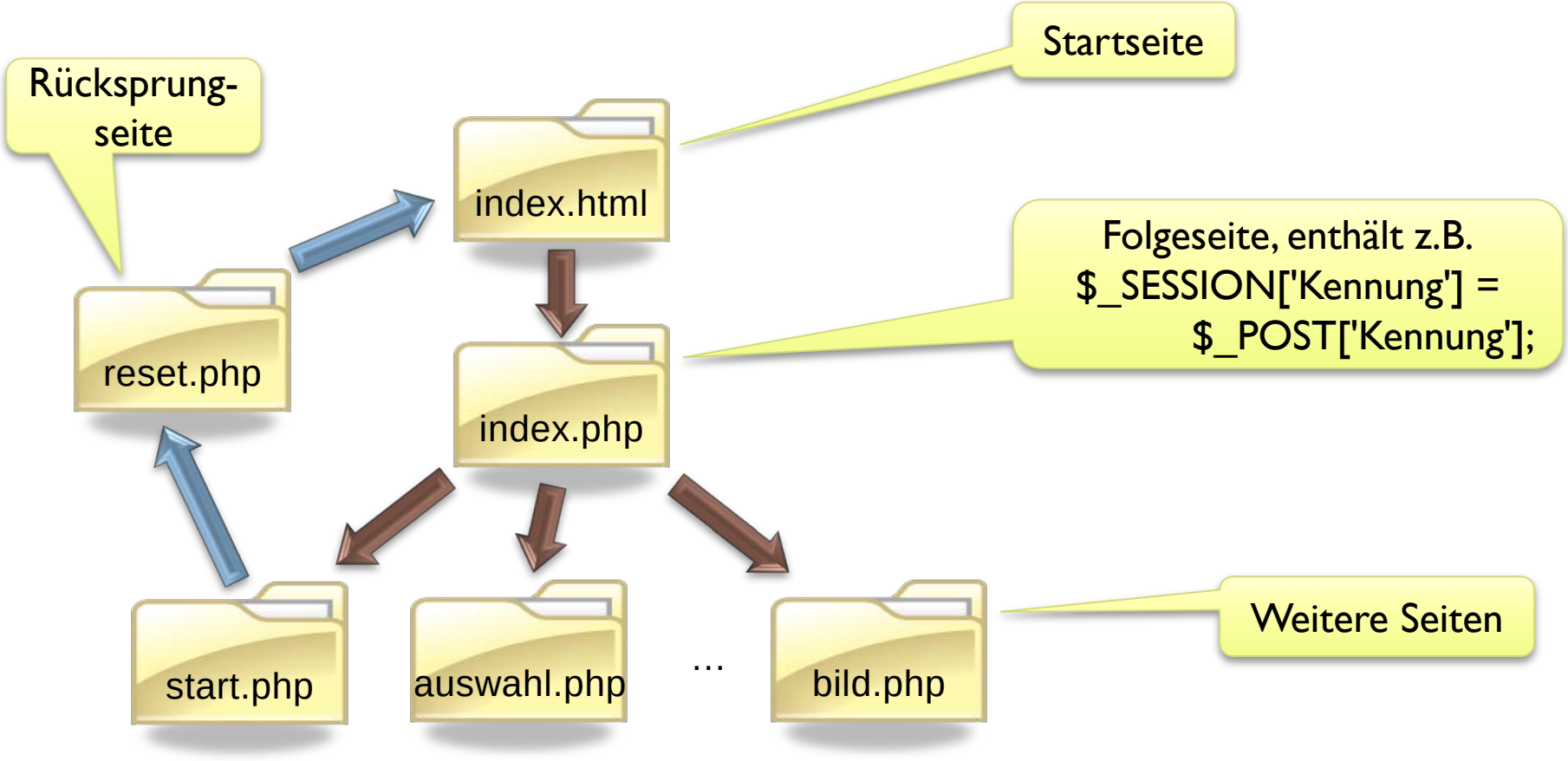
Sessionvariable Kennung existiert nun nicht mehr

Inhalt einer Sessionvariable ausgeben

Arbeiten mit Datenbanken und Session

- Startseite:
  - Benutzerdaten anfordern (z.B. Kennung und Passwort)
- Folgeseite:
  - Benutzerdaten aus POST-Variablen auslesen und in SESSION-Variablen speichern
- Weitere Seiten:
  - SESSION-Variablen verwenden (z.B. zum Einloggen in Datenbank)
  - Um Quereinstieg zu unterbinden: Existenz der SESSION-Variablen überprüfen
- Rücksprungseite
  - Zurücksetzen der SESSION-Variablen

Beispiel zu Session



Handling der weiteren Seiten

- **Verhindern des Quereinstiegs mit Vermeidung von Folgefehlern:**

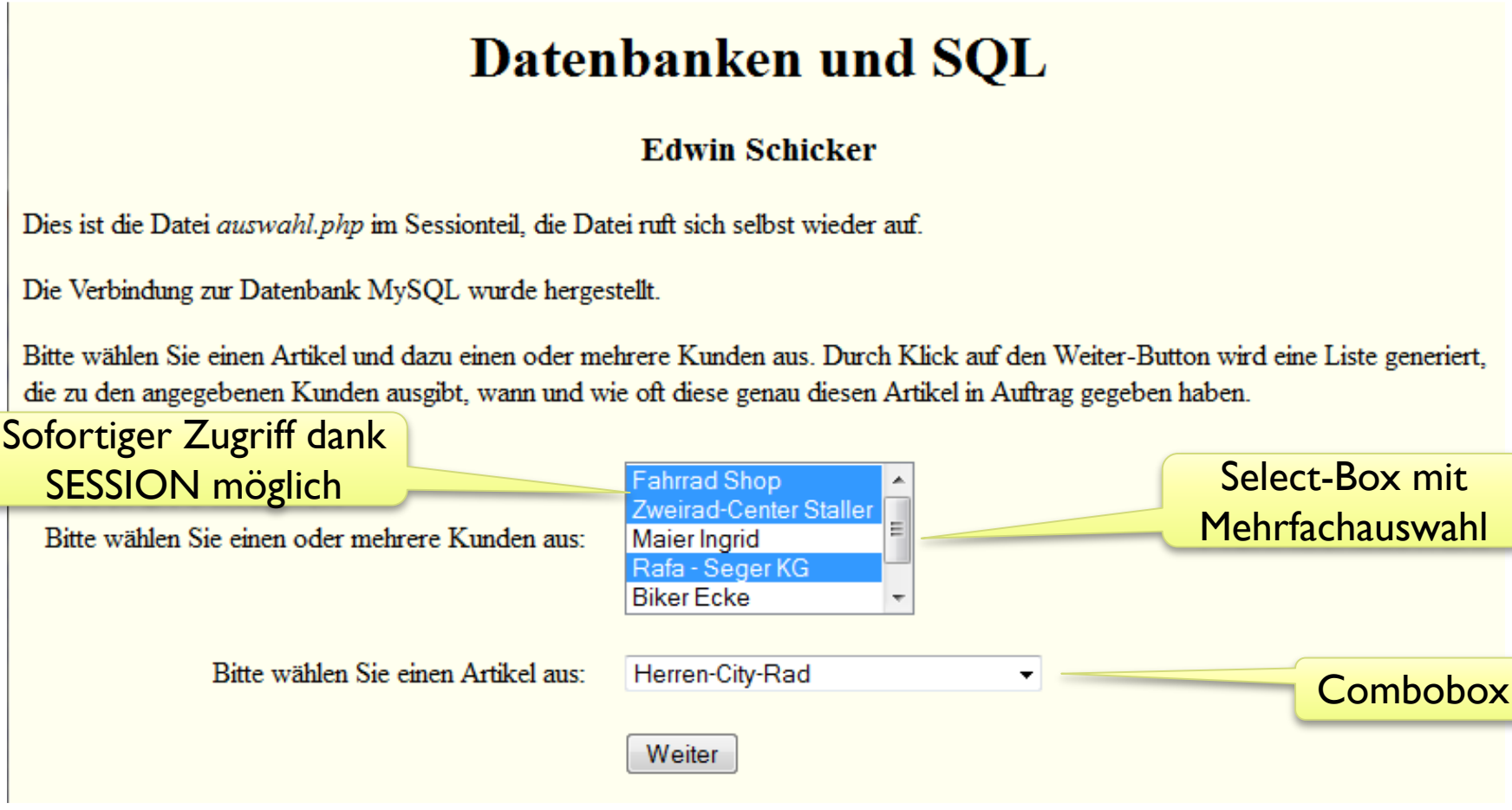
```
if ( !isset($_SESSION['Kennung']) ||
     !isset($_SESSION['Passwort']) ||
     !isset($_SESSION['Server']) ||
     !isset($_SESSION['Datenbank'])  )
   echo "Gehen Sie zur <a href="start.html">Startseite</a>");
else
{       $conn = new PDO( …
```

Überprüfung, ob SESSION-Variablen gesetzt

Wenn nein, dann zurück zur Startseite

Wenn ja, dann zugreifen

Auswahl mittels Select-Boxen



Ausgabe der Artikel in Combo-Box

```
$sql2 = "Select Anr, Bezeichnung From Artikel";
$stmt2 = $conn->query($sql2);

if ( $row2 = $stmt2->fetch() )
{  echo "<select name=\"Artikelnr\" size=\"1\">" ;

   do
   {  echo "<option value=\"$row2[ANR]\"";
      if (isset($_POST['Artikelnr']) and $_POST['Artikelnr'] == $row2['ANR'])
         echo " selected";
      echo "> $row2[BEZEICHNUNG]</option>";
   } while ($row2 = $stmt2->fetch());

   echo "</select>";
} else …
```

Alle Artikel lesen

Existieren Artikel?

Combobox: size=1

… aber Artikelnummern merken

Vorherige Auswahl merken

Artikelnamen anzeigen …

In Schleife alles ausgeben

Ausgabe der Kunden in Select-Box (1)

```
$sql1 = "Select Nr, Name From Kunde";
$stmt1 = $conn->query($sql1);

if ( $row1 = $stmt1->fetch() )
{  echo "<select multiple name=\"Kundnr[]\" size=\"5\">" ;
```

Alle Kunden lesen

Select-Box mit 5 Einträgen

Mehrfachauswahl

Feldübergabe

- Problem: Mehrfachauswahl
  - Lösung: Übergabe eines Feldes im Parameter Name
  - Im Feld werden nur alle angeklickten Elemente aufgenommen
- Problem: Wie merken wir uns die angeklickten Kunden?

Ausgabe der Kunden in Select-Box (2)

```
if (isset($_POST['Kundnr']))
   $alleKunden = implode( "  ", $_POST['Kundnr'] );
else
   $alleKunden ="";

do {
   echo "<option value=\"$row1[NR]\"" ;
   if (strpos($alleKunden, $row1['NR']) !== false) echo " selected";
   echo "> $row1[NAME]</option>";
} while ($row1 = $stmt1->fetch());

echo "</select>";
```

Trennzeichen zwischen Elementen

Implode: Fasst alle Feldelemente zu einem String zusammen

… aber Kundennummer merken

!==

Auswählen, falls Kundnr im String enthalten

Kundenname anzeigen …

Ausgabe der dazugehörigen Aufträge

- Die markierten Angaben sind als Tabelle auszugeben
- Die Ausgabe in eine Tabelle wurde bereits behandelt

Alle Auftragsdaten …

```
$sql = "Select A.Auftrnr, Kundnr, Datum, Persnr, Anzahl, Gesamtpreis
        From Auftrag A Inner Join Auftragsposten AP
                                        On A.Auftrnr=AP.Auftrnr
        Where  Kundnr  In (" . implode( "," ,  $_POST['Kundnr']) . ")
          And  Artnr = $_POST[Artikelnr]";

$stmt = $conn->query($sql);
```

… für die ausgewählten Kunden …

… und den Artikel

Implode: Fasst Feldelemente zu einer durch Komma getrennten Aufzählung zusammen

Verwendung einer Textarea

- **Textarea:**
  - Mehrzeiliges Eingabefeld
- **Beispiel:**

```
<form action="select.php" method="post">
  <textarea name="Eingabe" rows="10" cols="60" wrap="virtual" >
  <?php
    if (isset($_POST['Eingabe']))
        echo trim(htmlspecialchars(stripslashes($_POST['Eingabe'])));
  ?>
  </textarea><br/>
  <input type="Submit" value="Ausführen"/>
</form>
```

10 Zeilen à 60 Spalten

Automatischer Umbruch

Ab 2. Aufruf letzte Eingabe anzeigen

Verhindert nicht erwünschten HTML-Code

Entfernt Entwertungszeichen

Auswerten beliebiger SQL-Befehle

- **Befehl hängt vom ersten Wort ab**
  - Also: Erstes Wort der Eingabe isolieren
- **Realisierung:**

```
$Eingabe = trim(stripslashes($_POST['Eingabe']));
$stmt = $conn->query($Eingabe);
$pos = strpos($Eingabe, " ");
$erstesWort = substr($Eingabe, 0, $pos);
```

Eingabe bereinigen

SQL-Befehl abschicken

Erstes Leerzeichen suchen, eventuell zusätzlich: \n, \r, \t

Erstes Wort geht bis zum ersten Leerzeichen (bzw. \n, \t, \r)

Falls erstes Wort: SELECT

```
if (strCaseCmp($erstesWort,"SELECT")==0) {
  if ( $row = $stmt->fetch(PDO::FETCH_ASSOC) )  {
    echo "<table border=\"2\" cellpadding=\"2\"><tr>";
    foreach ( $row as $colname => $data )
      echo "<th> $colname </th>";
    echo "</tr>";
    do {
      echo "<tr>";
      foreach ( $row as $data )
        echo "<td>" . ($data == null ? "(null)" : $data) . "</td>";
      echo "</tr>";
    }  while ( $row = $stmt->fetch(PDO::FETCH_ASSOC) ) ;
    echo "</table>";
```

Select-Befehl

nur assoziative Werte, keine Index-Werte!

Key-Werte der ersten Zeile lesen, um Spaltennamen auszugeben

Schleife über alle Zeilen

Schleife über alle Spalten: Daten ausgeben

falls NULL: (null) ausgeben

Falls anderer Befehl

- Kein Select, also: keine Ergebnisausgabe
- DML-Befehl: Ausgabe der Anzahl der manipulierten Zeilen

Vergleich unabhängig von Groß- und Kleinschreibung

```
elseif (strCaseCmp($erstesWort,"INSERT")==0)
   echo $stmt->rowCount(), " Zeile(n) wurde(n) eingefügt.<br>";
elseif (strCaseCmp($erstesWort,"UPDATE")==0)
   echo $stmt->rowCount(), " Zeile(n) wurde(n) geändert.<br>";
elseif (strCaseCmp($erstesWort,"DELETE")==0)
   echo $stmt->rowCount(), " Zeile(n) wurde(n) gelöscht.<br>";
else
   echo "Ein DDL-Befehl wurde ausgeführt.<br>";
```

Anzahl der manipulierten Zeilen

Klasse PDOException

Eigenschaft/Methode	Beschreibung
getMessage( )	gibt eine ausführliche Fehlermeldung zurück
getCode( )	gibt die Fehlernummer zurück (SQLSTATE-Code)
getLine( )	gibt Fehlerzeile zurück
$errorInfo	Feld mit 3 Indizes: 0: Fehlercode (SQLSTATE-Code) 1: Fehlercode des Herstellers 2: Ausführliche Fehlermeldung

bekannt

Normierter Fehlercode

entspricht getCode

entspricht getMessage

In Oracle Zugriff auf interne Codes erforderlich, da Fehlercode meist "HY000"

SQLSTATE

Code	Ursache
'00'	Erfolgreiche Beendigung
'01'	Warnung
'02'	Daten nicht gefunden
'08'	Verbindungsaufbau-Fehler
'0A'	Merkmal wird nicht unterstützt
'22'	Datenfehler (z.B. Division durch Null)
'23'	(Tabellen/Spalten-)Bedingung ist verletzt
'24'	Ungültiger Cursor-Status
'25'	Ungültiger Transaktions-Status
'2A'	SQL-Syntax- oder Zugriffsfehler
'2B'	Abhängiges Privileg existiert
'2D'	Nichterlaubte Transaktionsbeendigung
'34'	Ungültiger Cursorname
'3D'	Ungültiger Katalogname
'3F'	Ungültiger Schemaname
'40'	Rollback
'42'	Syntax- oder Zugriffsfehler
'44'	Check-Bedingung ist verletzt

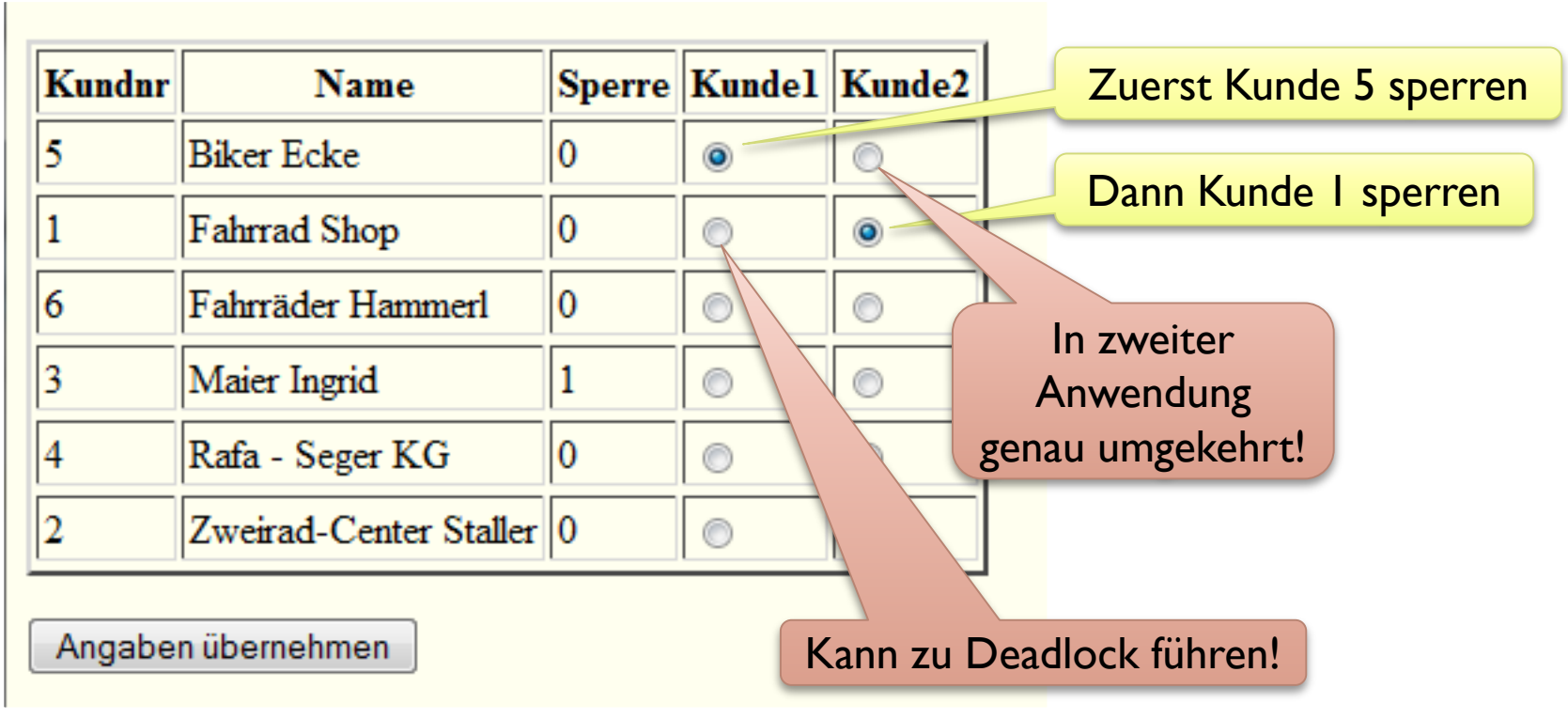
- SQLSTATE:
  - 5 stelliger String
  - Erste 2 Zeichen
    - Hauptcode
  - Weitere 3 Zeichen
    - Subcode
  - z.B. Erfolg:
    - "00000"
  - z.B. Deadlock:
    - "40001"

Beispiel: Transaktionsbetrieb

- Provozieren von Deadlocks (SQLSTATE: 40001)
- Deadlock:
  - Eine Verklemmung, bei der zwei oder mehr Transaktionen gegenseitig auf die Freigabe eines oder mehrerer Locks warten
- Deadlockbehandlung im Catch-Teil
- Realisierung:
  - 2 Internetanwendungen ändern Daten
  - Dabei holen die Transaktionen Locks
  - Bei ungeschickter Anforderung der Daten: Deadlock

Beispielanwendung

- Attribut Sperre in Relation Kunde manipulieren



Realisierung

```
$sql = "Update Kunde
            Set Sperre = Sperre + 1
            Where Nr = $_POST[Kunde1]";
$stmt1 = $conn->query($sql);

ob_flush();  flush();
sleep(10);

$sql = "Update Kunde
            Set Sperre = Sperre + 1
            Where Nr = $_POST[Kunde2]";
$stmt2 = $conn->query($sql);
```

Daten des ersten Kunden ändern

Änderungen sofort auf Browser anzeigen

10 Sekunden warten, um parallele Anwendung zu starten

Daten des zweiten Kunden ändern

Auftreten eines Deadlocks

- Anwendung 1:
  - Kunde1: Kundnr = 5
  - Kunde2: Kundnr = 1
  - Sperre für Kundnr=5, Warten von 10 sec., Sperre für Nr 1 besetzt
- Anwendung 2:
  - Kunde1: Kundnr = 1
  - Kunde2: Kundnr = 5
  - Sperre für Kundnr=1, Warten von 10 sec., Sperre für Nr 5 besetzt
- Beide warten auf Freigabe einer Sperre → DEADLOCK
- Voraussetzung: Anwendungen starten innerhalb von 10 sec.

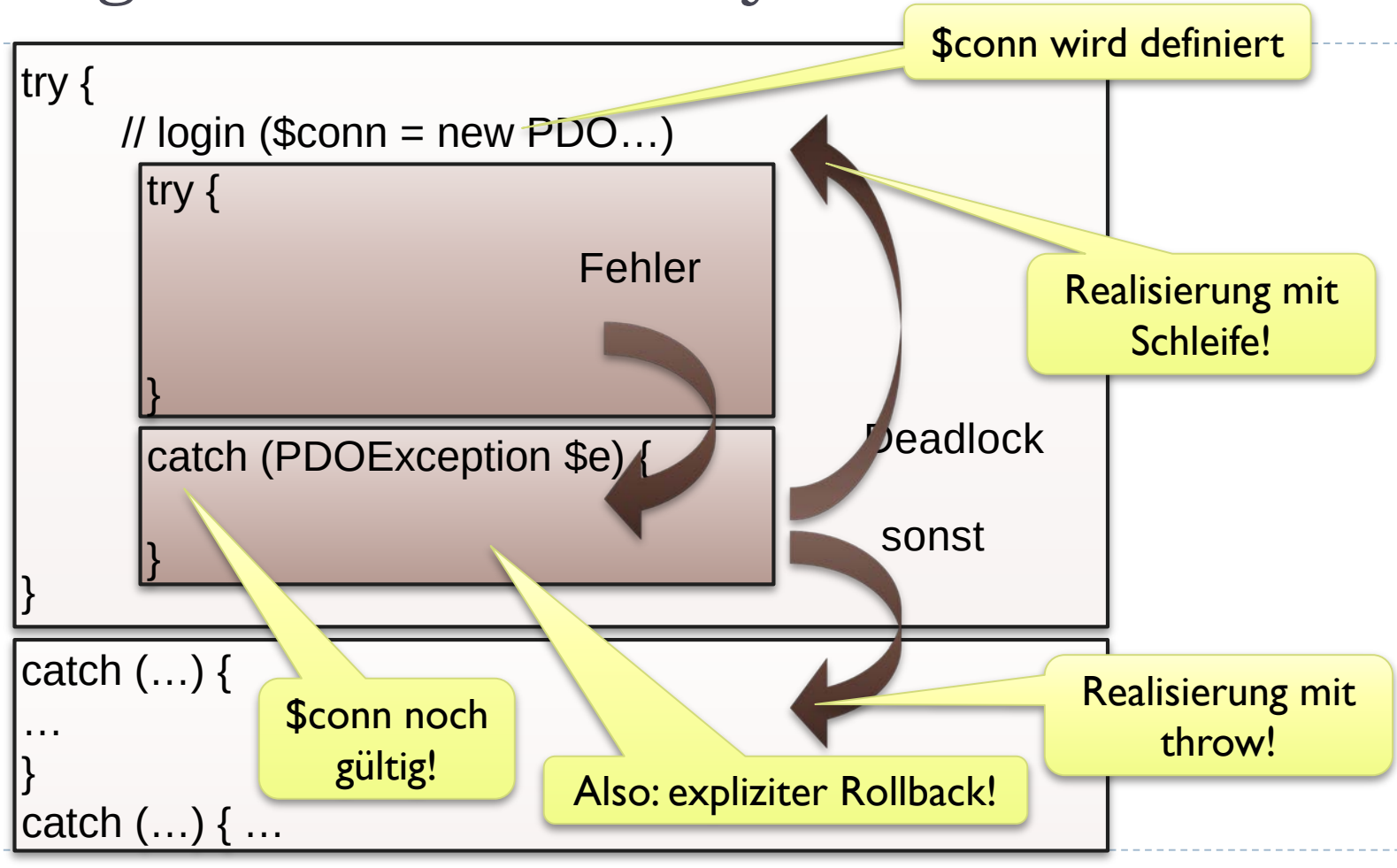
Kundnr	Name	Sperre	Kunde1	Kunde2
5	Biker Ecke	0	◉	○
1	Fahrrad Shop	0	○	◉

Kundnr	N…	Sperre	Kunde1	Kunde2
5	…r Ecke	0	○	◉
	Fahrrad…	0	◉	○

Behandlung des Deadlocks

- Falls Deadlock eintritt:
  - Eine der beiden Transaktionen erhält Fehlermeldung „40001“
  - Diese Transaktion führt einen Rollback durch und startet neu
    - Damit werden Sperren freigegeben
  - Dadurch kann andere Transaktion weiter arbeiten
- In unserer PDO Anwendung:
  - Fehler führen zum Sprung in Catch-Block
  - Damit wird automatisch ein Rollback durchgeführt
  - Aber: Wie können wir die Transaktion wiederholen?
  - Aber: Unsere Connection-Variable gibt es nicht mehr!

Lösung: Geschachtelte Try-Catch-Blöcke



Realisierung im Überblick

```
try  {
    $conn = new PDO( … ) ;        // Einloggen
    $versuch = 0;          // Noch keine Ausführung der Transaktion
    while ($versuch < 2)     {
        $versuch++;          // Erster Versuch!
        try  {
            $conn->beginTransaction();          // Transaktionsmodus
              // Zugriff1
            ob_flush();  flush();   sleep(10);       // 10 Sekunden warten
              // Zugriff 2
            $conn->commit();
            $versuch = 2;                // fertig, kein Wiederholen!
        }  //end try
        catch { … }        // → siehe naechste Folie
    }  // end while
}  // end try
```

Erster Update

Warten

Zweiter Update

Lösung in PHP

```
catch (PDOException $e) {
   $code = $e->getCode();
   if ($code == "HY000")
      $code = $e->errorInfo[1] ;

   if ( $versuch <= 1 and ($code == "40001" or $code == 60) )
   {
      echo "Synchronisationsprobleme! Es wird nochmals gestartet." ;
      $conn->rollback( ) ;
   } else
      throw $e;
}
```

Code abfragen

Falls kein korrekter Code …

… herstellerspezifischen Code auslesen

Erstversuch

Deadlock-Code

Oracle Deadlock-Code

Enorm wichtig!

Ausnahme weiterreichen zu äußerem Catch-Block

Weitere Anmerkungen

- Innerer Try-Catch-Block ist umhüllt von Schleife
  - Bei fehlerfreiem Ablauf wird diese nur einmal durchlaufen
  - Bei Deadlock-Fehler kann Zähler (z.B. $versuch) mitgeführt werden, um Endlosschleifen generell zu verhindern
- Problem:
  - Auch Browser synchronisieren Sessions!
  - Überlagerung von Lockproblemen
  - Empfehlung:
    - session_write_close(); // gleich nach session_start();
    - → keine Protokollierung der Session → keine Serialisierung der Browser

Arbeiten mit Large Objects (LOB)

- Large Objects:
  - Binary Large Objects (BLOB): Binär gespeicherte Objekte
  - Character Large Objects (CLOB): Sehr lange Zeichenketten

	Max. Größe eines BLOB	Max. Größe eines CLOB
Oracle	8 – 128 TByte	8 – 128 TByte
SQL Server	2 GByte (Image)	1 – 2 GByte
MySQL	4 GByte (LONGBLOB)	4 GByte (LONGTEXT)

Zusätzlich: FILESTREAM

Large Objects

- Large Objects (LOBs) können direkt in die Datenbank aufgenommen werden:
  - Gleiche Verwaltung wie die Daten
  - Gleiche Zugriffsrechte wie die Daten
  - Keine Zugriffe von außerhalb möglich
- LOBs können gespeichert und wieder ausgelesen werden
- CLOBs zusätzlich:
  - Direkte Bearbeitung und Durchsuchen in Datenbank möglich
- Speichern von Büchern, Bildern, Musik, Videos, …

CLOB BLOB BLOB BLOB

Beispiel: Ablegen von Bildern

- Idee:
  - In Personaltabelle werden die Fotos der Mitarbeiter mit aufgenommen
  - Technische Gründe: Bildtyp wird benötigt (JPEG usw.)
- Implementierung:

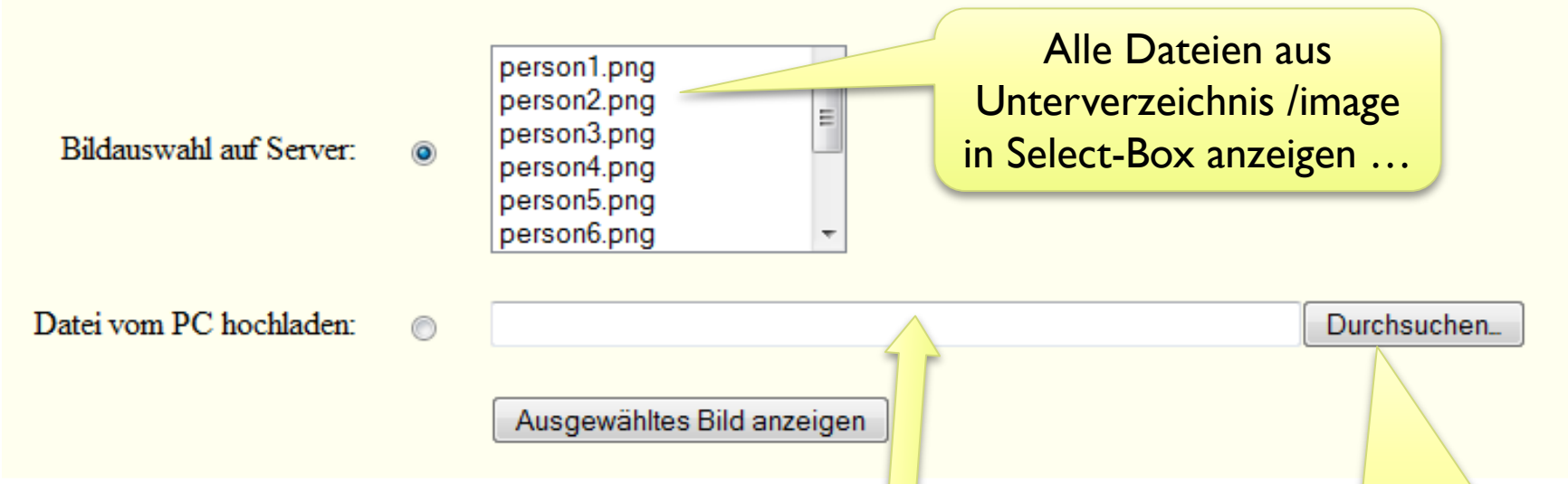
```
ALTER TABLE Personal
  ADD  Bild BLOB Default EMPTY_BLOB() ;
ALTER TABLE Personal
  ADD  Bildtyp CHAR(40) ;
```

Oracle

SQL Server: Add Bild Varbinary(max);

MySQL: Add Bild LongBlob;

Bilder und PHP: blob_ein1.php



Neue Funktionen:
- scandir
- is_dir

Neues Formularelement:
<input type="file" name=…

Implementierung: blob_ein1.php (1)

```
<form action="blob_ein2.php" method="post"
                                    enctype="multipart/form-data">
  <p>Bildauswahl auf Server:
  <input type="radio" name="Auswahl" value="Server" checked/>
  <select name="Bild" size="6">
<?php
   $dir = "./image";
   if (is_dir($dir))
    if ($filearray = scandir($dir))
     foreach ($filearray as $name)
      if (!is_dir($name))
        echo "<option> $name </option>";
?> </select>
```

Ruft blob_ein2.php auf

Notwendig, wenn file-Typ verwendet wird

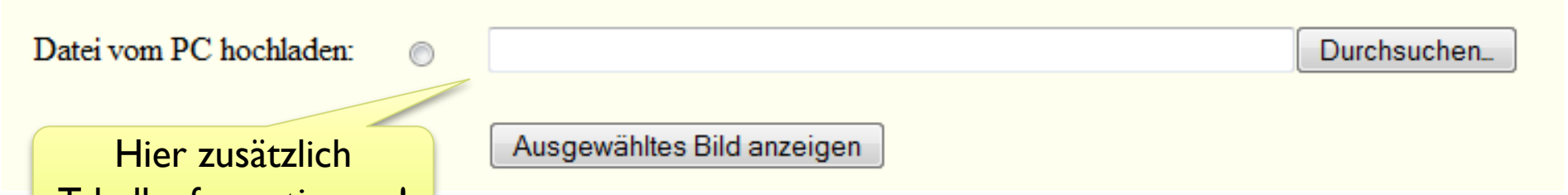
Existiert Verzeichnis ./image?

Liest alle Dateien des Verzeichnisses in Feld $filearray

Ver-zeich-nis nicht leer?

$filearray komplett auslesen. Dateien (keine Verzeichnisse) in Select-Box anzeigen

Implementierung: blob_ein1.php (2)



Hier zusätzlich Tabellenformatierung!

```
<p>Datei vom PC hochladen:
  <input type="radio" name="Auswahl" value="Lokal"/>

  <input type="file" name="Bilddatei" size="60" accept="image/*"/>  </p>

  <p> <input type="Submit" value="Ausgewählte Bild anzeigen"/></p>
</form>
```

Angabe der akzeptierten Dateitypen, hier: Bilder

Eigenschaften des Typs $_FILES

- Dateien werden mit $_FILES übertragen, nicht $_POST
- $_FILES besitzt vier assoziative Werte:
  - $_FILES['Bilddatei']['name']
    - Name der Datei, die auf dem PC ausgewählt wurde
    - Name des Input-Files
  - $_FILES['Bilddatei']['type']
    - Dazugehöriger Dateityp
  - $_FILES['Bilddatei']['tmp_name']
    - Name der Datei, in den die ausgewählte Datei temporär hochgeladen wurde (inkl. Pfad)
  - $_FILES['Bilddatei']['size']
    - Größe der Datei

Zwischenschritte (blob_ein2.php) (1)

- **Kopieren der hochgeladenen Datei ins Verzeichnis upload:**

```
copy( $_FILES['Bilddatei']['tmp_name'] ,
                                    './upload/'.$_FILES['Bilddatei']['name'] );
```

- **Merken der Bilddaten in Session-Variable:**

```
$_SESSION['Bild'] = "./upload/" . $_FILES['Bilddatei']['name'];
$_SESSION['Bildtyp'] = $_FILES['Bilddatei']['type'];
```

Falls Bild vom PC hochgeladen

bzw.

```
$_SESSION['Bild'] = "./image" . $_POST['Bild'];
$_SESSION['Bildtyp'] = "image/png";
```

Falls PNG-Bild auf Server ausgewählt

Zwischenschritte (blob_ein2.php) (2)

- Formular:
  - Anzeige des ausgewählten Bildes
  - Aufforderung zur Angabe der Personalnummer
    - um das Bild einem Mitarbeiter zuzuordnen
    - Besser: Alle Mitarbeiter auflisten (→ Übung)

Einfügen des Bildes in Datenbank

- **Problem:**
  - Die Datenbank muss wissen, dass ein LOB vorliegt!
  - Bei der Übergabe eines Bildes müssen wir also mitteilen, dass dies ein Element vom Typ PDO::PARAM_LOB ist!
  - Mit der Methode query ist dies nicht möglich
  - Wir verwenden die Methoden prepare und execute!
- **Gleichwertig:**

```
$stmt = $conn->query( $sql );
```

Parsen und Ausführen

Parsen des Befehls

```
$stmt = $conn->prepare( $sql );
$stmt->execute( );
```

Ausführen

gleichwertig

Parsen und Vorbereiten des Update

```
$sql = "Update Personal
          Set      Bildtyp = ?,
                   Bild = EMPTY_BLOB( )
         Where Persnr = $_POST[Persnr]
         Returning Bild Into ?" ;
$stmt = $conn->prepare($sql);

$fp = fopen( $_SESSION['Bild'], 'rb' ) ;

$stmt->bindParam( 1, $_SESSION['Bildtyp'] ) ;
$stmt->bindParam( 2, $fp, PDO::PARAM_LOB ) ;
$stmt->execute( ) ;

fclose( $fp ) ;
```

Platzhalter

Oracle

Bild einfügen: Zunächst leeren Blob vorhalten (nur in Oracle)

Platzhalter

Vorbereiten des Befehls

Bild binär zum Lesen öffnen

1. Fragezeichen mit Variable verknüpfen

2. Fragezeichen mit Variable $fp als LOB verknüpfen

Übertragen und ausführen

Ist Bild wirklich in Datenbank?

- Zum Überprüfen:
  - Wir testen die Länge des LOB
    - in Oracle: Dbms_Lob.Getlength(*Binärdatei*)
    - in SQL Server: DataLength(*Binärdatei*)
    - in MySQL: Length(*Binärdatei*)
- z.B.:

Gibt Bildgröße in Byte zurück

```
$sql = "Select Persnr, Name, Ort, Gebdatum, Gehalt,
                Vorgesetzt, Dbms_Lob.Getlength(Bild) As Bildlaenge
          From Personal
          Where Upper(Name) Like Upper( '%$Name%' ) " ;
```

Bild auslesen (mitarbeiterblob.php)

```
do {
?><tr>
      <td> <?php echo $row["PERSNR"]; ?> </td>
    // analog: NAME, ORT, GEBURTSDATUM, GEHALT
      <td> <?php echo ($row["VORGESETZT"] == null)? "Ja" : "Nein";?> </td>
<?php
      if ($row["BILDLAENGE"] > 0)
        echo "<img src=\"bild.php?id=$row[PERSNR]\" height=250/> " ;
      else
        echo "<td> <p>-- kein Bild --</p> </td>";
   echo "</tr>";
}  while ( $row = $stmt->fetch() ) ;
?>
```

Alle Daten ausgeben (wie in mitarbeiter.php)

Liegt Bild vor? (falls Bildlänge größer 0!)

… dann Bild anzeigen! (siehe nächste Folie)

Schleife über alle Mitarbeiter

Funktionsweise des Auslesens

- Auszulesen ist ein Bild → IMG-Tag
- Bild steht in Datenbank und liegt nicht als Bilddatei vor
- Also: Datenbank öffnen und mit Select-Befehl auslesen
- Dies geschieht in einer eigenen PHP-Datei
- Aufruf dieser PHP-Datei mit GET-Parameter
  - um das Bild des gewünschten Mitarbeiters zu erhalten

```
echo "<img src=\"bild.php?id=$row[PERSNR]\" height=250/> "
```

z.B. Bild von Mitarbeiter 2:
src="bild.php?id=2"

Get-Parameter

Datei bild.php

```
<?php
session_start();
$conn = new PDO($_SESSION['Parameter1'],
                                $_SESSION['Kennung'], $_SESSION['Passwort']);
$sql =    " Select Bild, Bildtyp From Personal
            Where Persnr = $_GET[id] " ;
$stmt = $conn->query($sql);

$stmt->bindColumn(1, $blob, PDO::PARAM_LOB);
$stmt->bindColumn(2, $bildtyp);
$stmt->fetch(PDO::FETCH_BOUND);

header("Content-Type: $bildtyp");
fpassthru($blob);
?>
```

Nur Bildausgabe, daher: Keine Tags, auch kein <html>

Get-Parameter!

1. Parameter Bild → $blob als LOB !

2. Parameter Bildtyp → $bildtyp

Keine Ausgabe in Feld, sondern in Variable mittels bindColumn!

