Teil III

Von Datenbanken und ihren Systemen

Robert Hartmann (SoSe 2024)

basierend auf Folien von
Prof. Dr. Harm Knolle
und
Prof. Dr. Sascha Alda

Fachbereich Informatik
Hochschule Bonn-Rhein-Sieg

- Kapitel 7 - Datenbanksysteme -

Inhalt

Inhalt

- Datenbanksysteme -

Ziel

- Aus welchen Komponenten besteht ein Datenbanksystem?
- Wie wird ein Auftrag vom Datenbanksystem ausgeführt?
- Welche Hilfsmittel stehen dem Datenbanksystem zur Verfügung?

Hilfsmittel

- Die Ideen von CODD
- Architektur, Komponenten und Funktionsweise kommerzieller Datenbanksysteme

Literatur

- KeEi15 (nur teilweise)
  - Kapitel 1 „Einleitung und Überblick“
    - Abschnitt 1.7
  - Kapitel 7 „Physische Datenorganisation“
    - bis einschl. 7.1
    - 7.4 - 7.5
  - Kapitel 8 „Anfragebearbeitung“
    - Einleitung von Kapitel 8

Inhalt

- Die zwölf CODD’schen Regeln
- Datenbankmanagementsystem
- Data Communication System
- Data Dictionary System
- Utilities (System Support Routines)
- Exkurs: Datenbanken in Java (JDBC)

- Ku15 (nur teilweise)
  - Kapitel 1: „Grundlagen und Überblick“
    - Abschnitt 1.4
  - Kapitel 7: „Komponenten eines Datenbankmanagementsystems“
    - bis einschl. 7.3
- SSH18 (nur teilweise)
  - Kapitel 2: „Datenbanksystemkonzepte und Architektur“
    - bis einschl. 2.4

- Die zwölf CODD'schen Regeln -

Inhalt

- Die zwölf CODD'schen Regeln
- Datenbankmanagementsystem
- Data Communication System
- Data Dictionary System
- Utilities (System Support Routines)
- Exkurs: Datenbanken in Java

Überblick

- Anforderungen an relationale DBMS
- Erläuterung der Regeln
- Interpretation

Die zwölf CODD'schen Regeln

- Anforderungen an relationale DBMS -

Anmerkungen

- 1985 wurden von E. F. Codd zwölf Regeln veröffentlicht, die eine relationale Datenbank im strengen Sinne definieren
- die zwölf Regeln dürfen bei der Beurteilung eines Datenbanksystems nicht gleich gewichtet werden
- viele Hersteller bezeichnen ihre Datenbank bereits als relational, wenn sie nur einige der wichtigsten Kriterien erfüllt
- die Erfüllung dieser einfachen Bedingungen genügt jedoch noch nicht, ein Datenbanksystem als relational zu klassifizieren

Vollständigkeit

- Ende 1990 veröffentlichte E. F. Codd ein Buch über relationale Datenbanken, in dem er die einstigen zwölf Regeln des relationalen Modells auf 333 Regeln differenziert
  - E. F. Codd, The Relational Model for Database Management - Version 2, Addison-Wesley 1990
- dadurch wird die Relationalität einer Datenbank bis ins letzte Detail festgeschrieben

Die zwölf CODD'schen Regeln

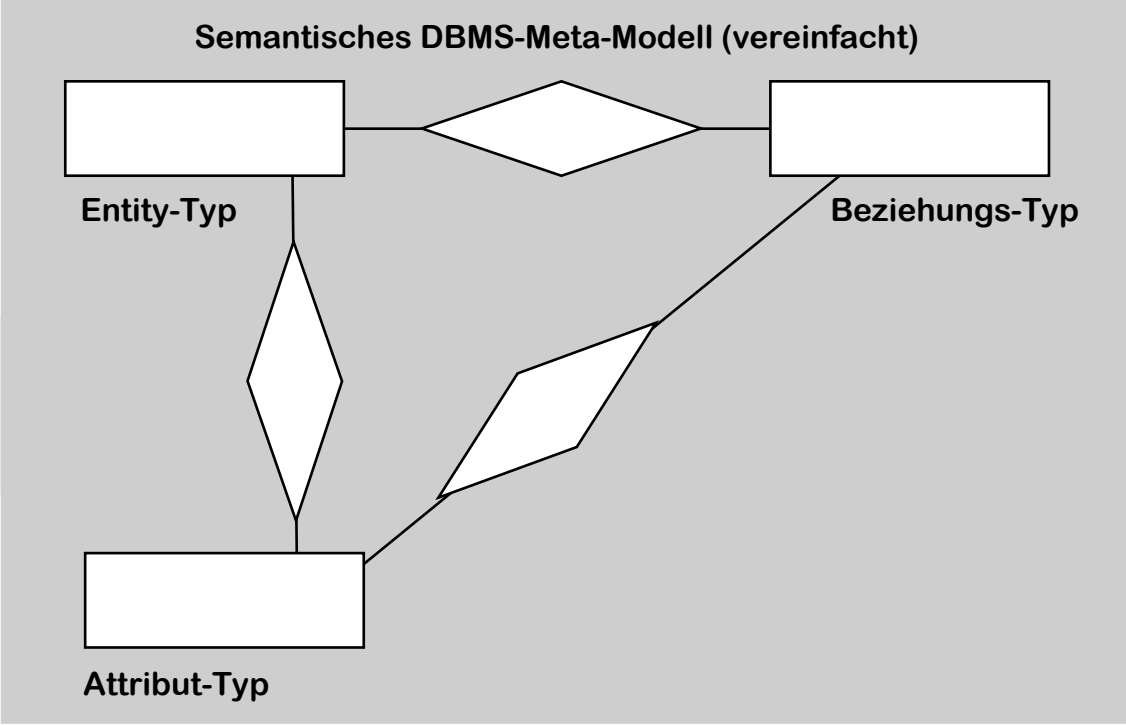
- Regel 0 für relationale DBMS - Das DBMS-Meta-Modell -