Ausgabe nicht mit echo, da $blob nur Zeiger (Oracle)

Datenbankunabhängiges Programmieren

- PDO ermöglicht weitgehend datenbankunabhängige Zugriffe (Problem: LOBs)
- Herstellerspezifisches SQL sollte beachtet werden, z.B.:

zu beachten	wegen
PDO::ATTR_CASE auf PDO::CASE_UPPER setzen	Oracle
Kein Semikolon am Befehlsende	Oracle
Im Join nur den Operator ON verwenden	SQL Server
In der From-Klausel auf den Bezeichner AS verzichten	Oracle
Kein Leerzeichen nach einer Aggregatfunktion	MySQL
Kein Full Outer Join	MySQL

Zusammenfassung zu PDO

- Klasse PDO:
  - zur Verwaltung der Datenbankverbindung
- Klasse PDOStatement:
  - zur Bearbeitung eines SQL-Befehls
- Klasse PDOException:
  - zur Fehlerbehandlung

Methoden der PDO-Klassen

Methode	aus Klasse	Kurzbeschreibung
Konstruktor PDO()	PDO	baut Verbindung zur Datenbank auf
SetAttribute()	PDO	setzt Attribute einer Verbindung
BeginTransaction()	PDO	startet eine Transaktion
Commit()	PDO	beendet eine Transaktion
Rollback()	PDO	setzt eine Transaktion zurück
Query()	PDO	führt einen SQL-Befehl aus
Prepare()	PDO	bereitet eine Abfrage vor
Execute()	PDOStatement	führt eine vorbereitete Abfrage aus
Fetch()	PDOStatement	liest die nächste Zeile
RowCount()	PDOStatement	gibt Anzahl manipulierter Zeilen aus
BindParam()	PDOStatement	bindet Parameter an eine Abfrage
BindColumn()	PDOStatement	verbindet Spalten mit Variablen

PHP-Funktionen (1)

trim(str)	entfernt Leerzeichen am Anfang und Ende von str
strlen(str)	gibt die Länge der Zeichenkette str zurück
strpos(str1,str2)	gibt das erste Vorkommen von str2 in der Zeichenkette str1 zurück; bzw. false, falls nicht enthalten
strcmp(str1,str2)	vergleicht die Zeichenketten str1 und str2
strcasecmp(str1,str2)	vergleicht die Zeichenketten str1 und str2 unabhängig von Groß- und Kleinschreibung
substring(str,pos,len)	gibt Teilstring von str der Länge len ab Position pos zurück
htmlspecialchars(str)	wandelt HTML-Zeichen (z.B. <, >) in Ersatzzeichen um
implode(str,feld)	liefert Zeichenkette mit allen Feldelementen zurück, die mittels der Zeichenkette str verknüpft werden
stripslashes(str)	entfernt Entwertungszeichen ‚\' in der Zeichenkette str
echo param1,…	gibt die Parameterliste auf HTML aus
isset(var)	liefert true, wenn Variable var existiert, sonst false
unset(var)	setzt die Variable var zurück, var existiert nicht mehr

PHP-Funktionen (2)

copy(file1, file2)	kopiert Datei file1 in die Datei file2
fopen(file, str)	öffnet Datei file mit der im String str angegebenen Methode und liefert einen Dateizeiger zurück
fclose(fp)	schließt Datei, auf die der Dateizeiger fp verweist
header(str)	gibt den im String str angegebenen Header aus
fpassthru(blob)	gibt das Binärobjekt aus, auf den blob verweist
is_dir(str)	gibt true zurück, falls str ein Verzeichnis ist, sonst false
scandir(str)	liefert alle Dateinamen des Verzeichnisses str in einem Feld zurück
session_start()	erster Befehl einer Session-Seite
session_write_close()	schreibt kein Session-Protokoll und vermeidet dadurch Synchronisierung der Browser
ob_flush()	gibt Inhalt auf dem Webserver sofort aus
flush()	gibt Inhalt im lokalen Browser sofort aus
sleep(sec)	wartet für sec Sekunden

Datenbanken und SQL

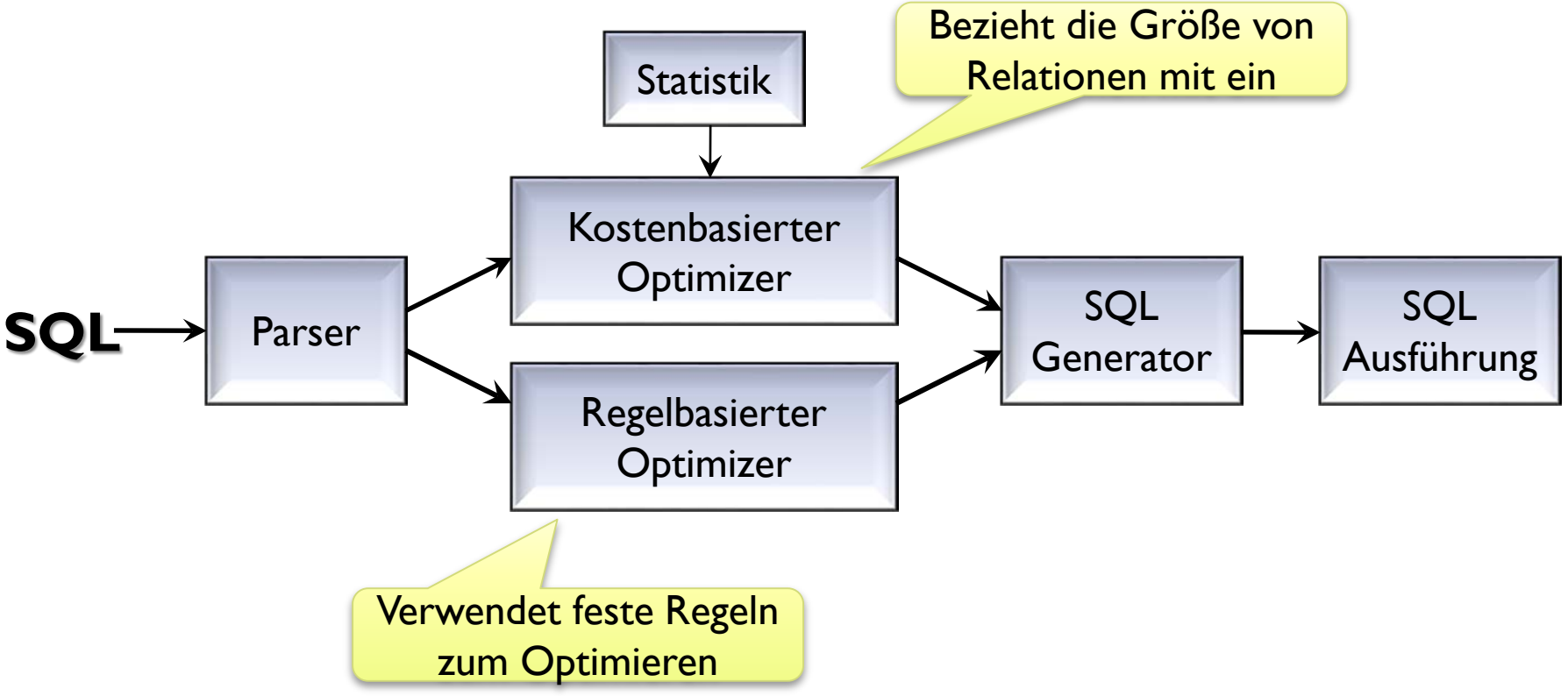
Kapitel 7

Performance in Datenbanken

Performance in Datenbanken

- Datenbankenoptimizer und Ausführungsplan
- Index
- Partitionierung
- Materialisierte Sichten
- Optimierungen im Select-Befehl
- Stored Procedures
- Weitere Optimierungen

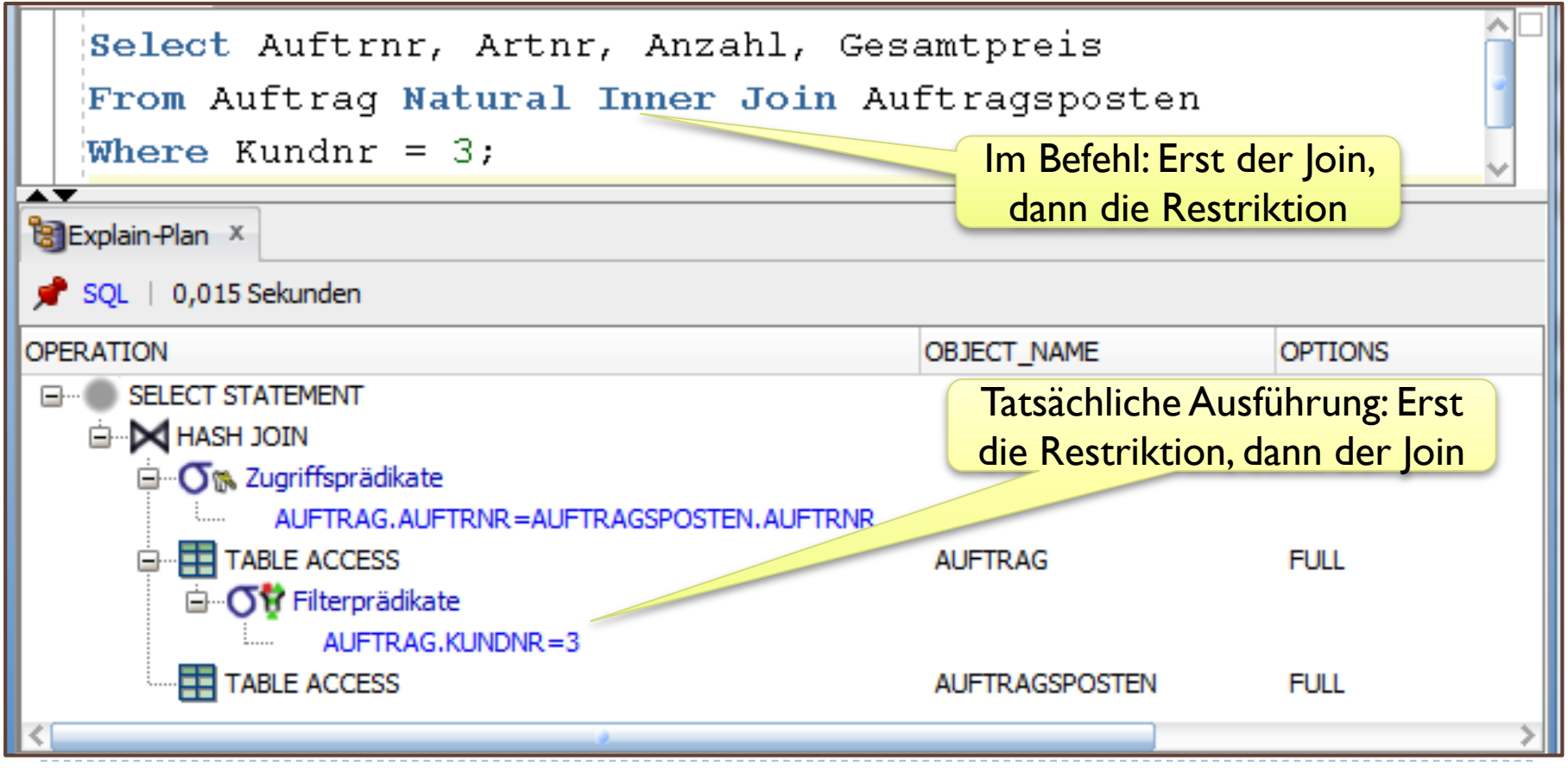
Abfrageoptimierung



Optimizer

- **Regelbasierte Optimierung**
  - Optimierung nach vorgegebenen Regeln. Beispiele:
    - Verwende einen Index, falls vorhanden
    - Führe erst Projektion durch, dann einen Verbund
    - Führe erst Restriktion durch, dann einen Verbund
    - Verwende Merge Join, wenn Relationen bereits sortiert sind
- **Kostenbasierte Optimierung**
  - Optimierung zusätzlich in Abhängigkeit von den Kosten
  - Berücksichtigung der Größe (und des Inhalt) der Relationen
    - Beispiel: Verwende im Verbund erst die kleinere Relation

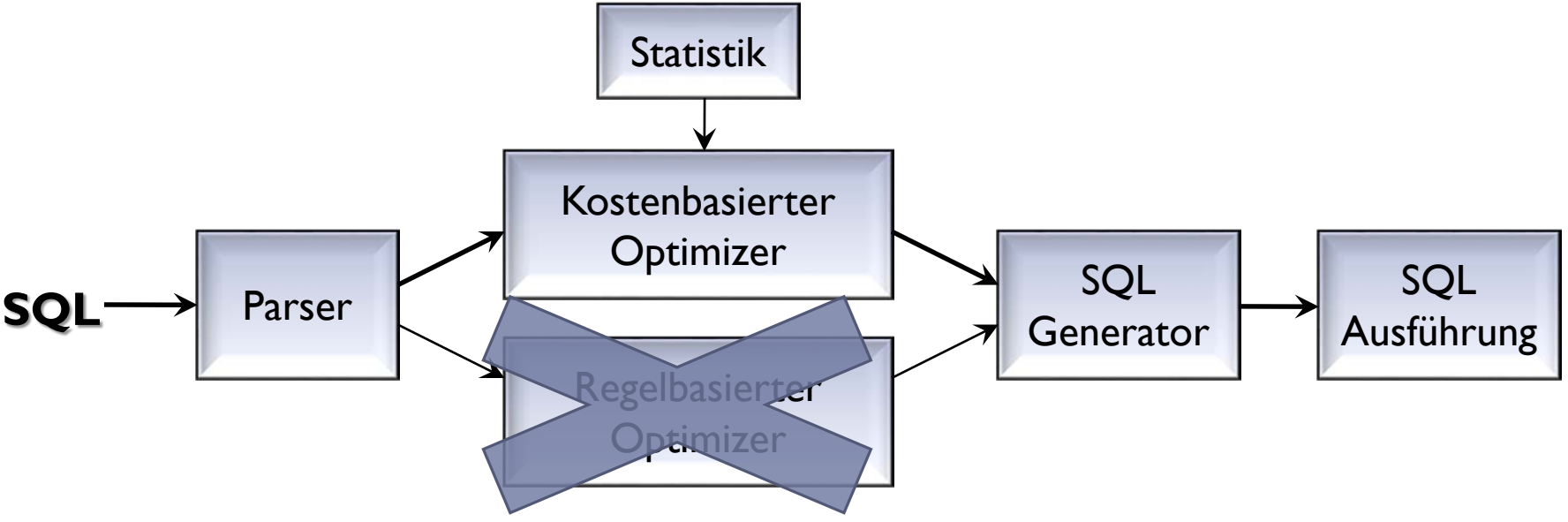
Beispiel zur Optimierung



Regelbasiert versus kostenbasiert

- Regelbasierter Optimizer
  - \+ Relativ einfach implementierbar
  - – Ungenau, da nicht an die reale Tabelle angepasst
- Kostenbasierter Optimizer
  - \+ Berücksichtigt den Aufbau einer Tabelle
  - \+ Liefert deshalb sehr genaue Ergebnisse
  - – Benötigt zusätzliche Statistiken
  - – Relativ komplex und umfangreich
- Fazit:
  - Kostenbasierter Optimizer ist der Standard

Abfrageoptimierung



Kostenbasierter Optimizer: Aufgaben

- Analyse des Select-Befehls
- Überprüfen der Umgebung nach
  - vorhandenen Indexen
  - Aufteilung von Relationen in Partitionen
  - internen Strukturen (z.B. Aufteilung auf Festplatten)
- Auswertung der Statistiken
- Optimierung des Select-Befehls
  - unter Verwendung der obigen Gesichtspunkte
- Weiterleitung an den SQL-Generator

Statistiken

- Jede Datenbank sammelt umfangreiche Statistiken
- Notwendig für den kostenbasierten Optimizer
- Wichtige Statistiken:
  - Anzahl der Zeilen jeder Relation
  - Anzahl unterschiedlicher Einträge je Attribut und Relation
  - Zusätzlich: Histogramme zu jedem Attribut
- In der Praxis wichtig:
  - Selektivität eines Attributs → nächste Folie

Selektivität

- **Definition (Selektivität eines Attributs):**

  Die Selektivität eines Attributs sel ist der Kehrwert zur Anzahl der unterschiedlichen Attributswerte.

- Beispiele:
  - Primärschlüssel: sel = 1 / Anzahl der Zeilen
  - Geschlecht (m/f): sel = 0,5
  - Wochentage: sel = 0,14
- Die Selektivität wird benötigt:
  - bei Restriktionen: zur Abschätzung der Größe der Restrelation
  - ebenso bei Gruppierungen

Selektivität (2)

- Die Selektivität sel ist ein Mittelwert
  - Sind die Attributswerte ungleich verteilt, so sollten zusätzlich Histogramme verwendet werden
- Die Selektivität mehrerer Attribute kann einfach aus den Einzelattributen berechnet werden. Es gilt:

sel(Attr1 AND Attr2) = sel(Attr1) * sel(Attr2)

sel(Attr1 OR Attr2) = sel(Attr1) + sel(Attr2) – sel(Attr1)*sel(Attr2)

sel(NOT Attr1) = 1 – sel(Attr1)

  - Diese Gleichungen gelten exakt nur bei Gleichverteilungen

Optimierung in Oracle

Systemtabellen	Wichtige Attribute
USER_TABLES	Table_Name, Num_Rows, Avg_Row_Len
USER_TAB_STATISTICS	Table_Name, Num_Rows, Avg_Row_Len
USER_TAB_COLUMNS	Table_Name, Column_Name, Num_Distinct, Density, Num_Nulls, Avg_Col_Len, Histogram
USER_TAB_COL_STATISTICS	Table_Name, Column_Name, Num_Distinct, Density, Num_Nulls, Avg_Col_Len, Histogram

Anzahl Zeilen

entspricht Selektivität

für manche Anfragen sehr wichtig

Anzahl unterschiedliche Einträge

Statistiken in Oracle

- **Automatische Aktualisierung der Statistiken mit:**

  DBMS_AUTO_TASK_ADMIN.ENABLE

  Administratorrechte!

- **Manuelle Aktualisierung der Statistiken:**
  - Statistikpaket DBMS_STATS
  - Beispiele:

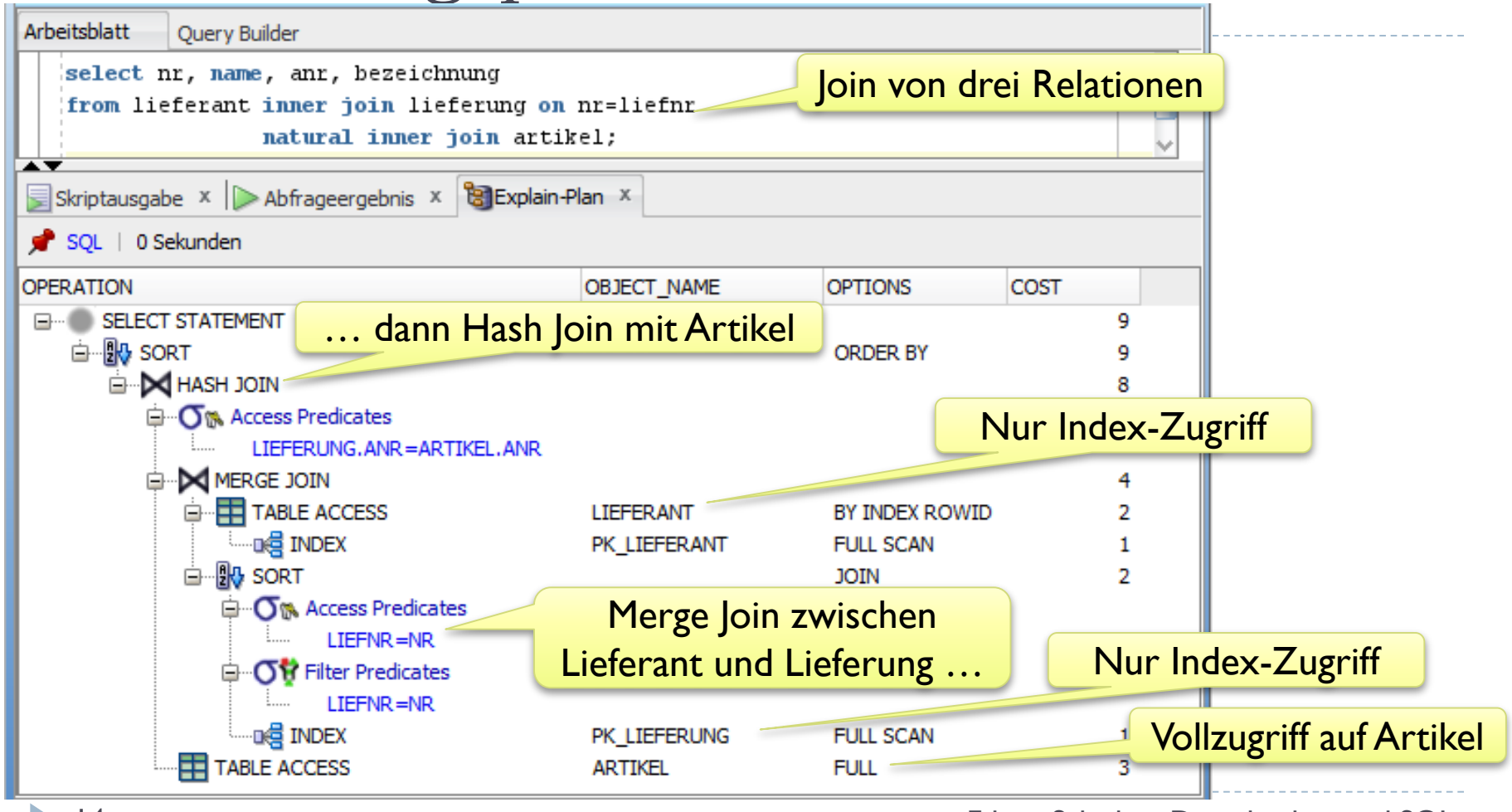
  Aktualisierung einzelner Tabellen

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS( 'Schema', 'Tabelle' );
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS( 'Schema' );
```

Aktualisierung im gesamten Schema

Achtung! Performance!

Ausführungsplan in Oracle



Optimierung in SQL Server

- SQL Server erzeugt zu jedem Schlüssel einen Index
- SQL Server führt Statistiken zu allen Indexen
- Ansehen von Statistiken:
  - DBCC SHOW_STATISTICS(Relation, Index);
  - oder komfortabel mit SQL Server Management Studio
- Statistiken anlegen, aktualisieren:

10% der Daten sind Basis für Hochrechnung

```
CREATE STATISTICS Statistikname ON Tabelle (Spalte) WITH FULLSCAN;
CREATE STATISTICS Statistikname ON Tabelle (Spalte) WITH SAMPLE 10 PERCENT;
ALTER DATABASE Datenbank SET AUTO_CREATE_STATISTICS ON;
ALTER DATABASE Datenbank SET AUTO_UPDATE_STATISTICS ON;
```

Automatisches Erzeugen bzw. Ändern

Beispiel einer Statistik (im SSMS)

Tabellenname: dbo.Personal

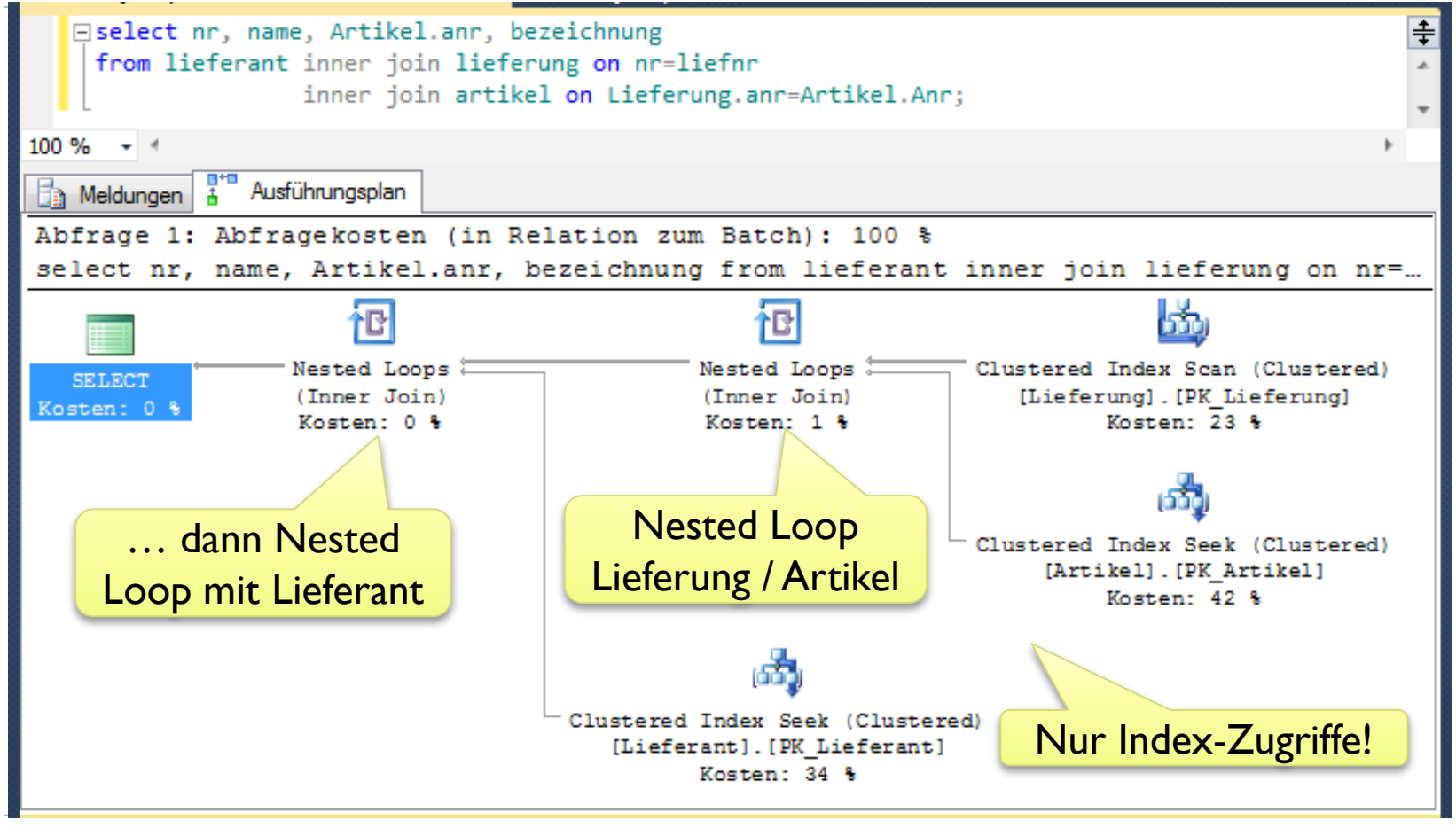
Statistikname: PK_Personal

```
Statistiken für INDEX 'PK_Personal'.
------------------------------------------------------------------

Name                                Updated
------------------------------------------------------------------
PK_Personal                         Feb  5 2013  9:00PM

All Density                         Average Length
------------------------------------------------------------------
0.1111111                           4

Histogram Steps
RANGE_HI_KEY                        RANGE_ROWS
------------------------------------------------------------------
1                                   0
3                                   1
5                                   1
7                                   1
9                                   1
```

= 1/9, da 9 Einträge (Selektivität)

Einfaches Histogramm

Ausführungsplan in SQL SErver



Optimierung in MySQL

- Statistiken werden automatisch angelegt für
  - Primärschlüssel
  - Alternative Schlüssel
  - Fremdschlüssel
- Statistiken stehen in:
  - Information_Schema.Statistics
- Statistiken enthalten keine Selektivität und keine Histogramme

Index

- **Definition (Index)**:

  Ein Index in einer Datenbank ist eine Struktur zur Beschleunigung von Suchvorgängen, in der Daten auf- oder absteigend sortiert angelegt werden.

- Ein Index in einer relationalen Datenbank wird in der Regel intern als eine eigenständige sortierte Tabelle angelegt, auf den mittels eines B*-Baums zugegriffen wird.

Index in SQL-1

Falls eindeutig

Index über mehrere Spalten möglich

```
CREATE [UNIQUE] INDEX Name ON
    Tabellenname ( { Spalte [ ASC | DESC ] } [ , ... ] )
```

aufsteigend (Standard)

absteigend

```
DROP INDEX Name
```

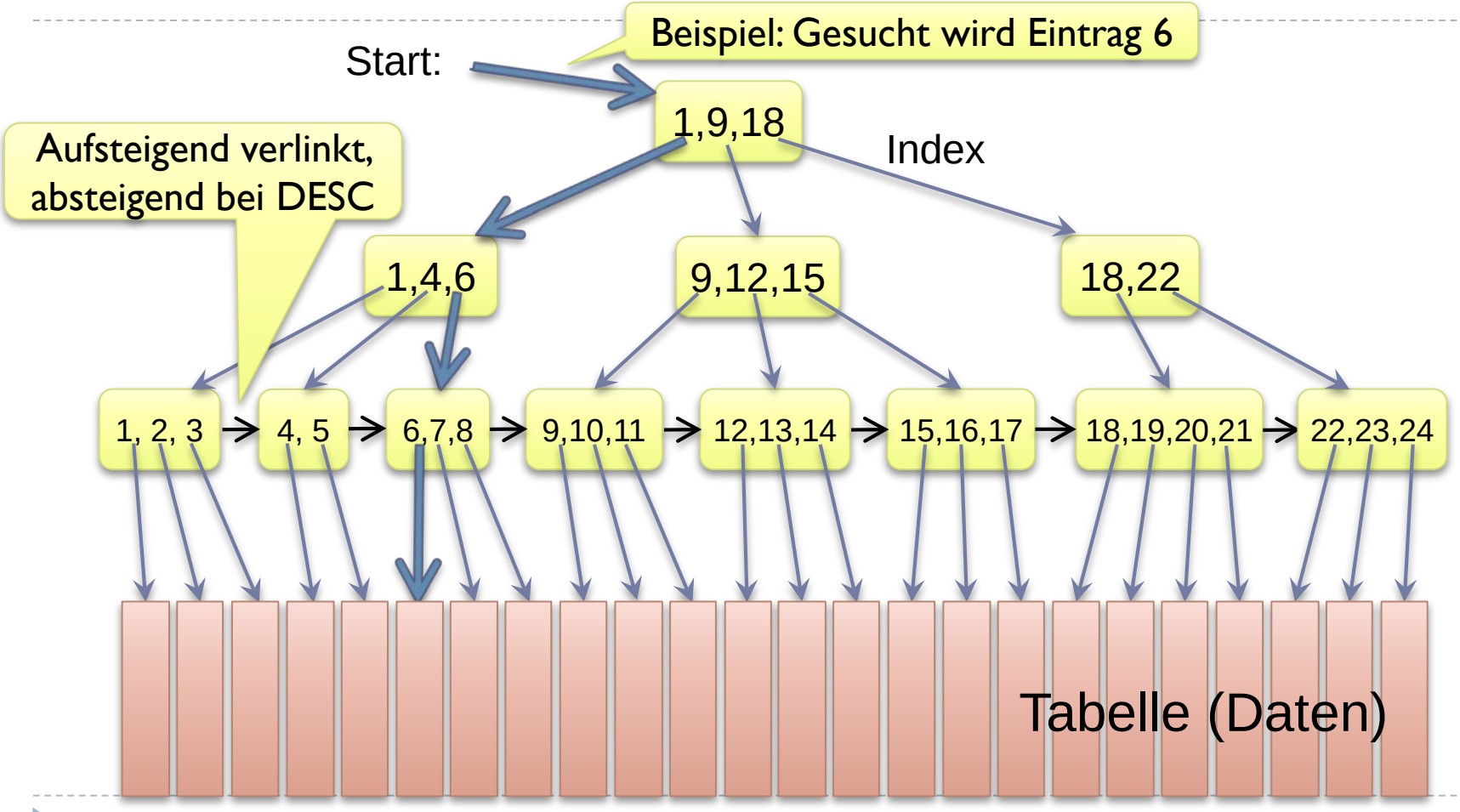
Seit SQL-2 nicht mehr normiert, aber immer noch üblich!

Systemtabellen und Indexe

Hersteller	Systemtabelle / Befehl	Inhalt
Oracle	USER_INDEXES	alle Indexe der Benutzerrelationen
Oracle	USER_IND_COLUMNS	alle Attribute mit gesetzten Indexen
SQL Server	SYS.INDEXES	alle Indexe
MySQL	SHOW INDEX FROM Tabelle	alle Indexe der Tabelle

Achtung:
nicht in INFORMATION_SCHEMA,
da Indexe nicht normiert!

Realisierung von Indexen: B*-Baum



Beispiel zu Indexen: Relation Umfrage

Name	Wert
NUM_ROWS	800000
BLOCKS	6409
AVG_ROW_LEN	51
SAMPLE_SIZE	800000
LAST_ANALYZED	08.06.13
LAST_ANALYZED_SINCE	08.06.13

800000 Einträge

Spaltenstatistik

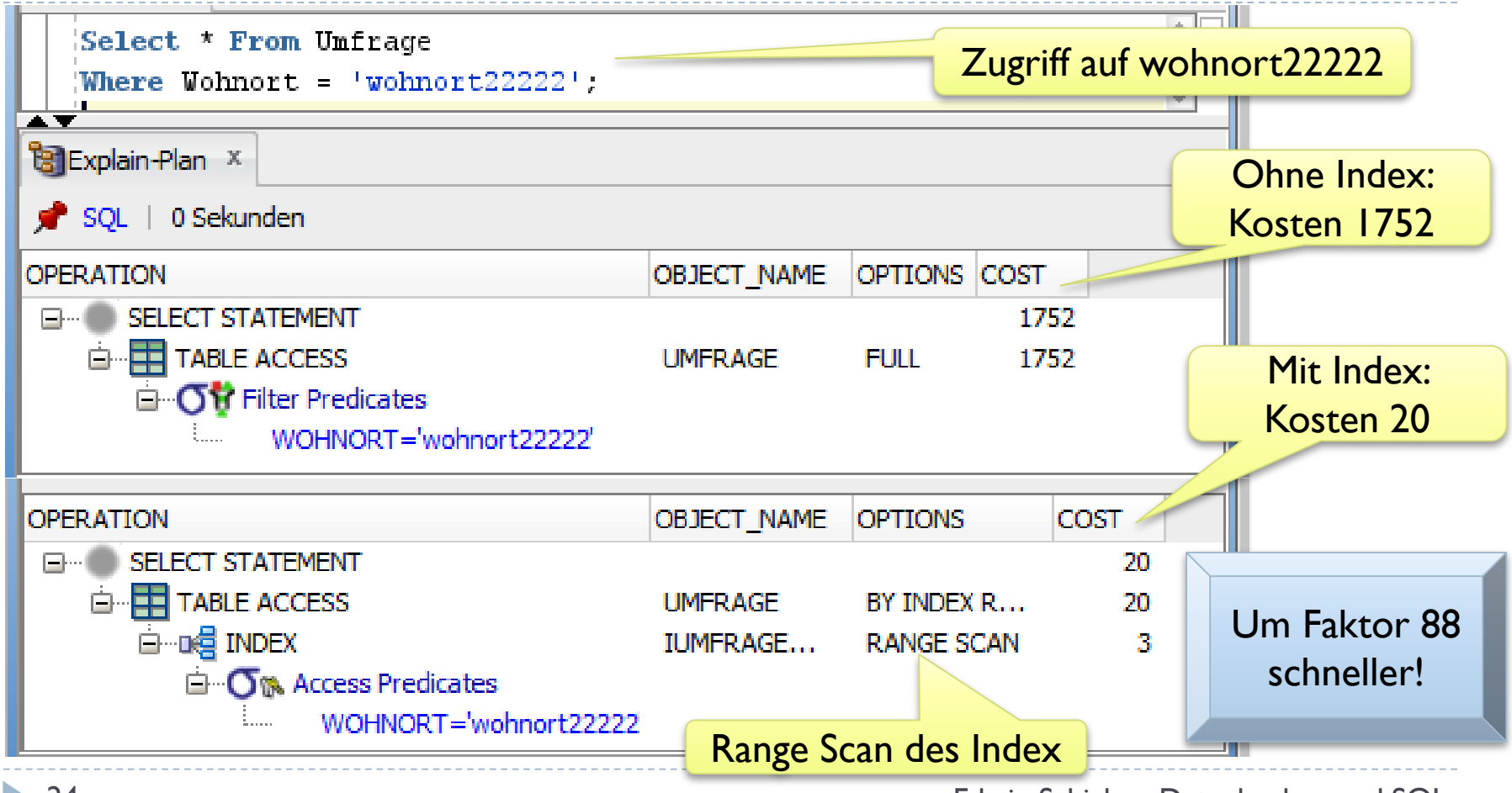
Aktualisieren: 0

Selektivität

OWNER	TABLE_NAME	COLUMN_NAME	NUM_DISTINCT	...	...	DENSITY	NUM_NULLS
BIKE	UMFRAGE	NR	800000	...		0,00000125	0
BIKE	UMFRAGE	GESCHLECHT	2	...		0,5	0
BIKE	UMFRAGE	FAMILIENSTAND	4	...		0,25	160000
BIKE	UMFRAGE	GEBURTSJAHR	50	...		0,02	0
BIKE	UMFRAGE	AUTOMARKE	1000	...		0,001	0
BIKE	UMFRAGE	WOHNORT	49548	...		0,0000201824493...	0

50000 Orte

Zugriff auf Relation Umfrage



Ergebnis

- Das Setzen und Entfernen des Index erfolgte mit:

  ```
  CREATE INDEX IUmfrageWohnort ON Umfrage(Wohnort);
  DROP INDEX IUmfrageWohnort;
  ```

- Das Erzeugen eines Index erfordert etwas Zeit und benötigt Speicherplatz
- Der lesende Zugriff wird erheblich beschleunigt
- Der schreibende Zugriff wird langsamer, da Index immer mit aktualisiert werden muss
- Folgerung: So viele Indexe wie nötig, so wenige wie möglich

Partitionierung

- Nachteil sehr großer Relationen
  - Keine Parallelzugriffe, da nur ein Medium
  - Bei Suche ohne Index: Komplette Relation durchsuchen
- Zerlegen in viele kleine Teilrelationen
  - Vorteil:
    - Obige Nachteile fallen weg
  - Nachteil:
    - Anwender muss Strukturen kennen
    - Neue Teile sind den Anwendungsprogrammen nicht bekannt, daher inflexibel
- Lösung: Partitionen

Definition: Partitionierung

- **Definition:**
  - Unter einer Partitionierung verstehen wir eine horizontale, vollständige und transparente Aufteilung einer Relation in disjunkte Teilrelationen.
  - Diese Teilrelationen bezeichnen wir als Partitionen.

- **Begriffe:**
  - Horizontal: zeilenweise (nicht spaltenweise)
  - Vollständig: 
  - Transparent: Nicht sichtbar für den Anwender
  - Disjunkt: Keine redundante Aufteilung

Partitionierung am Beispiel

Gegeben: Relation Produktion, partitioniert nach Monaten

Zugriff

Produktion

Zugriff wird durchgereicht

Prod_Jan

Prod_Feb

...

Prod_Dez

In Wirklichkeit: Monatspartitionen

Anwender kennt nur die Relation Produktion

Unterstützung der Partitionierung

Hersteller	Partitionierungstypen
Oracle	Bereichs-Partitionierung List-Partitionierung Hash-Partitionierung Intervall-Partitionierung Referenz-Partitionierung Virtuelle spaltenbasierte Partitionierung
SQL Server	Bereichs-Partitionierung Index-Partitionierung
MySQL	Bereichs-Partitionierung List-Partitionierung Hash-Partitionierung Schlüssel-Partitionierung

Bereichspartitionierung am Beispiel

- **Bereichspartionierung (Range-Partitioning) in Oracle:**

Nach außen sichtbar

```
CREATE TABLE Auftrag (
    Auftrnr    INT       PRIMARY KEY,
    Datum      DATE      NOT NULL,
    Kundnr     INT       NOT NULL REFERENCES Kunde,
    Persnr     INT       REFERENCES Personal     )
 PARTITION BY RANGE (Datum)
 (  PARTITION Auftrag2010 VALUES LESS THAN DATE '2011-01-01',
    PARTITION Auftrag2011 VALUES LESS THAN DATE '2012-01-01',
    PARTITION Auftrag2012 VALUES LESS THAN DATE '2013-01-01',
    PARTITION Auftrag2013 VALUES LESS THAN DATE '2014-01-01'       );
```

Relation Auftrag wird nach Jahren partitioniert

Intern real existent

Partitionierungsarten (1)

- **Bereichspartitionierung:**
  - Einteilung in disjunkte Bereiche (meist nach Datum/Zeit)
  - Sehr häufig eingesetzt
  - In allen Datenbanken implementiert
- **List-Partitionierung**
  - Einteilung nach Listen
  - Beispiel: Einteilung nach Verkaufsländern (Liste aller Länder)
- **Hash-Partitionierung**
  - Datenbank übernimmt die Einteilung nach Hash-Codes
  - Nur interessant, wenn es sonst keine sinnvolle Einteilung gibt

z.B. eine Partition für Deutschland, eine Partition für Österreich und Schweiz gemeinsam usw.

Partitionierungsarten (2)

- Intervall-Partitionierung
  - Spezielle Bereichspartitionierung
  - Partitionierungsintervalle werden vorgegeben
  - Beispiel: Monatsintervall → Jeden Monat wird automatisch neue Partition erzeugt
- Virtuelle spaltenbasierte Partitionierung
  - In Oracle gibt es virtuelle Spalten (aus realen abgeleitet)
  - Partitionierung mit Hilfe dieser virtuellen Spalten
- Index-Partitionierung
  - In SQL Server: Index kann mit Relation partitioniert werden

Partitionierungsarten (3)

- Schlüsselpartitionierung
  - Spezielle Hashpartitionierung
  - Primär- oder alternativer Schlüssel dienen als Hash-Code
- Referenzpartitionierung → nächste Folie

- Viele Kombinationen sind möglich
  - Range – List (erst Rangepartitionierung, dann Listpart.)
  - Range – Hash
  - List – Hash usw.

Referenzpartitionierung

- Auftrag ist nach Jahren partitioniert (siehe oben)
- Auftragsposten enthält kein Datum, soll aber ebenso aufgeteilt werden → Referenzpartitionierung

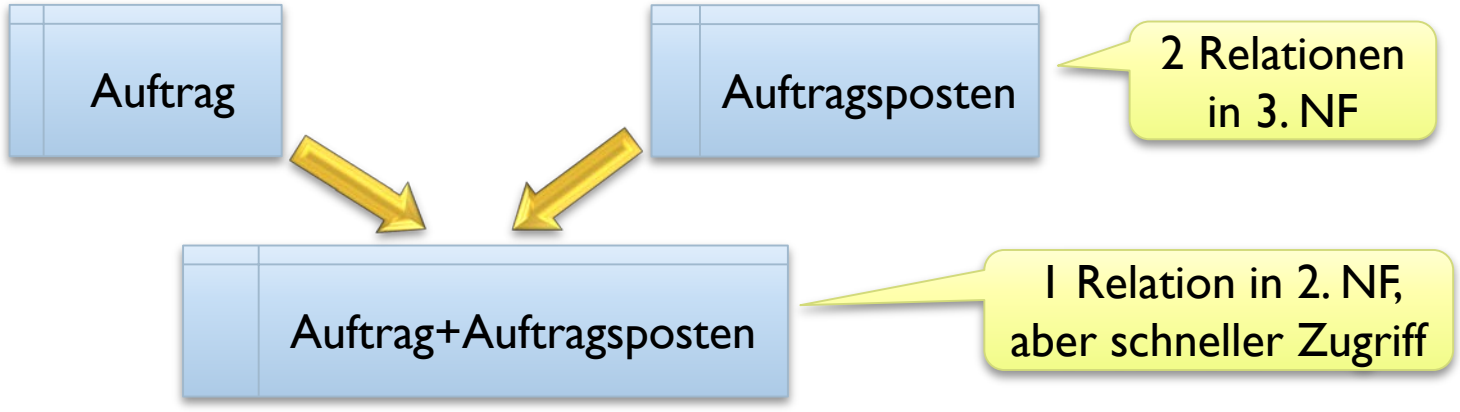
```
CREATE TABLE Auftragsposten (
    Posnr          INT       PRIMARY KEY
    Auftrnr        INT       NOT NULL,
    Teilenr        INT       NOT NULL,
    Gesamtpreis    NUMERIC(8,2),
    Anzahl         INT,
    CONSTRAINT Auftrpos_FK FOREIGN KEY(Auftrnr)
                                    REFERENCES Auftrag      )
  PARTITION BY REFERENCE( Auftrpos_FK )      ;
```

Partitionierung wie Auftrag!

Partitionierung: Zusammenfassung

- Partitionierung bei sehr großen Relationen wichtig
- Vorteile:
  - Transparenz: nicht sichtbar für Anwender
  - Parallele Verarbeitung möglich
  - Gezielter Zugriff auf nur eine oder wenige Partitionen statt auf gesamte Relation
- Folgerung:
  - Teils erhebliche Performancesteigerung

Materialisierte Sichten (Problem)



- Zusammenfassung von Auftrag und Auftragsposten?
  - Kein Join → Schnellere Zugriffe → Bessere Performance
  - Redundanz → Mehr Speicher → Gefahr von Inkonsistenz

Relation AuftragKomplett

Auftrag JOIN Auftragsposten

Posnr	Auftrnr	Datum	Kundnr	Persnr	Artnr	Anzahl	Gesamtpreis
101	1	04.01.2013	1	2	200002	2	800,00
201	2	06.01.2013	3	5	100002	3	1.950,00
202	2	06.01.2013	3	5	200001	1	400,00
301	3	07.01.2013	4	2	100001	1	700,00
302	3	07.01.2013	4	2	500002	2	100,00
401	4	18.01.2013	6	5	100001	1	700,00
402	4	18.01.2013	6	5	500001	4	30,00
403	4	18.01.2013	6	5	500008	1	94,00
501	5	03.02.2013	1	2	500010	1	40,00
502	5	03.02.2013	1	2	500013	1	30,00

Materialisierte Sichten (Idee)

- Sicht anlegen (Join zwischen Auftrag und Auftragsposten)
- Sicht physisch speichern!
- Vorteile:
  - Schnelle Zugriffe über (physische) Sicht
  - 3. NF der Basisrelationen bleibt erhalten
- Nachteile:
  - Es gibt jetzt Basisrelationen und (physische) Sicht parallel
  - Also: Redundanz und Gefahr der Inkonsistenz
- Wunsch:
  - Datenbank muss Datenabgleich intern übernehmen

Materialisierte Sicht AuftragKomplett

```
CREATE MATERIALIZED VIEW AuftragKomplett
    REFRESH FAST ON COMMIT
AS SELECT  Posnr, AuftrNr, Datum, Kundnr, Persnr,
           Artnr, Anzahl, Gesamtpreis
    FROM   Auftrag NATURAL INNER JOIN Auftragsposten ;
```

Analog zu Create View

- Refresh Fast / Refresh Complete [On Commit]

Nur Änderungen nachvollziehen

Alternativ: Inhalt komplett neu erzeugen

Beim Commit aktualisieren

Alternative: Zu bestimmten Zeitpunkten aktualisieren: START NEXT …

Materialisierte Sichten: Resümee

- Bei selten geänderten Relationen hervorragend geeignet
- Reduziert aufwändige Joins
- Allerdings:
  - Hoher interner Aufwand der Aktualisierung
  - Bei Refresh Fast wird ein Logbuch benötigt:

    CREATE MATERIALIZED VIEW LOG …

- Entfernen einer materialisierten Sicht:

  DROP MATERIALIZED VIEW MV_Name

Optimierung von Select-Befehlen

- Optimizer arbeitet nicht immer optimal
- Beispiel:
  - Frauenverband mit wenigen männlichen Mitgliedern
  - Suche aller männlichen Mitglieder
  - Optimizer:
    - Selektivität für Geschlecht ist 0,5
    - Index lohnt nicht, da sowieso jedes 2. Mitglied gesucht wird
  - Realität:
    - Suche der wenigen männlichen Mitglieder über Index wäre effektiv
- Folgerung:
  - Wir beschäftigen uns ein wenig mit Optimierung

Vorteil des Optimizers

- Optimizer kennt alle Regeln der Relationalen Algebra!
- Wichtig sind insbesondere:

Vertauschung von Restriktion und Verbund

(1) $\sigma_{Bedingung}(R1 \bowtie R2) = \sigma_{Bedingung}(R1) \bowtie \sigma_{Bedingung}(R2)$

Spezialfall von (1)

(2) $\sigma_{Bedingung\_an\_R2}(R1 \bowtie R2) = R1 \bowtie \sigma_{Bedingung\_an\_R2}(R2)$

Vertauschung von Projektion und Restriktion

(3) $\pi_{Auswahl}(\sigma_{Bedingung}(R)) = \sigma_{Bedingung}(\pi_{Auswahl}(R))$

Vertauschung von Projektion und Verbund

(4) $\pi_{Auswahl}(R1 \bowtie R2) = \pi_{Auswahl}(R1) \bowtie \pi_{Auswahl}(R2)$

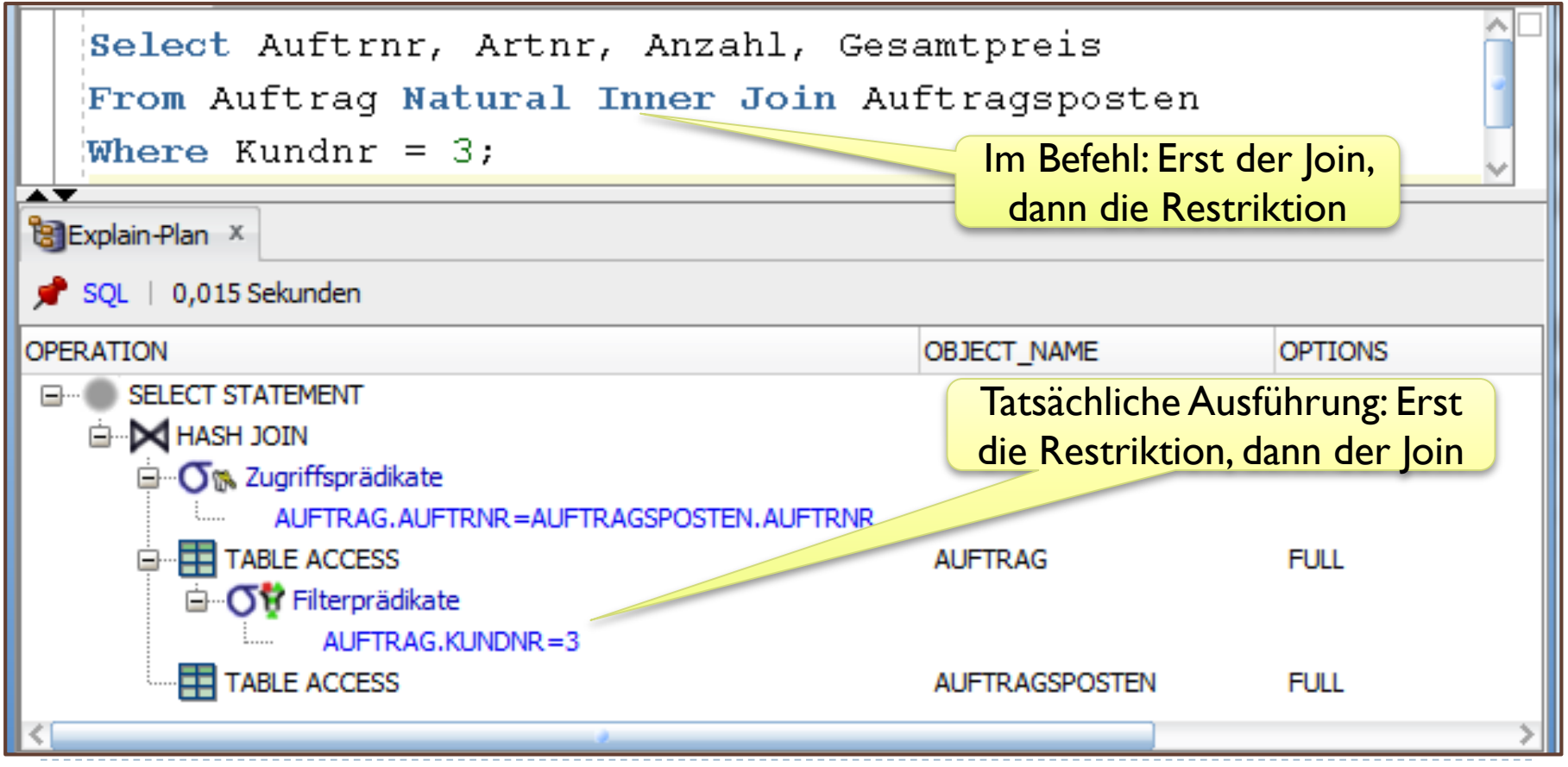
Achtung: Projektion muss verbindende Attribute enthalten!

Wichtige Regeln (1)

- Ein Verbund (und ein Produkt) zweier Relationen ist aufwändig
- Ziel: Beide Relationen vorher verkleinern
- Regel:
  - Erst Restriktion und Projektion
  - Dann Gruppierung
  - Und dann erst Verbund (Produkt)
- Der Optimizer kennt diese Regeln

Restriktion ist besonders effektiv, da dann meist weniger Daten eingelesen werden müssen

Beispiel zur Optimierung (Wiederholung)



Sicht VAuftrag (Wiederholung)

AuftrNr	Datum	Kundname	Persname	Summe
1	04.01.2013	Fahrrad Shop	Anna Kraus	800
2	06.01.2013	Maier Ingrid	Johanna Köster	2350
3	07.01.2013	Rafa – Seger KG	Anna Kraus	800
4	18.01.2013	Fahrräder Hammerl	Johanna Köster	824
5	06.02.2013	Fahrrad Shop	Anna Kraus	70

- Verbund aus Auftrag, Kunde, Personal und Auftragsposten

Beispiel zu den Grenzen des Optimizers

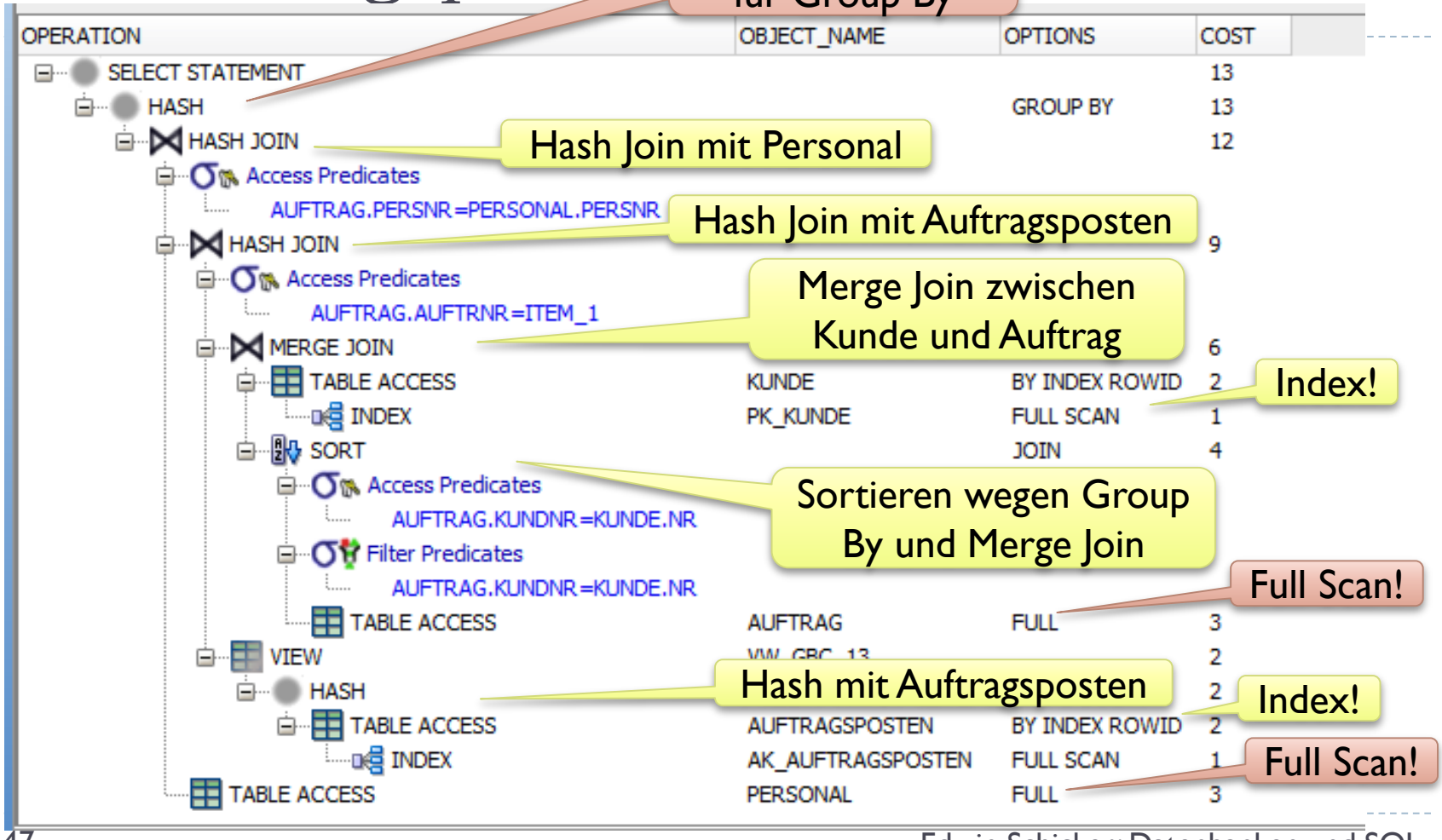
- **Select-Befehl zur Sicht VAuftrag:**

```
SELECT    AuftrNr, Datum, Kunde.Name, Personal.Name,
          SUM (Gesamtpreis)
FROM      Auftrag    JOIN Kunde  ON Kunde.Nr = Auftrag.Kundnr
                     JOIN Personal  USING (Persnr)
                     JOIN Auftragsposten USING (Auftrnr)
GROUP BY Auftrnr, Datum, Kunde.Name, Personal.Name ;
```

- **Schwäche:**
  - Group By kommt nach JOIN
  - Group By bezieht sich aber nur auf Auftragsposten

Woher soll das der Optimizer wissen?

Ausführungsplan



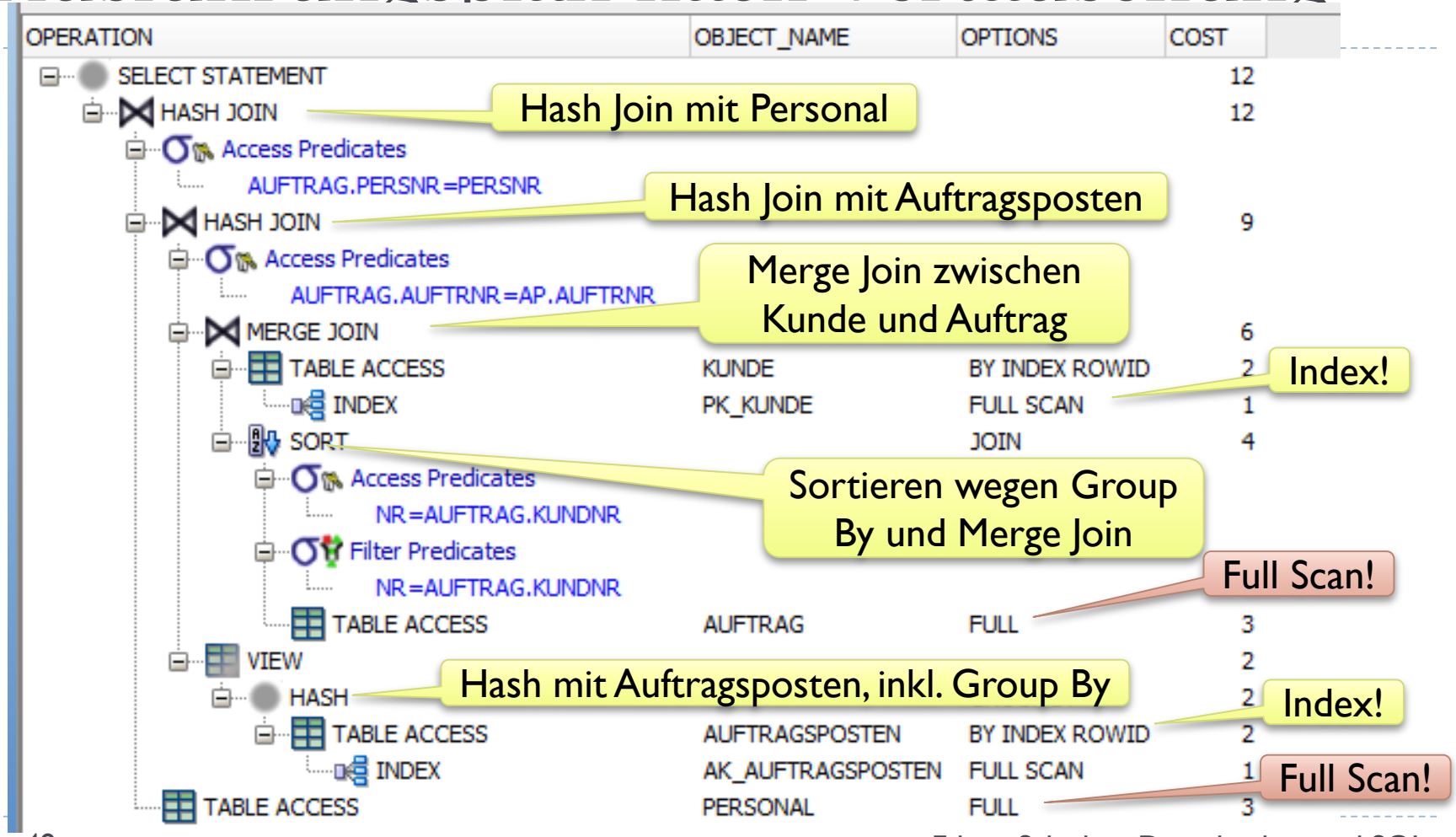
OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			13
HASH		GROUP BY	13
HASH JOIN			12
Access Predicates			
AUTFRAG.PERSNR=PERSONAL.PERSNR			
HASH JOIN			9
Access Predicates			
AUFTRAG.AUFTRNR=ITEM_1			
MERGE JOIN			6
TABLE ACCESS	KUNDE	BY INDEX ROWID	2
INDEX	PK_KUNDE	FULL SCAN	1
SORT		JOIN	4
Access Predicates			
AUFTRAG.KUNDNR=KUNDE.NR			
Filter Predicates			
AUFTRAG.KUNDNR=KUNDE.NR			
TABLE ACCESS	AUFTRAG	FULL	3
VIEW	VW_GBC_13		2
HASH			2
TABLE ACCESS	AUFTRAGSPOSTEN	BY INDEX ROWID	2
INDEX	AK_AUFTRAGSPOSTEN	FULL SCAN	1
TABLE ACCESS	PERSONAL	FULL	3

Gruppierung vor Verbund

- **Der Select-Befehl lautet jetzt:**

```
SELECT  AuftrNr, Datum, K.Name, P.Name, AP.Summe
FROM    Auftrag JOIN
        (Select Nr, Name From Kunde) K ON K.Nr = Auftrag.Kundnr
        JOIN (Select Persnr, Name From Personal) P USING (Persnr)
        JOIN (  Select Auftrnr, Sum(Gesamtpreis) As Summe
                From Auftragsposten
                Group By Auftrnr      ) AP     USING (Auftrnr)
```

Ausführungsplan nach Vertauschung



Wichtige Regeln (2)

- Sortieren ist in großen Relationen extrem aufwändig
- Sortieren steckt indirekt in vielen Operationen
- Wir versuchen daher einige Operationen zu vermeiden:
  - Kein Union (stattdessen: Union All)
  - Kein Order By
  - Kein Select Distinct
  - Kein Group By

Union entfernt gleiche Einträge, aufwändig! Union All tut dies nicht!

Sortiert! Alternative: Limit, Rownum

Entfernt gleiche Einträge, aufwändig!

Gruppieren heißt: Gleiche Einträge suchen!

Der Verbund (JOIN)

- Der Verbund wird sehr oft benötigt
- Der Verbund ist aufwändig
- Regel:
  - Die kleinere Relation steht links vom Join-Operator
  - Eine vorherige Restriktion wird dabei berücksichtigt!
- Wichtige (interne) Verbund-Operationen:
  - Nested Loop Join
  - Hash Join
  - Merge Join

Der Optimizer kennt diese Regel

Geschachtelte Schleife: Äußere Schleife über die linke Relation

Mit Hashtabellen

Erfordert vorheriges Sortieren der beiden Relationen

Nested Loop Join (R1 ⋈ R2)

- **Schleife (grob):**

  foreach (linke Relation R1 as row1)

  foreach (rechte Relation R2 as row2)

  vergleiche row1 mit row2

- **Optimierung:**
  - Vergleiche datenblockweise, nicht zeilenweise
  - Dies optimiert die Einlesevorgänge
- **Aufwand:**
  - R1 wird komplett eingelesen
  - Ist Hauptspeicher knapp, so wird R2 bis zu |R1|-mal gelesen
  - In Summe: |R1| + |R1|*|R2| (|R| = Anzahl der Blöcke von R)

Wenn R1 kleiner als R2, dann |R1|<|R2|
Also: kleinere Relation steht links!

Hash Join (R1 ⋈ R2)

- Aufbau einer Hash-Tabelle zu R1 und R2
- Aufwand
  - Lesen von R1 und R2 wegen Hash-Tabelle
  - Lesen von R1 und R2 wegen des Verbindens
  - Lesen von R1 und R2 wegen abschließenden Mischens
- Bei extrem knappem Speicher gilt für den max. Aufwand:

  3 * (|R1| + |R2|) (|R| = Anzahl der Blöcke von R)
- Ohne Beweis:
  - Bei knappem Speicher hat Hash große Vorteile, wenn die linke Relation kleiner ist!

Merge Join (R1 ⋈ R2)

- Idee:
  - Sortieren von R1
  - Sortieren von R2
  - Zusammenmischen der beiden Relationen
- Aufwand:
  - Das Sortieren erfordert c*n*log(n) Schritte, n=Anzahl, c=const
- Problem:
  - Bei knappem Speicher können sortierte Relationen nicht im Arbeitsspeicher gehalten werden.
  - Daher: Aufwand vergleichbar mit Hash und Nested Loop

Vergleich der drei Joins

- Je nach Anwendung kann jeder Join der beste sein
- Grob gilt:
  - Merge Join kommt mit wenig Arbeitsspeicher aus
  - Nested Loop Join benötigt viel Arbeitsspeicher, ist dann schnell
  - Hash Join ist optimal, wenn eine Relation relativ klein
- Die kleinere Relation als erste Relation hat Vorteile
  - bei Nested Loop Join
  - bei Hash Join

Index und Select-Befehl

- Index zu einem Attribut kann nur verwendet werden, wenn direkt dieses Attribut im Select verwendet wird!
- Negatives Beispiel:
  - Suche nach einem Namen, der indiziert ist
  - Verwendet wird: … Where Trim(Name) Like …
  - Index kann nicht mehr verwendet werden, da auch führende Leerzeichen entfernt werden!
  - Hervorragende Alternative:
    - … Where RTrim(Name) Like …

Das noch verbleibende rechtsseitige Trimmen stört den Index nicht

Index und Oracle

- Standard: Setzen von Indexen auf Attribute
- Oracle: Zusätzlich Setzen von Indexen auf Ausdrücke!
- Beispiel:

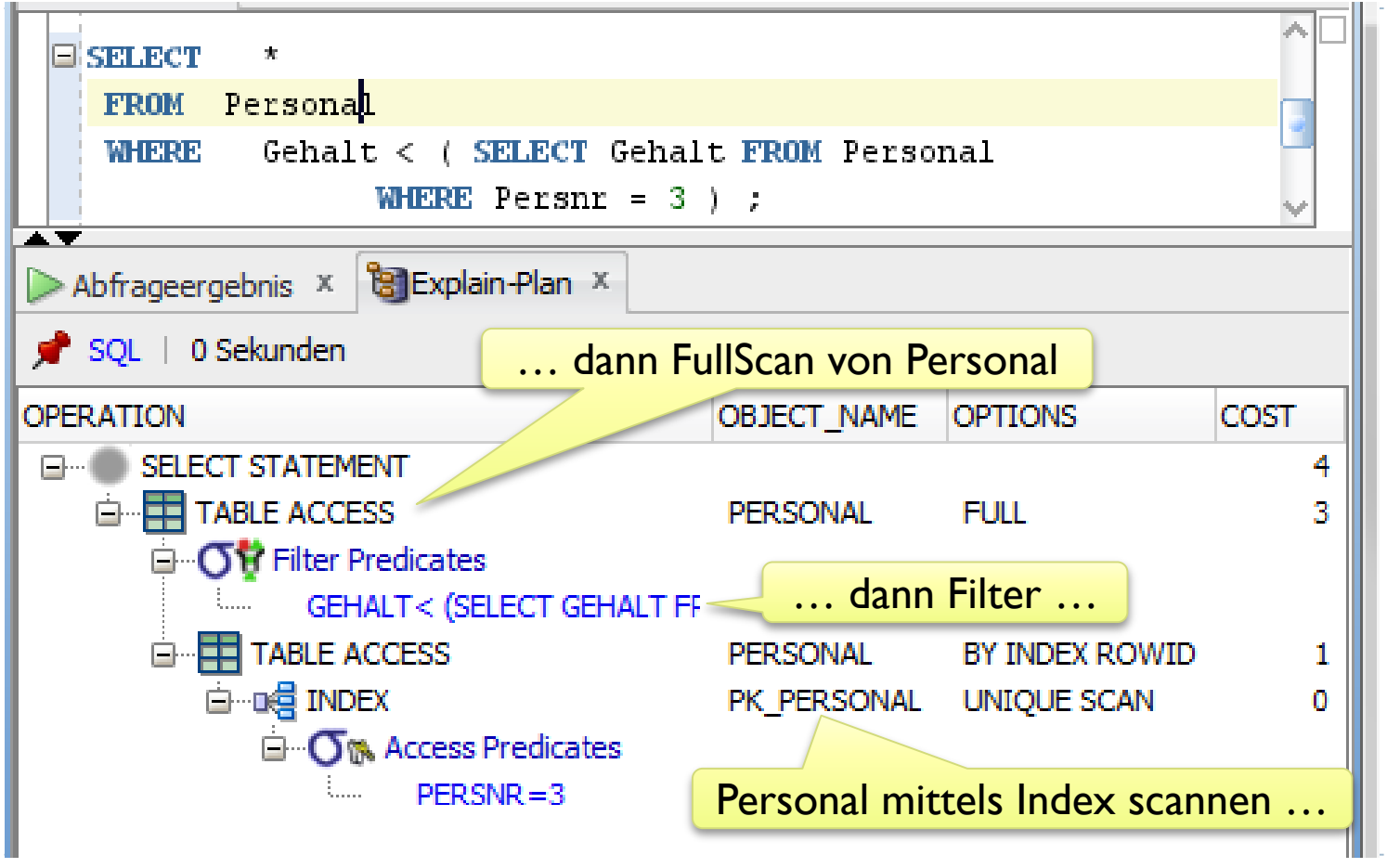
```
CREATE INDEX IUmfrageWohnort
                                ON Umfrage(Upper(Wohnort)) ;
```

- Der Index wird auf die Großbuchstaben angewendet
- Nur bei Verwendung von Upper(Wohnort) ist die Suche jetzt schnell

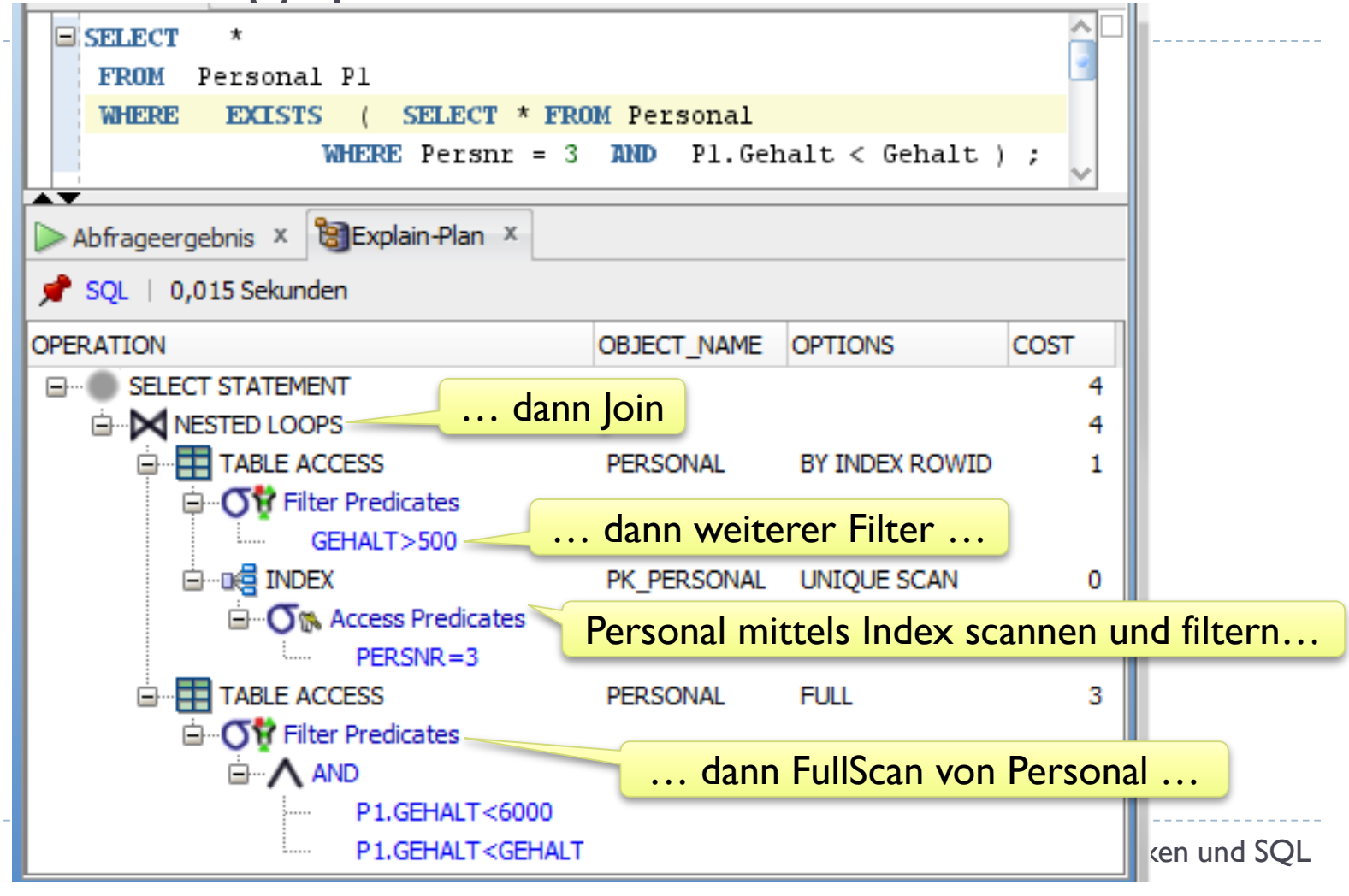
Vergleich zwischen In und Exists

- Trugschluss:
  - Der Exists-Operator ist in der Regel nicht langsamer als der In-Operator!
  - Das Gleiche gilt für Not Exists und Not In
- Beispiel:
  - Ausgabe aller Mitarbeiter, die weniger als Mitarbeiter 3 verdienen, siehe Kapitel 5
    - Lösung mit Vergleichsoperator
    - Lösung mit Verbund
    - Lösung mit Exists-Operator

Ausführungsplan mit Vergleich



Ausführungsplan mit Exists



Hinweise (HINTS)

- Nicht immer kennt der Optimizer die Daten optimal
- Dem Optimizer können daher Hinweise gegeben werden
- Beispiel:
  - Wir wollen, dass bei der Abfrage auf den Wohnort in der Relation Umfrage kein Index verwendet wird
  - Lösung in Oracle:

    Quasi als Kommentar

    Select /*+ NO_INDEX(Umfrage) */ * From Umfrage Where …
  - Lösung in SQL Server:

    Erzwingt das Scanner der Tabelle

    Select * From Umfrage With (FORCESCAN) Where …
  - Lösung in MySQL:

    Select * From Umfrage Ignore Index(IUmfrageWohnort) Where …

Stored Procedures

- Eine Stored Procedure
  - ist eine Prozedur (procedure)
  - wird in der Datenbank abgespeichert (stored)
- Eine Stored Procedure enthält
  - SQL Befehle
  - Variablen-Deklarationen
  - Anweisungen
- Stored Procedure sind normiert mit
  - SQL 3 (SQL 1999)
  - SQL 2003

z.B. Select, Insert, Update, Delete, Create Index

z.B.: DECLARE nummer INT;

z.B.: SET nummer = 3;

Stored Procedure versus Trigger

- Gemeinsam:
  - Prozeduren
  - In Datenbank gespeichert
- Unterschiede:
  - Trigger werden bei Ereignissen automatisch aufgerufen
  - Stored Procedures werden explizit aufgerufen
- Syntax:
  - CREATE TRIGGER
  - CREATE PROCEDURE

Beispiel

- Prozedur RABATT:
  - Parameter:
    - ARTNR: Artikelnummer, auf den Rabatt gewährt wird
    - NACHLASS: Der Preisnachlass für den Artikel
  - Idee:
    - Wenn der Nachlass unter 50% des eingetragenen Preises liegt, so wird der angegebenen Nachlass vom Preis des angegebenen Artikels abgezogen. Der neue Preis wird in der Relation eingetragen.
    - Sonst wird genau ein Nachlass von 50% gewährt. Der neue Preis wird entsprechend eingetragen.
  - Realisierung:
    - CREATE PROCEDURE Rabatt

Prozedur RABATT in Oracle

nicht NUMERIC(8,2)

```
CREATE OR REPLACE PROCEDURE Rabatt ( artnr INT , nachlass NUMERIC ) AS
    altPreis  NUMERIC(8,2);
    neuPreis  NUMERIC(8,2);
BEGIN
    SELECT Preis     INTO altPreis
    FROM Artikel     WHERE anr = artnr;

    neuPreis := altPreis - nachlass;
    IF neuPreis < 0.5 * altPreis     THEN neuPreis := 0.5 * altPreis;   END IF;

    UPDATE Artikel
    SET  Preis = neuPreis, Netto = Netto * neuPreis / altPreis, Steuer = Steuer * neuPreis / altPreis
    WHERE Anr = artnr ;
EXCEPTION WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE ('Fehler beim Ausfuehren der Prozedur Rabatt');
END;
/
```

kein: DECLARE … (gemäß Norm)

speichert Attribut Preis in Variable altPreis

kein: SET var = … (gemäß Norm)

IF: normkonform!

Ausnahmebehandlung in Oracle
WHEN OTHERS: alle sonstigen Ausnahmen

Begrenzer in Oracle

gibt Zeile aus

Prozedur RABATT in SQL Server

Keine Klammern!

```
CREATE PROCEDURE Rabatt @artnr INT, @nachlass NUMERIC(8,2) AS
DECLARE @altPreis NUMERIC(8,2), @neuPreis NUMERIC(8,2);
BEGIN TRY
    SELECT @altPreis = Preis
    FROM Artikel     WHERE anr = @artnr;

    SET @neuPreis := @altPreis - @nachlass;
    IF @neuPreis < 0.5 * @altPreis     SET @neuPreis := 0.5 * @altPreis;

    UPDATE Artikel
    SET Preis = @neuPreis, Netto = Netto * @neuPreis / @altPreis,
                                                    Steuer = Steuer * @neuPreis / @altPreis
    WHERE Anr = @artnr ;
END TRY
BEGIN CATCH
    SELECT 'Fehler beim Ausfuehren der Prozedur Rabatt';
END CATCH
GO
```

Variablen beginnen mit @

speichert Attribut Preis in Variable altPreis

IF: nicht normkonform!

TRY - CATCH

Begrenzer in SQL SERVER

Prozedur RABATT in MySQL

```
DELIMITER //

CREATE PROCEDURE Rabatt ( artnr INT , nachlass NUMERIC(8,2) )
BEGIN
    DECLARE   altPreis NUMERIC(8,2);
    DECLARE   neuPreis NUMERIC(8,2);

    SELECT Preis     INTO altPreis
    FROM Artikel     WHERE anr = artnr;

    SET neuPreis = altPreis - nachlass;
    IF neuPreis < 0.5 * altPreis    THEN SET neuPreis = 0.5 * altPreis;  END IF;

    UPDATE Artikel
    SET Preis = neuPreis, Netto = Netto * neuPreis / altPreis, Steuer = Steuer * neuPreis / altPreis
    WHERE Anr = artnr ;
END;
//
```

Definition eines Begrenzers

DECLARE-Teil

speichert Attribut Preis in Variable altPreis

normgemäß

Aufruf von Prozeduren

- Aufruf erfolgt in Oberfläche (Oracle, SQL Server)
- Aufruf erfolgt in Kommandozeile:
  - Oracle: `execute rabatt ( 100002, 50 )`
  - SQL Server: `execute rabatt 100002, 50`
  - MySQL `call rabatt ( 100002, 50 )`
- Damit wird Artikel 100002 um 50 Euro günstiger

Optimierung: Transaktionsbetrieb

- In MySQL
  - Abschalten von Autocommit
  - SET Autocommit = 0
    - oder
  - START TRANSACTION … COMMIT

Optimierung: bindParam

- Oft werden Befehle mit modifizierten Parametern mehrfach aufgerufen
- Dann: Arbeiten mit bindParam (prepare + execute, statt query)

```
$sql = "Select Auftrnr, Artnr, Anzahl, Gesamtpreis
        From Auftragsposten  Where Auftrnr = ?" ;
$stmt = $conn->prepare($sql);

$stmt->bindParam( 1, 1 ) ;
$stmt->execute( ) ;
$stmt->bindParam( 1, 2 ) ;
$stmt->execute( ) ;
$stmt->bindParam( 1, 4 ) ;
$stmt->execute( ) ;
```

variabel

Nur einmal geparst und aufbereitet!

Ersetzt 1. Fragezeichen durch den Wert 1

Ersetzt 1. Fragezeichen durch den Wert 2

Ersetzt 1. Fragezeichen durch den Wert 4

Mehrfach ausgeführt

Zusammenfassung

- Eine Datenbank führt selbst Optimierungen durch
- Eine Datenbank stellt viele Werkzeuge zur Optimierung zur Verfügung:
  - Indexe
  - Partitionierungen
  - Materialisierte Sichten
  - Stored Procedures
- Der Anwender und der Systemverwalter setzen diese Werkzeuge gezielt ein
- Der Anwender optimiert zusätzlich

Datenbanken und SQL

Kapitel 8

Concurreny und Recovery

Concurrency und Recovery

- Transaktionen
- Recovery
  - Einführung in die Recovery
  - Logdateien
  - Checkpoints
- Conncurrency
- Sperrmechanismen
- Deadlocks
- SQL-Norm und Concurrency
- Concurrency in Oracle, SQL Server und MySQL

Transaktionen in Datenbanken

- **Garantie an den Anwender:**
  - Eine Transaktion wird im Fehlerfall komplett zurückgesetzt, wenn sie nicht mehr beendet werden kann
  - Alle Daten einer abgeschlossenen Transaktion sind persistent
  - Eine Transaktion läuft unabhängig von anderen parallel laufenden Transaktionen
  - Die Datenbank ist immer in sich schlüssig und korrekt
- **Dies heißt:**
  - Die Garantie gilt lebenslang und unter allen Umständen!
  - Die Garantie gilt auch bei Blitzeinschlag oder Attentaten!
  - Die Garantie gilt auch bei tausenden parallelen Anwendungen!

Atomarität

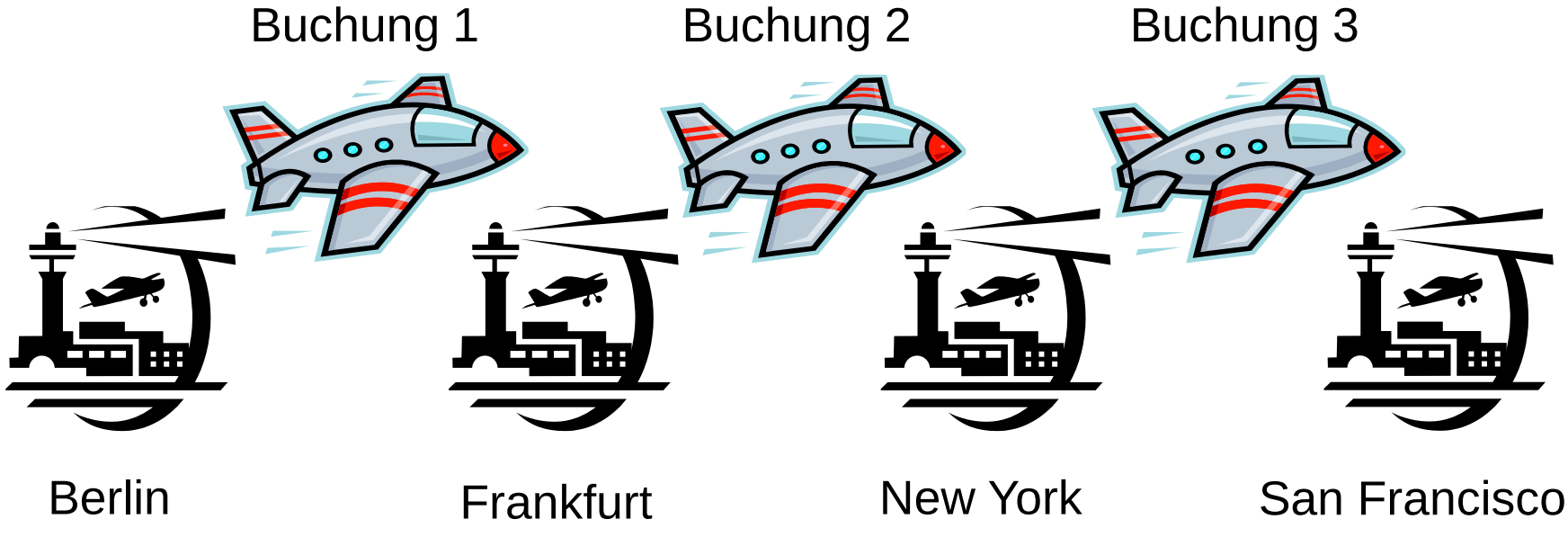
Dauerhaftigkeit

Isolation

Konsistenz

Beispiel: Flug Berlin – San Francisko

- Flug via Frankfurt und New York
- Drei Buchungen ergeben eine Transaktion



Flugbuchung