Die 12 CODD'schen Regeln basieren auf einer nicht ausgesprochenen Regel 0

- ein relationales DBMS muss fähig sein, Datenbanken vollständig und eigenständig durch seine relationalen Fähigkeiten zu verwalten

Die Konsequenz

- Tabellen über Tabellen
- Tabellen über andere Datenbank-Komponenten



Die zwölf CODD'schen Regeln

- Erläuterung der Regeln (I) -

Regel 1: Darstellung von Information

- sämtliche Informationen in relationalen Datenbanken muss logisch in Tabellen dargestellt sein, insbesondere
  - Daten
  - Definitionen von Tabellen und Attributen
  - Integritätsbedingungen und die Aktion bei deren Verletzung
    - siehe auch Regel 10
  - Sicherheitsinformationen
    - z. B. Zugangsberechtigungen

Regel 2: Zugriff auf Daten

- jeder Wert einer relationalen Datenbank muss logisch durch Kombination der Ausprägung folgender Strukturen auffindbar sein
  - Tabellenname
  - Primärschlüssel
  - Attributname (Spaltenname)
- dies bedeutet, dass in einer Tabelle an jedem Schnittpunkt einer Zeile mit einer Spalte nur ein Wert stehen darf, z.B.
  - Tabellenname: Angestellte
    Primärschlüssel: Angestellter = Meier
    Attributname: Gehalt = 5300

Die zwölf CODD'schen Regeln

- Erläuterung der Regeln (II) -

Regel 3: Systematische Behandlung von Nullwerten

- Nullwerte stellen in Attributen, die nicht Teil eines Primärschlüssels sind, fehlende Information dar und werden durchgängig gleich, insbesondere unabhängig vom Datentyp des Attributes, behandelt
- fehlende Informationen werden heute meist mit NULL bezeichnet
- Beispiel
  - man kann also nicht in numerischen Feldern bei fehlenden Daten das Feld einfach leer lassen
  - oder bei Textfeldern das Zeichen '-' einfügen

Regel 4: Struktur einer Datenbank

- die Datenbankstruktur wird in derselben logischen Struktur wie die Daten gespeichert, also in Tabellen
- dazu muss die Struktur aller Tabellen, die zu einer Datenbank gehören, in einer Tabelle (dem Katalog) zugänglich sein
- diese Forderung bedingt, dass sich eine Änderung im Katalog automatisch in einer geänderten Datenbankstruktur auswirkt

Die zwölf CODD'schen Regeln

- Erläuterung der Regeln (III) -

Regel 5: Die Abfragesprache

- ein relationales System enthält mindestens eine befehlsgesteuerte Abfragesprache, die mindestens die folgenden Funktionen unterstützt:
  - Datendefinition
  - Definition von Views
  - Definition von Integritätsbedingungen
  - Definition von Transaktionen
    - eine Transaktion ist eine Folge von Befehlen, die eine Datenbank von einem konsistenten Zustand in einen anderen überführt
    - sie muss entweder vollständig durchgeführt oder, bei einem Abbruch, vollständig zurückgesetzt werden
  - Definition von Berechtigungen
- eine weit verbreitete Abfragesprache ist SQL

Regel 6: Aktualisieren von Views

- alle Views, die theoretisch aktualisiert werden können, können auch vom System aktualisiert werden
- im Allgemeinen kann jedoch nicht entschieden werden, ob eine View theoretisch aktualisiert werden kann
- Beispiel
  - hat man zwei Spalten A und B mit Zahlen, so kann man sich eine weitere Spalte definieren, die A*B enthält
  - ändert man einen Wert in dieser Spalte, so kann daraus im allgemeinen nicht der Wert der Spalten A und B bestimmt werden
  - diese View kann also theoretisch nicht aktualisiert werden