```
$conn->query( UPDATE Flugbuchung  SET ...  ) ;  // BER - FRA
$conn->query( UPDATE Flugbuchung  SET ...  ) ;  // FRA - NYC
$conn->query( UPDATE Flugbuchung  SET ...  ) ;  // NYC - SFO
$conn->commit() ;
```

Zusammen:
Eine Transaktion

- Konsistenzmodell ACID:
  - Buchung erfolgt nur komplett oder überhaupt nicht
  - Dies gilt auch bei Rechnerabsturz!

ACID:
- Atomarität
- Konsistenz
- Isolation
- Dauerhaftigkeit

Recovery

- Recovery
  - Wiederherstellung von Daten bei schweren Fehlern
- HW-Fehler:
  - Stromausfall, Wackelkontakt, Festplattenausfall, Arbeitsspeicherausfall, Brand (Feuer, Löschwasser)
- SW-Fehler:
  - Fehler in Datenbank-SW, Betriebssystem-SW, Netz-SW, Anwendungsprogramm

Auch: Anschlag, Terrorismus, Flugzeugabsturz

In der Verantwortung des Datenbankprogrammierers!

Vorsorge

- Hardware-Einkauf
  - Nur Geräte, die lange Standzeiten garantieren
  - Keine Geräte von der Stange
- Software-Einkauf
  - SW-Komponenten aufeinander abstimmen
  - Nur zuverlässige und zertifizierte Software einsetzen
- Sicherung
  - Tägliche Sicherung der Daten (Differenzsicherung)
  - Mindestens einmal pro Woche: Komplettsicherung
  - Sichere Aufbewahrung der Sicherungen

In der Regel: Nur zertifizierte Rechner

Updates vorher testen!

Oft viele GB pro Tag …

… und viele TB pro Woche

Sicherung des Datenbestands

Zeitlicher Rahmen	Sicherung
wöchentlich	Komplettsicherung des Datenbestandes
täglich	Differenzsicherung
im laufenden Betrieb	Protokollierung jeder Änderung in einer Logdatei

- Aufwand:
  - Nächtliche Sicherungen
    - Stören den Regelbetrieb nur geringfügig
  - Protokollierung
    - Erhebliche Störung des Regelbetriebs wegen der Erstellung des Protokolls und der Speicherung in Datei (Logdatei)

Performance-Problem!

Wie gegensteuern?

Sicherer Datenbankbetrieb

- Voraussetzungen
  - Datenbank befindet sich auf externen nichtflüchtigen Medien
  - Logdaten befinden sich auf externen nichtflüchtigen Medien
  - Datenbankdaten werden im Arbeitsspeicher zwischengespeichert (Performance)
    - Bezeichnung: Datenbank-Puffer, Datenbank-Cache
  - Daten werden vom Datenbank-Puffer gelesen
  - Sind Datum nicht im Puffer, so werden sie von der Datenbank geholt
  - Das Schreiben der Daten geschieht im Datenbank-Puffer
  - Aktualisierung der Datenbank erfolgt asynchron

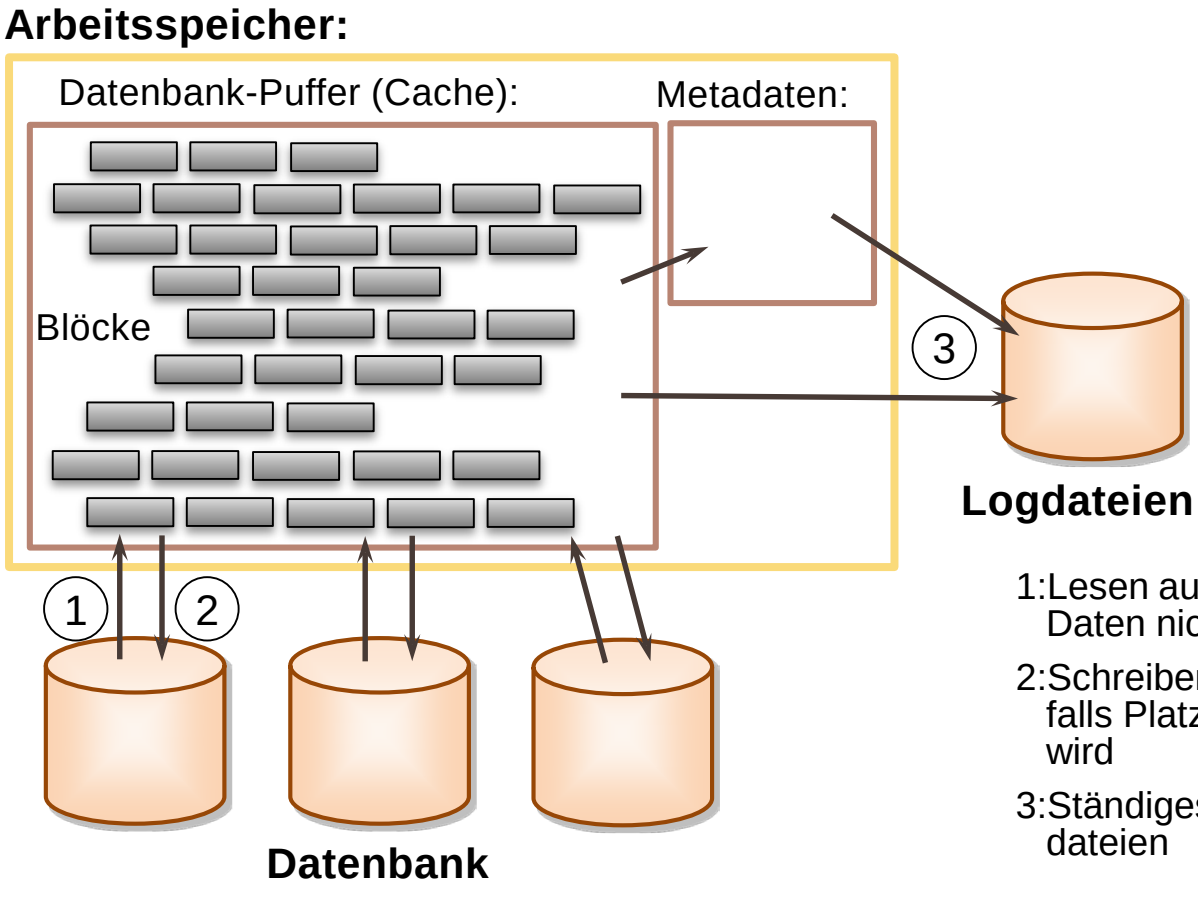
Medien sind direkt zugreifbar

Reduziert Anzahl der I/Os

Metadaten

- Metadaten sind Daten, die Informationen zu und Zustände über eine Datenbank merken
- Metadaten sind u.a.
  - Informationen zu laufenden Transaktionen
  - Zustände zu Synchronisationsmechanismen
  - Aktualität und Gültigkeit der Daten im Datenbankpuffer
  - Zustand der aktuellen Logdaten
- Aktuelle Metadaten werden vor allem im Arbeitsspeicher gehalten

Datenbankpuffer



1:Lesen aus der Datenbank, falls Daten nicht im Cache

2:Schreiben in die Datenbank, falls Platz im Cache benötigt wird

3:Ständiges Sichern in die Log-dateien

Datenbankpuffer - Erläuterung

- Datenbankpuffer
  - Nimmt einen großen Teil des Arbeitsspeichers in Anspruch
  - In großen Datenbanken auch mehr als 1 TB
  - Bereits gelesene Daten müssen nicht nochmals gelesen werden!
- Daten werden bei Bedarf von der Festplatte geholt und im Datenbankpuffer gehalten
- Daten werden ausschließlich im Datenbankpuffer bearbeitet
- Geänderte Daten werden bei Engpässen im Puffer in die Datenbank zurückgeschrieben
- Parallel werden Änderungen sofort in die Logdateien geschrieben und damit persistent gesichert

Die Idee der Pufferung!

Also: keine ständigen I/Os

Damit müssen geänderte Daten nicht sofort in die Datenbank geschrieben werden

Before- und After-Image

- Jedes Ändern, Löschen oder Einfügen führt dazu, dass Daten manipuliert werden. Wir unterscheiden:
- Before-Image:
  - Die zu ändernden Daten
- After-Image:
  - Die geänderten Daten
- Ein Before-Image wird bis Transaktionsende gespeichert
- Ein After-Image wird bis zur nächsten Sicherung gespeichert

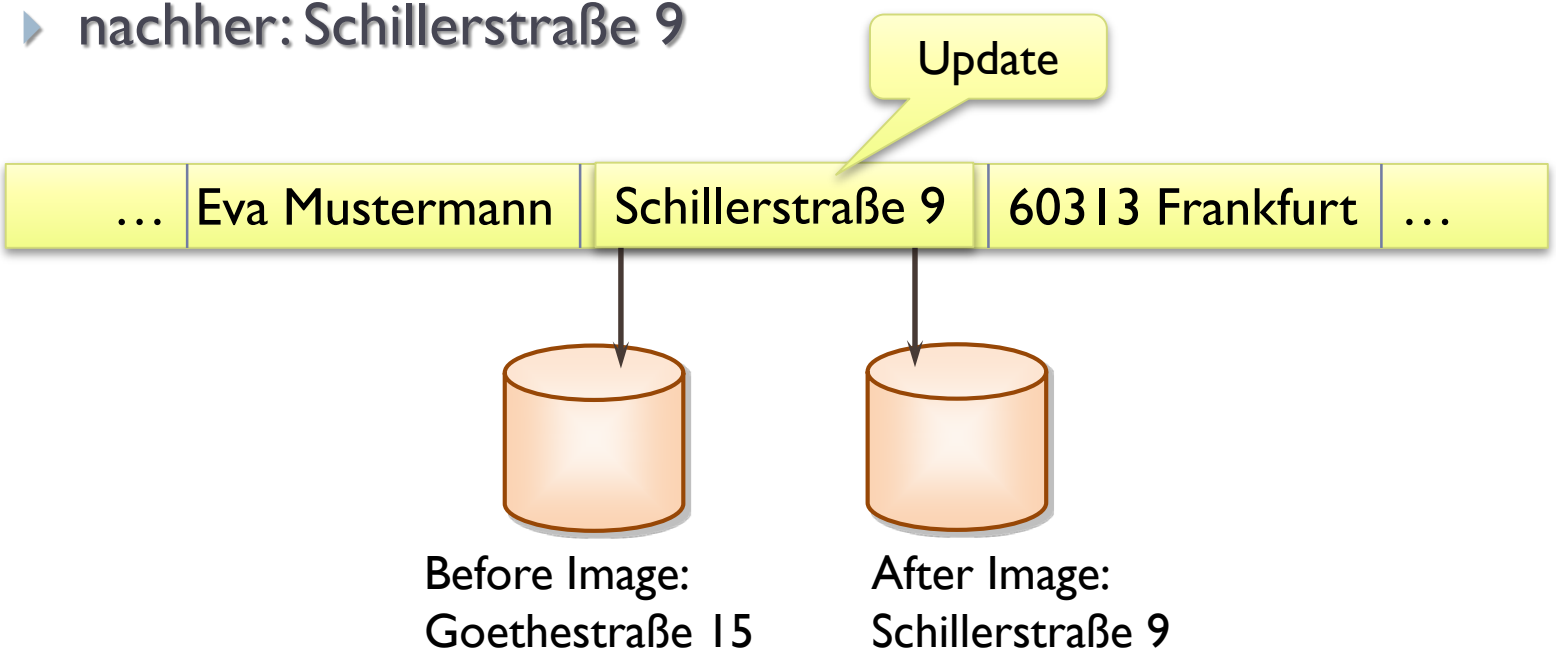
Das Abbild, bevor geändert wird

Das Abbild, nachdem geändert wurde

Warum wohl?

Before- und Afterimage am Beispiel

- Wir wollen die Adresse eines Mitarbeiters ändern
  - vorher: Goethestraße 15
  - nachher: Schillerstraße 9



Einfacher Transaktionsbetrieb

Lesen der Daten	Die Daten werden von der Datenbank eingelesen, falls sie sich nicht bereits im Puffer des Arbeitsspeichers befinden.
Merken der bisherigen Daten	Die zu ändernden Daten werden in die Logdatei geschrieben (Before Image).
Ändern der Daten	Ändern (Update, Delete, Insert) der Daten im Arbeitsspeicher, Sperren dieser Einträge für andere Benutzer.
Merken der geänderten Daten	Die geänderten Daten werden in die Logdatei geschrieben (After Image).
. . .	Obige vier Schritte können sich innerhalb einer Transaktion mehrmals wiederholen.
Transaktionsende mit COMMIT	Transaktionsende in der Logdatei vermerken. Sperren freigeben. Schreiben aller Änderungen in die Datenbank.
Transaktionsende mit Rollback	Rücksetzen der Metadaten der Transaktion. Geänderte Daten im Arbeitsspeicher mittels Before-Images restaurieren. Sperren freigeben.

Schwächen des einfachen TA-Betriebs

- Daten von sehr lange laufenden Transaktionen werden im Arbeitsspeicher gehalten
- Absturz während des Schreibens der geänderten Daten am Transaktionsende:
  - Komplexe Recovery: Es muss überprüft werden, welche Daten schon geschrieben wurden und welche nicht
- Das Schreiben immer zu Transaktionsende kann zu punktuellen Überlasten führen
- Das Schreiben immer zu Transaktionsende ist inflexibel

Transaktionsende in der Praxis

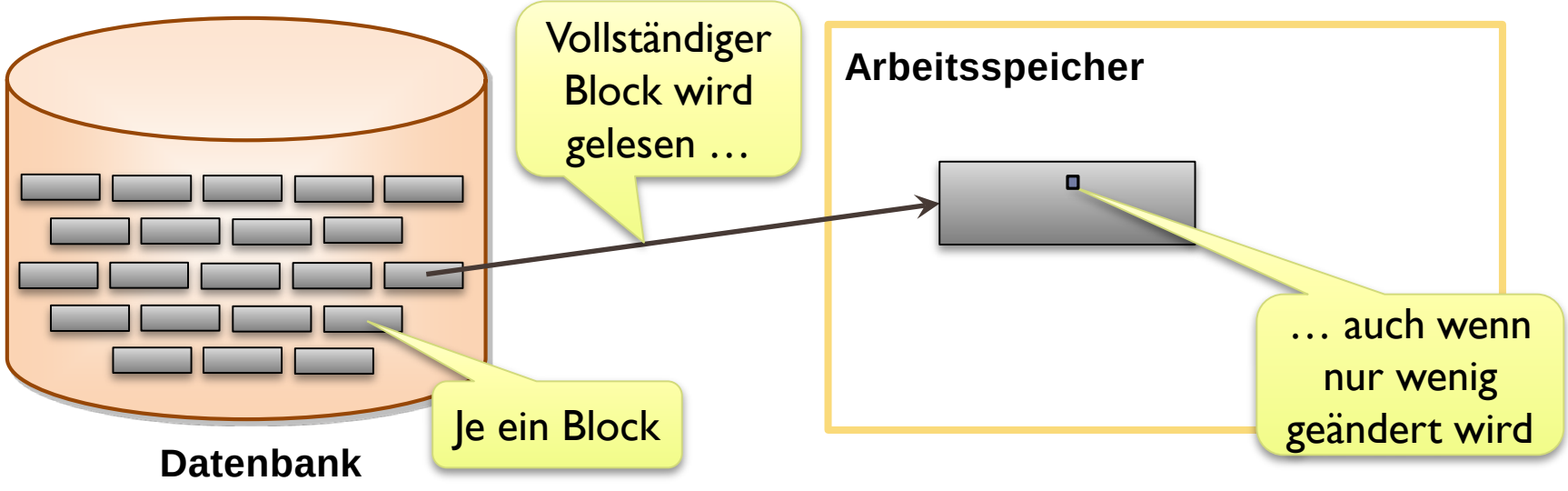
Erste vier Schritte	Wie bisher, auch wiederholt
Transaktionsende mit COMMIT	Transaktionsende in der Logdatei vermerken. Sperren freigeben.
Transaktionsende mit ROLLBACK	Rücksetzen der Metadaten der Transaktion. Geänderte Daten im Arbeitsspeicher mittels der Before-Images restaurieren. Alle geänderten Daten, die bereits in die Datenbank geschrieben wurden, werden für ungültig erklärt. Sperren freigeben.
Änderungen speichern	Die geänderten Daten werden asynchron (unabhängig vom Transaktionsbetrieb) in die Datenbank geschrieben.

Transaktionsbetrieb mit Pufferung

- Hochperformant
  - Der I/O-Verkehr wird minimiert
  - Werden Daten mehrmals geändert, so müssen diese nicht jedes Mal geschrieben werden
  - Werden gelesene Daten wieder gelesen, so stehen diese bereits zur Verfügung
- Die Konsistenz der Daten hängt wesentlich von den Logdateien ab
  - Dauerhaftigkeit nur dank Logdateien gesichert

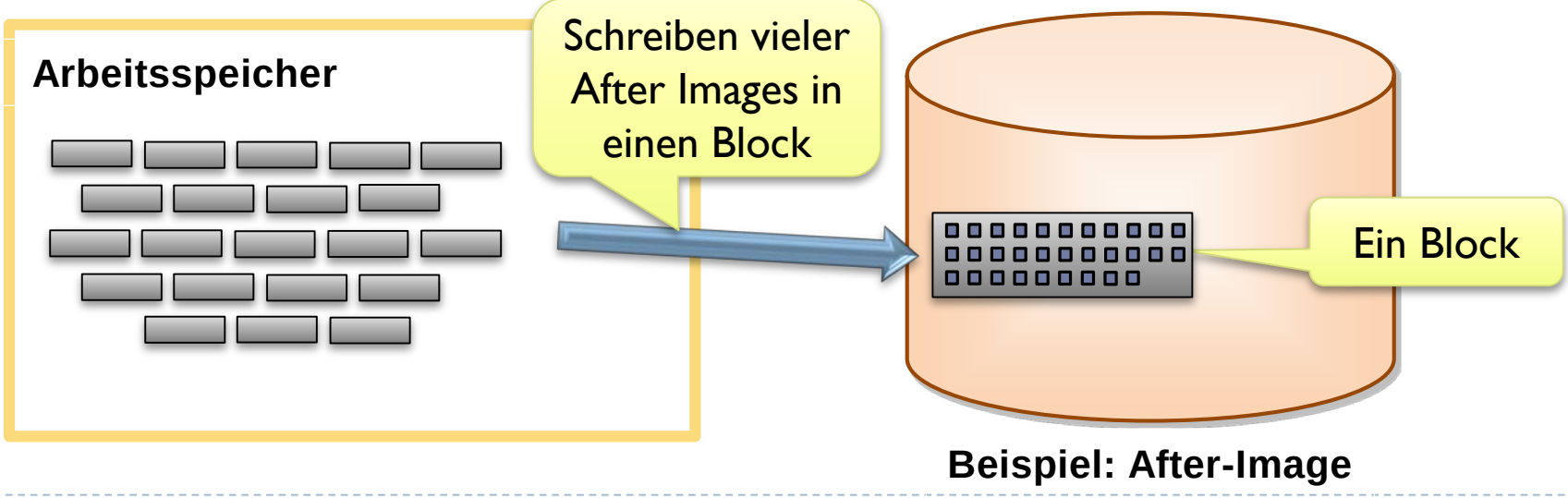
Wiederholung: Blockstruktur

- Alle Festplatten besitzen Blockstruktur
  - Formatiert in 2kB, 4kB, 8kB, 16kB, 32kB Blöcke
- Datenbank übernimmt die Struktur
  - Kann Blöcke noch zusammenfassen: bis zu 64kB Blöcke



Struktur der Before- und After-Images

- Logdateien besitzen ebenfalls Blockstruktur
- Aber: Before- und Afterimages enthalten nur die Änderungen!
- Viele Änderungen werden in einem Block zusammengefasst
- Daher: Geringer Schreibverkehr; zusätzlich: gestreamt



Inhalt der Logdateien

- Die Logdateien enthalten
  - alle Before-Images mit Transaktionsnummer und Zeitstempel
  - alle After-Images mit Transaktionsnummer und Zeitstempel
  - dazugehörige Metadaten
    - Transaktionsnummer,
    - Transaktionsende
    - gehaltene Sperren
    - weitere Infos
- Alle Logdaten müssen aber nicht gleich lange aufbewahrt werden!

Undo-Log und Redo-Log

- **Undo-Log:**
  - Eine Logdatei, die alle Before-Images und dazugehörige Metadaten enthält
  - Ein Undo-Log-Eintrag muss nur bis Transaktionsende aufgehoben werden
- **Redo-Log:**
  - Eine Logdatei, die alle After-Images und dazugehörige Metadaten enthält
  - Ein Redo-Log muss bis zur nächste Sicherung aufgehoben werden

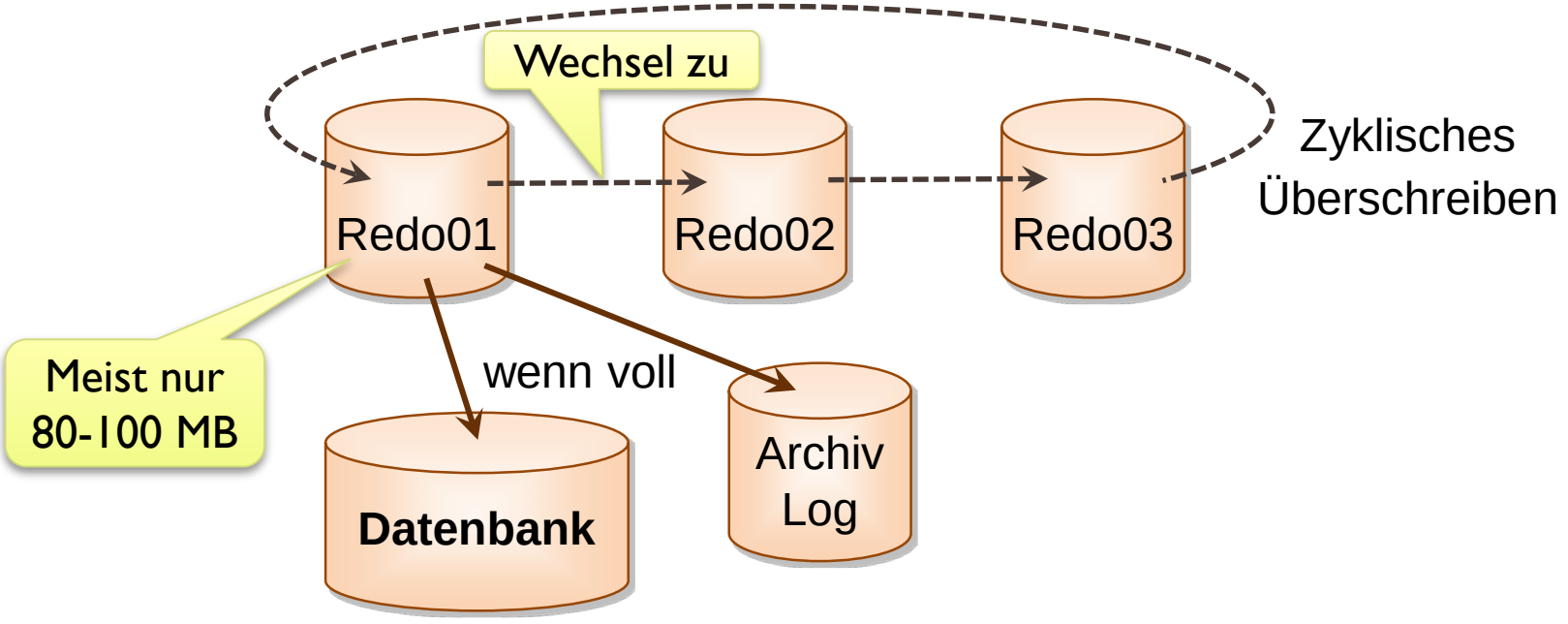
Undo-Log

- Undo-Log wird in speziellem Datenbankbereich gehalten
  - Oracle: Tablespace UNDOTBS1
- Undo-Log wird für Rollback benötigt
- Von einem Undo-Log wird auch gelesen
- Undo-Log wird in der Regel zyklisch überschrieben
- Undo-Log-Einträge müssen auf Festplatte stehen, bevor Daten nicht abgeschlossener Transaktionen in die Datenbank geschrieben werden!

Redo-Log

- Redo-Log ist „Lebensversicherung“ des aktuellen Daten-bestands
- Redo-Logs werden ausschließlich sequentiell beschrieben
- Größe eines Redo-Logs:
  - in MySQL: max. 512 GB (ab V5.6)
  - in Oracle: zusätzlich Archive Log
  - in SQL Server: Log Backup
- Redo-Logs werden auf eigenem externen Medium angelegt
- Redo-Logs werden häufig gespiegelt (Raid 1) (Sicherheit)

Redo-Logs in Oracle



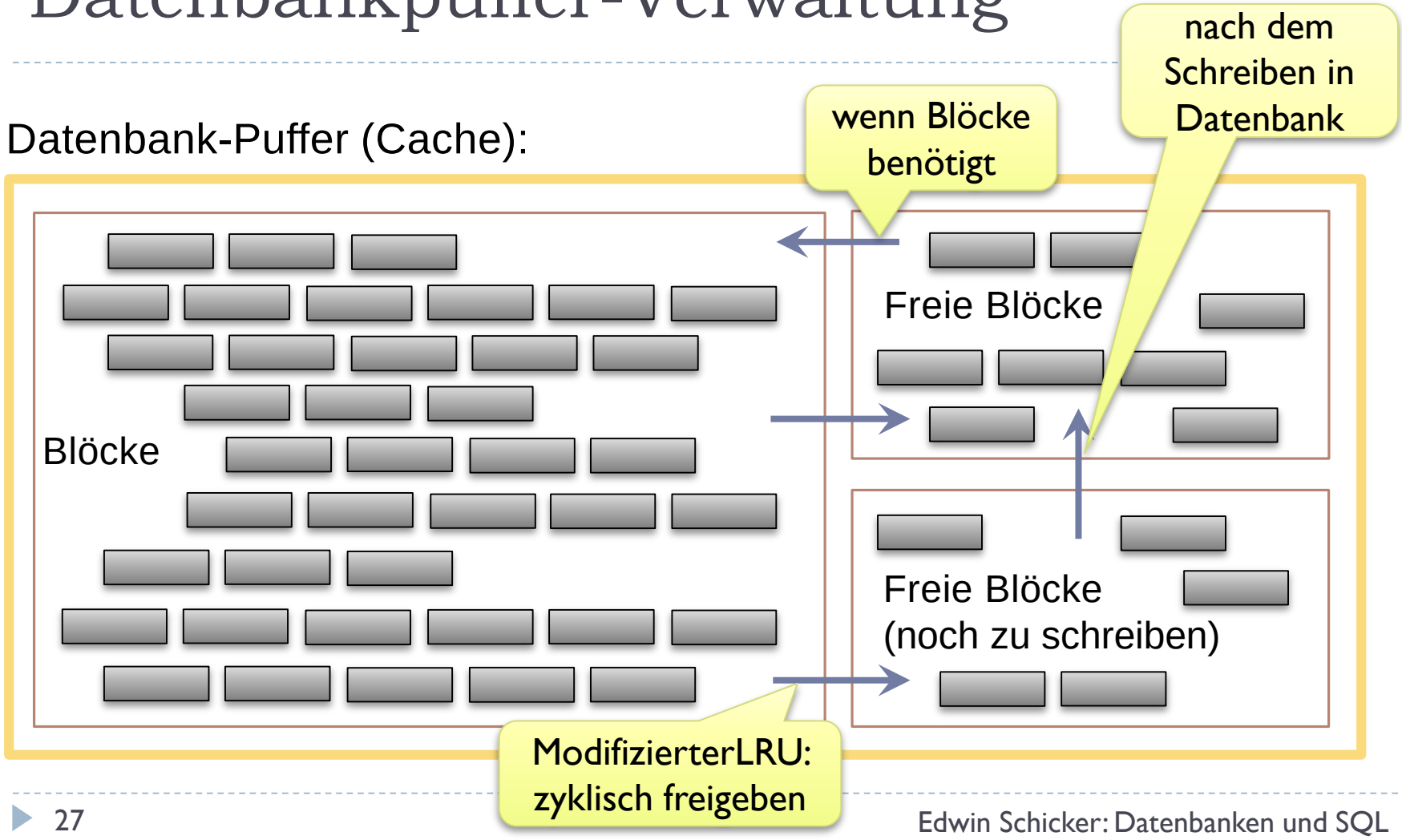
- Es existieren mindestens 2 Redo-Logs und 1 Archiv-Log
- Archiv-Log kann „billigeres“ Medium sein

LRU Algorithmus

- Wenn Datenbankpuffer voll, müssen Seiten verdrängt werden → LRU Algorithmus
- LRU (Least Recently Used) Algorithmus:
  - Ausgewählt wird die Seite, die am längsten nicht mehr verwendet wurde
- Gute Erfahrung, da
  - ältere Seiten häufig nicht mehr benötigt
  - jüngere Seiten eventuell nochmals verwendet werden

Datenbankpuffer-Verwaltung

Datenbank-Puffer (Cache):



Hot Spots

- Hot Spots sind Daten,
  - auf die immer wieder zugegriffen wird
  - die deshalb nie verdrängt werden

  - Sehr gute Performance
    - da weniger I/Os
  - Alter Datenbestand auf Festplatte

    - Aufwändige Recovery
      - Alle Änderungen zu den Hot Spots sind nachzuvollziehen
    - Viele Metadaten
      - Kein zyklisches Überschreiben der Metadaten möglich

Zyklisches Überschreiben erlaubt einfache Handhabung

Checkpoints

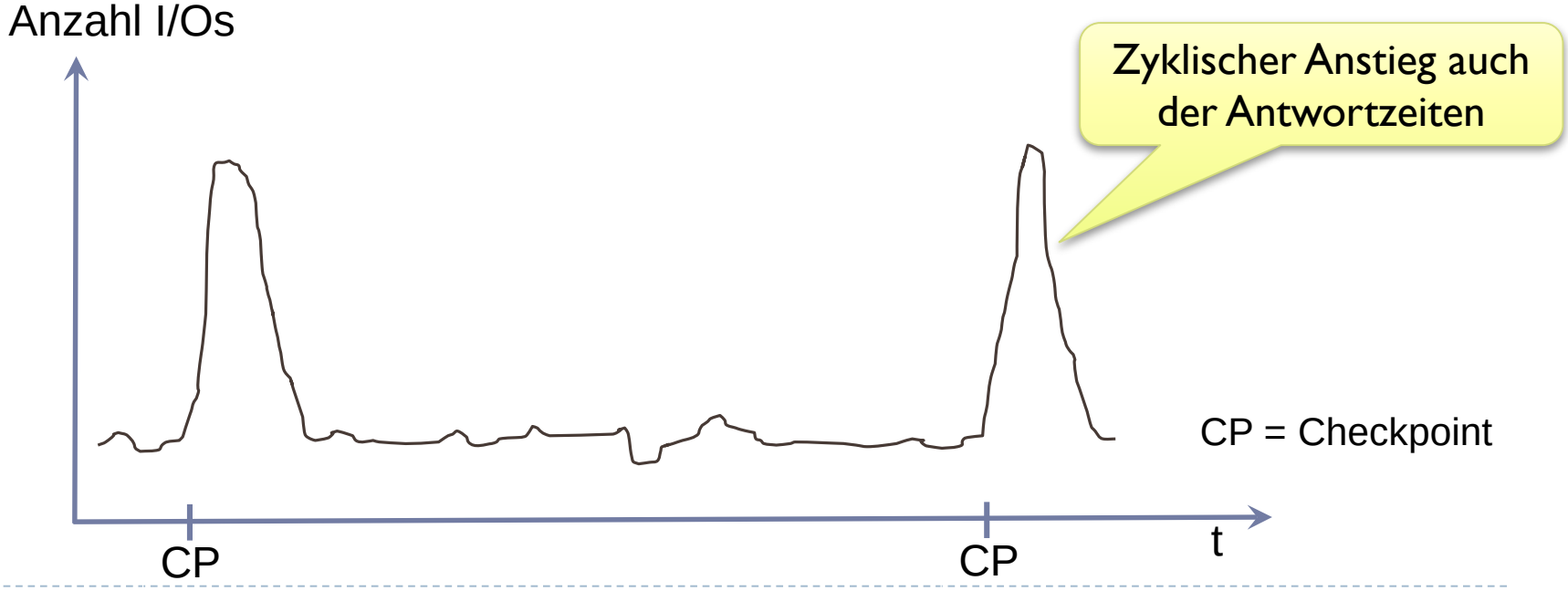
- Checkpoints sind Zeitpunkte, wo alle geänderten Daten zwangsweise in die Datenbank geschrieben werden

  - Nachteil:
    - Punktuell sehr viele I/Os
    - Dadurch auch Behinderung laufender Transaktionen

  - Vorteil:
    - Im Recoveryfall sind nur Redo-Daten seit dem letzten Checkpoints nachzuvollziehen
    - Viele Metadaten können gelöscht werden

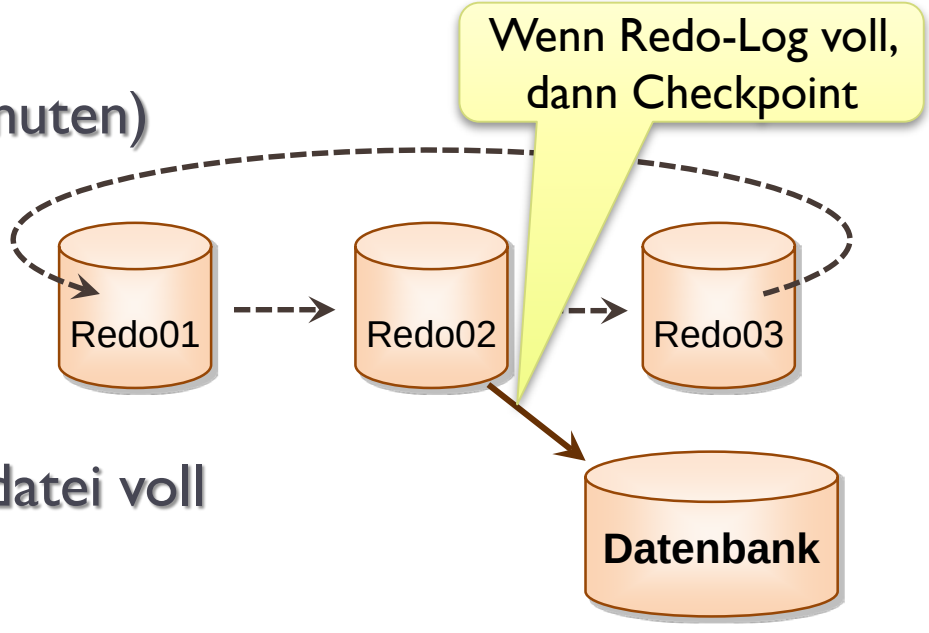
Nachteil von Checkpoints

- Die hohe I/O-Last behindert alle Transaktionen
- Bei jedem Checkpoint steigen die Antwortzeiten

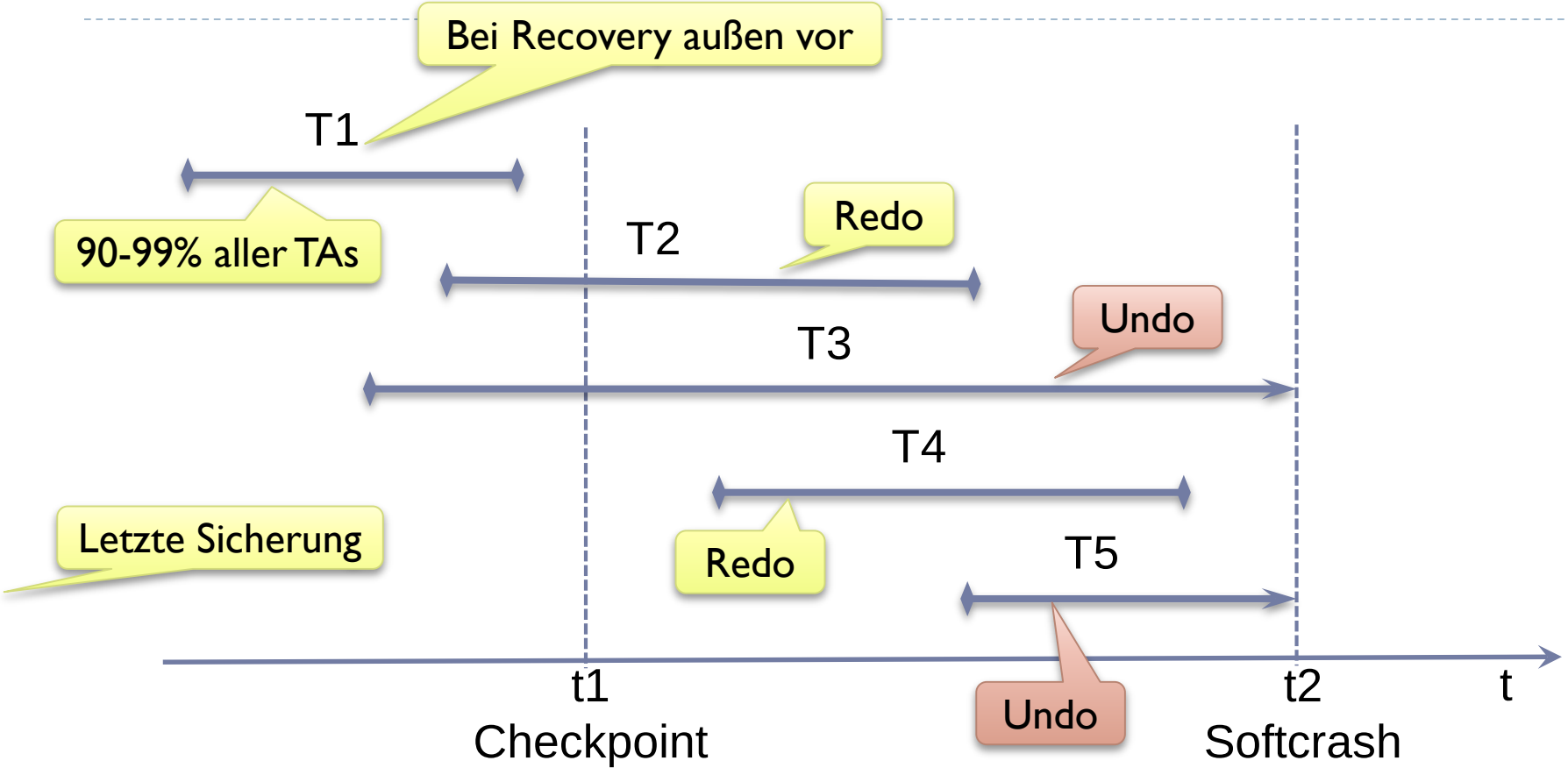


Häufigkeit von Checkpoints

- Zwei Möglichkeiten
  - Zeitgesteuert (z.B. alle 15 Minuten)
  - Ereignisgesteuert
- In Oracle:
  - Checkpoint, wenn Redo-Logdatei voll
- Optimale Einstellung:
  - Parameter sind: Zeitintervall bzw. Größe des Redo-Logs
  - Werte hängen von Erfahrung ab



Transaktionen beim Crash



Sicherheit geht über alles!

- Normalfall:
  - Datenbank, Redo-Logs, Undo-Log, Checkpoints
  - Problem: Während einer Recovery darf keine weitere Komponente ausfallen
- Mehr Sicherheit:
  - Zusätzlich: Redo-Logs werden gespiegelt (z.B. Raid 1)
- Noch mehr Sicherheit:
  - Zusätzlich: Datenbank wird gespiegelt (räumlich getrennt)
- Extreme Sicherheit:
  - Zwei komplett autarke Rechenzentren im Parallelbetrieb

Concurrency

- Concurrency beschäftigt sich mit dem Parallelbetrieb in Datenbanken
- Grundregel der Concurrency:

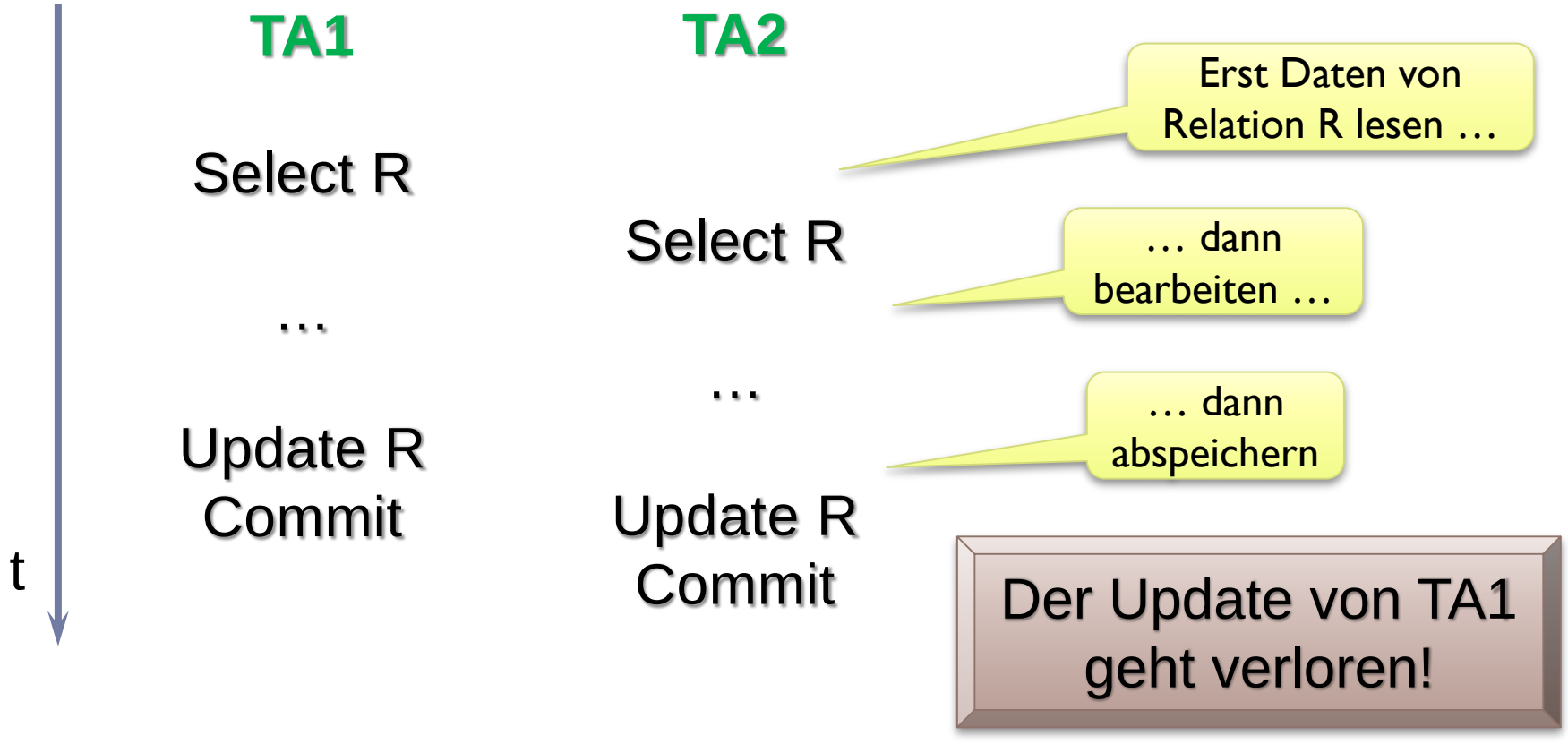
Jede Transaktion läuft so ab, als sei sie allein im System

- Insbesondere muss eine Transaktion Ergebnisse liefern, die unabhängig von anderen Transaktionen sind

Drei Concurrency Probleme

- Problem der verlorengegangenen Änderung
  - Zwei Transaktionen ändern (fast) gleichzeitig. Eine Änderung geht verloren
- Problem der Abhängigkeit von nicht abgeschlossenen Transaktionen
  - Daten werden gelesen, die mittels Rollback rückgesetzt werden
- Problem der Inkonsistenz der Daten
  - Fehlerhafte Daten werden gelesen, wenn andere Transaktionen gleichzeitig ändern

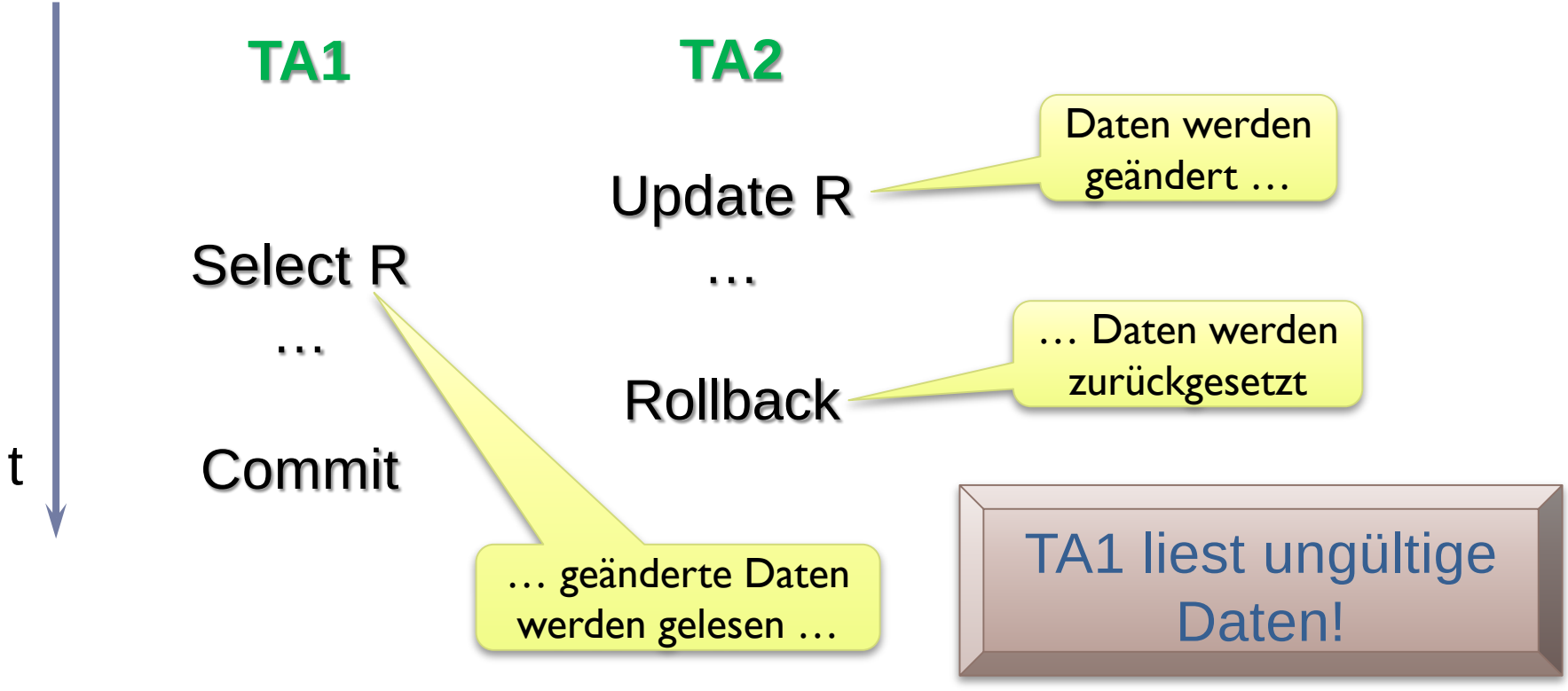
Verlorengegangene Änderung



Verlorengegangene Änderung

- Grundsätzlich nicht erlaubt
- Großes Problem in verteilten Systemen
- Beispiel:
  - Flugbuchung:
    - Die beiden letzten Plätze werden reserviert
    - Die Tickets werden ausgedruckt
    - Parallel dazu werden diese Plätze in anderem Reisebüro vergeben
    - Überbuchung trotz Sitzplatzbestätigung und Tickets!

Abhängigkeit von nicht abgeschl. TAen



Abhängigkeit von nicht abgeschl. TAs

- Problem sieht harmlos aus
- Es ist jedoch ein Problem bei konsequenter Ausnutzung dieser Lücke
- Beispiel:
  - Person hat Schulden, darf Konto nicht überziehen
  - Person muss 1000 Euro überweisen, Konto ist aber leer
  - Freund überweist 1000 Euro
  - Person kann überweisen
  - Freund führt einen Rollback durch!

Inkonsistenz der Daten

TA1	TA2
Summiere drei Kontoinhalte	*Überweise 600 € von Konto3 auf Konto1*
Select Konto1 (400 €)	Select Konto3 (700 €)
Select Konto2 (300 €)	Update Konto3 (700-600=100 €)
…	Select Konto1 (400 €)
Select Konto3 (100 €)	Update Konto1 (400+600=1000 €)
Ausgabe: Summe (**800 €**)	Commit
Commit	

In Wahrheit: 1400€

Inkonsistenz der Daten

- Problem sieht harmlos aus
- Aber:
  - Mit diesen falschen Werten könnte jetzt intern weiter gearbeitet werden!

- Konsequenz:
- Alle drei Probleme sind zu vermeiden
  - Problem 1 ist grundsätzlich sehr kritisch
  - Probleme 2 und 3 bei „harmlosen“ Transaktionen vorstellbar

z.B. Sammeln von einfachen Statistiken

Concurrency Strategien: optimistisch

- Optimistische Strategie:
  - Jede Transaktion darf beliebig lesen und ändern
  - Die drei Concurrency Probleme werden in Kauf genommen
- Aber:
  - Bei Transaktionsende wird auf parallele Zugriffe überprüft
  - Wenn keine parallelen Zugriffe: Alles OK
  - Wenn doch: Rücksetzen der Transaktion und Neustart
- Realisierung:
  - Zugriffszähler
- Nachteil:
  - Nur bei extrem niedriger Kollisionswahrscheinlichkeit einsetzbar
  - Gegenseitiges Aufschaukeln ist möglich: Immer wieder Neustarts

Concurrency Strategien: pessimistisch

- Pessimistische Strategie:
  - Alle von einer Transaktion angefassten Daten sind für andere Transaktionen gesperrt
  - Freigabe der Sperre am Transaktionsende
- Folgerung:
  - Andere Transaktionen müssen gegebenenfalls warten
- Realisierung:
  - Mit Sperrmechanismen (Locks)
- Nachteil:
  - Einschränkung der Parallelität
  - Hoher Verwaltungsaufwand für die Sperrmechanismen

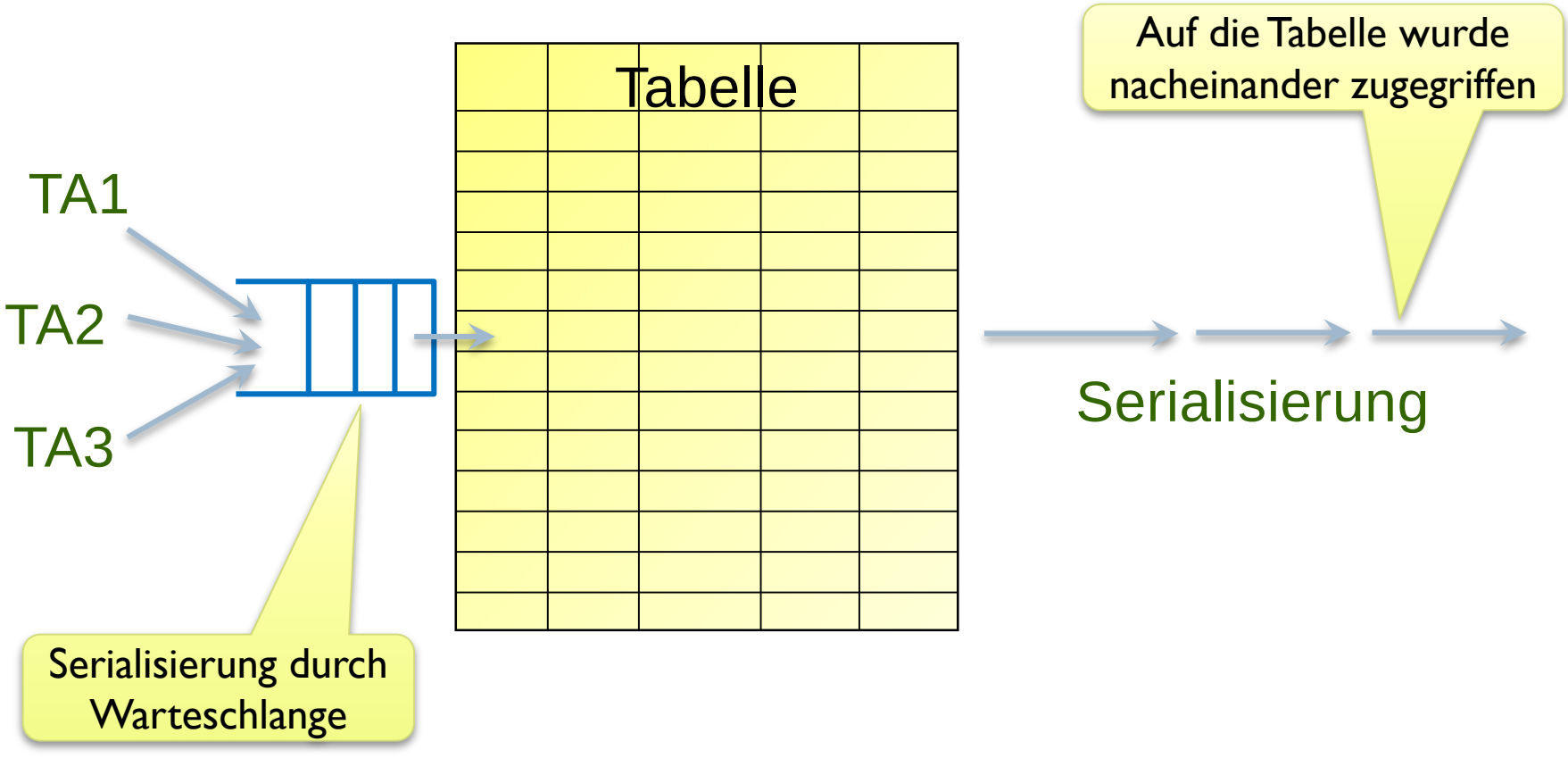
Vergleich der Concurrency Strategien

	Optimistische Strategie	Pessimistische Strategie
Vorteile	Einfache Implementierung Gute Performance Grundregel der Concurrency kann garantiert werden	Universell einsetzbar Grundregel der Concurrency kann garantiert werden
Nachteile	Kann sich aufschaukeln: daher nur in Spezialfällen einsetzbar	Aufwändige Implementierung Provoziert Wartezeiten anderer Transaktionen

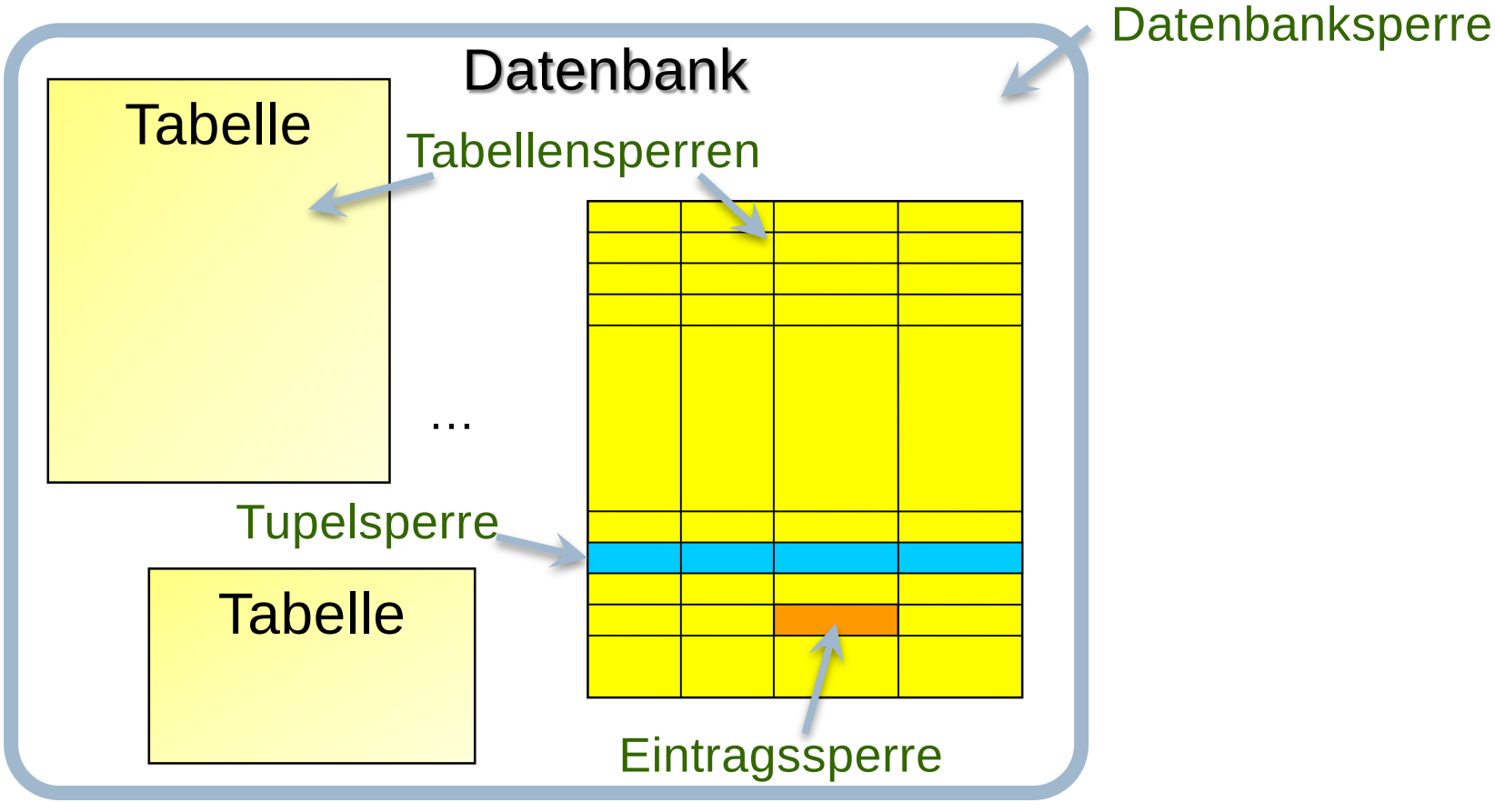
Sperrmechanismen

- Sperrmechanismen werden mit Locks realisiert
- Grundidee zu Locks in Datenbanken:
  - Zu jeder Relation existiert ein Lock
  - Eine Transaktion holt vor jedem Zugriff auf eine Relation automatisch den Lock dieser Relation
  - Ist der Lock von einer anderen Transaktion belegt, so wartet die Transaktion in einer Warteschlange, bis sie nach der Freigabe des Locks an der Reihe ist
  - Bei Transaktionsende werden alle gehaltenen Locks freigegeben

Lockmechanismus



Sperrgranulat in Datenbanken



Sperrgranulat in der Praxis (1)

- Datenbanksperre
  - Erlaubt keinen Parallelbetrieb
  - Nur im Einzelplatzbetrieb vorstellbar
- Tabellensperren
  - Relativ flexibel
  - Problem: Auf einzelne Tabellen wird intensiv zugegriffen
  - Bei geringer Parallelität gut einsetzbar
- Eintragssperren
  - Sehr sehr aufwändig
  - Daher kaum implementiert

Sperrgranulat in der Praxis (2)

- Tupelsperren
  - Alle wichtigen Datenbanken unterstützen Tupelsperren
  - Ist Standard in allen größeren Datenbanken
- Problem bei großen Datenbanken:
  - Beispiel: 1000 Tabellen mit je 100.000 Zeilen
  - Also: 100 Millionen unterschiedliche Locks!
- Implementierung:
  - Lockpool mit Poolverwaltung
  - Locks werden bei Bedarf eingerichtet

In Wirklichkeit noch viel mehr!

Lesende Zugriffe und Concurrency

- In der Praxis:
  - 80-90% Lesezugriffe
- Lesezugriffe sollten sich nicht gegenseitig behindern
- Aber:
  - Lesende dürfen Änderungen nicht lesen
  - Schreibende dürfen Daten nicht ändern, wenn vorher von anderen gelesen

vor dem Commit der anderen Transaktion!

Inkonsistenz der Daten!

Exklusiv- und Share-Lock

- **Definition (Exklusiv-Lock, Share-Lock)**
  - Ein Exklusiv-Lock auf ein Objekt weist alle weiteren Exklusiv- und Share-Lockanforderungen auf dieses Objekt zurück.
  - Ein Share-Lock auf ein Objekt gestattet weitere Share-Lockzugriffe auf dieses Objekt, weist aber exklusive Lockanforderungen zurück.

- Bei Zurückweisung wird bis zur Lockfreigabe gewartet

Locks in Datenbanken

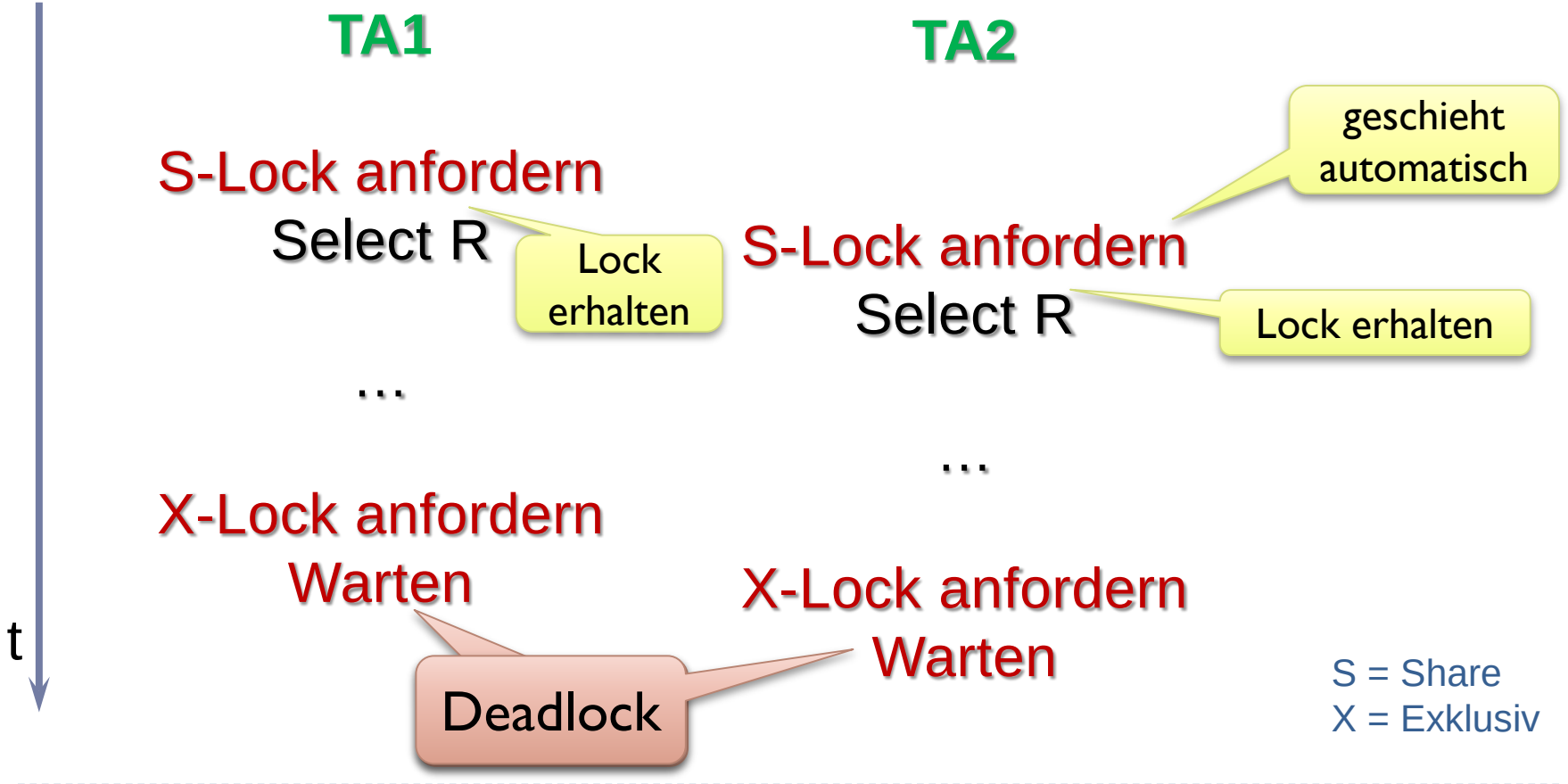
- Vor dem lesenden Zugriff auf eine Zeile:
  - Share-Lock für diese Zeile wird geholt
- Vor dem schreibenden Zugriff auf eine Zeile:
  - Exklusiv-Lock für diese Zeile wird geholt

- Gegebenenfalls wird so lange gewartet, bis der Lock verfügbar ist

Sperren in Datenbanken

- Vor Lesezugriff:
  - Share-Lock wird automatisch angefordert
  - Transaktion erhält Share-Lock, wenn es keine anderen Exklusiv-Lock-Anforderungen anderer Transaktionen gibt
- Vor Schreibzugriff:
  - Exklusiv-Lock wird automatisch angefordert
  - Transaktion erhält Exklusiv-Lock, wenn es keine anderen Share- oder Exklusiv-Lock-Anforderungen gibt
  - Hält Transaktion bereits den Share-Lock, so wird dieser in Exklusiv-Lock umgewandelt, sobald verfügbar
- Misslingt Lock-Anforderung, so wird bis Lockfreigabe gewartet
- Bei Transaktionsende
  - Freigabe aller gehaltenen Locks

Verlorengegangene Änderung (2)

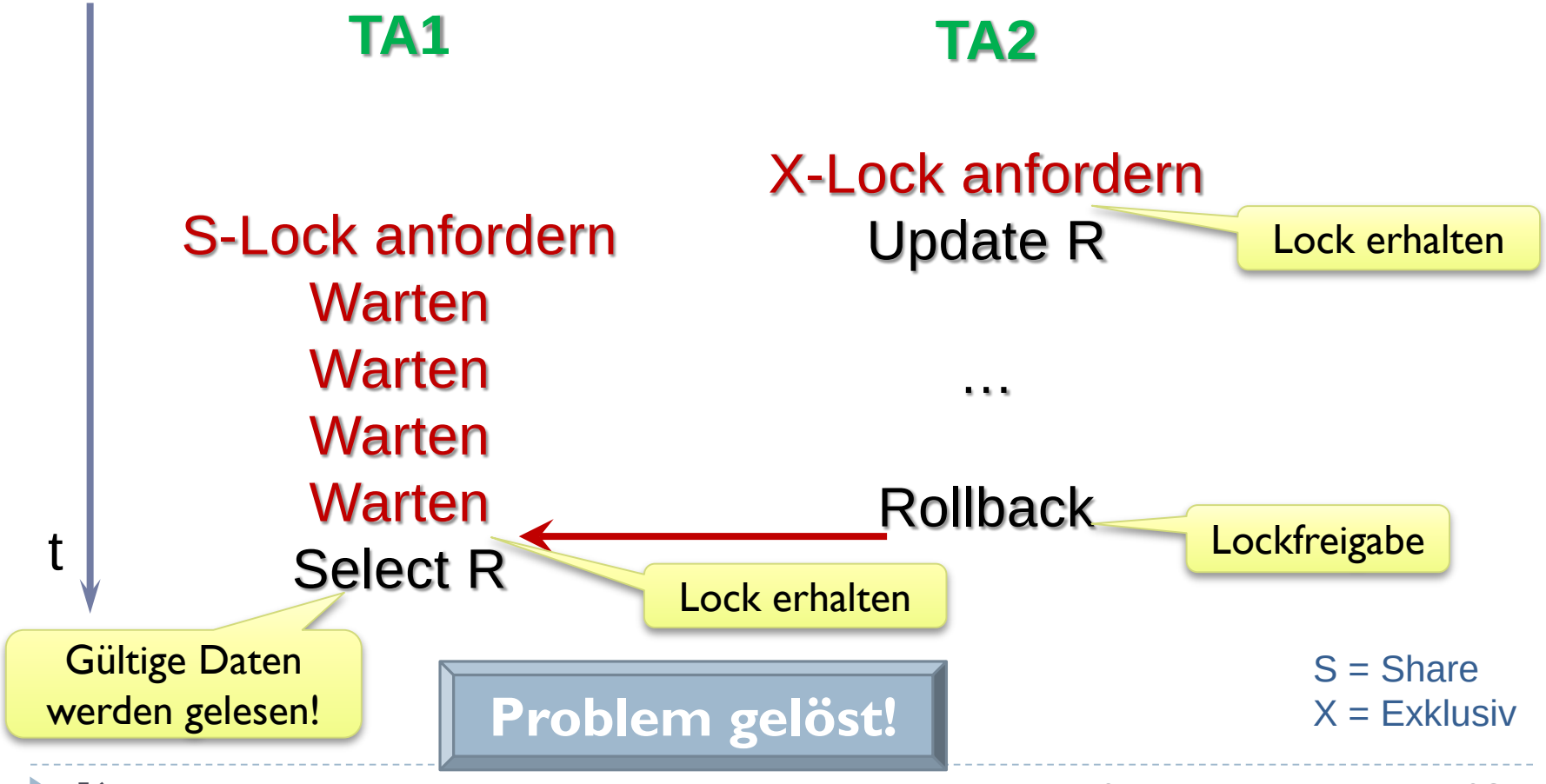


Deadlock

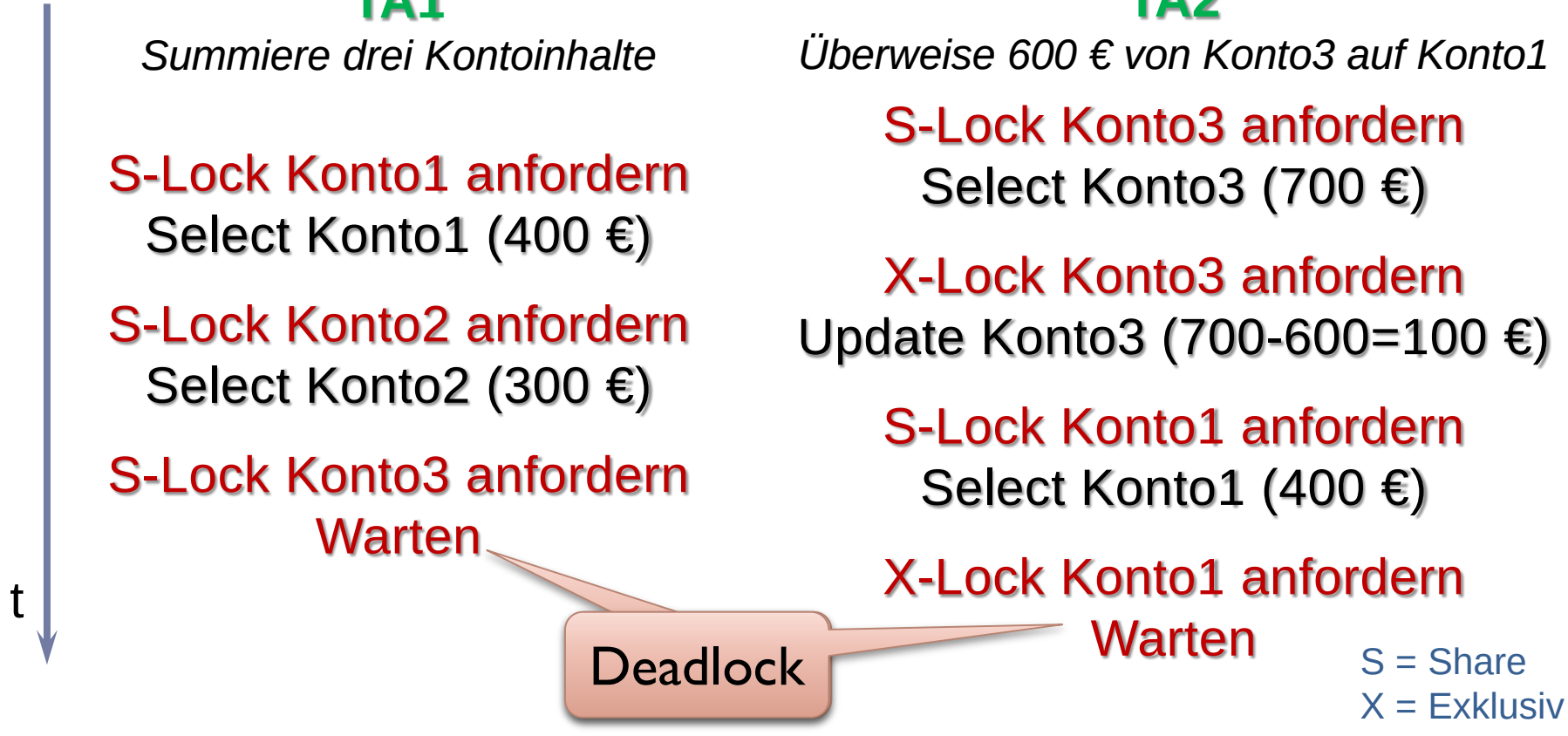
- **Definition (Deadlock)**
  - Eine Verklemmung, bei der mindestens zwei Transaktionen gegenseitig auf die Freigabe eines oder mehrerer Locks warten, heißt Deadlock.

Wir stellen Deadlocks zunächst zurück und betrachten erst die beiden anderen Concurrency-Probleme

Abhängigkeit von nicht abgeschl. TAs (2)



Inkonsistenz der Daten (2)

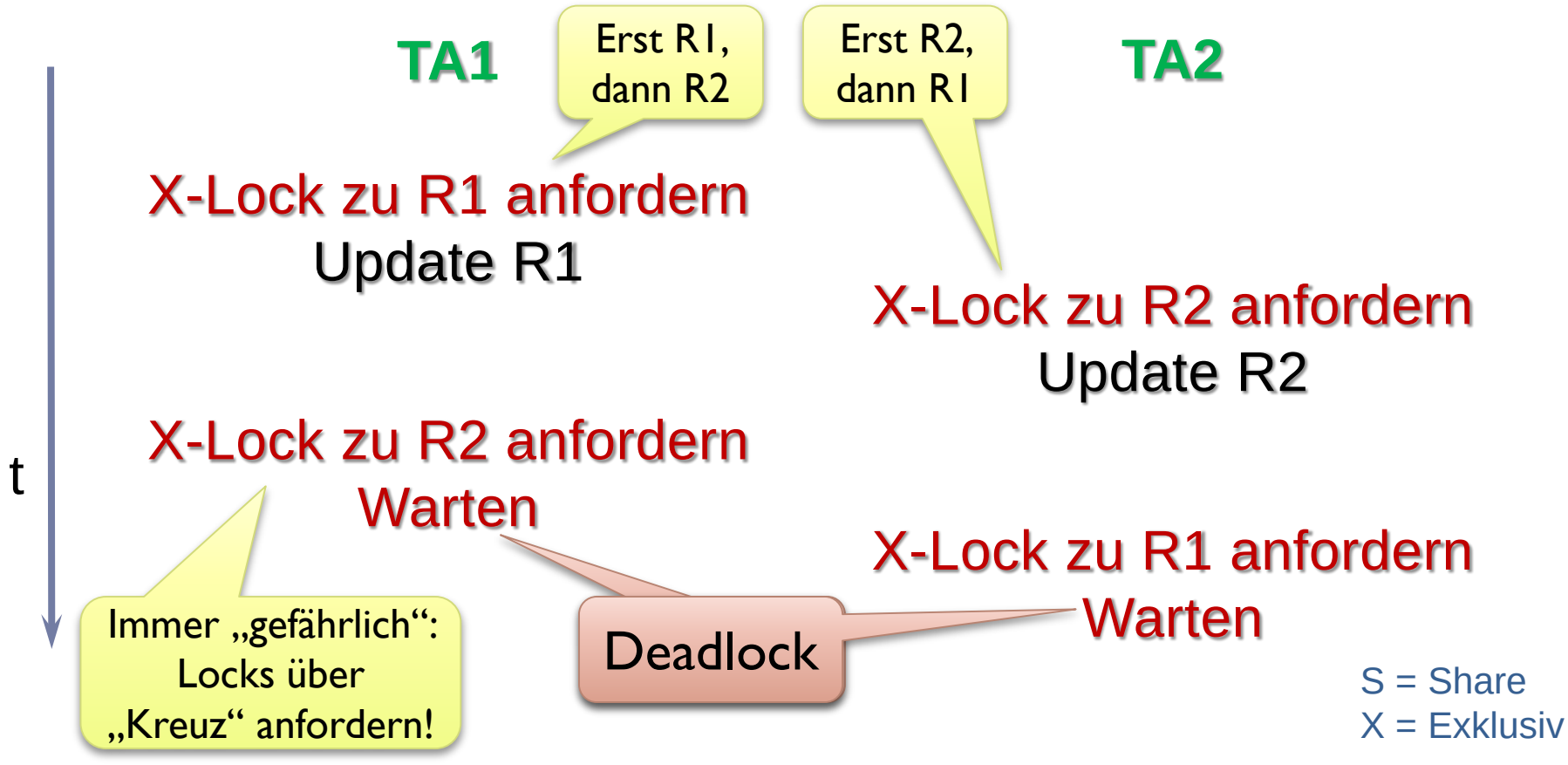


Ergebnis

- Einsatz von Exklusiv- und Share-Locks
  - löst ein Concurrency-Problem
  - führt zweimal zu Deadlock

- Folgerung:
  - Können wir das Deadlockproblem lösen, so sind auch die Concurrency-Probleme gelöst
  - Lesende behindern Schreibende und umgekehrt
  - Deadlocks treten auch zwischen Lesenden und Schreibenden auf

Deadlocks bei Schreibzugriffen



Deadlockvermeidung

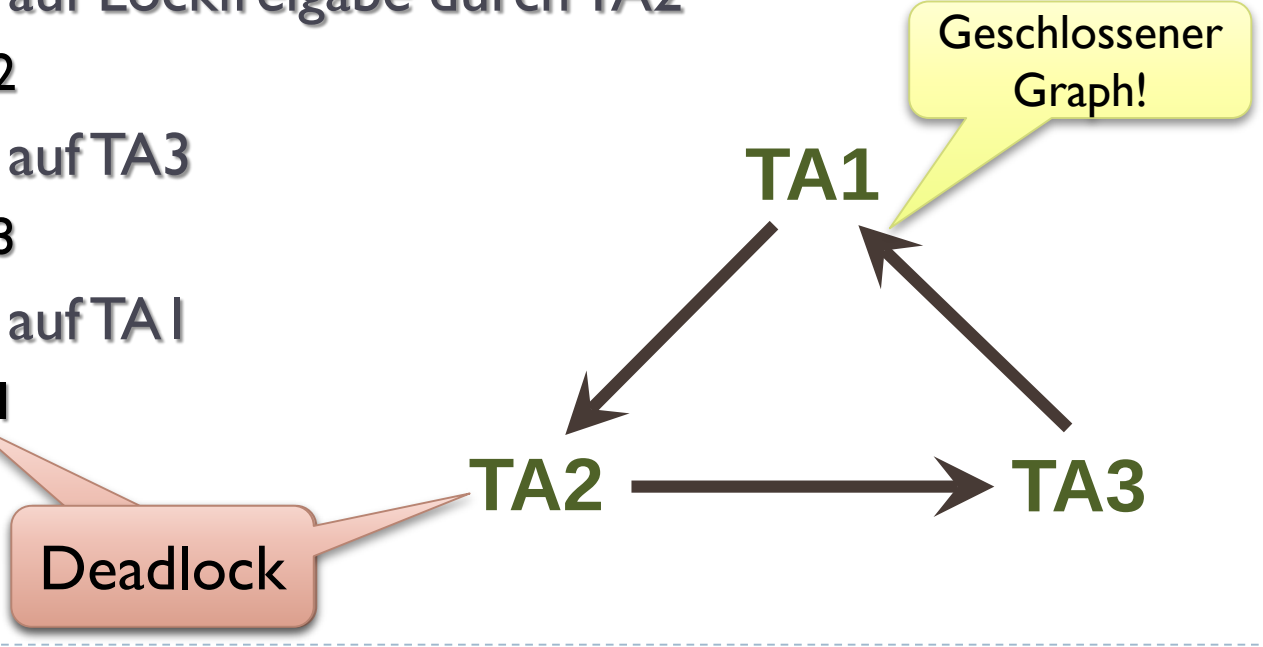
- Es entstehen keine Deadlocks, wenn
  - Locks in einer vorgegebenen Reihenfolge angefordert werden
- Beispiel:
  - Relationen werden alphabetisch geordnet
  - Tupel werden nach Primärschlüssel geordnet
  - Auf alle während einer Transaktion verwendeten Tupel wird in der Reihenfolge gemäß obiger Ordnung zugegriffen
- Aber:
  - Nicht immer ist zu Beginn einer Transaktion bekannt, auf welche Tupel zugegriffen wird
  - Eventuell muss Transaktion zurückgesetzt und neu gestartet werden
  - In der Praxis zu unflexibel

Deadlock-Erkennung (1)

- Einfache Strategie:
  - Beobachten von Wartezeiten
  - Bei langen Wartezeiten: Transaktion mit Fehler abbrechen
- Nachteile:
  - Eine lange wartende Transaktion muss nicht im Deadlock sein
  - Die optimale Wartezeit bis zum Abbruch ist nicht bekannt:
    - Ein zu kurzes Warten bricht auch „unschuldige“ Transaktionen ab
    - Ein zu langes Warten verlängert die Antwortzeit dieser Transaktionen

Deadlockerkennung (2)

- Sicheres Erkennen mittels Wartegraphen
- Beispiel (3 Transaktionen TA1, TA2, TA3):
  - TA1 wartet auf Lockfreigabe durch TA2
    - TA1 → TA2
  - TA2 wartet auf TA3
    - TA2 → TA3
  - TA3 wartet auf TA1
    - TA3 → TA1



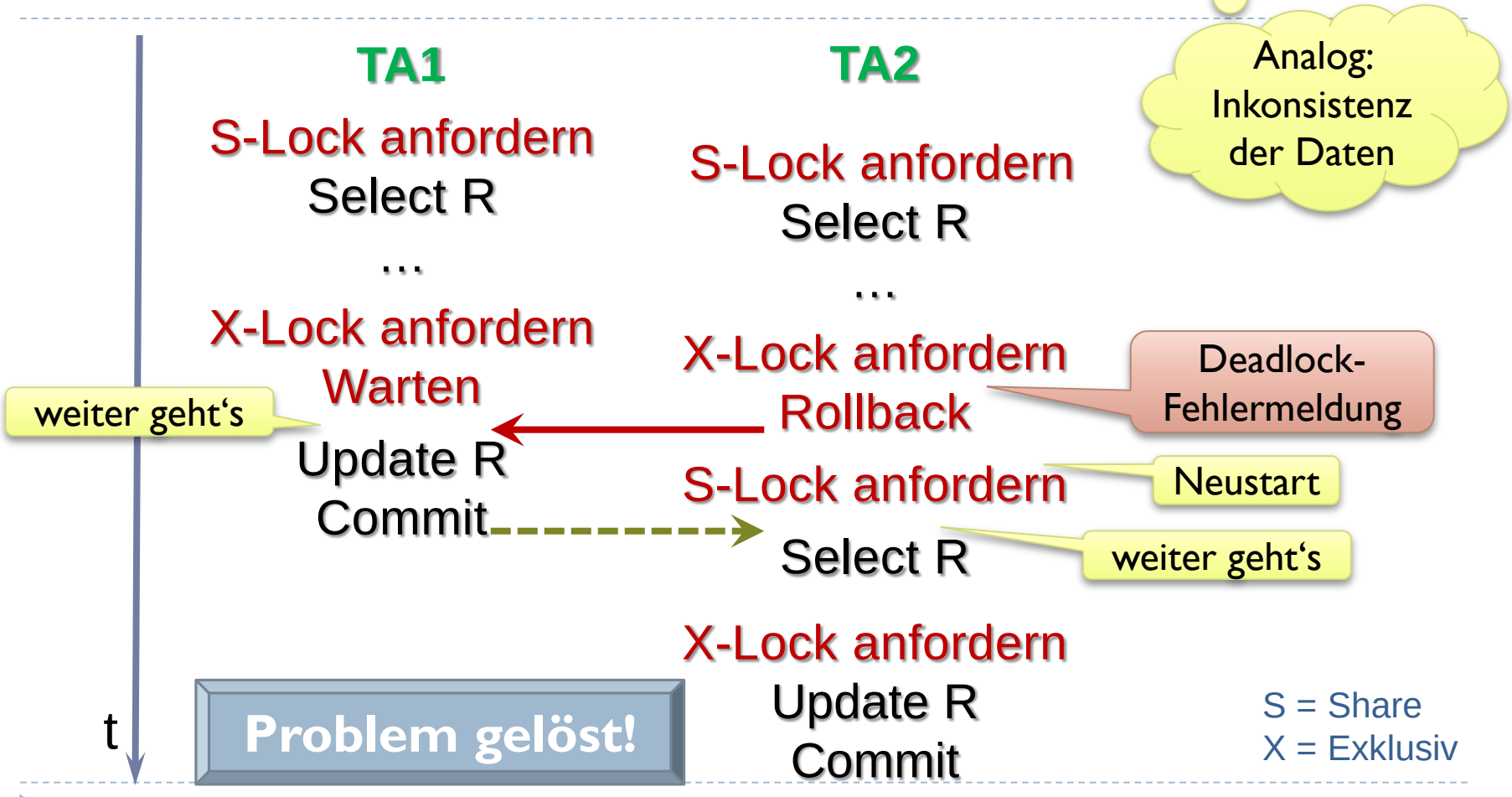
Deadlockerkennung in der Praxis

- Meist warten nur wenige Transaktionen auf Lockfreigabe
- Muss eine Transaktion warten,
  - wird ein gerichteter Graph hinzugefügt (Pfeil)
  - wird überprüft, ob dadurch ein geschlossener Zyklus entsteht
- Der Aufwand dieser Implementierung ist nicht allzu hoch
- Heute: Standard in modernen Datenbanken

Deadlockauflösung

- Datenbank:
  - Zuständig für Deadlockerkennung
  - DML-Befehl führt zu Fehlermeldung
    - Fehlervariable SQLSTATE: 40001 (SQL-Norm)
- Datenbankanwendung:
  - Reagiert auf Fehlermeldung
  - Fast immer die einzige sinnvolle Reaktion:
    - Zurücksetzen der Transaktion
    - Neustart der Transaktion
  - Warum Transaktion zurücksetzen?
    - Deadlock entstand, weil Transaktion Locks hält
    - Nur die Freigabe aller Locks löst Verklemmung sicher auf

Verlorengegangene Änderung (3)



Endergebnis

- Die Probleme der Concurrency lassen sich lösen
  - mit Exklusiv- und Share-Locks
  - und mit Lockerkennung
  - und mit Rücksetzen und Neustart einer Transaktion
- Merke:
  - Bei Deadlockfehlermeldung: Transaktion zurücksetzen
  - Gegebenenfalls Neustart dieser Transaktion

Ein Wiederholen eines Datenbankzugriffs im Deadlockfall führt sofort wieder zum Deadlock!

Concurrency in der SQL-Norm

- Concurrency Probleme nach SQL-Norm:
  - Dirty Write
    - (Verlorengegangene Änderung)
  - Dirty Read
    - (Abhängigkeit von nicht abgeschlossenen Transaktionen)
  - Non Repeatable Read
    - (Wiederholtes Lesen führt zu anderen Ergebnissen)
  - Phantom
    - (Eine bisher nicht vorhandene Zeile erscheint)

trotz Sperren möglich, da andere Transaktion neue Zeile einfügen kann

Isolationslevel gemäß SQL-Norm

Isolationslevel	Dirty Write erlaubt?	Dirty Read erlaubt?	Non Repeatable Read erlaubt?	Phantom erlaubt?
Read Uncommitted	Nein	Ja	Ja	Ja
Read Committed	Nein	Nein	Ja	Ja
Repeatable Read	Nein	Nein	Nein	Ja
Serializable	Nein	Nein	Nein	Nein

Isolationslevel setzen

- Befehl SET TRANSACTION

```
SET TRANSACTION ISOLATION LEVEL Level
```

- Level:
  - Read Uncommitted
  - Read Committed
  - Repeatable Read
  - Serializable
- Meist erster Befehl einer Transaktion

Mögliche Realisierung der Level

z.B. S-Locks auf Tabellen; Insert benötigt X-Lock

Isolationslevel	Exklusiv-Locks zum Schreiben	Schreiben auf Kopie	Share-Locks zum Lesen	Range-Locks
Read Uncommitted	Ja	Nein	Nein	Nein
Read Committed	Ja	Ja	Nein	Nein
Repeatable Read	Ja	Ja	Ja	Nein
Serializable	Ja	Ja	Ja	Ja

Concurrency und Oracle

Set Transaction Isolation Level Serializable;

Erster Befehl einer Transaktion!

Set Transaction Isolation Level Read Committed;

Standard

Set Session Isolation Level Serializable;

Ab jetzt für gesamte Session

Set Session Isolation Level Read Committed;

Read Uncommitted und Repeatable Read nicht implementiert

- Oracle verwendet keine Lesesperren!
- Im Level Serializable:
  - Optimistische Lesestrategie!
  - Gegebenenfalls Serialisierungsfehler ORA-08177
- Ausnahme:
  - Share-Lock bei `SELECT … FOR UPDATE`

Concurrency und SQL Server

- Alle vier Isolationslevel implementiert
- Intern:
  - Exklusiv- und Share-Locks
  - Phantomvermeidung durch Key-Range-Locks
- Notwendig:
  - Transaktion explizit starten: `BEGIN TRANSACTION;`
- Überprüfen des Isolationslevels:

  `dbcc useroptions;`

Sonst keine Transaktion und keine Concurrency

Concurrency und MySQL

SET TRANSACTION ISOLATION LEVEL Level;

SET GLOBAL TRANSACTION ISOLATION LEVEL Level;

- Standard: Repeatable Read
- Voraussetzung:
  - Engine INNODB
  - Transaktionsmodus
    - Entweder: SET AUTOCOMMIT=0;
    - Oder: START TRANSACTION;
- Intern:
  - Exklusiv- und Share-Locks
  - Phantomvermeidung durch Next-Key-Locking-Algorithmus

Ab der nächsten Transaktion für gesamte Session

Ab jetzt: wie in Oracle

Wie in SQL Server expliziter Start

Zusammenfassung

- Ohne Transaktionen:
  - Keine sichere Recovery
  - Keine Concurrency
- Recovery:
  - Datenbank-Pufferung, Redo-Logs, Undo-Log, Checkpoints
- Concurrency:
  - Sperren (Exklusiv, Share), Deadlock, Deadlockerkennung, Rücksetzen der Transaktion und Neustart, Isolationslevel

Datenbanken und SQL

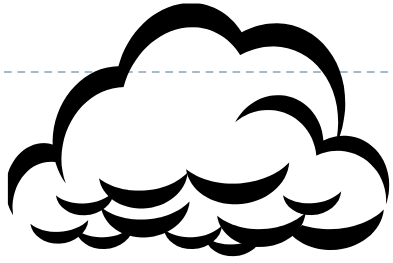
Kapitel 9

Moderne Datenbankkonzepte

Moderne Datenbankkonzepte

- Verteilte Datenbanken
  - Vorteile der verteilten Datenbanken
  - Die 12 Regeln von Date zu verteilten Datenbanken
  - Das CAP-Theorem
  - BASE versus ACID
  - Überblick über moderne verteilte Datenbanken
  - Zwei-Phasen-Commit
- Objektorientierte Datenbanken
  - Definition
  - Objektrelationale Datenbanken
  - Objektrelationale Erweiterungen in Oracle
  - Eingebettete Relationen in Oracle

Übersicht



- Neue wichtige Begriffe:
  - Cloud-Computing
    - Umfassender Begriff dafür, dass Daten im Netz gehalten werden
    - Der exakte Speicherort ist in der Regel nicht festgelegt und nicht bekannt
    - Beispiele:
      - Drop Box, OneDrive, IMAP (Email) usw.
      - Verteilte Datenbankserver von Amazon, Google, Facebook usw.
  - NoSQL
    - Begriff steht für: Not Only SQL
    - Erweiterung der Sprache SQL für neue Datenbankkonzepte
    - SQL ist sehr weit verbreitet, optimiert für relationale Datenbanken
    - Erweiterungen für nicht relationale Datenbanken sind erforderlich

Vorteile verteilter Datenbanken

- Schnelle Verfügbarkeit
  - da Daten parallel zugreifbar
  - da Daten eventuell mehrfach gehalten werden
  - da deshalb Daten eventuell lokal direkt verfügbar
- Ausfallsicherheit
  - Wenn ein Knoten ausfällt, sind die anderen noch verfügbar
- Im Extremfall
  - gibt es tausende von Knoten
  - Daten werden redundant gehalten
- Aber: Synchronisierung geänderter Daten ist komplex

Fundamentales Prinzip

- **Fundamentales Prinzip verteilter Datenbanken:**

  Ein verteiltes System sollte sich dem Anwender gegenüber genauso wie ein zentrales verhalten.

- **Dies heißt:**
  - Der Anwender bemerkt bei seinen Zugriffen keinen Unterschied, ob er zentral auf eine Datenbank oder auf viele verteilte Daten zugreift
  - Dies gilt auch für den Anwendungsprogrammierer!
- **Die folgenden 12 Regeln von Date basieren auf diesem Prinzip**

Die 12 Regeln von Daten

- J.F. Date stellte 12 Regeln zu verteilten Datenbanken auf:

1.	Lokale Eigenständigkeit jedes Rechners
2.	Keine zentrale Verwaltungsinstanz
3.	Ständige Verfügbarkeit
4.	Lokale Unabhängigkeit
5.	Unabhängigkeit gegenüber Fragmentierung
6.	Unabhängigkeit gegenüber Datenreplikation
7.	Optimierte verteilte Zugriffe
8.	Verteilte Transaktionsverwaltung
9.	Unabhängigkeit von der Hardware
10.	Unabhängigkeit von Betriebssystemen
11.	Unabhängigkeit vom Netz
12.	Unabhängigkeit von den Datenverwaltungssystemen

Regeln 1 und 2

- Lokale Eigenständigkeit jedes einzelnen Rechners
  - Jeder einzelne Rechner arbeitet möglichst autonom
  - Dies garantiert eine hohe Ausfallsicherheit
  - Dies erfordert einen hohen internen Kommunikationsaufwand
- Keine zentrale Instanz, die das System verwaltet
  - Fast eine Folgerung aus Regel 1: Wenn jede Instanz eigenständig ist, benötigen wir keine zentrale Instanz
  - Damit müssen sich alle einzelnen Rechner gegenseitig verwalten
  - Dies führt zu einem hohen internen Kommunikationsaufwand

Regeln 3, 4 und 5

- Ständige Verfügbarkeit
  - Das gesamte System sollte nie abgeschaltet werden
- Lokale Unabhängigkeit
  - Jeder Zugriff ist unabhängig davon, wo sich die gewünschten Daten derzeit befinden, ob lokal oder entfernt
- Unabhängigkeit gegenüber Fragmentierung
  - Fragmentierung: Relationen werden auf mehrere Rechner verteilt (auch sharding genannt)
  - Der Anwender greift unabhängig von der Fragmentierung zu
  - Die Fragmentierung kann dynamisch sein: Daten werden meist dort gespeichert, wo sie häufig zugegriffen werden (Regionalisierung)

Regeln 6, 7 und 8

- Unabhängigkeit gegenüber Datenreplikation
  - Die Zugriffe ändern sich nicht, falls Replikate existieren
  - Die Verwaltung der Replikate und die Konsistenz der Daten übernimmt das verteilte System
- Optimierung verteilter Zugriffe
  - Das Suchen der Daten im verteilten System und das Lesen und Schreiben werden intern optimiert
- Verteilte Transaktionsverwaltung
  - Transaktionen werden als atomare Einheiten voll unterstützt
  - Recovery und Concurrency werden voll unterstützt

Regeln 9 bis 12

- Unabhängigkeit von der verwendeten Hardware
  - PCs, Großrechner, unterschiedliche Netze werden unterstützt
- Unabhängigkeit von den verwendeten Betriebssystemen
  - Windows, MacOS, Unix, usw.
- Unabhängigkeit vom verwendeten Netzwerk
  - Unterstützung aller wichtigen Protokolle, z.B. TCP/IP
- Unabhängigkeit vom verwendeten DBMS
  - Verwendung gemeinsamer Zugriffssprachen wie JSON, PDO
  - Unterstützung von SQL und NoSQL

Zusammenfassung zu den 12 Regeln

- Regel 9 bis 11 (unabhängig von HW, OS, Netz):
  - Heute weitgehend erfüllt
- Regel 1 bis 6 (eigenständig, verfügbar, fragmentiert, repliziert):
  - Ehrgeizige Ziele, die teilweise schon erfüllt werden
- Regel 7, 8 und 12 (optimiert, Transaktion, DBMS-unabhängig):
  - Nicht widerspruchsfrei
  - Regel 7 und 12 erwarten einheitliche Schnittstellen und einen Transaktionsbetrieb
  - Regel 8 fordert verteilte Transaktionen, mit SQL allein kaum zu erfüllen

Das CAP-Theorem

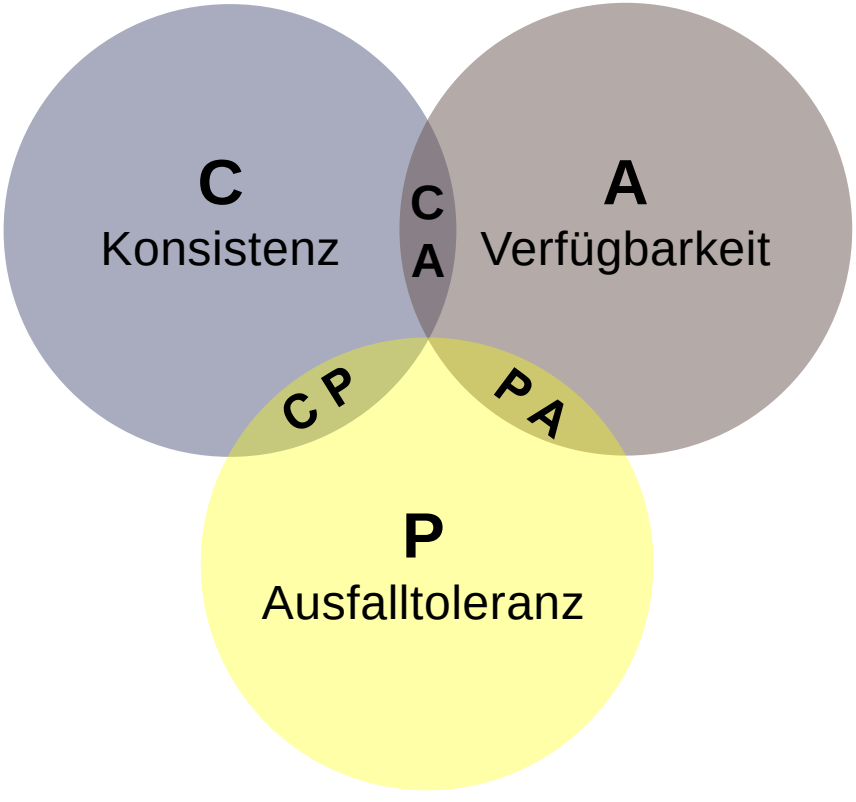
- Konsistenz (**C**onsistence),
  Verfügbarkeit (**A**vailability) und
  Ausfalltoleranz (Tolerance of Network **P**artitions)
  können in verteilten Datenbanken nicht gleichzeitig erfüllt werden.

  - Hier wird sehr gezielt auf das Problem des Widerspruchs der 12 Regeln von Date eingegangen.
  - Nur jeweils 2 der obigen Eigenschaften C, A und P können gleichzeitig vollständig erfüllt werden.

C – A – P

- **C**onsistence (Konsistenz)
  - → Transaktionsbetrieb, ACID, Regel 8
  - Erfordert die Atomarität, Konsistenz bei redundanten Daten
- **A**vailability (Verfügbarkeit)
  - → Regel 3 und 7
  - Ein Teilausfall des Systems sollte nicht zum Gesamtausfall führen
- **P**artition-Toleranz (Ausfalltoleranz)
  - → in mehreren Regeln enthalten
  - Ein (vorübergehender) Verlust von Daten muss toleriert werden
  - Ein verspätetes Zustellen ist zu tolerieren

CAP-Eigenschaften



- CA-Systeme:
  - Klassischer Bereich der relationalen Datenbanken
- CP-Systeme:
  - Daten in einzelnen Knoten können ausfallen. Eventuell Neustart des Systems
- PA-Systeme:
  - Hochverfügbares System auf Kosten der sofortigen Konsistenz

Das Konsistenzmodell BASE

- **B**asically **A**vailable
  - Die Verfügbarkeit ist wichtiger als die Konsistenz
- **S**oft State
  - Konsistenz wird nach Transaktionsende fließend (soft) erreicht
- **E**ventual Consistency
  - Letztendlich wird die Konsistenz erreicht und garantiert

- Im klassischen Modell steht konträr dazu: **ACID**

Überblick über moderne DB-Systeme (1)

- Key/Value-Datenbanken
  - Daten werden mit dem Schlüssel abgelegt
  - z.B. Amazon System Dynamo, Riak
  - Leicht zu skalieren, in Cloud-Systemen gerne angewendet
- Dokumentenbasierte Datenbanken
  - Als Dokumente abgelegt ohne fest vorgegebene Strukturen
  - Basiert auf Lotus Notes
  - z.B. MongoDB, CouchDB

Überblick über moderne DB-Systeme (2)

- Spaltenorientierte Datenbanken
  - Daten werden spaltenweise verwaltet und gespeichert
  - Meist aber Mischformen mit Key/Value,
  - Anwendung in Big Table Konzept von Google
  - z.B. HBase von Microsoft, Cassandra von Facebook
- Graphen-Datenbanken
  - Grundlage ist die Graphentheorie, basiert auf Graphen
  - Sehr gut für rekursive Suche geeignet, für Navis, für Geodaten-banken, für soziale Netzwerke
  - z.B. GraphDB von Sones, Open Source Neo4J

Zuordnung moderner DB-Systeme

	CA-System	CP-System	PA-System
Relationale Datenbanken	Oracle SQL Server MySQL, …		
Key-Value Datenbanken		BerkeleyDB	Dynamo Riak
Dokumentenbasierte Datenbanken		MongoDB	CouchDB
Spaltenorientierte Datenbanken		Big Table HBase	Cassandra

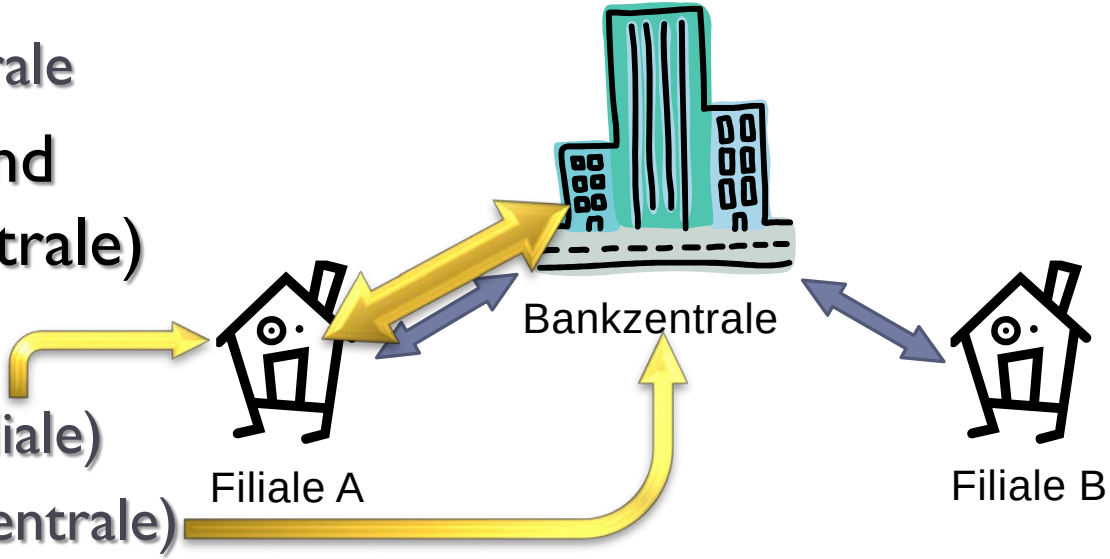
Zwei-Phasen-Commit

- **Gegeben:**
  - Daten sind auf mindestens 2 Datenbanken verteilt
  - CA-System ist erforderlich
  - z.B. Überweisung von einer Bank auf eine andere

- **Idee:**
  - Jede Datenbank führt lokal ein eigenes Transaktionsprotokoll
    - → Phase 1
  - Übergreifend gibt es ein globales Transaktionsprotokoll
    - → Phase 2

Die Idee des Zwei-Phasen-Commits

- Überweisung
  - von Filiale A zur Zentrale
- Zwei Datenbanken sind involviert (Filiale, Zentrale)
- Also:
  - Lokale Transaktion (Filiale)
  - Lokale Transaktion (Zentrale)
  - Koordination (Globale Transaktion)



Das Zwei-Phasen-Commit

1. Jede Datenbank arbeitet getrennt im Transaktionsbetrieb
   - Jede Datenbank macht eigene Recovery (z.B. Logs, Checkpoints)
   - Jede Datenbank beendet mit einem „lokalen“ Commit
2. Eine Datenbank ist zusätzlich der Koordinator
   - Der Koordinator startet eine „globale“ Transaktion
   - Diese überwacht die „lokalen“ Transaktionen
   - Bei Erfolg aller lokalen Transaktionen wird dies an alle zurückgemeldet
   - Die „lokalen“ Commits werden dann in „globale“ umgewandelt

Algorithmus des 2-Phasen-Commits

Lokales Abarbeiten einer Transaktion	Jede betroffene Datenbank startet eine lokale Transaktion: Lokale Änderungen werden in der lokalen Logdatei protokolliert. (Beginn der **Phase 1**)
Melden des Transaktionsendes	Am Ende einer lokalen Transaktion erfolgt eine entsprechende Meldung (Commit bzw. Rollback) an den Koordinator (Ende der **Phase 1**).
Globales Transaktionsende	Der Koordinator sammelt alle lokalen Meldungen. Liegen nur erfolgreiche Rückmeldungen vor, so wird ein globales Commit, ansonsten ein Rollback eingetragen (**Phase 2**).
Endgültiges lokales Transaktionsende	Das Ergebnis der globalen Transaktion wird an alle lokalen Rechner geschickt. Jeder lokale Rechner übernimmt das globale Ergebnis (Commit oder Rollback) als endgültiges. Erst jetzt ist die Transaktion abgeschlossen (Ende der **Phase 2**).

Zusammenfassung: 2-Phasen-Commit

- Der transaktionsübergreifende Commit wird sichergestellt
- Die Konsistenz wird datenbankübergreifend garantiert
- Der 2-Phasen-Commit wird kommerziell angewendet
- Das Protokoll ist extrem aufwendig
  - da der Koordinator ständig die lokalen Transaktionen überwachen muss
  - da auch Netzausfälle mit einkalkuliert werden müssen
  - da Netzverzögerungen nicht sofort zum Abbruch der globalen Transaktionen führen sollen

Objektorientierte Datenbanken

- Objektorientierte Datenbanken entstanden
  - in Folge zu den objektorientierten Programmiersprachen
  - ab 1990
  - als rein objektorientierte Datenbanken
    - Diese sind heute praktisch ohne Bedeutung
  - als Erweiterung der relationalen Datenbanken
    - Diese wurden in die SQL 3 und SQL 2003 Norm aufgenommen
- Die wichtigsten objektorientierten Datenbank sind:
  - Oracle
  - PostgreSQL

Definition (objektorientierte Datenbank)

- Eine Datenbank heißt objektorientiert, wenn sie grundlegende objektorientierte Konzepte wie Objekte, Klassen, Methoden, Kapselung und Vererbung enthält und verwendet.

- Diese Definition ist sehr allgemein gehalten und bezieht damit die objektrelationalen Datenbanken voll mit ein.

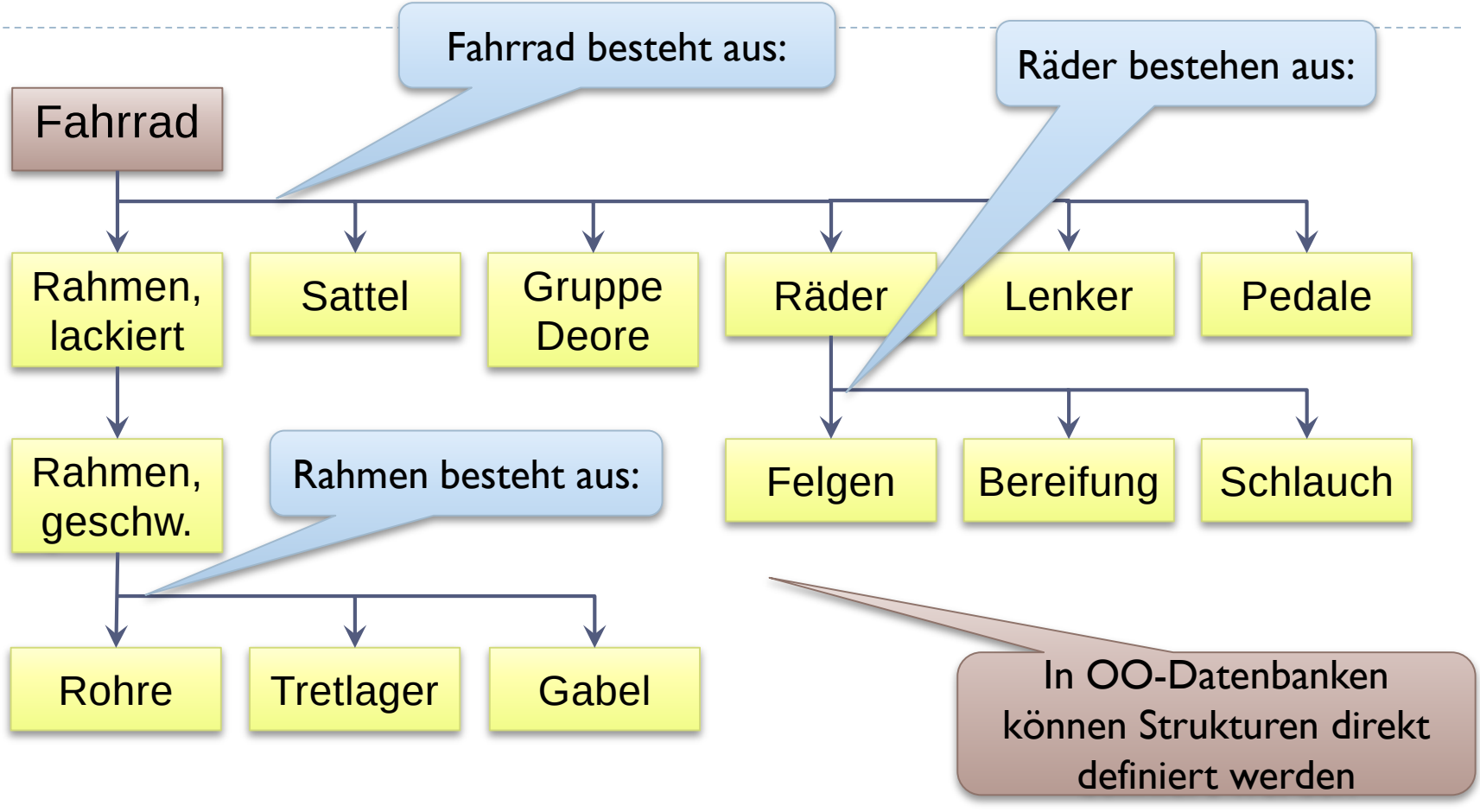
Objektorientierte Konzepte (Überblick)

- **Objekt:** Einzelne Gegenstände und Entitäten, die sehr komplex sein können, z.B. ein bestimmtes Flugzeug
- **Klasse:** Ein Objekttyp, z.B. ein Airbus A300
- **Eigenschaft einer Klasse:** Eigenschaft eines Objekttyps, z.B. der Preis, das Gewicht
- **Methode einer Klasse:** Routine, die auf Objekte angewendet werden, z.B. StarteFlugzeuge, LandeFlugzeug
- **Kapselung:** Eigenschaften und Methoden sind nur für bestimmte Anwendungen erlaubt.
- **Vererbung:** Spezialisierung von Klassen. Aus einem Flugzeug wird beispielsweise ein Segel- oder Motorflugzeug abgeleitet

Nachteile relationaler Datenbanken

Nachteile	Auswirkung
Flache Struktur	Komplexe Objekte werden „flachgeklopft“; ihr Aufbau ist aus dem Design nicht mehr direkt ersichtlich
Keine Rekursion	Der Aufbau komplexer Objekte kann nur schwer nachvollzogen werden (Stücklistenproblem)
Viele Relationen	Häufige Joins auf zusammengehörige Relationen verlängern die Laufzeit

Vorteil OO-Datenbanken: Strukturen



Objektrelationale Datenbanken

- Echte Erweiterung der relationalen Datenbanken
- Erweiterung um objektorientierte Konzepte
- Idee
  - Relationale Datenbanken sind weit verbreitet
  - Diese können weiter verwendet werden
  - Zusätzlich wird eine Erweiterung angeboten, die Schritt für Schritt eingesetzt werden kann
- Verbreitung:
  - Oracle, PostgreSQL

Definition (NF^2-Normalform)

- **Eine Relation ist in der NF^2-Normalform, falls sie bis auf die Atomarität alle Bedingungen an eine Relation erfüllt.**

- Idee:
  - Ein Attribut kann aus komplexen Strukturen bestehen, z.B.
    - Aufzählung (Liste)
    - Struktur (Objekt)
    - Relation (eingebettete Relation)
  - Damit lassen sich sehr komplexe Strukturen in nicht normalisierten Relationen nachbilden

Objektrelational in Oracle

- Objektorientierte Datenbanken benötigen Programmier-sprache
  - In Oracle: PL/SQL
- Oracle unterstützt:
  - Variable Felder
  - Objekte
  - Objekt-Sichten
  - Eingebettete Relationen
  - Objekt-Methoden (Prozeduren / Funktionen)

Variable Felder in Oracle (1)

CREATE TYPE Typname AS
{ VARRAY | VARRYING ARRAY } (Anzahl) OF Datentyp

- Ein Feld wird erzeugt
- Die maximale Größe des Feldes ist anzugeben (Anzahl)
- Beispiel:
  - Feld TProdukt aus max. 50 Zeichenketten mit 30 Zeichen:

```
CREATE TYPE TProdukt AS VARRAY ( 50 ) OF CHAR ( 30 ) ;
```

Nicht atomare Relation VerkaeuferProdukt

VerkNr	VerkName	PLZ	VerkAdresse	Produktname	Umsatz
V1	Meier	80331	München	Waschmaschine, Herd, Kühlschrank	17000
V2	Schneider	70173	Stuttgart	Herd, Kühlschrank	7000
V3	Müller	50667	Köln	Staubsauger	1000

- **Realisierung mit Oracle:**

```
CREATE TABLE VerkaeuferProdukt
(     VerkNr       CHAR(4)        PRIMARY KEY ,
      VerkName     CHAR(20)       NOT NULL ,
      PLZ          CHAR(5) ,
      Adresse      CHAR(60) ,
      Produktname  TProdukt ,
      Umsatz       NUMERIC (10, 2)        ) ;
```

Einfügen der ersten Zeile:

```
INSERT INTO VerkaeuferProdukt VALUES
(   'V1', 'Meier', '80331', 'München',
    TProdukt( 'Waschmaschine', 'Herd', 'Kühlschrank' ), 17000 );
```

- Das Feld kann nicht direkt eingegeben werden
- Es muss mit dem Objekttyp spezifiziert werden
- Die weiteren Zeilen werden entsprechend eingefügt
- Aufruf mittels:

```
SELECT * FROM VerkaeuferProdukt;
```

Objekte in Oracle

```
CREATE [ OR REPLACE ] TYPE Typname AS OBJECT
(  Spalte   Datentyp  [ , … ] ,
   [ { MEMBER { Prozedurname | Funktionsname } }  [ , … ] ]  )
```

- Es werden zuerst Attribute (Spalten) definiert
- Dann folgt die Deklaration von Memberfunktionen und –prozeduren (Methoden)

Objekttyp: TAdresse

- **Beispiel:**
  - Als Objekt wird eine Adresse mit PLZ, Ort und Straße definiert. Zusätzlich enthält das Objekt eine Methode Anschrift

```
CREATE OR REPLACE TYPE TAdresse AS OBJECT
(   Strasse      VARCHAR2 ( 30 ) ,
    PLZ          VARCHAR2 ( 5 ) ,
    Ort          VARCHAR2 ( 20 ) ,
    MEMBER FUNCTION Anschrift RETURN VARCHAR2    ) ;
/
```

3 Attribute: Strasse, PLZ, Ort

1 Methode (Funktion): Anschrift

In Oracle wichtig, wenn weitere Zeilen folgen

Anwendung von TAdresse

- In Relation Lieferant (analog in Relation Kunde, Personal):

```
CREATE TABLE LieferantNeu
(   Nr          INTEGER             PRIMARY KEY ,
    Name        VARCHAR ( 30 )      NOT NULL ,
    Adresse     TAdresse ,
    Sperre      CHAR                ) ;
```

Neuer Datentyp

- Ausgabe aller Mitarbeiter mit Wohnort:

```
SELECT  Name, L.Adresse.Ort
FROM    LieferantNeu L;
```

Objekt Adresse mit Attribut Ort

Oracle benötigt einen Aliasnamen!

Einfügen in LieferantNeu

- **Existiert die Relation Lieferant, so können alle Daten direkt übernommen werden:**

```
INSERT INTO LieferantNeu
SELECT Nr, Name, TAdresse( Strasse, PLZ, Substr(Ort, 1, 20) ),
       Sperre
FROM   Lieferant ;
```

… mit 3 Attributen

Objekt TAdresse …

Achtung! Ort ist CHAR(20), in Lieferant jedoch CHAR(25)!

- **Die Funktion Substr erzeugt ein passendes Attribut!**

Objektsichten

- In rein relationalen Datenbanken können alternativ auch Objektsichten verwendet werden.
- Beispiel
  - Relation SLieferant als Objektsicht

```
CREATE VIEW SLieferant ( Nr, Name, Adresse, Sperre )  AS
    SELECT   Nr, Name, TAdresse( Strasse, PLZ, Ort ), Sperre
    FROM     Lieferant ;
```

Sicht verwendet Objekttyp!

- Die Zugriffe sind analog wie in LieferantNeu!
- Die Sicht ist änderbar!

Probleme mit Objekten

- **Ausgabe in SQL Developer:**

  SELECT * FROM LieferantNeu;

NR	NAME	ADRESSE	SPERRE
1	Firma Gerti Schmidtner	[BIKEOO.TADRESSE]	0
2	Rauch GmbH	[BIKEOO.TADRESSE]	0
3	Shimano GmbH	[BIKEOO.TADRESSE]	0
4	Suntour LTD	[BIKEOO.TADRESSE]	0
5	MSM GmbH	[BIKEOO.TADRESSE]	0

- **Alternative 1:**

```
SELECT   Nr, Name, L.Adresse.Strasse, L.Adresse.PLZ,
         L.Adresse.Ort, Sperre
FROM     LieferantNeu L ;
```

Sehr umständlich

Objektmethode Anschrift

- **Alternative 2:**
  - Objektmethode Anschrift verwenden
- **Definition der Methode in CREATE TYPE BODY:**

```
CREATE OR REPLACE TYPE BODY TAdresse AS
  MEMBER FUNCTION Anschrift RETURN VARCHAR2 IS
  Ausgabe  VARCHAR2(60);
  BEGIN
     Ausgabe := TRIM(Strasse) || ', ' || TRIM(PLZ) || '  ' || TRIM(Ort);
     RETURN Ausgabe;
  END;
END;
/
```

Ergebnistyp ist VARCHAR2

Lokale Variable Ausgabe

Konkatenieren von Strasse, PLZ und Ort

Ergebnisrückgabe

Ausgabe von LieferantNeu

- **Elegante Ausgabe mittels der Methode Anschrift()**

Arbeitsblatt | Query Builder

```
SELECT Nr, Name, L.Adresse.Anschrift(), Sperre
FROM LieferantNeu L ;
```

Skriptausgabe | Abfrageergebnis

SQL | Alle Zeilen abgerufen:5 in 0 Sekunden

	NR	NAME	L.ADRESSE.ANSCHRIFT()	SPERRE
1	1	Firma Gerti Schmidtner ...	Dr. Gesslerstr. 59, 93051 Regensburg	0
2	2	Rauch GmbH ...	Burgallee 23, 90403 Nürnberg	0
3	3	Shimano GmbH ...	Rosengasse 122, 51143 Köln	0
4	4	Suntour LTD ...	Meltonstreet 65, London	0
5	5	MSM GmbH ...	St-Rotteneckstr. 13, 93047 Regensburg	0

Eingebettete Relationen

- Bisher:
  - Ein Attribut kann eine Liste oder auch ein Objekt enthalten
  - Ein Attribut kann eine Liste aus Objekten enthalten!
  - Aber: Die maximale Größe der Liste ist beschränkt
- Neu:
  - Ein Attribut kann eine Relation enthalten
  - Eine Beschränkung der Größe der Relation existiert nicht
  - Diese Relation heißt:

Eingebettete Relation

Eingebettete Relation am Beispiel

- Die Relation Auftrag enthält Auftragspositionen
- Die Positionen werden in die Relation Auftrag eingebettet
- Definition eines geeigneten Objekts TEinzelposten:

```
CREATE OR REPLACE TYPE TEinzelposten AS OBJECT
(   Artnr          INTEGER,
    Anzahl         INTEGER,
    Gesamtpreis    NUMERIC(10,2)     ) ;
```

Objekt TEinzelposten

- Definition der eingebetteten Relation:

ER_Einzelposten ist Relation vom Typ TEinzelposten

```
CREATE  TYPE  ER_Einzelposten  AS TABLE OF  TEinzelposten ;
```

Erzeugen der Relation AuftragNeu

```
CREATE TABLE  AuftragNeu
(   AuftrNr        INTEGER   PRIMARY KEY,
    Datum          DATE,
    Einzelposten   ER_Einzelposten,
    Kundnr         INTEGER   REFERENCES Kunde,
    Persnr         INTEGER   REFERENCES Personal
                             ON DELETE SET NULL
)   NESTED TABLE Einzelposten
                   STORE AS  ER_Einzelposten_TABLE ;
```

Eingebettete Relation ER_Einzelposten

Angabe der Tabelle, in der die Daten der eingebetteten Relation gespeichert werden

Einfügen von Auftrag 2 in AuftragNeu

- Auftrag 2 besitzt zwei Auftragspositionen
- Wir müssen wieder Cast-Operatoren verwenden
- ER_Einzelposten besteht aus Objekten (TEinzelposten):

```
INSERT INTO AuftragNeu VALUES
(   2,  DATE '2013-01-06',
    ER_Einzelposten (   TEinzelposten (100002, 3, 1950),
                        TEinzelposten (200001, 1, 400)      ),
    3,  5
) ;
```

Auftrnr, Datum

ER_Einzelposten, bestehend aus TEinzelposten

Artnr, Anzahl, Gesamtpreis als Objekt TEinzelposten

Kundnr, Persnr

Vorteile objektrelationaler Datenbanken

- Die Relation AuftragNeu muss nicht künstlich in die Relationen Auftrag und Auftragsposten zerlegt werden
- Ein Fremdschlüssel (Auftrnr in Auftragsposten) entfällt
- Die interne Struktur des Auftrags bleibt erhalten
- Es ist kein Join erforderlich, um die Daten zu lesen
- Hohe Performance (da kein Join erforderlich)

Nachteile objektrelationaler Datenbanken

- Der Fremdschlüssel Artnr (auf Relation Auftrag) kann nicht angegeben werden!
- Die Zugriffsbefehle sind sehr komplex
- Beispiel 1:

```
SELECT * FROM AuftragNeu ;
```

- Beispiel 2:

```
SELECT   Einzelposten
 FROM    AuftragNeu
WHERE    AuftrNr = 2 ;
```

Einzelposten(Artnr, Anzahl, Gesamtpreis)

ET_Einzelposten(TEinzelposten(100002, 3, 1950), TEinzelposten(200001, 1, 400))

Der Operator THE

- Der Operator THE wandelt eine eingebettete Relation in eine „normale“ Relation um
- Beispiel:

Eingebettete Relation

```
SELECT  *
 FROM   THE (   SELECT  Einzelposten
                FROM    AuftragNeu
                WHERE AuftrNr = 2    ) ;
```

Ergebnis: Relation

Artnr	Anzahl	Gesamtpreis
100002	3	1950
200001	1	400

Beispiele mit Operator THE

- **Einfügen eines weiteren Auftragspostens (2 Tretlager):**

```
INSERT INTO THE (  SELECT Einzelposten FROM AuftragNeu
                   WHERE AuftrNr = 2 )
VALUES ( 500013, 2, 60 ) ;
```

„Normale“ Relation dank THE-Operator

- **Ausgabe aller Positionen zu Auftrag 2, die mehr als 100 Euro kosten:**

„Normale“ Relation dank THE-Operator

```
SELECT  *
 FROM   THE (   SELECT  Einzelposten
                FROM    AuftragNeu
                WHERE   AuftrNr = 2   )
 WHERE  Gesamtpreis > 100 ;
```

Cast-Operator MULTISET

- Der Operator THE wandelt eine eingebettete Relation in eine Relation um
- Um alle Auftragspositionen, die mehr als 100 Euro Umfang haben, auszugeben, sind viele THE-Operatoren erforderlich
- Zur gemeinsamen Ausgabe müssen diese wieder in eine Objektrelation zurückverwandelt werden.
- Dies leistet der Cast-Operator Multiset:

Relation

Ergebnis: eingebettete Relation

CAST (MULTISET (Unterabfrage) AS Objekttyp)

Beispiel zu MULTICAST

- Ausgabe der Auftragsnummern mit den dazugehörigen Auftragspositionen über 100 Euro:

```
SELECT A.AuftrNr,
       CAST(MULTISET(
            SELECT *
            FROM THE (  SELECT Einzelposten
                        FROM   AuftragNeu
                        WHERE AuftrNr = A.AuftrNr )
            WHERE Gesamtpreis > 100
                  )   AS ER_Einzelposten )
 FROM AuftragNeu A;
```

Liefert je A.Auftrnr alle Auftragspositionen als Relation

davon nur die Positionen mit Gesamtpreis > 100

Rückumwandeln in eingebettete Relation, um diese Positionen mit der Auftrnr anzuzeigen

Einfügen mittels MULTISET

- Übernahme aller Daten aus Auftrag und Auftragsposten:

```
INSERT INTO AuftragNeu (  Auftrnr, Datum, Einzelposten,
                          Kundnr, Persnr)
 SELECT    AuftrNr, Datum, NULL, Kundnr, Persnr
 FROM      Auftrag;

UPDATE Auftragneu
SET Einzelposten = CAST(MULTISET(
                   SELECT  Teilenr, Anzahl, Gesamtpreis
                   FROM    Auftragsposten
                   WHERE  AuftrNr = Auftragneu.AuftrNr
                                  )  AS ER_Einzelposten ) ;
```

Alle Daten außer Einzelposten aus Relation Auftrag übernehmen

Alle Positionen werden je Auftrag in Attribut Einzelposten übernommen

Zusammenfassung

- Verteilte Datenbanken
  - Verteilte Datenbanken erhöhen die Zugriffsgeschwindigkeit und die Verfügbarkeit auf Kosten der sofortigen Konsistenz
  - ACID wird durch BASE ersetzt!
  - Zwei-Phasen-Commit garantiert sofortige Konsistenz, aber sehr aufwendig
- Objektrelationale Datenbanken
  - Beliebig komplexe Objekte sind direkt abbildbar
  - Dies erfordert jedoch eine sehr aufwendige Programmierung

Datenbanken und SQL

Kapitel 11

Anhang – Die Beispieldatenbank BIKE

Die Beispieldatenbank BIKE

- Hinweise zur Installation
- Die Datenbank BIKE
- Die Relationen der Datenbank BIKE
- Das Erzeugen der Relationen mit Create Table
- Eine PHP-Beispielanwendung

Installation einer Datenbank

- **Kostenlose voll funktionsfähige Datenbanksysteme:**
  - In Oracle ist GUI extra herunterzuladen (SQL Developer)

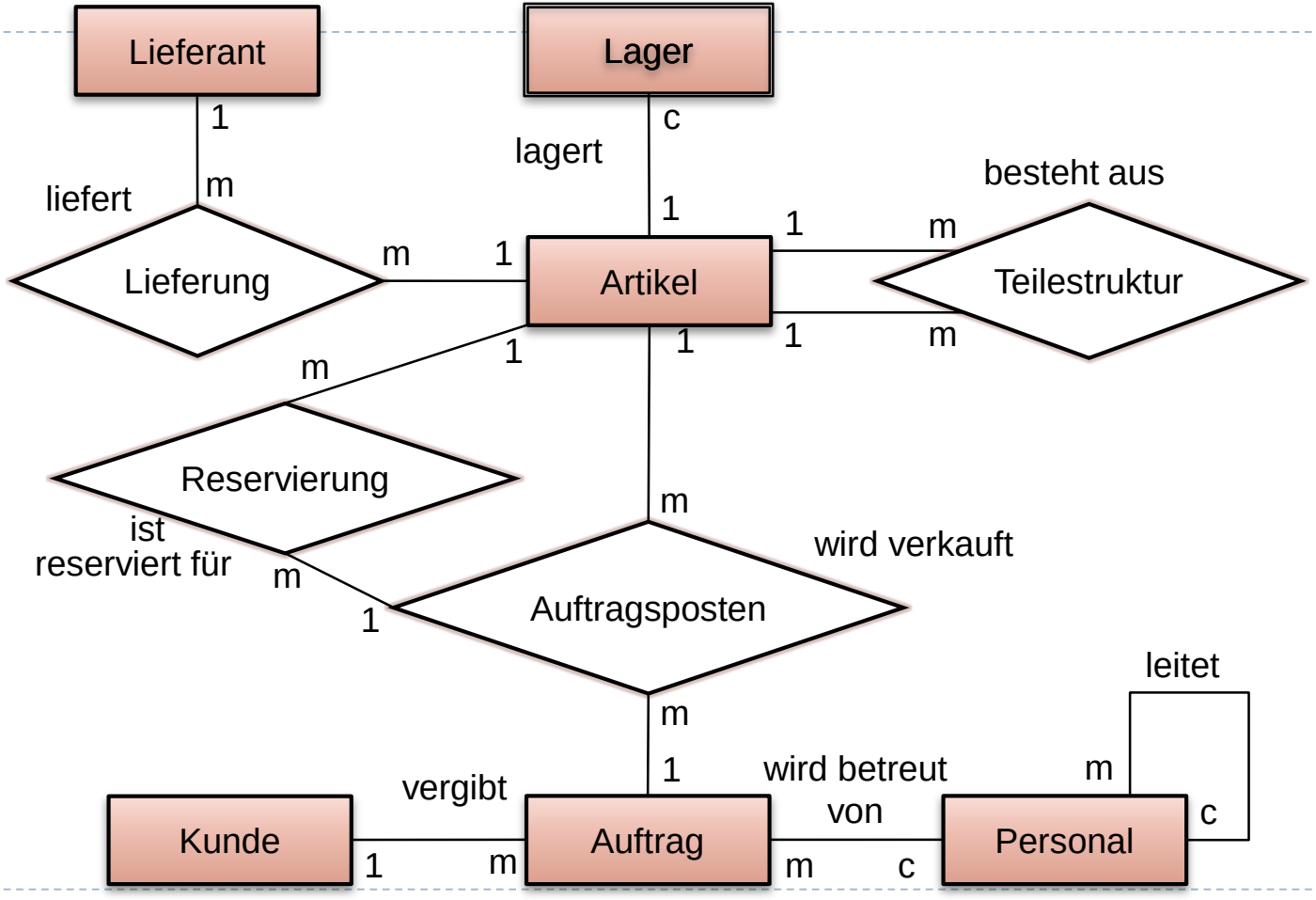
Oracle	Oracle Database Express Edition Oracle SQL Developer	www.oracle.de (Downloads)
Microsoft	Microsoft SQL Server Express Edition	www.microsoft.com/de-de/download (Tools für Entwickler, Serverprodukte)
Oracle	MySQL Community Server	www.mysql.de (Downloads)

Installationsskripte für BIKE

- Jedes Datenbanksystem besitzt Spezifika
  - Daher: Eigene Installationsskripte
  - Jedes Skript ist eine Textdatei
  - Jedes Skript enthält zu Beginn eine Kurzbeschreibung zur Installation

Oracle	bikeOracle.sql
Microsoft	bikeSQLServer.sql
Oracle	bikeMySQL.sql

ER-Diagramm der Datenbank BIKE



Relation LIEFERANT

Nr	Name	Strasse	PLZ	Ort	Sperre
1	Firma Gerti Schmidtner	Dr.Gesslerstr.59	93051	Regensburg	0
2	Rauch GmbH	Burgallee 23	90403	Nürnberg	0
3	Shimano GmbH	Rosengasse 122	51143	Köln	0
4	Suntour LTD	Meltonstreet 65	NULL	London	0
5	MSM GmbH	St-Rotteneckstr.13	93047	Regensburg	0

Relation KUNDE

Nr	Name	Strasse	PLZ	Ort	Sperre
1	Fahrrad Shop	Obere Regenstr. 4	93059	Regensburg	0
2	Zweirad-Center Staller	Kirschweg 20	44267	Dortmund	0
3	Maier Ingrid	Universitätsstr. 33	93055	Regensburg	1
4	Rafa - Seger KG	Liebigstr. 10	10247	Berlin	0
5	Biker Ecke	Lessingstr. 37	22087	Hamburg	0
6	Fahrräder Hammerl	Schindlerplatz 7	81739	München	0

Relation PERSONAL (1)

Persnr	Name	Strasse	PLZ	Ort
1	Maria Forster	Ebertstr. 28	93051	Regensburg
2	Anna Kraus	Kramgasse 5	93047	Regensburg
3	Ursula Rank	Dreieichstr. 12	60594	Frankfurt
4	Heinz Rolle	In der Au 5	90455	Nürnberg
5	Johanna Köster	Wachtelstr. 7	90427	Nürnberg
6	Marianne Lambert	Fraunhofer Str. 3	92224	Landshut
7	Thomas Noster	Mahlergasse 10	93047	Regensburg
8	Renate Wolters	Lessingstr. 9	86159	Augsburg
9	Ernst Pach	Olgastr. 99	70180	Stuttgart

Relation PERSONAL (2)

Persnr	GebDatum	Stand	Vorgesetzt	Gehalt	Beurteilung	Aufgabe
1	05.07.79	verh	NULL	4800.00	2	Manager
2	09.07.75	led	1	2300.00	3	Vertreter
3	04.09.67	verh	6	2700.00	1	Facharbeiterin
4	12.10.57	led	1	3300.00	3	Sekretär
5	07.02.84	gesch	1	2100.00	5	Vertreter
6	22.05.74	verh	NULL	4100.00	1	Meister
7	17.09.72	verh	6	2500.00	5	Arbeiter
8	14.07.79	led	1	3300.00	4	Sachbearbeit.
9	29.03.92	led	6	800.00	NULL	Azubi

Relation AUFTRAG

AuftrNr	Datum	Kundnr	Persnr
1	04.01.2013	1	2
2	06.01.2013	3	5
3	07.01.2013	4	2
4	18.01.2013	6	5
5	03.02.2013	1	2

Relation AUFTRAGSPOSTEN

PosNr	AuftrNr	Artnr	Anzahl	Gesamtpreis
101	1	200002	2	800,00
201	2	100002	3	1.950,00
202	2	200001	1	400,00
301	3	100001	1	700,00
302	3	500002	2	100,00
401	4	100001	1	700,00
402	4	500001	4	30,00
403	4	500008	1	94,00
501	5	500010	1	40,00
502	5	500013	1	30,00

Relation ARTIKEL

ANr	Bezeichnung	Netto	Steuer	Preis	Farbe	Mass	Einh.	Typ
100001	Herren-City-Rad	588.24	111.76	700.00	blau	26 Zoll	ST	E
100002	Damen-City-Rad	546.22	103.78	650.00	rot	26 Zoll	ST	E
200001	He-Rahmen lack.	336.13	63.87	400.00	blau	NULL	ST	Z
200002	Da-Rahmen lack.	336.13	63.87	400.00	rot	NULL	ST	Z
300001	He-Rahmen geschw.	310.92	59.08	370.00	NULL	NULL	ST	Z
300002	Da-Rahmen geschw.	310.92	59.08	370.00	NULL	NULL	ST	Z
400001	Rad	58.82	11.18	70.00	NULL	26 Zoll	ST	Z
500001	Rohr 25CrMo4 9mm	6.30	1.20	7.50	NULL	9 mm	CM	F
500002	Sattel	42.02	7.98	50.00	NULL	NULL	ST	F
500003	Gruppe Deore LX	5.88	1.12	7.00	NULL	LX	ST	F
500004	Gruppe Deore XT	5.04	0.96	6.00	NULL	XT	ST	F
500005	Gruppe XC-LTD	6.72	1.28	8.00	NULL	Xc-Ltd	ST	F
500006	Felgensatz	33.61	6.39	40.00	NULL	26 Zoll	ST	F
500007	Bereifung Schwalbe	16.81	3.19	20.00	NULL	26 Zoll	ST	F
500008	Lenker + Vorbau	78.99	15.01	94.00	NULL	NULL	ST	F
500009	Sattelstütze	4.62	0.88	5.50	NULL	NULL	ST	F
500010	Pedalsatz	33.61	6.39	40.00	NULL	NULL	ST	F
500011	Rohr 34CrMo4 2.1	3.36	0.64	4.00	NULL	2,1 mm	CM	F
500012	Rohr 34CrMo3 2.4	3.61	0.69	4.00	NULL	2,4 mm	CM	F
500013	Tretlager	25.21	4.79	30.00	NULL	NULL	ST	F
500014	Gabelsatz	10.08	1.92	12.00	NULL	NULL	ST	F
500015	Schlauch	6.72	1.28	8.00	NULL	26 Zoll	ST	F

Relation TEILESTRUKTUR

Artnr	EinzelteilNr	Anzahl	Einheit
100001	200001	1	ST
100001	500002	1	ST
100001	500003	1	ST
100001	400001	1	ST
100001	500008	1	ST
100001	500009	1	ST
100001	500010	1	ST
100002	200002	1	ST
100002	500002	1	ST
100002	500004	1	ST
100002	400001	1	ST
100002	500008	1	ST
100002	500009	1	ST
100002	500010	1	ST

Artnr	EinzelteilNr	Anzahl	Einheit
200001	300001	1	ST
200002	300002	1	ST
300001	500001	180	CM
300001	500011	161	CM
300001	500012	20	CM
300001	500013	1	ST
300001	500014	1	ST
300002	500001	360	CM
300002	500011	106	CM
300002	500012	20	CM
300002	500013	1	ST
300002	500014	1	ST
400001	500007	2	ST
400001	500006	1	ST
400001	500015	2	ST

Relation LAGER

Artnr	Lagerort	Bestand	Mindbest	Reserviert	Bestellt
100001	001002	3	0	2	0
100002	001001	6	0	3	0
200001	NULL	0	0	0	0
200002	004004	2	0	0	0
300001	NULL	0	0	0	0
300002	002001	7	0	2	0
400001	005001	1	0	0	0
500001	003005	8050	6000	184	0
500002	002002	19	20	2	10
500003	001003	15	10	0	0
500004	004001	18	10	0	0
500005	003002	2	0	0	0
500006	003004	21	20	0	0
500007	002003	62	40	0	0
500008	003003	39	20	1	0
500009	002007	23	20	0	0
500010	001006	27	20	1	0
500011	001007	3250	3000	161	0
500012	004002	720	600	20	0
500013	005002	20	20	2	0
500014	005003	27	20	1	0
500015	002004	55	40	0	0

Relation RESERVIERUNG

Posnr	Artnr	Anzahl
101	300002	2
201	100002	3
202	500001	180
202	500011	161
202	500012	20
202	500013	1
202	500014	1
301	100001	1
302	500002	2
401	100001	1
402	500001	4
403	500008	1
501	500010	1
502	500013	1

Relation LIEFERUNG

ANr	Liefnr	Lieferzeit	Nettopreis	Bestellt
500001	5	1	6.50	0
500002	2	4	71.30	10
500002	1	5	73.10	0
500003	3	6	5.60	0
500003	4	5	6.00	0
500003	2	4	5.70	0
500004	3	2	5.20	0
500004	4	3	5.40	0
500005	4	5	6.70	0
500006	1	1	31.00	0
500007	1	2	16.50	0
500008	1	4	83.00	0
500009	1	2	4.10	0
500009	2	1	4.60	0
500010	1	3	35.20	0
500011	5	1	3.10	0
500012	5	1	3.40	0
500013	1	4	21.00	0
500014	1	5	9.20	0
500015	1	1	6.20	0

Create Table Befehle: ARTIKEL

```
CREATE TABLE  Artikel
( ANr           INTEGER              PRIMARY KEY,
  Bezeichnung   CHARACTER (35)       NOT NULL,
  Nettopreis    NUMERIC(7,2)         CHECK ( Nettopreis > 0),
  Steuer        NUMERIC(7,2)         CHECK ( Steuer > 0),
  Preis         NUMERIC(7,2)         CHECK ( Preis > 0),
  Farbe         CHARACTER (10),
  Mass          CHARACTER (15),
  Einheit       CHARACTER (2)        NOT NULL,
  Typ           CHARACTER (1)  NOT NULL  CHECK (Typ IN ('E', 'Z', 'F')),
) ;
```

Create Table Befehle: AUFTRAG

```
CREATE TABLE  Auftrag
( AuftrNr      INTEGER    PRIMARY KEY,
  Datum        DATE,
  Kundnr       INTEGER    NOT NULL  REFERENCES Kunde
                          ON DELETE NO ACTION
                          ON UPDATE CASCADE,
  Persnr       INTEGER    REFERENCES Personal
                          ON DELETE SET NULL
                          ON UPDATE CASCADE
);
```

Create Table Befehle: AUFTRAGSPOSTEN

```
CREATE TABLE Auftragsposten
( PosNr        INTEGER          PRIMARY KEY,
  AuftrNr      INTEGER          NOT NULL REFERENCES Auftrag
                                ON DELETE CASCADE
                                ON UPDATE CASCADE,
  Artnr        INTEGER          NOT NULL REFERENCES Artikel
                                ON DELETE NO ACTION
                                ON UPDATE CASCADE,
  Anzahl       SMALLINT,
  Gesamtpreis  NUMERIC(10,2)    CHECK ( Gesamtpreis > 0 ),
  UNIQUE ( AuftrNr, Artnr )
);
```

Create Table Befehle: LIEFERUNG

```
CREATE TABLE Lieferung
( ANr          INTEGER        REFERENCES Artikel
                              ON DELETE CASCADE
                              ON UPDATE CASCADE,
  Liefnr       INTEGER        REFERENCES Lieferant
                              ON DELETE CASCADE
                              ON UPDATE CASCADE,
  Lieferzeit   SMALLINT,
  Nettopreis   NUMERIC(7,2),
  Bestellt     SMALLINT,
  PRIMARY KEY ( ANr,  Liefnr )
);
```

Create Table Befehle: TEILESTRUKTUR

```
CREATE TABLE  Teilestruktur
( Artnr          INTEGER          REFERENCES Artikel
                                  ON DELETE CASCADE
                                  ON UPDATE CASCADE,
  Einzelteilnr   INTEGER          REFERENCES Artikel
                                  ON DELETE NO ACTION
                                  ON UPDATE CASCADE,
  Anzahl         INTEGER,
  Einheit        CHARACTER (2),
  PRIMARY KEY ( ANr, Einzelteilnr )
) ;
```

Zugriff auf BIKE-DB mit PHP (PDO) (1)

```
<html>
<head>
<title>Löschen eines Auftrags aus der Datenbank BIKE</title>
<meta name="description" content="Löschen in der BIKE-DB">
<meta name="author" content="Edwin Schicker">
</head>
<body>
<center><h1>Datenbanken und SQL</h1></center>
<center><h3>Edwin Schicker</h3></center>
<!-- Einloggen in die Datenbank: -->
<p>Programm zum Löschen eines Auftrags<br></p>
<p>Bitte Kennung, Passwort und zu löschenden Auftrag eingeben:</p>
```

Zugriff auf BIKE-DB mit PHP (PDO) (2)

```
<!-- Eingabeformular definieren: -->
<form action="loeschauftrag.php" method="post">
<table cellpadding=10>
<tr>  <td align=right>Datenbank-Kennung: </td>
      <td><input type="Text" name="Kennung" size="20"></td> </tr>
<tr>  <td align=right>Datenbank-Passwort: </td>
      <td><input type="Password" name="Passwort" size="20"> </td> </tr>
<tr>  <td>Datenbankname(z.B. ora11g oder xe):</td>
      <td><input type="Text" name="Connect" size="20“> </td></tr>
<tr>  <td align=right>Nummer des zu löschenden Auftrags: </td>
      <td><input type="Text" name="Nr" size="10"></td></tr>
<tr>  <td></td>  <td align=right> <input type="Submit" value="Weiter">
</td>
</tr>
</table> </form>
```

Zugriff auf BIKE-DB mit PHP (PDO) (3)

```
<?php
// Code wird ausgefuehrt, wenn Variable Kennung verwendet wurde!
if (isset($_POST['Kennung'])) {
  try {
    // neues PDO-Objekt anlegen und mit Oracle-Datenbank verbinden:
    $conn = new PDO("oci:dbname=$_POST[Connect]",$_POST['Kennung'],
                                                    $_POST['Passwort']);
    $conn->beginTransaction();          // Transaktionsmodus
     // Ueberpruefen, ob Auftrag existiert:
    $sql =          "Select Auftrnr
                     From Auftrag
                     Where Auftrnr = $_POST[Nr]";
    $stmt = $conn->query($sql);     // Select Befehl ausfuehren
     if (!$stmt->fetch())
      die ("Die angegebene Auftragsnummer existiert nicht!");    // Abbruch
```

Zugriff auf BIKE-DB mit PHP (PDO) (4)

```
// Loeschen der Reservierungen zu dem Auftrag:
$sql = "Delete From Reservierung
        Where Posnr In ( Select Posnr
                         From Auftragsposten
                         Where Auftrnr = $_POST[Nr])";
$stmt = $conn->query($sql);
// Loeschen des Auftrags:
$sql = "Delete From Auftrag
        Where Auftrnr = $_POST[Nr]";
$stmt = $conn->query($sql);
// Loeschen der Auftragspositionen erfolgt automatisch (Delete Cascade)
echo "Der Auftrag $_POST[Nr] wurde geloescht.";
```

Zugriff auf BIKE-DB mit PHP (PDO) (5)

```
 // Die Datenbank wird jetzt geschlossen:
    $conn->commit();
  }  // end try
  catch (Exception $e)  {
    echo "Das Programm endete mit folgendem Fehler: ".$e->getMessage();
  }
}  // endif isset
?>
</body>
</html>
```

Lösungen zu den Übungsaufgaben aus Kapitel 1

Edwin Schicker

Lösung zu Aufgabe 1

Vorteile: Wissen über die physischen Strukturen ist nicht erforderlich, einfache Programmierung mittels mächtiger Schnittstellenfunktionen, hohe Datensicherheit (Zugriffsschutz, Integrität), Mehrfachzugriff und Recovery werden unterstützt.

Nachteile: Datenbankverwaltung erforderlich, langsamere Zugriffe, höherer Speicherverbrauch.

Lösung zu Aufgabe 2

In allen 4 DML-Befehlen von SQL (Select, Update, Insert, Delete) spielt die Reihenfolge der gespeicherten Daten keine Rolle, weder die Reihenfolge der Zeilen, noch der Spalten! SQL ist eine Sprache, die nur nach einer Information fragt, nicht aber nach der Speicherstelle dieser Information. Es gibt in Standard-SQL keinen Befehl, der die 5. Zeile einer Tabelle ausgibt, stattdessen fragen wir nach dem Inhalt, etwa nach einem Paulaner Weißbier.

Die gleiche Zeile zweimal in einer Tabelle zu halten bringt keinerlei Vorteile. Da aber Nachteile existieren, etwa Redundanz, werden doppelte Datensätze in relationalen Datenbanken grundsätzlich verboten, siehe Kapitel 2.

Lösung zu Aufgabe 3

Sammlung logisch verbundener Daten: Daten, die nicht miteinander in Verbindung stehen, werden in getrennten Datenbanken verwaltet.

Speicherung der Daten mit möglichst wenig Redundanz: Je größer der Datenbestand ist, umso wichtiger wird eine geringe Redundanz, da Widersprüche in den Daten kaum noch erkannt werden können (Suche einer Stecknadel im Heuhaufen!).

Abfragemöglichkeit und Änderbarkeit von Daten: wichtig, denn wozu speichern wir sonst Daten?

Logische Unabhängigkeit der Daten von der physischen Struktur: nicht zwingend erforderlich, aber es erleichtert den Zugriff und die Verwaltung der Daten ungemein.

Zugriffsschutz: zwingend, da beispielsweise ein Bankkunde nur seine eigenen Kontodaten lesen und ändern darf.

Integrität: zwingend, da beispielsweise das Buchen korrekt ablaufen muss oder die Versicherungsdaten korrekt gespeichert sein müssen.

Mehrfachzugriff: zwingend, da Bankkunden oder Sachbearbeiter gleichzeitig auf die Datenbank zugreifen wollen.

Zuverlässigkeit: zwingend, da ein unerlaubtes Eindringen von außen einen enormen Schaden verursachen kann.

Ausfallsicherheit: zwingend, da beispielsweise ein Rechnerabsturz nicht den dauerhaften Ausfall aller oder einzelner Daten nach sich ziehen darf.

Kontrolle: zwingend, um beispielsweise auf Überlast rechtzeitig reagieren zu können. Nichts ist schlimmer, als eine Großdatenbank sozusagen „blind“ laufen zu lassen.

Lösung zu Aufgabe 4

Der Administrator ist der einzige, der den Datenbankaufbau ändern, also erweitern oder löschen, darf. Er vergibt außerdem Zugriffsrechte an alle Benutzer und übernimmt damit die Verantwortung über die Zugriffe auf die Datenbank. Diese Verantwortung kann und darf nicht breit über alle Benutzer gestreut werden.

Lösung zu Aufgabe 5

Sind für eine Transaktion mehrere Mutationen erforderlich, so muss der Benutzer darauf achten, dass er immer alle diese Mutationen hintereinander ausführt. Dies erfordert eine hohe Disziplin, kann aber per Programm automatisiert werden. Werden aus Versehen eine oder mehrere Mutationen ausgelassen oder stürzt der Rechner zwischen diesen Mutationen ab, so müssen nachträglich die fehlenden Mutationen ermittelt und zurückgenommen werden. Dies kann einen enormen Aufwand bedeuten, im schlimmsten Fall müssen alle Datenbankdaten überprüft werden. Dies wird in der Praxis automatisiert ablaufen.

Viele kleine Datenbanken arbeiten ohne Transaktionsmechanismus, weil die Programmierung aufwändiger wird, oder weil dadurch die Antwortzeiten merklich anwachsen. Heute unterstützen alle

auf dem Markt befindlichen Datenbanken Transaktionsmechanismen. Das hohe Fehlerrisiko beim Betrieb ohne Transaktionen sollte Grund genug sein, auf Transaktionen nie zu verzichten. Reine Retrieval-Datenbanken mit nur lesendem Zugriff sind hier eine der wenigen Ausnahmen.

Lösung zu Aufgabe 6

Relationale Datenbanken sind Datenbanken, die ausschließlich aus Tabellen bestehen und der Zugriff nur über diese Tabellen erfolgt. Ältere nicht relationale Datenbanken enthalten zwar ebenfalls Tabellen (Knoten), die jedoch mittels spezieller Verknüpfungen miteinander verbunden sind. Oder anders gesagt: In relationalen Datenbanken wird alles, auch Verknüpfungen, auf Tabellen abgebildet.

Lösung zu Aufgabe 7

a)

Sorte	Hersteller
Export	Löwenbräu
Hell	EKU
Märzen	Hofbräu

b)

Sorte	Hersteller	Anzahl
Märzen	Hofbräu	3
Alkoholfreies Pils	Clausthaler	1

c)

Hersteller	Anzahl
Löwenbräu	22

Lösung zu Aufgabe 8

a) Es kommt folgende Zeile zum Bierdepot hinzu:

Nr	Sorte	Hersteller	Typ	Anzahl
18	Export	EKU	6er Pack	8

b) Die Artikel mit den Nummern 24 und 47 werden gelöscht.

c) Von den Artikeln mit den Nummern 28 und 47 steht jetzt in der Spalte *Anzahl* die Zahl 5 bzw. 3.

Lösung zu Aufgabe 9

Folgende SQL-Anweisungen sind erforderlich:

```
a) SELECT Sorte, Hersteller FROM Bierdepot WHERE Typ = '6er Pack' ;
b) SELECT Sorte FROM Bierdepot WHERE Hersteller = 'Löwenbräu' AND Anzahl>0
                                        AND Typ = 'Kasten';
c) DELETE FROM Bierdepot WHERE Hersteller = 'Kneitinger' ;
e) UPDATE Bierdepot SET Anzahl = Anzahl - 10
      WHERE Hersteller = 'Löwenbräu' AND Sorte = 'Pils' ;
f) INSERT INTO Bierdepot
      VALUES (10, 'Dunkles Weißbier', 'Schneider', 'Kasten', 6) ;
```

Lösung zu Aufgabe 10

Der Transaktionsmechanismus kann in folgenden Fällen etwas aufgeweicht werden:

- Reine Abfrage von Daten (Retrieval-Datenbank)
- In Datenbanken, wo ungefähre Datenangaben reichen (Beispiel: „Zur Zeit sind ca. 5600 Kunden im System“)
- In Datenbanken, die nicht sekundengenau aktuell sein müssen

In all diesen Fällen wird jede Transaktion natürlich ebenfalls beendet, der genaue Zeitpunkt der Beendigung ist aber unbekannt! Somit entfällt der hohe Aufwand für die sofortige Beendigung einer Transaktion.

Lösung zu Aufgabe 11

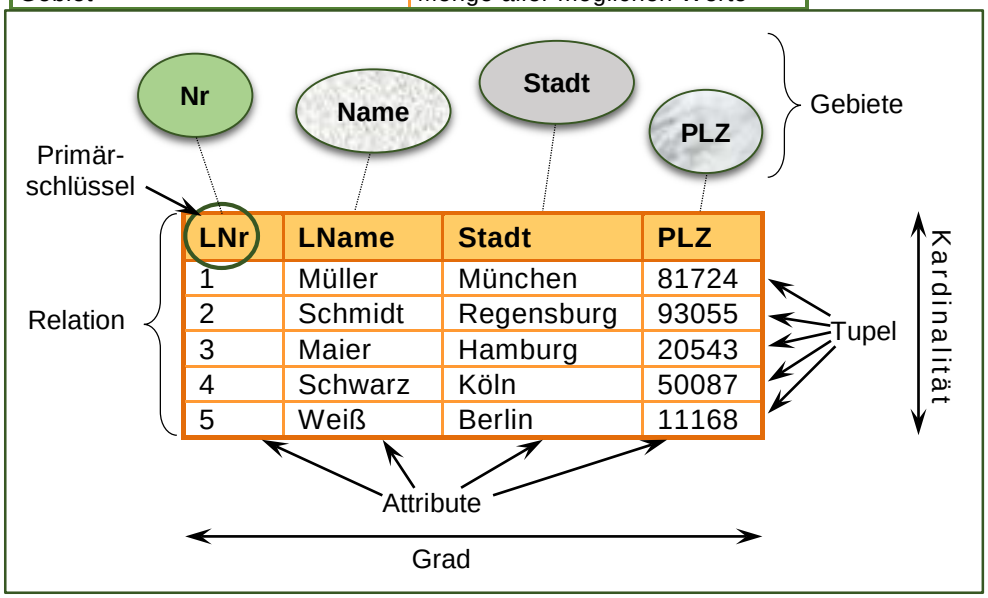
Ohne die Isolation der Benutzer voneinander könnte die Konsistenz verletzt werden. Beispiel: Ein Händler hat von einem bestimmten Artikel nur noch ein Exemplar auf Lager. Zwei Kunden möchten dieses fast gleichzeitig kaufen und werden von zwei Verkäufern an getrennten Theken bedient. Beim Blick in den Rechner bekommen beide Kunden mitgeteilt, dass sie es sofort erhalten könnten. Das Missverständnis lässt sich durch ein entsprechendes Kundengespräch klären, im automatischen Online-Handel könnte der nur einmal vorhandene Artikel aber zweimal verkauft werden!

Lösungen zu den Übungsaufgaben aus Kapitel 2

Edwin Schicker

Lösung zu Aufgabe 1

Formale relationale Bezeichner	Informelle Bezeichnung
Relation	Tabelle
Tupel	eine Zeile (Reihe) einer Tabelle
Kardinalität	Anzahl der Zeilen einer Tabelle
Attribut	eine Spalte (Feld) einer Tabelle
Grad	Anzahl der Spalten einer Tabelle
Primärschlüssel	eindeutiger Bezeichner
Gebiet	Menge aller möglichen Werte



Lösung zu Aufgabe 2

Dagegen spricht: Ein chemisches Element wird identifiziert durch die Protonenzahl, nicht durch seinen Namen oder sein Symbol. Ein eventuell neu entdecktes Element könnte erst dann eingetragen werden, wenn es einen Namen hat. Zusätzlich lassen sich Zahlen als Primärschlüssel besser handhaben als Zeichen.

Lösung zu Aufgabe 3

VerkNr und Produktname zusammen, also (*Verknr*, *Produktname*)

Lösung zu Aufgabe 4

Der Primärschlüssel muss nach der ersten Integritätsregel zwingend angegeben werden. Hinzu kommen alle Attribute mit der Vorgabe *Not Null*.

Lösung zu Aufgabe 5

Primärschlüssel:

Relation Lieferant: Nr

Relation Kunde: Nr

Relation Personal: Persnr

Relation Artikel: ANr

Relation Lager: Artnr

Relation Auftrag: AuftrNr

Relation Reservierung: Posnr + Artnr

Relation Lieferung: ANr + Liefnr

Relation Teilestruktur: Artnr + Einzelteilnr

Relation Auftragsposten: PosNr

Alternative Schlüssel:

Relation Auftragsposten: *AuftrNr* + *Artnr* (je Auftrag gibt es jeden Artikel höchstens einmal)

Lösung zu Aufgabe 6

Fremdschlüssel (Relation.Attribut)	bezieht sich auf Relation
Lieferung.ANr	Artikel
Lieferung.Liefnr	Lieferant

Teilestruktur.Artnr	Artikel
Teilestruktur.Einzelteilnr	Artikel
Lager.Artnr	Artikel
Personal.Vorgesetzt	Personal
Auftrag.Kundnr	Kunde
Auftrag.Persnr	Personal
Auftragsposten.AuftrNr	Auftrag
Auftragsposten.Artnr	Artikel
Reservierung.Posnr	Auftragsposten
Reservierung.Artnr	Artikel

Lösung zu Aufgabe 7

a)

Tupel (Relation: Primärschlüssel)
Personal: 5
Auftrag: 2
Auftrag: 4
Auftragsposten: 201
Auftragsposten: 202
Auftragsposten: 401
Auftragsposten: 402
Auftragsposten: 403
Reservierung: 201 + 100002
Reservierung: 202 + 500001
Reservierung: 202 + 500011
Reservierung: 202 + 500012
Reservierung: 202 + 500013
Reservierung: 202 + 500014
Reservierung: 401 + 100001
Reservierung: 402 + 500001
Reservierung: 403 + 500008

b)

Tupel (Relation: Primärschlüssel)
Artikel: 500001
Teilestruktur: 300001 + 500001
Teilestruktur: 300002 + 500001
Lager: 500001
Reservierung: 22 + 500001
Reservierung: 42 + 500001
Lieferung: 500001 + 5

Lösung zu Aufgabe 8

P.Persnr	P.Name	P.Vorgesetzt	C.Persnr	C.Name	C.Vorgesetzt
1	Maria Forster	NULL	NULL	NULL	NULL
2	Anna Kraus	1	1	Maria Forster	NULL
3	Ursula Rank	6	6	Marianne Lambert	NULL
4	Heinz Rolle	1	1	Maria Forster	NULL
5	Johanna Köster	1	1	Maria Forster	NULL
6	Marianne Lambert	NULL	NULL	NULL	NULL
7	Thomas Noster	6	6	Marianne Lambert	NULL
8	Renate Wolters	1	1	Maria Forster	NULL
9	Ernst Pach	6	6	Marianne Lambert	NULL

Folgende Abkürzungen wurden verwendet: *P* für *Personal* und *C* für *Chef*.

Lösung zu Aufgabe 9

Beim Verbund wird vorausgesetzt, dass beide Relationen ein gemeinsames Attribut besitzen. Bei der Vereinigung zählen eventuell gleiche Einträge nur einmal.

	min. Kardinalität	max. Kardinalität
$A \cup B$	max (M, N)	M + N
$A \bowtie B$	0	M * N
$A \setminus B$	0	M
$A \times B$	M * N	M * N
$A \cap B$	0	min (M, N)

Lösung zu Aufgabe 10

AuftrNr	Datum	Kundnr	Persnr	Nr	Name	PLZ	Ort
1	04.01.13	1	**2**	1	Fahrrad Shop	93059	Regensburg
2	06.01.13	3	**5**	3	Maier Ingrid	93055	Regensburg
3	07.01.13	4	**2**	4	Rafa-Seger KG	10247	Berlin
4	18.01.13	6	**5**	6	Fahrräder Hammerl	81739	München
5	06.02.13	1	**2**	1	Fahrrad Shop	93059	Regensburg

Aus Platzgründen wurden die Attribute *Kunde.Strasse* und *Kunde.Sperre* weggelassen.

Lösung zu Aufgabe 11

Schnitt: $R_3 = R_1 \setminus R_2$ enthält alle Elemente aus R_1, die nicht zur Schnittmenge $R_1 \cap R_2$ gehören. Mit $R_1 \setminus R_3$ erhalten wir dazu das Komplement, also genau die Schnittmenge $R_1 \cap R_2$ selbst.

Verbund: $R_3 = R_1 \times R_2$ liefert alle Kombinationen aus den beiden Relationen. Mit der Restriktion $R_4 = \sigma_{R_1.Y=R_2.Y}(R_3)$ erhalten wir nur noch alle Elemente, die im Attribut Y übereinstimmen, also den Equi-Join! Mit Hilfe der Projektion $\pi_{R_1.X,R_1.Y,R_2.Z}(R_4)$ entfernen wir eines der beiden Y-Attribute. Wir erhalten damit den natürlichen Verbund.

Division: Der Ausdruck $R_3 = \pi_{R_1.X}(R_1) \times R_2$ liefert eine Relation, die die gleichen Attribute wie R_1 enthält, aber alle Kombinationsmöglichkeiten zwischen den Werten $R_1.X$ und $R_2.Y$ enthält. Die Differenz $R_4 = R_3 \setminus R_1$ liefert folglich eine Relation, die alle Kombinationsmöglichkeiten enthält, die nicht in R_1 vorkommen. Mit der Restriktion $R_5 = \pi_{R_1.X}(R_4)$ erhalten wir also alle X-Werte, deren Kombinationsmöglichkeiten nicht in der Relation R_1 enthalten sind. Gesucht sind aber gerade alle Kombinationsmöglichkeiten. Dies liefert dann zuletzt noch die Differenz $\pi_{R_1.X}(R_1) \setminus R_5$.

Lösungen zu den Übungsaufgaben aus Kapitel 3

Edwin Schicker

Lösung zu Aufgabe 1

Der Primärschlüssel beschreibt ein Tupel eindeutig. Jedes Attribut einer Relation ist daher funktional abhängig vom Primärschlüssel. Besteht zusätzlich der Primärschlüssel nur aus einem einzelnen Attribut, so sind alle anderen Attribute sogar voll funktional abhängig vom Primärschlüssel. Dies ist aber gerade die Eigenschaft der zweiten Normalform.

Lösung zu Aufgabe 2

a)	VerkaeuferProdukt:	$VerkNr \Rightarrow (VerkName, PLZ, VerkAdresse)$
		$(VerkNr, Produktname) \Rightarrow Umsatz$
b)	Lieferant, Kunde:	$Nr \Rightarrow (Name, Strasse, PLZ, Ort, Sperre)$
	Personal:	$Persnr \Rightarrow (Name, Strasse, PLZ, Ort, GebDatum, Stand)$
		$Persnr \Rightarrow (Vorgesetzt, Gehalt, Beurteilung, Aufgabe)$
	Artikel:	$ANr \Rightarrow (Bezeichnung, Mass, Einheit, Typ, Liefernr, Zeit)$
		$ANr \Rightarrow AVOnr$
		$(Netto, Steuer) \Rightarrow Preis$
	Teilestruktur:	$(Artnr, Einzelteilnr) \Rightarrow Anzahl$
		$Einzelteilnr \Rightarrow Einheit$
	Lager:	$Artnr \Rightarrow (Lagerort, Bestand, Mindbest, Reserviert)$
		$Artnr \Rightarrow Bestellt$
	Auftrag:	$AuftrNr \Rightarrow (Datum, Kundnr, Persnr)$
	Auftragsposten:	$Posnr \Rightarrow (Auftrnr, Artnr, Anzahl, Gesamtpreis)$
		$(Auftrnr, Artnr) \Rightarrow (Posnr, Anzahl, Gesamtpreis)$
	Reservierung:	$(Posnr, Artnr) \Rightarrow (Anzahl)$
	Lieferung:	$(ANr, Liefnr) \Rightarrow (Lieferzeit, Nettopreis, Bestellt)$

Lösung zu Aufgabe 3

a)	VerkaeuferProdukt:	*VerkNr, (VerkNr, Produktname)*
b)	Lieferant, Kunde:	*Nr*
	Personal:	*PersNr*
	Artikel:	*ANr, (Nettopreis, Steuer)*
	Teilestruktur:	*(Artnr, Einzelteilnr), Einzelteilnr*
	Lager:	*Artnr*
	Auftrag:	*AuftrNr*
	Auftragsposten:	*PosNr, (AuftrNr, Artnr)*
	Reservierung:	*(Posnr, Artnr)*
	Lieferung:	*(ANr, Liefnr)*

Lösung zu Aufgabe 4

Verkaeufer: *VerkNr*; Produkt: *ProdNr*; Verknuepfung: (*VerkNr*, *ProdNr*)

Lösung zu Aufgabe 5

3. NF: *Lieferant*, *Kunde*, *Personal*, *Lager*, *Auftrag*, *Auftragsposten*, *Reservierung*, *Lieferung*; 2. NF: *Artikel*; 1. NF: *Teilestruktur*

Lösung zu Aufgabe 6

Betrachten wir die Relation *VerkaeuferProdukt*. Nehmen wir an, dass ein Verkäufer durch die drei Attribute *VerkName*, *PLZ* und *VerkAdresse* eindeutig identifiziert wird, dann gilt:

(*VerkNr, Produktname*) und (*VerkName, PLZ, VerkAdresse, Produktname*) sind Schlüsselkandidaten und Determinanten. Weitere Determinanten sind: *VerkNr* und (*VerkName, PLZ, VerkAdresse*). Es gibt

keine transitiven Abhängigkeiten außerhalb von Schlüsselkandidaten. Also: 3. NF nach Codd, keine 3. NF nach Boyce und Codd!

Lösung zu Aufgabe 7

Voraussetzung: Es werden alle Kombinationsmöglichkeiten zwischen *VerkNr*, *Produktname* und *KFZNr* für jedes angegebene Jahr aufgelistet. Gegebenenfalls ist bei den Kilometerangaben die Zahl 0 einzutragen.

Primärschlüssel: *(VerkNr, Produktname, KFZNr, Jahr)*

$Primärschlüssel \Rightarrow KM, VerkNr \twoheadrightarrow Produktname, VerkNr \twoheadrightarrow KFZNr$

Dritte Normalform nach Boyce und Codd

Lösung zu Aufgabe 8

Es gilt: Waschmaschinen und Kühlschränke werden nur von den KFZ mit M-E 515 und S-H 654 ausgeliefert. Alle anderen Geräte haben keine Einschränkung. Es fehlen daher die Einträge: (Waschmaschine, S-H 654), (Staubsauger, M-E 515), (Staubsauger, M-X 333), (Staubsauger, S-H 654).

Dies hat keine Auswirkung auf den Verbund der drei Relationen und damit auch nicht auf die Normalform. Kommen noch weitere Verkäufe hinzu, so wird der Verbund die notwendigen und richtigen Einträge erzeugen.

Lösung zu Aufgabe 9

Personal.Vorgesetzt verweist auf *Personal*: on delete set null, on update cascade

Auftrag.Persnr verweist auf *Personal*: on delete set null, on update cascade

Auftrag.Kundnr verweist auf *Kunde*: not null, on delete no action, on update cascade

Lösung zu Aufgabe 10

Lager ist schwach. Beziehungsrelationen sind: *Teilestruktur*, *Reservierung*, *Lieferung* und *Auftragsposten*. *Lager* ist ein Subtyp von *Artikel*. *Artikel* ist ein Supertyp zu *Lager*.

Lösung zu Aufgabe 11

Relation *Rechnung* mit den Attributen *RechNr, Datum, Mahnung, Rabatt, Endpreis, bezahlt*

Primärschlüssel: *Rechnung.ReNr*

Rechnung.ReNr verweist auf *Auftrag*: *On Delete No Action, On Update Cascade*.

Es liegt eine 1 zu *c* Beziehung vor. Rechnung ist aber nicht schwach! Liegt nämlich bereits eine Rechnung vor, so kann der Auftrag nicht einfach gelöscht werden, daher *On Delete No Action*!

Lösungen zu den Übungsaufgaben aus Kapitel 4

Edwin Schicker

Lösung zu Aufgabe 1

```
SELECT Name FROM Personal WHERE Gehalt > 3000 ;
```

Lösung zu Aufgabe 2

```
SELECT SUM(Anzahl) FROM Reservierung ;
```

Lösung zu Aufgabe 3

```
SELECT Artnr, Bezeichnung FROM Lager, Artikel
WHERE ANr = Artnr AND Bestand - Mindbest - Reserviert < 3 ;
```

Lösung zu Aufgabe 4

```
SELECT Artnr, SUM(Anzahl) FROM Teilestruktur GROUP BY Artnr;
```

Lösung zu Aufgabe 5

```
SELECT R.Artnr, A.Bezeichnung, R.Anzahl
FROM  Auftragsposten AP, Reservierung R, Artikel A
WHERE AP.PosNr = R.Posnr AND A.ANr = R.Artnr AND AP.AuftrNr = 2 ;
SELECT R.Artnr, A.Bezeichnung, R.Anzahl
FROM  Auftragsposten AP  INNER JOIN Reservierung R ON AP.PosNr = R.Posnr
                         INNER JOIN Artikel A ON A.ANr = R.Artnr
WHERE AP.AuftrNr = 2 ;
```

Lösung zu Aufgabe 6

```
SELECT R.Artnr, A.Bezeichnung, R.Anzahl
 FROM  Auftragsposten AP, Reservierung R, Artikel A
 WHERE AP.PosNr = R.Posnr AND A.ANr = R.Artnr AND AP.AuftrNr = 2
UNION
SELECT ANr, Bezeichnung, 0 FROM Artikel
 WHERE ANr NOT IN (   SELECT Artnr FROM Reservierung
                      WHERE Posnr IN (SELECT PosNr FROM Auftragsposten
                                      WHERE AuftrNr = 2 ) );

SELECT R.Artnr, A.Bezeichnung, COALESCE(R.Anzahl, 0)
FROM  (SELECT * FROM Auftragsposten WHERE Auftrnr=2) AP
          INNER JOIN Reservierung R ON AP.PosNr = R.Posnr
          RIGHT OUTER JOIN Artikel A ON A.ANr = R.Artnr ;
```

Lösung zu Aufgabe 7

```
UPDATE Lager SET Bestand = Bestand+7 WHERE Artnr IN
       ( SELECT ANr FROM Artikel WHERE Bezeichnung = 'SATTEL' ) ;
```

Lösung zu Aufgabe 8

```
INSERT INTO
 Artikel (ANr, Bezeichnung, Preis, Netto, Steuer, Mass, Einheit, Typ)
 VALUES (100003,'Damen-Mountainbike',650.00,560.34,89.66,'26 Zoll','ST','E') ;
```

Lösung zu Aufgabe 9

```
INSERT INTO Kunde SELECT 10, Name, Strasse, PLZ, Ort, Sperre
          FROM Lieferant WHERE Name = 'Firma Gerda Schmidt' ;
```

Lösung zu Aufgabe 10

```
DELETE FROM Lager WHERE Bestand = 0 ;
```

Lösung zu Aufgabe 11

```
UPDATE Personal SET Beurteilung = 2, Gehalt = Gehalt+100
WHERE Beurteilung = 1 ;
```

Lösung zu Aufgabe 12

```
SELECT P.Persnr, P.Name,
          COALESCE(SUM (AP.Gesamtpreis),0) AS Auftragsvolumen
 FROM    Personal P LEFT OUTER JOIN
          (Auftrag A INNER JOIN Auftragsposten AP ON A.Auftrnr = AP.Auftrnr) ON P.Persnr=A.Persnr
 GROUP BY P.Persnr, P.Name ;
```

```
SELECT P.Persnr, P.Name, SUM (AP.Gesamtpreis) AS Auftragsvolumen
 FROM    Personal P  INNER JOIN Auftrag A ON P.Persnr=A.Persnr
                     INNER JOIN Auftragsposten AP ON A.Auftrnr = AP.Auftrnr
 GROUP BY P.Persnr, P.Name
UNION
SELECT Persnr, Name, 0 FROM Personal
 WHERE Persnr NOT IN ( SELECT Persnr FROM Auftrag) ;
```

Lösung zu Aufgabe 13

```
SELECT Liefnr FROM Lieferung
EXCEPT
SELECT Liefnr FROM
(       SELECT L1.Liefnr,L2.Anr FROM Lieferung L1, Lieferung L2
        WHERE L2.Liefnr = 3
        EXCEPT
        SELECT Liefnr, Anr FROM Lieferung   )  ;
```

Lösung zu Aufgabe 14

```
DELETE FROM Personal ;
SELECT * FROM Personal; SELECT * FROM Auftrag;
ROLLBACK ;
SELECT * FROM Personal; SELECT * FROM Auftrag;
```

Die Relationen sind wieder im vorherigen Zustand.

Lösungen zu den Übungsaufgaben aus Kapitel 5

Edwin Schicker

Lösung zu Aufgabe 1

Relationen *Artikel*, *Auftrag*, *Auftragsposten*, *Teilestruktur*, *Lieferung*: siehe Anhang in Kapitel 10.

```
CREATE TABLE Lieferant ( Nr        INTEGER           PRIMARY KEY,
                         Name      CHARACTER (30) NOT NULL,
                         Strasse   CHARACTER (30),
                         PLZ       CHARACTER (5),
                         Ort       CHARACTER (25),
                         Sperre    CHARACTER   ) ;
```

Analog wird die Relation *Kunde* erzeugt.

```
CREATE TABLE Personal ( PersNr    INTEGER           PRIMARY KEY,
                        Name      CHARACTER (30) NOT NULL,
                        Strasse   CHARACTER (30),
                        PLZ       CHARACTER (5),
                        Ort     CHARACTER (20),
                        GebDatum  DATE              NOT NULL,
                        Stand     CHARACTER (6),
                        Vorgesetzt  INTEGER    REFERENCES Personal
                        ON DELETE SET NULL  ON UPDATE CASCADE,
                        Gehalt    NUMERIC (10, 2),
                        Beurteilung  CHARACTER,
                        Aufgabe  CHARACTER (18) ) ;
CREATE TABLE Lager ( Artnr      INTEGER    PRIMARY KEY  REFERENCES Artikel
                                           ON DELETE CASCADE  ON UPDATE CASCADE,
                     Lagerort  CHARACTER (6),
                     Bestand  SMALLINT   NOT NULL,
                     Mindbest SMALLINT,
                     Reserviert    SMALLINT,
                     Bestellt   SMALLINT   ) ;
CREATE TABLE Reservierung (   Posnr     INTEGER    REFERENCES Auftrag
                                 ON DELETE NO ACTION  ON UPDATE CASCADE,
                              Artnr      INTEGER    REFERENCES Artikel
                                 ON DELETE NO ACTION  ON UPDATE CASCADE,
                              Anzahl    SMALLINT,
                              PRIMARY KEY (Posnr, Artnr)        ) ;
```

Lösung zu Aufgabe 2

```
ALTER TABLE Auftragsposten   ADD COLUMN Einzelpreis Numeric (10,2) ;
UPDATE Auftragsposten SET Einzelpreis = Gesamtpreis / Anzahl
  WHERE Anzahl IS NOT NULL AND Anzahl <> 0;
```

Die neue Relation ist in 2. NF.

Lösung zu Aufgabe 3

```
CREATE VIEW VPers AS
SELECT Persnr, Name, Strasse, PLZ, Ort, GebDatum, Stand, Vorgesetzt, Aufgabe
FROM Personal
WHERE Vorgesetzt IS NOT NULL ;
```

Die Sicht ist änderbar.

Lösung zu Aufgabe 4

```
CREATE VIEW VAuftragsPosten( PosNr, AuftrNr, Artnr, Anzahl, Einzelpreis, Gesamtpreis)  AS
SELECT PosNr, AuftrNr, Artnr, Anzahl, Gesamtpreis/Anzahl, Gesamtpreis
FROM Auftragsposten ;
```

Die Sicht ist nicht änderbar.

Lösung zu Aufgabe 5

Das Hinzufügen von Schemaelementen zu einem Schema erfolgt einfach mittels Qualifizierung. Das Hinzufügen einer Sicht namens *VPers* zum Schema *Verkauf* geschieht beispielsweise wie folgt:

```
CREATE VIEW Verkauf.VPers AS SELECT ...
```

Lösungen zu den Übungsaufgaben aus Kapitel 5

Edwin Schicker

Lösung zu Aufgabe 6

```
CREATE TRIGGER Artikelpreis_Trigger  BEFORE INSERT OR UPDATE
  ON Artikel  REFERENCING NEW AS neu FOR EACH ROW
  BEGIN ATOMIC
    neu.Steuer = neu.Netto * 0.19;
    neu.Preis = neu.Netto * 1.19;
  END ;
```

Lösung zu Aufgabe 7

```
CREATE SEQUENCE Kundnr_Sequenz START WITH 21 INCREMENT BY 2;
INSERT INTO Kunde (Nr, Name)
   VALUES (NEXT VALUE FOR Kundnr_Sequenz, 'Dummy');
```

Lösung zu Aufgabe 8

```
CREATE SEQUENCE Persnr_Sequenz START WITH 10 INCREMENT BY 1;
CREATE TRIGGER Persnr_Trigger BEFORE INSERT
  ON Personal  REFERENCES NEW AS neu FOR EACH ROW
  BEGIN ATOMIC
    neu.Persnr = NEXT VALUE FOR Persnr_Sequenz;
  END ;
```

Lösung zu Aufgabe 9

Diese Aussage stimmt im Prinzip. Ein DBMS nimmt aber in der Regel Optimierung vor und hält die wichtigsten Daten im Cache. Damit gibt es meist doch nur einen Zugriff auf die Festplatte.

Lösung zu Aufgabe 10

```
GRANT Select, Update (Bestand, Reserviert, Bestellt) ON TABLE Lager TO Gast ;
```

Lösung zu Aufgabe 11

```
REVOKE Select, Update ON TABLE Lager FROM Gast ;
```

Lösung zu Aufgabe 12

```
CREATE VIEW VLager AS  SELECT Artnr, Lagerort, Bestand FROM Lager
                                  WHERE Bestand >= Mindbest + Reserviert ;
REVOKE ALL PRIVILEGES ON TABLE Lager FROM Gast ;
GRANT Select ON TABLE VLager TO Gast WITH GRANT OPTION ;
```

Lösung zu Aufgabe 13

```
ALTER TABLE Personal ADD  CONSTRAINT PersIntegr CHECK (
   GebDatum BETWEEN DATE '1940-01-01' AND DATE '1998-12-31'  AND
   Stand IN ('led', 'verh', 'ges', 'verw')  AND  Gehalt BETWEEN 500 AND 6000  AND
   Beurteilung BETWEEN 1 AND 10  )  ;
```

Lösung zu Aufgabe 14

```
CREATE ASSERTION PersIntegr CHECK ( EXISTS
   ( SELECT * FROM Personal WHERE GebDatum BETWEEN DATE '1940-01-01'
      AND DATE '1998-12-31'  AND Stand IN ('led', 'verh', 'gesch', 'verw')  AND
      Gehalt BETWEEN 500 AND 6000  AND Beurteilung BETWEEN 1 AND 10  ) ) ;
```

Lösung zu Aufgabe 15

```
CREATE DOMAIN Berufsbezeichnung AS CHAR(12) CONSTRAINT Berufe1
  CHECK ( VALUE IN ( 'Manager', 'Vertreter', 'Vertreterin', 'Facharbeiter',
              'Facharbeiterin', 'Sekretär', 'Sekretärin', 'Meister', 'Arbeiter', 'Arbeiterin',
              'Sachbearbeiter', 'Sachbearbeiterin', 'Azubi' ) ) ;
```

Lösungshinweise zu den Übungsaufgaben aus Kapitel 6

Edwin Schicker

Die Übungsaufgaben dieses Kapitels sind alle aufwendig. Die Lösungen werden daher im als eigenständige HTML- und PHP-Dateien zur Verfügung gestellt. Hier erfolgen Hinweise zur Lösungsfindung. Es wird empfohlen, das PHP-Programmpaket dieses Kapitels herunterzuladen und zu entpacken. Dann können diese Übungen im Sessionteil hinzugefügt werden.

Lösung zu Aufgabe 1

Geübt werden HTML-Formulare. Bitte verwenden Sie ein Eingabefeld, zwei oder drei Radiobuttons, zwei oder drei Checkboxen und eine Select-Box. Ausgegeben wird der Inhalt des Eingabefeldes, welcher Radiobutton geklickt wurde, welche Checkboxen mit einem Haken versehen wurden und welches Element der Select-Box angegeben wurde.

Lösung zu Aufgabe 2

Bitte beachten Sie, dass statt einer Zahl auch Buchstaben eingegeben werden könnten. Arbeiten Sie daher unbedingt mit einem Try-Catch-Block.

Lösung zu Aufgabe 3

Wir lesen zunächst alle gültigen Personalnummern in eine Combobox mittels einer Schleife ein. Von Vorteil ist jetzt, dass im Programm immer ein existierender Mitarbeiter ausgewählt wird.

Lösung zu Aufgabe 4

Es müssen mindestens der Name und das Geburtsdatum eingegeben werden, da diese Attribute nicht leer sein dürfen (Not Null!). Beim Geburtsdatum wird dringend empfohlen, dies in der Form *jjjj-mm-tt* einzulesen. Verwenden Sie dieses Format zusammen mit dem Operator *Date* (nicht in SQL Server!). Beenden Sie die Transaktion mit Commit!

Die automatische Vergabe der Personalnummer ist einfach: Lesen Sie das bisherige Maximum der Personalnummern ein und addieren die Zahl 1 hinzu. Aber auch das Arbeiten mit Trigger und Sequenz ist möglich.

Lösung zu Aufgabe 5

Arbeiten Sie mit der Vereinigung der beiden Relationen *Kunde* und *Lieferant*. Merken Sie sich in einem zweiten Attribut, aus welcher Relation die Personen stammten. Verwenden Sie die Sortierung mit *Order By*. Die Zahl zu Beginn jeder Zeile ist der Schleifenzähler.

Lösung zu Aufgabe 6

Diese Übung ist sehr aufwendig und folgt den beiden Programmen *transaktion1.php* und *transaktion2.php*. Es wird empfohlen, diese beiden Programme als Basis zu verwenden. Der vorgeschlagene Radiobutton dient nur dazu, Verklemmungen zu provozieren. Starten Sie das fertige Programm zweimal, ändern Sie in beiden Programmen gleiche Artikel und starten Sie das eine Programm mit aufsteigender Änderung, das andere mit absteigender.

Lösung zu Aufgabe 7

Nehmen Sie Aufgabe 4 als Basis, ebenso *blob_ein1.php*, *blob_ein2.php* und *blob_ein3.php*. Statt eines Update-Befehls liegt hier ein Insert-Befehl vor. Kontrollieren Sie das Ergebnis mit Hilfe der Programme aus diesem Kapitel.

Lösung zu Aufgabe 8

Kombinieren Sie die Dateien *blob_ein2.php* und *select.php*. In der ersten Datei ersetzen Sie die Eingabe einer Personalnummer durch ein Select-Feld. Die Verwendung von Select-Feldern wird in der Datei *select.php* gezeigt.

Lösung zu Aufgabe 9

Hier wird es schnell sehr aufwendig. Wir empfehlen für einen Web-Shop keine eigene aufwendige Programmierung. Ein Content-Management-System wie beispielsweise Joomla (www.joomla.de) oder Drupal (www.drupal.org) seien hier empfohlen.

Lösungen zu den Übungsaufgaben aus Kapitel 7

Edwin Schicker

Lösung zu Aufgabe 1

Der kostenbasierte Optimizer benötigt zusätzlich umfangreiche Statistiken zu den einzelnen Relationen. Diese Statistiken sollten immer aktuell sein. Dadurch kann der Optimizer aber die einzelnen Befehle wesentlich besser optimieren.

Lösung zu Aufgabe 2

Die Restriktion erfolgt vor dem Verbund. Der Befehl lautet nun:

```
SELECT A.Bezeichnung, AP.Anzahl, Datum
FROM Auftrag NATURAL INNER JOIN Auftragsposten AP
    INNER JOIN (SELECT Nr FROM KUNDE WHERE Name='Fahrrad Shop') ON Nr=Kundnr
    INNER JOIN Artikel A ON Anr=Artnr
```

Lösung zu Aufgabe 3

Ein eindeutiger Index ist für Namen in der Regel zu einschränkend.

```
CREATE INDEX IBezeichng ON Artikel (Bezeichnung) ;
```

Lösung zu Aufgabe 4

```
CREATE INDEX Suche ON Personal (Einsatzort, Name) ;
```

Lösung zu Aufgabe 5

Vorteil 1: Zugriff gezielt nur auf kleine Relationen (z.B. Monat Juni 2013)

Vorteil 2: Parallele Zugriffe bei großen Suchvorgängen.

Hash-Partitionierung: Vorteil 1 entfällt, aber Vorteil 2 bleibt. Insbesondere liefert die Hashpartitionierung eine sehr gleichmäßige Aufteilung.

Lösung zu Aufgabe 6

Die Referenzpartitionierung ermöglicht es, eine Relation nach einem Kriterium zu partitionieren, das in dieser Relation als Attribut nicht existiert. Beispiel: Partitionierung der Relation *Auftragsposten* nach dem Datum.

In SQL Server und MySQL müsste in Relation *Auftragsposten* das Attribut *Datum* hinzugefügt werden. Dann wäre eine Partitionierung möglich. Die Relation wäre aber nicht mehr in der dritten Normalform.

Lösung zu Aufgabe 7

Einsatz in Datenbanken, wo nur selten Änderungen stattfinden, die Performance aber sehr wichtig ist. Dies trifft vor allem auf Data Warehouses zu. Hier wird nicht geändert, die Abfragen sind gleichzeitig extrem komplex.

Lösung zu Aufgabe 8

Mit der Restriktion wird die Relation *Kunde* von 6 auf einen Eintrag reduziert. Alle folgenden Joins liefern entsprechend kleinere Zwischenrelationen. Bei großen Relationen wirkt sich dies enorm aus.

Lösung zu Aufgabe 9

Sind die betroffenen Relationen bereits sortiert, so müssen diese nur noch zusammen gemischt werden. Dieses Mischen ist relativ performant, ein Merge-Join ist daher dann üblich. Liegen beide zu verknüpfenden Attribute als Indexe vor, so gilt diese Aussage ebenfalls. SQL Server legt daher auch für Fremdschlüssel meist Indexe automatisch an.

Lösung zu Aufgabe 10

Eine Gruppierung ist eine Zusammenfassung und setzt in der Regel eine vorherige Sortierung voraus. Diese Sortierungen sind sehr aufwendig, ein Index kann hier etwas Unterstützung leisten. Eine Group-By-Klausel macht aber erst Sinn, wenn auch Aggregatfunktionen verwendet werden. In diesen Fällen muss aber dann doch auf die Daten zugegriffen werden. Letztlich muss die gesamte Relation in den Arbeitsspeicher geladen werden.

Lösung zu Aufgabe 11

```
CREATE PROCEDURE Preisnachlass( nachlass Numeric ) AS
BEGIN
```

```
  IF nachlass <= 10 THEN
    UPDATE Artikel
    SET Preis = Preis*(100-nachlass)/100, Netto = Netto*(100-nachlass)/100,
          Steuer = Steuer*(100-nachlass)/100;
    UPDATE Auftragsposten
    SET Gesamtpreis = Gesamtpreis*(100-nachlass)/100;
  END IF;
EXCEPTION WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE ('Fehler in der Prozedur Preisnachlass');
END;
```

Lösung zu Aufgabe 12

```
$sql =    "Select Nr, Name FROM Lieferant
            Where Nr IN (SELECT Liefnr FROM Lieferung WHERE Anr = ?" ;
$stmt = $conn->prepare($sql);
foreach ($artnr in $liste)                  // $liste enthalte die gesuchten Artikel
{  echo "<p>Artikelnummer: $artnr</p><p>Lieferanten:</p>";
  $stmt->bindParam(1, $artnr);
  $stmt->execute( ) ;
  while ($row = $stmt->fetch( ) )
    echo "<p>Lieferantnr: $row[NR]; Name: $row[NAME]</p>";
}
```

Lösungen zu den Übungsaufgaben aus Kapitel 8

Edwin Schicker

Lösung zu Aufgabe 1

Die Transaktion ist gültig, da nur die Information aus der Logdatei zählt.

Lösung zu Aufgabe 2

Checkpoints sind Zeitpunkte, wo alle aktuellen Datenbankdaten im Arbeitsspeicher auf das nicht flüchtige externe Medium (meist Festplatte) geschrieben werden. Ohne Checkpoints würden häufig angefasste Daten (Hot Spots) immer im Arbeitsspeicher stehenbleiben und nie auf dem externen Medium aktualisiert werden. Im Fehlerfall müsste dann zwecks Restaurierung immer die gesamte Logdatei durchsucht werden, was sehr zeitaufwendig wäre.

Lösung zu Aufgabe 3

Recovery senkt wegen des zusätzlichen Overheads den Durchsatz und die mittleren Antwortzeiten des Datenbankbetriebs deutlich ab. Entsprechend schnellere Rechner und Speichermedium sind wesentlich teurer. Es werden daher niedrige Betriebskosten und ein zuverlässiger Datenbankbetrieb gegeneinander abgewogen.

Lösung zu Aufgabe 4

Rücksetzen dieser Transaktion mittels eines *Rollback* inklusive der Freigabe von gehaltenen Sperren. Fehlermeldung an den Benutzer. Empfohlen wird eine Protokollierung des Fehlers zwecks späterer Analyse.

Lösung zu Aufgabe 5

Anhalten des gesamten Datenbankbetriebs; Rücksetzen aller noch nicht abgeschlossener Transaktionen mittels *Rollback*; Abklemmen der Festplatte; Montieren einer neuen Festplatte; Kopieren der letzten Sicherungen auf die neue Festplatte; Nachfahren der Änderungen seit der letzten Sicherung mittels der Logdateiinformationen; Neustart des Datenbankbetriebs.

Lösung zu Aufgabe 6

Eigentlich nie! Denn dann kann kein Rollback mehr durchgeführt werden.

Lösung zu Aufgabe 7

Nie! Ein Ausfall der Festplatte mit den gespeicherten Datenbankdaten würde sonst zu Datenverlusten führen.

Lösung zu Aufgabe 8

Undo-Daten (Before-Images) müssen nur bis Transaktionsende aufgehoben werden, Redo-Daten (After-Images) hingegen bis zur nächsten Sicherung der Datenbank.

Lösung zu Aufgabe 9

In der Praxis werden Deadlocks meist mittels Wartegraphen erkannt. Eine der betroffenen Transaktionen wird zurückgesetzt und neu gestartet.

Lösung zu Aufgabe 10

Optimistisches Verfahren. Steigt die Konfliktwahrscheinlichkeit auf nur wenige Prozent, so schaukelt sich das System auf. Es wird unbrauchbar.

Lösung zu Aufgabe 11

Ja, wenn es Share- und Exklusivlock gibt. Sonst: Nein.

Lösung zu Aufgabe 12

Nur teilweise: Transaktionen, die auf gemeinsame Daten zugreifen, werden serialisiert; eine Reihenfolge kann also angegeben werden. Für alle anderen Transaktionen, die gleichzeitig ablaufen, gilt dies nicht.

Lösung zu Aufgabe 13

Nein. Auch ohne Sperrmechanismen wäre (zufälligerweise) keine Inkonsistenz aufgetreten.

Lösungen zu den Übungsaufgaben aus Kapitel 9

Edwin Schicker

Lösung zu Aufgabe 1

Fragmentierung: Daten einer Relation können fragmentiert, d.h. auf mehrere Rechner verteilt sein.

Replikation: Daten können auf mehreren Rechnern gleichzeitig gehalten werden.

Unabhängigkeit bedeutet in beiden Fällen, dass sich das System an der Schnittstelle nicht anders verhält, ob nun mit oder ohne Fragmentierung bzw. Replikation gearbeitet wird.

Lösung zu Aufgabe 2

Eine Datenbank heißt verteilt, wenn die zusammengehörigen Daten dieser Datenbank auf mindestens zwei Rechnern aufgeteilt sind und von einem gemeinsamen Datenbankverwaltungssystem verwaltet werden.

Lösung zu Aufgabe 3

In einem verteilten Datenbanksystem können selbstverständlich auch lokale Deadlocks auftreten. Denken wir nur an eine Transaktion, die nur lokale Zugriffe durchführt. Es können also jederzeit sowohl globale als auch lokale Deadlocks vorkommen.

Lösung zu Aufgabe 4

Die Regel 2 ist dann erfüllbar, wenn der Koordinator nicht zentral auf einem Rechner gehalten wird. Übernimmt beispielsweise immer derjenige Rechner die Koordination, auf dem eine globale Transaktion gestartet wurde, so ist Regel 2 nicht verletzt.

Lösung zu Aufgabe 5

Individuell zu beantworten. In der Praxis verbreitet ist das Zwei-Phasen-Commit-Protokoll. Hier werden mehrere Regeln von Date verletzt: 1, 2, 4, 5, 6, 8, 12. Die Verteilung ist sehr eingeschränkt, dafür wird sehr hoher Wert auf die Konsistenz gelegt. Es liegt ein CA-System vor.

Lösung zu Aufgabe 6

Das CAP-Theorem sagt aus, dass es nur entweder CA-, CP- oder PA-Datenbanken gibt. Im Falle von PA-Datenbanken wird Konsistenz „vernachlässigt“. Das Base-Modell zeigt Möglichkeiten auf, um die Konsistenz zwar nicht unmittelbar, aber im Endeffekt doch noch sicherzustellen. Die letztendliche Konsistenz ist auch in PA-Datenbanken extrem wichtig.

Lösung zu Aufgabe 7

Mittels der Einführung von NF^2-Relationen und des Recursive-Union-Operators.

Lösung zu Aufgabe 8

Zunächst sind Datentypen (insbesondere variable Felder und eingebettete Relationen) mittels des Create-Type-Befehls zu definieren. Hier sind als Datentypen auch Unterobjekte zugelassen. Das Erzeugen einer Relation dieses Typs erfolgt dann mit einem syntaktisch erweiterten Create-Table-Befehl.

Lösung zu Aufgabe 9

```
CREATE OR REPLACE TYPE TPerson AS OBJECT
      ( Name CHARACTER (20),  Adresse TAdresse,  GebDatum DATE ) ;
CREATE TABLE PersonalNeu
(     Persnr INTEGER,  Person TPerson,  Stand CHARACTER(6),
      Vorgesetzt INTEGER REFERENCES PersonalNeu,  Gehalt NUMERIC(10,2),
      Beurteilung SMALLINT,  Aufgabe CHARACTER(10) );
```

Lösung zu Aufgabe 10

```
SELECT *  FROM THE (    SELECT Einzelposten FROM AuftragNeu
                        WHERE AuftrNr = 4  )   T
          WHERE T.Anzahl > 1 ;
```

Lösung zu Aufgabe 11

In relationalen Systemen ist dies notwendig, sobald eine Transaktion auf mehr als eine Datenbank schreibend zugreift, für die kein gemeinsames Log geführt wird.

Lösungen zu den Übungsaufgaben aus Kapitel 9

Edwin Schicker

Lösung zu Aufgabe 12

Je Datenbank muss ein Log geführt werden, zusätzlich ist ein globales Log erforderlich. Der Hauptaufwand ist in der Regel jedoch der hohe Kommunikationsoverhead.

Lösung zu Aufgabe 13

```
INSERT INTO AuftragNeu ( Auftrnr, Datum, Einzelposten, Kundnr, Persnr )
 SELECT Auftrnr, Datum,
            CAST( MULTISET(   SELECT Teilenr, Anzahl, Gesamtpreis
                              FROM Auftragsposten
                              WHERE AuftrNr = A.AuftrNr )
                    AS ER_Einzelposten ) ,        Kundnr, Persnr
 FROM Auftrag A ;
```

Aufgabe 1

Welches sind die Vorteile einer Datenbank gegenüber der direkten Verwaltung von Daten? Welches sind die Nachteile?

Aufgabe 2

Beantworten Sie an Hand des Beispiels des Bierdepots folgende Fragen intuitiv: Hängt die SQL-Datenabfrage von der Reihenfolge der Datensätze ab? Hängt die Manipulation von Daten von der Reihenfolge der Datensätze ab? Spielt vielleicht die Reihenfolge der Spalten beim Lesen oder Schreiben eine Rolle? Sind doppelte vollständig identische Datensätze in irgendeiner Weise von Nutzen?

Aufgabe 3

Betrachten Sie eine Großdatenbank (z.B. Datenbank einer Versicherung oder eines Geldinstituts). Geben Sie zu allen Anforderungen aus Abschnitt 1.2 einzeln an, ob und wenn ja, warum diese Anforderungen notwendig sind. Die Anforderungen sind:

- Sammlung logisch verbundener Daten
- Speicherung der Daten mit möglichst wenig Redundanz
- Abfragemöglichkeit und Änderbarkeit von Daten
- Logische Unabhängigkeit der Daten von ihrer physischen Struktur
- Zugriffsschutz
- Integrität
- Mehrfachzugriff (Concurrency)
- Zuverlässigkeit
- Ausfallsicherheit
- Kontrolle

Aufgabe 4

Warum ist die Trennung zwischen Administrator und Anwendern (Benutzer) in Datenbanken so wichtig?

Aufgabe 5

Wenn eine (kleinere) Datenbank im transaktionslosen Einbenutzerbetrieb verwendet wird, so muss die Konsistenz der Daten vom Anwender selbst kontrolliert werden. Wie könnte dies aussehen? Diskutieren Sie den Aufwand? Warum arbeiten heute trotzdem noch einige kleine Datenbanken ohne Transaktionsmechanismen?

Aufgabe 6

Was ist eine relationale Datenbank? Wie unterscheidet sie sich von nicht relationalen?

Aufgabe 7

Geben Sie die Ergebnisse folgender SQL-Abfrage-Operationen zum Bierdepot aus:

a)
```
SELECT Sorte, Hersteller FROM  Bierdepot
                         WHERE Typ = 'Fass';
```

b)
```
SELECT Sorte, Hersteller, Anzahl FROM  Bierdepot
                                 WHERE Anzahl < 4;
```

c)
```
SELECT Hersteller, Anzahl FROM  Bierdepot
                          WHERE Sorte = 'Pils'
                           AND  Typ = 'Kasten';
```

Aufgabe 8

Geben Sie die Ergebnisse folgender SQL-Änderungs-Operationen zum Bierdepot aus:

a)
```
INSERT INTO Bierdepot
VALUES (18, 'Export', 'EKU', '6er Pack', 8);
```

b)
```
DELETE FROM Bierdepot
WHERE Typ = 'Kasten'  AND  Anzahl < 5;
```

c)
```
UPDATE Bierdepot
SET      Anzahl = Anzahl + 2
WHERE Nr = 28  OR  Nr = 47;
```

Aufgabe 9

Schreiben Sie die entsprechenden SQL-Anweisungen:

a) Geben Sie alle Sorten mit Hersteller an, die 6er-Packs vertreiben.

b) Geben Sie alle Sorten an, die vom Hersteller Löwenbräu im Depot vorrätig sind.

c) Entfernen Sie alle Sorten des Herstellers Kneitinger.

d) Entfernen Sie 10 Kasten Pils des Herstellers Löwenbräu.

e) Fügen Sie die Biersorte Dunkles Weißbier der Firma Schneider mit der Nummer 10, dem Typ Kasten und der Anzahl 6 hinzu.

Aufgabe 10

In NOSQL-Datenbanken wird der Transaktionsmechanismus aufgeweicht. In welchen Einsatzbereichen scheint dies hinnehmbar zu sein, wo nicht?

Aufgabe 11

Warum ist Isolation (Isolation steht für den Buchstaben *I* in ACID) so wichtig? Geben Sie ein Beispiel an, warum fast gleichzeitige ungeschützte parallele Zugriffe auf Daten die Konsistenz verletzen können.

Aufgabe 1

Erklären Sie folgende Begriffe: Tupel, Attribut, Relation, Gebiet, Grad, Kardinalität.

Aufgabe 2

Was spricht dagegen, in der Relation der chemischen Elemente den Schlüsselkandidaten *Symbol* als Primärschlüssel zu wählen?

Aufgabe 3

Geben Sie den Primärschlüssel der Relation *VerkaeuferProdukt* an.

Aufgabe 4

Welche Attribute eines neuen einzutragenden Tupel müssen immer mindestens angegeben werden? Denken Sie dabei an die erste Integritätsregel.

Aufgabe 5

Geben Sie die Primärschlüssel aller Relationen der Beispieldatenbank *Bike* an. Finden Sie auch alle alternativen Schlüssel.

Aufgabe 6

Geben Sie alle Fremdschlüssel der Beispieldatenbank *Bike* an.

Aufgabe 7

Nehmen wir an, in der Beispieldatenbank *Bike* gelte für alle Fremdschlüssel die Eigenschaft *on delete cascade*. Geben Sie alle Tupel an, die kaskadierend gelöscht werden, wenn

a) der Eintrag von Fr. Köster in der Relation *Personal*

b) der Eintrag 500001 in der Relation *Artikel* gelöscht wird.

Aufgabe 8

Bilden Sie einen Verbund der Relation *Personal* auf sich, also $Personal \bowtie Personal$. Hierbei ist das Attribut *Vorgesetzt* der einen Relation mit dem Attribut *Persnr* der anderen Relation zu verknüpfen. Geben Sie nur die Attribute *Persnr*, *Name* und *Vorgesetzt* in beiden Relationen aus, also:

$$\pi_{Persnr,Name,Vorgesetzt}(Personal) \bowtie_{Personal.Vorgesetzt=Chef.Persnr} \rho_{Personal \to Chef}\left(\pi_{Persnr,Name,Vorgesetzt}(Personal)\right)$$

Aufgabe 9

Es seien zwei Relationen *A* und *B* mit der Kardinalität *M* respektive *N* gegeben. Geben Sie jeweils die minimale und die maximale Kardinalität der folgenden Ergebnisrelationen an (in Abhängigkeit von *M* und *N*):

$A \cup B$, $A \bowtie B$, $A \setminus B$, $A \times B$, $A \cap B$

Aufgabe 10

Geben Sie die Relation *R* aus mit $R = Kunde \bowtie_{Nr=Kundnr} Auftrag$.

Aufgabe 11

Die Operatoren Schnitt, Verbund und Division können aus den verbleibenden relationalen Operatoren hergeleitet werden. Vollziehen Sie die entsprechenden Formeln nach:

$$R_1 \cap R_2 = R_1 \setminus (R_1 \setminus R_2)$$

$$R_1 \bowtie R_2 = \pi_{R_1.X,R_1.Y,R_2.Z}\left(\sigma_{R_1.Y=R_2.Y}(R_1 \times R_2)\right)$$

$$R_1 \div R_2 = \pi_{R_1.X}(R_1) \setminus \pi_{R_1.X}\left(\left(\pi_{R_1.X}(R_1) \times R_2\right) \setminus R_1\right)$$

Aufgabe 1

Zeigen Sie, dass sich jede Relation mit nicht zusammengesetztem Primärschlüssel in der zweiten Normalform befindet.

Aufgabe 2

Geben Sie alle vollen funktionalen Abhängigkeiten

a) in der Relation *VerkaeuferProdukt*

b) in allen Relationen der *Bike*-Datenbank an.

Aufgabe 3

Geben Sie zu allen Relationen aus Aufgabe 2 die Determinanten an.

Aufgabe 4

Geben Sie zu den Relationen *Verkaeufer*, *Produkt* und *Verknuepfung* alle Determinanten an.

Aufgabe 5

Bestimmen Sie die höchste Normalform (nur bis zur 3.NF!) aller Relationen der *Bike*-Datenbank.

Aufgabe 6

Geben Sie eine Relation an, die in der dritten Normalform nach Codd, jedoch nicht in der dritten Normalform nach Boyce und Codd ist. (Hinweis: Diese Relation muss zwei zusammengesetzte Schlüsselkandidaten besitzen, die sich in mindestens einem Attribut überlappen.)

Aufgabe 7

Diskutieren Sie die Relation *VerkaeuferProduktKFZ*. Bestimmen Sie den Primärschlüssel und die funktionalen und mehrwertigen Abhängigkeiten. Schließen Sie daraus auf die Normalform.

Aufgabe 8

Die Relation *ProduktKFZ* ist aus einer Projektion entstanden. Sie enthält aber in der Praxis nicht alle Kombinationsmöglichkeiten. Welche Tupel sollten daher noch hinzugefügt werden? Welches Ergebnis ergibt dann der Verbund aus dieser Relation mit den Relationen *VerkaeuferProduktname* und *VerkaeuferKFZ*? Kommen eventuell gegenüber der Relation *VerkaeuferProduktKFZ4NF* noch weitere Tupel hinzu? Wirkt sich dies gegebenenfalls auf die Normalform aus?

Aufgabe 9

Geben Sie zu allen Fremdschlüsseln der Relationen *Personal*, *Kunde* und *Auftrag* der Datenbank *Bike* die drei Fremdschlüsseleigenschaften an.

Aufgabe 10

Geben Sie zur Beispieldatenbank *Bike* an, welche Entitäten schwach sind, bei welchen Relationen es sich um Beziehungsrelationen handelt und ob Sub- und Supertypen vorliegen.

Aufgabe 11

Das Auftragswesen der Datenbank *Bike* ist nur rudimentär implementiert. Erweitern Sie daher die Datenbank durch ein einfaches Rechnungswesen. Die neue Relation *Rechnung* sollte mindestens ein Rechnungsdatum, den Rechnungsbetrag, Informationen über einen

Rabatt, erfolgte Mahnung und erfolgte Bezahlung enthalten. Ergänzen Sie das Entity-Relationship-Modell zur Datenbank *Bikek* (siehe Anhang). Geben Sie alle neuen Fremd- und Primärschlüssel und die Eigenschaften der Fremdschlüssel an. Ist die Relation *Rechnung* schwach?

Übungsaufgaben zu Kapitel 4

Edwin Schicker

Die folgenden Aufgaben beziehen sich ausnahmslos auf die Relationen der Beispieldatenbank *Bike*.

Aufgabe 1

Schreiben Sie einen *Select*-Befehl, der aus der Relation *Personal* die Namen aller Personen ermittelt, die mehr als 3000 Euro verdienen.

Aufgabe 2

Geben Sie mittels eines *Select*-Befehls die Gesamtanzahl der für Aufträge reservierten Artikel aus (die benötigten Informationen stehen in der Relation *Reservierung*).

Aufgabe 3

Geben Sie mittels eines *Select*-Befehls alle Artikel der Relation *Lager* aus, dessen Bestand abzüglich des Mindestbestands und der Reservierungen unter den Wert 3 gesunken ist. Als Ausgabe werden Artikelnummer und Artikelbezeichnung erwartet.

Aufgabe 4

Aus wie vielen Einzelteilen bestehen alle zusammengesetzten Artikel? Bestimmen Sie diese Stückzahlen mittels eines *Select*-Befehls. Falls ein Einzelteil wieder aus noch kleineren Einzelteilen besteht, so ist dies nicht weiter zu berücksichtigen. Ausschlaggebend zur Ermittlung der Anzahl der Einzelteile ist das Attribut *Anzahl* ohne Rücksichtnahme auf die Einheit (‚ST' oder ‚CM').

Aufgabe 5

Geben Sie alle Artikel aus, die vom Auftrag mit der Auftragsnummer 2 reserviert sind. Geben Sie dazu zu jedem Artikel die Artikelnummer, die Artikelbezeichnung und die Anzahl der für diesen Auftrag reservierten Artikel aus. Lösen Sie die Aufgabe einmal unter Verwendung des Operators *Join* und einmal ohne diesen Operator (also mit Kreuzprodukt und Restriktion).

Aufgabe 6

Modifizieren Sie die Aufgabe 5 dahingehend, dass alle Artikel der Relation *Artikel* ausgegeben werden (äußerer Verbund) und nicht nur die reservierten. Für die nicht für Auftrag 2 reservierten Teile ist die entsprechende Spaltenangabe zu den Reservierungen auf den Zahlenwert 0 zu setzen. Schreiben Sie den *Select*-Befehl einmal ohne und einmal mit Verwendung des Operators *Outer Join*.

Achtung: Ein Where-Teil nach dem Outer Join zerstört diesen. Die benötigte Restriktion muss deshalb vor dem Outer Join ausgeführt werden!

Aufgabe 7

Es sind sieben neue Sättel eingetroffen. Modifizieren Sie die Datenbank.

Aufgabe 8

Es wird ein Damen-Mountainbike ins Sortiment aufgenommen. Weitere Angaben sind: ANr 100003, Preis 650.00, Nettopreis 560.34, Steuer 89.66, Mass 26 Zoll, Einheit ST, Typ E. Nehmen Sie dieses Fahrrad in die Relation *Artikel* auf.

Aufgabe 9

Die Lieferantin „Firma Gerda Schmidt" wird auch Kundin mit der Kundennummer 10. Nehmen Sie die Lieferantin auch in die Kundenrelation auf. Die benötigten Daten sind direkt der Lieferantenrelation zu entnehmen.

Aufgabe 10

Löschen Sie alle Artikel aus der Relation *Lager*, deren Bestand auf 0 gesunken ist.

Aufgabe 11

Erhöhen Sie das Gehalt aller Mitarbeiter um 100 Euro, bei denen die Beurteilung 1 eingetragen ist. Senken Sie gleichzeitig die Beurteilung um eine Notenstufe.

Aufgabe 12

Geben Sie zu allen Mitarbeitern (*Persnr*, *Name*) die Gesamtumsatzsumme der von ihnen betreuten Aufträge ist. Es liegt ein äußerer Verbund vor. Schreiben Sie diesen Befehl ohne Verwendung des Bezeichners *Natural* und einmal mit und einmal ohne Verwendung des Outer-Join-Operators.

Aufgabe 13

Geben Sie alle Lieferanten an, die mindestens die gleichen Artikel liefern wie Lieferant 3. Verwenden Sie dazu die Division. Bilden Sie die Division mit der Ersatzdarstellung mittels Kreuzprodukt und Differenz nach.

Aufgabe 14

Testen Sie den Transaktionsbetrieb. Löschen Sie alle Mitarbeiter in Relation *Personal*. Überprüfen Sie, ob das Löschen erfolgte. Beachten Sie wegen der Fremdschlüssel auch Relation *Auftrag*. Setzen Sie jetzt die Transaktion zurück. Wie wirkt sich dies aus?

Übungsaufgaben zu Kapitel 5

Edwin Schicker

Die folgenden Aufgaben beziehen sich auf die Relationen der Beispieldatenbank *Bike*.

Aufgabe 1

Schreiben Sie alle Create-Table-Befehle zum Erzeugen der Beispieldatenbank *Bike*. Geben Sie alle Entitäts- und Referenz-Integritätsregeln in Form von Tabellen- und Spaltenbedingungen an. Ergänzen Sie diese Angaben durch sinnvolle weitere Integritätsbedingungen (*Unique, Not Null, Check*).

Aufgabe 2

Fügen Sie zur Relation *Auftragsposten* das Attribut *Einzelpreis* hinzu. Füllen Sie dieses Attribut mit Daten auf, ermittelt aus den Attributen *Anzahl* und *Gesamtpreis*. In welcher Normalform befindet sich die Relation *Auftragsposten* jetzt?

Aufgabe 3

Erzeugen Sie in der Beispieldatenbank *Bike* eine Sicht *VPers*, die der Relation *Personal* ohne die Attribute *Gehalt* und *Beurteilung* entspricht. Weiter sind in dieser Sicht nur die Personen aufzunehmen, denen ein Vorgesetzter zugeordnet ist. Liegt eine änderbare Sicht vor?

Aufgabe 4

Die Relation *Auftragsposten* enthält aus Redundanzgründen nur den Gesamtpreis jedes einzelnen Auftragspostens. Schreiben Sie daher eine Sicht *VAuftragsposten*, die alle Daten der Relation *Auftragsposten* enthält und zusätzlich ein Attribut *Einzelpreis*. Ist diese Sicht änderbar?

Aufgabe 5

In SQL gibt es keinen *Alter-Schema*-Befehl. Überlegen Sie, wie in ein existierendes Schema weitere Schemaelemente aufgenommen werden können. Zeigen Sie dies an einem Beispiel.

Aufgabe 6

Beim Einfügen und Ändern von Artikeln soll automatisch aus dem Nettopreis die Mehrwertsteuer (19%) und der Gesamtpreis ermittelt werden. Schreiben Sie einen geeigneten Trigger. Testen Sie den Trigger.

Aufgabe 7

Alle neuen Kunden sollen automatisch mit einer Kundennummer versehen werden. Diese Nummern beginnen bei 21. Es sollen nur ungeradzahlige Kundennummern vergeben werden. Schreiben Sie eine geeignete Sequenz. Probieren Sie diese Sequenz durch Hinzufügen von neuen Kunden aus.

Aufgabe 8

In MySQL gibt es die Spaltenbedingung *AutoIncrement*. Damit erhält dieses Attribut immer eine eindeutige automatische Nummer. Bilden Sie diese Funktion mittels Sequenzen und Trigger für das Attribut *Persnr* der Relation *Personal* nach.

Aufgabe 9

In einem sicheren Datenbankverwaltungssystem wird vor jedem Zugriff auf eine Relation überprüft, ob der Benutzer die erforderlichen Zugriffsrechte besitzt. Ein Datenbankzugriff besteht demnach faktisch aus zwei Zugriffen (Sicherheitsüberprüfung und eigentlicher Zugriff). Stimmt diese Aussage wirklich?

Aufgabe 10

Schreiben Sie einen Befehl, der dem Benutzer *Gast* Änderungsrechte auf die Attribute *Bestand*, *Reserviert* und *Bestellt* der Relation *Lager* und Leserechte auf die gesamte Relation einräumt.

Aufgabe 11

Entziehen Sie dem Benutzer *Gast* die in der vorherigen Aufgabe gewährten Rechte wieder.

Aufgabe 12

Schreiben Sie alle notwendigen Befehle, damit der Benutzer *Gast* nur Leserechte auf die Attribute *Artnr*, *Lagerort* und *Bestand* der Relation *Lager* bekommt. Weiter darf er Tupel dieser Relation nicht sehen, falls Mindestbestand plus reservierte Teile größer als der tatsächliche Bestand ist. Diese Rechte darf der Benutzer *Gast* auch weiterreichen.

Aufgabe 13

Um die Integrität zu optimieren, sollen die Attribute *GebDatum*, *Stand*, *Gehalt* und *Beurteilung* der Relation *Personal* auf zulässige Werte überprüft werden. Es ist bekannt, dass alle Mitarbeiter zwischen 1940 und 1998 geboren sind, entweder ledig, verheiratet, geschieden oder verwitwet sind, das Gehalt zwischen 500 und 6000 Euro liegt und die Beurteilung entweder *Null* oder einen Wert zwischen 1 und 10 besitzt. Fügen Sie diese Bedingungen mittels geeigneter *Alter-Table*-Befehle hinzu, wobei sicherzustellen ist, dass diese Bedingungen, falls gewünscht, auch wieder entfernt werden können (bitte Constraintnamen vergeben!).

Aufgabe 14

Lösen Sie die letzte Aufgabe mit Hilfe eines *Create-Assertion*-Befehls.

Aufgabe 15

Im Attribut *Aufgabe* der Relation *Personal* gibt es nur eine beschränkte Anzahl von möglichen Aufgaben. Definieren Sie ein Gebiet *Berufsbezeichnung*, das eine Ansammlung von möglichen Berufen enthält.

Übungsaufgaben zu Kapitel 6

Edwin Schicker

Die meisten der folgenden Aufgaben beziehen sich auf die Relationen der Beispieldatenbank *Bike*.

Aufgabe 1

Schreiben Sie eine HTML-Seite, die mehrere Formulareingaben anfordert (Eingabefeld, Checkbuttons, Radiobuttons). Durch Klick auf den Submit-Button werden diese Daten erfasst und tabellarisch ausgegeben. Verwenden Sie die Methode *post*.

Aufgabe 2

Schreiben Sie ein PHP-Programm, das eine Personalnummer einliest und dann Daten zu diesem Mitarbeiter ausgibt. Existiert dieser Mitarbeiter nicht, so erfolgt eine entsprechende Information.

Aufgabe 3

Ergänzen Sie Aufgabe 2 dahingehend, dass alle vorhandenen Personalnummern in einer Combobox vorgegeben werden. Nach Auswahl einer dieser Nummern werden dann Daten dieses Mitarbeiters ausgegeben.

Aufgabe 4

Fügen Sie neues Personal hinzu. Die Personalnummer soll dabei automatisch vergeben werden.

Aufgabe 5

Geben Sie alle Kunden und Lieferanten zusammen in alphabetischer Reihenfolge aus. Die Ausgabe habe exakt folgende Form (inklusive der Aufzählung und der Klammern!).

1. Biker Ecke (Kunde)
2. Fahrrad Shop (Kunde)
3. Fahrräder Hammerl (Kunde)
4. Firma Gerti Schmidtner (Lieferant) usw.

Aufgabe 6

Eine neue Lieferung eines Lieferanten ist eingetroffen. Schreiben Sie ein Programm, das zunächst alle Lieferanten in einer Combobox auflistet. Nach Auswahl eines Lieferanten werden alle Artikel (*ANr*, *Bezeichnung*, *Mass*, *Einheit*), die dieser Lieferant liefert, angezeigt. In einer zusätzlichen Eingabespalte wird zu jedem Artikel die Anzahl der Lieferungen angegeben. Das Programm erhöht dementsprechend den Bestand in der Relation *Lager*.

Da mehrere Daten geändert werden, können Verklemmungen eintreten. In diesem Fall werden alle Änderungen zurückgesetzt und bis zu zweimal wiederholt. Testen Sie diese Verklemmungen durch paralleles Ausführen von zwei Anwendungen, und indem zwischen den Änderungen mindestens 3 Sekunden gewartet wird. Mit einem Radiobutton wird darüber hinaus angegeben, ob die Artikel absteigend oder aufsteigend im Lager geändert werden, um Deadlocks provozieren zu können.

Aufgabe 7

Fügen Sie neues Personal hinzu. Zusätzlich zu Aufgabe 4 wird ein Bild angefordert. Personaldaten und Bild werden mit einem einzigen *Insert Into* Befehl eingefügt.

Aufgabe 8

Erweitern Sie die Datei *blob_ein2.php*, indem statt der Eingabe einer Personalnummer ein Mitarbeiter aus einer Select-Box ausgewählt werden muss. Diese Select-Box enthält alle derzeitigen Mitarbeiter der Relation *Personal*.

Aufgabe 9 (als Anregung)

Der Fantasie sind keine Grenzen gesetzt: Als kleine Anregung wäre es denkbar, alle Artikel mit Bildern zu versehen. Wir könnten einen kleinen Web-Shop erstellen, wo die Artikel mit Bild, Preis und Beschreibung angegeben werden.

Aufgabe 1

Worin besteht der Mehraufwand des kostenbasierten Optimizers gegenüber dem regelbasierten? Warum werden heute trotzdem kostenbasierte Optimizer eingesetzt?

Aufgabe 2

Der folgende Befehl gibt alle Auftragspositionen aus, die der Kunde *Fahrrad Shop* in Auftrag gegeben hat:

```
SELECT A.Bezeichnung, AP.Anzahl, Datum
FROM    Auftrag  NATURAL INNER JOIN Auftragsposten AP
                 INNER JOIN KUNDE K  ON Nr=Kundnr
                 INNER JOIN Artikel A ON Anr=Artnr
WHERE K.Name = 'Fahrrad Shop' ;
```

Wie ändert der Optimizer diesen Befehl ab? Geben Sie den geänderten Befehl an.

Aufgabe 3

Versehen Sie das Attribut *Bezeichnung* der Relation *Artikel* aus der Beispieldatenbank *Bike* mit einem Index. Ist ein eindeutiger Index mittels des Bezeichners *Unique* sinnvoll?

Aufgabe 4

In einem großen Dienstleistungsunternehmen werden zentral Aufträge entgegengenommen und an Sachbearbeiter weitergeleitet, deren Einsatzort in der Nähe des Kunden liegt. Dazu existiere in der Datenbank eine Relation *Personal*, die die benötigten Daten der Sachbearbeiter speichert, insbesondere auch den Namen und den Einsatzort. Wegen einer großen Anzahl von Sachbearbeitern ist die Suche nach möglichen Sachbearbeitern in der Nähe des Kunden zeitaufwendig. Schreiben Sie einen Index, um die alphabetische Suche nach Sachbearbeitern an bestimmten Einsatzorten zu beschleunigen.

Aufgabe 5

Die Range-Partitionierung ist sehr weit verbreitet. Welche zwei großen Vorteile bietet diese Partitionierung bei sehr großen Relationen?

Worin liegt der Vorteil einer Hash-Partitionierung?

Aufgabe 6

Was ist der große Vorteil der Referenzpartitionierung? Wie könnte diese Partitionierung in SQL Server und MySQL nachgebildet werden?

Aufgabe 7

Materialisierte Sichten müssen aktuell gehalten werden. Dies bedingt einen hohen Aufwand. Wo werden daher diese Sichten fast ausschließlich eingesetzt und warum?

Aufgabe 8

Der Optimizer änderte den Befehl in Aufgabe 2 ab. Warum wohl?

Aufgabe 9

Wann ist ein Merge-Join gegenüber Nested-Loop-Join und Hash-Join von Vorteil?

Aufgabe 10

Warum beeinflusst eine Group-By-Klausel die Performance so negativ? Warum kann in der Regel ein existierender Index auf das zu gruppierende Attribut nicht verwendet werden?

Aufgabe 11

Schreiben Sie eine Stored Procedure *Preisnachlass*, die den Preis aller Artikel um den als Parameter angegebenen Prozentsatz senkt. Dies gilt auch für die Preise aller Aufträge. Die Prozedur überprüft, dass der Nachlass nicht größer als 10 Prozentpunkte beträgt.

Aufgabe 12

Unsere Firma Bike hätte einen sehr großen Lieferantenstamm. Gesucht werden häufig Lieferanten, die bestimmte Artikel liefern. Nacheinander sollen dann zu einer Liste von Artikeln die entsprechenden Lieferanten angezeigt werden. Wie könnte ein dazugehöriger PHP-Programmausschnitt aussehen?

Übungsaufgaben zu Kapitel 8

Edwin Schicker

Aufgabe 1

In einem sicheren Datenbankbetrieb wurde gerade das Transaktionsende in die Logdatei geschrieben. Doch noch vor der Rückmeldung der im Prinzip beendeten Transaktion an den Benutzer stürzt das ganze System ab. Wird beim nächsten Hochfahren die Transaktion deshalb zurückgesetzt? Begründen Sie Ihre Antwort!

Aufgabe 2

Was sind Checkpoints? Beschreiben Sie weiter den Nachteil, wenn eine Datenbank ohne Checkpoints arbeiten würde.

Aufgabe 3

In einigen einfachen Datenbankanwendungen ist die Recovery-Unterstützung deaktiviert oder zumindest stark eingeschränkt. Woran liegt das?

Aufgabe 4

Welche Schritte müssen vom Systemadministrator bzw. vom Datenbankverwaltungssystem im Einzelnen durchgeführt werden, wenn eine einzelne Transaktion wegen eines Software-Fehlers abstürzt?

Aufgabe 5

Der Systemverwalter bemerkt im laufenden Datenbankbetrieb, dass die Festplatte, auf der die Daten der Datenbank gespeichert sind, nicht mehr fehlerfrei arbeitet (Schreibfehler). Welche Maßnahmen müssen im Einzelnen ergriffen werden, um mögliche fehlerbehaftete Schreibvorgänge seit der letzten Sicherung zu eliminieren?

Aufgabe 6

Unter welchen Voraussetzungen kann auf ein Before-Image verzichtet werden?

Aufgabe 7

Unter welchen Voraussetzungen kann auf ein After-Image verzichtet werden?

Aufgabe 8

Warum wird bei den Logdaten zwischen Undo-Log und Redo-Log unterschieden?

Aufgabe 9

Wie erkennt man Deadlocks? Wie beseitigt man sie, ohne die Konsistenz der Datenbank zu zerstören?

Aufgabe 10

Im Parallelbetrieb kann man auch ohne Sperrmechanismen auskommen. Um welches Verfahren handelt es sich, und warum wird es kaum eingesetzt?

Aufgabe 11

In einem kleinen Mehrbenutzer-Datenbankverwaltungssystem existiere als Sperrmechanismus nur ein einziger Lock (globaler Datenbanklock). Kann in diesem System ein Deadlock entstehen?

Aufgabe 12

Im Parallelbetrieb ist es nicht immer leicht zu sagen, ob eine Transaktion *A* vor oder nach einer Transaktion *B* ablief. Kann dieses Problem immer entschieden werden, wenn wir mit Sperrmechanismen arbeiten?

Aufgabe 13

Betrachten wir das Problem der Inkonsistenz der Daten und deren Lösung mittels Locks in der folgenden Abbildung. Laufen wir ebenfalls in einen Deadlock, wenn wir in Transaktion *TA1* zunächst Konto 3, dann Konto 2 und Konto 1 abgefragt hätten? Wäre dann ohne Locks das Problem der Inkonsistenz der Daten überhaupt aufgetreten?

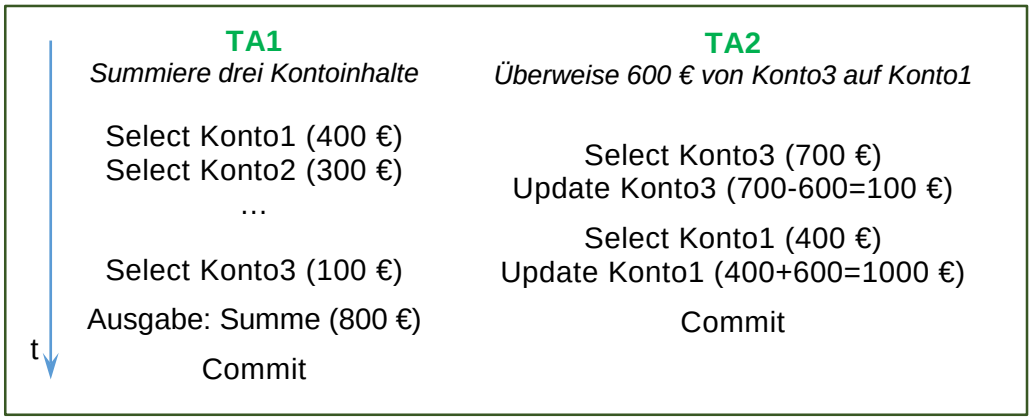


Abbildung: Problem der Inkonsistenz

Übungsaufgaben zu Kapitel 9

Edwin Schicker

Aufgabe 1

Was wird unter Unabhängigkeit gegenüber Fragmentierung und gegenüber Replikation verstanden?

Aufgabe 2

Definieren Sie den Begriff *Verteilte Datenbanken*!

Aufgabe 3

Wir haben globale Deadlocks in verteilten Datenbanken kennengelernt. Können in verteilten Datenbanken auch lokale Deadlocks auftreten?

Aufgabe 4

Ist unter Verwendung eines Zwei-Phasen-Commit-Protokolls die Regel 2 von Date erfüllbar, obwohl für dieses Protokoll ein zentraler Koordinator benötigt wird?

Aufgabe 5

Kennen Sie eine verteilte Datenbank aus der Praxis? Wenn ja, welche der zwölf Regeln verletzt sie? Welche Nachteile werden deshalb in Kauf genommen? Wie ist diese Datenbank bezüglich des CAP-Theorems einzuordnen?

Aufgabe 6

Wie hängen das CAP-Theorem und das Konsistenzmodell *BASE* zusammen?

Aufgabe 7

Wie löst SQL3 das rekursive Stücklistenproblem?

Aufgabe 8

Wie werden in SQL3 NF^2-Relationen definiert?

Aufgabe 9

Schreiben Sie ein Objekt *TPerson*, das den Namen, die Adresse und das Geburtsdatum einer Person enthält. Greifen Sie auf das Objekt *TAdresse* zurück. Schreiben Sie eine Relation *PersonalNeu*, die das Objekt *TPerson* verwendet.

Aufgabe 10

Schreiben Sie einen Select-Befehl, der aus der Relation *AuftragNeu* alle Einzelpositionen zu Auftrag 4 ausgibt, falls das Attribut *Anzahl* größer als 1 ist.

Aufgabe 11

Wann wird ein Zwei-Phasen-Commit benötigt?

Aufgabe 12

Was macht einen Zwei-Phasen-Commit Ihrer Meinung nach so extrem zeitaufwendig?

Aufgabe 13

Kopieren Sie mit einem einzigen Insert-Befehl alle Daten aus den Relationen *Auftrag* und *Auftragsposten* in die Relation *AuftragNeu*.