Die zwölf CODD'schen Regeln

- Erläuterung der Regeln (IV) -

Regel 7: Abfragen und Editieren ganzer Tabellen

- Abfrage- und Editieroperationen müssen als Operanden ganze Tabellen und nicht nur einzelne Sätze erlauben
- Mengenorientierung der Sprache

Regel 8: Physikalische Unabhängigkeit

- der Zugriff auf die Daten durch den Benutzer muss unabhängig davon sein, wie die Daten gespeichert werden oder wie physikalisch auf sie zugegriffen wird
- dies bedeutet, dass Anwendungen nur auf die logische Struktur des Systems zugreifen dürfen
- Beispiel
  - die Daten dürfen auf einem Datenträger durchaus hierarchisch gespeichert sein
  - nur die logische Struktur der Datenbank muss relational sein
  - ändert die Administration des DBMS die physikalische Struktur der Datenbank, darf der Anwender davon nichts mitbekommen

Die zwölf CODD'schen Regeln

- Erläuterung der Regeln (V) -

Regel 9: Logische Unabhängigkeit der Daten

- Anwendungen und Zugriffe dürfen sich logisch nicht ändern, wenn Tabellen so geändert werden, dass alle Informationen erhalten bleiben
  - z. B. beim Aufspalten einer Tabelle in zwei Tabellen

Regel 10: Integritätsregeln

- alle Integritätsbedingungen müssen in der Abfragesprache definierbar sein und in Tabellen dargestellt werden
- das System muss mindestens die folgenden Integritätsbedingungen prüfen:
  - Vollständigkeitsintegrität (Entity Integrity, Existential Integrity)
    - ein Primärschlüssel muss eindeutig sein und darf insbesondere keinen Nullwert enthalten
  - Beziehungsintegrität (Referentielle Integrität, Referential Integrity)
    - zu jedem Fremdschlüssel existiert ein Primärschlüssel
- Beispiel
  - zu jeder verwendeten Personalnummer muss es auch einen Haupteintrag mit den Daten eines Angestellten geben

Die zwölf CODD'schen Regeln

- Erläuterung der Regeln (VI) -

Regel 11: Verteilung der Daten

- Anwendungen für eine nicht-verteilte Datenbank dürfen sich beim Übergang zu einer verteilten Datenbank logisch nicht ändern
- Beispiel
  - wenn die oben beschriebenen Tabellen in einem Netzwerk auf zwei verschiedenen Rechnern gespeichert sind, darf sich bei der Anwendung nichts ändern, wenn die Tabellen irgendwann auf demselben Rechnern gespeichert werden

Regel 12: Unterlaufen der Abfragesprache

- unterstützt ein relationales Datenbanksystem neben der High-Level-Abfragesprache eine Low-Level-Abfragesprache, so darf diese die Integritätsbedingungen der High-Level-Sprache nicht unterlaufen
- Beispiel
  - die Low-Level Abfragesprache darf z. B. nicht direkt auf die physikalischen Eigenschaften der gespeicherten Daten zugreifen
  - Im Oracle DBMS ist SQL die High-Level-Sprache; und PL/SQL (Procedural Language/Structured Query Language) die Low-Level-Sprache.
    - PL/SQL ist Low-Level, weil es SQL-Anweisungen mit prozeduralen Konstrukten wie Unterprogrammen mischt.

Die zwölf CODD'schen Regeln

- Interpretation -

Interpretation

- die Regeln 1 bis 5 sowie 7 und werden im Prinzip von den diversen SQL-Dialekten eingehalten
  - Darstellung von Information
  - Zugriff auf Daten
  - Systematische Behandlung von Nullwerten
  - Struktur einer Datenbank
  - Die Abfragesprache
  - Abfragen und Editieren ganzer Tabellen
  - Physikalische Unabhängigkeit
- insgesamt sind, wenn man das heutige Produktspektrum betrachtet, vor allem die Regel 6, 10 und 11 bei der Einschätzung von großer Bedeutung
  - Aktualisieren von Views
  - Integritätsregeln
  - Verteilung der Daten
- Regel 9 fällt mit 6 zusammen
  - Logische Unabhängigkeit der Daten
  - Aktualisieren von Views
- die Regel 12 schließlich kennzeichnet einen Spezialfall, nämlich ein um ein relationales Interface erweitertes konventionelles (nicht relationales) DBMS
  - Unterlaufen der Abfragesprache

- Datenbankmanagementsystem (DBMS) -

Inhalt

- Die zwölf CODD'schen Regeln
- Datenbankmanagementsystem
- Data Communication System
- Data Dictionary System
- Utilities (System Support Routines)
- Exkurs: Datenbanken in Java

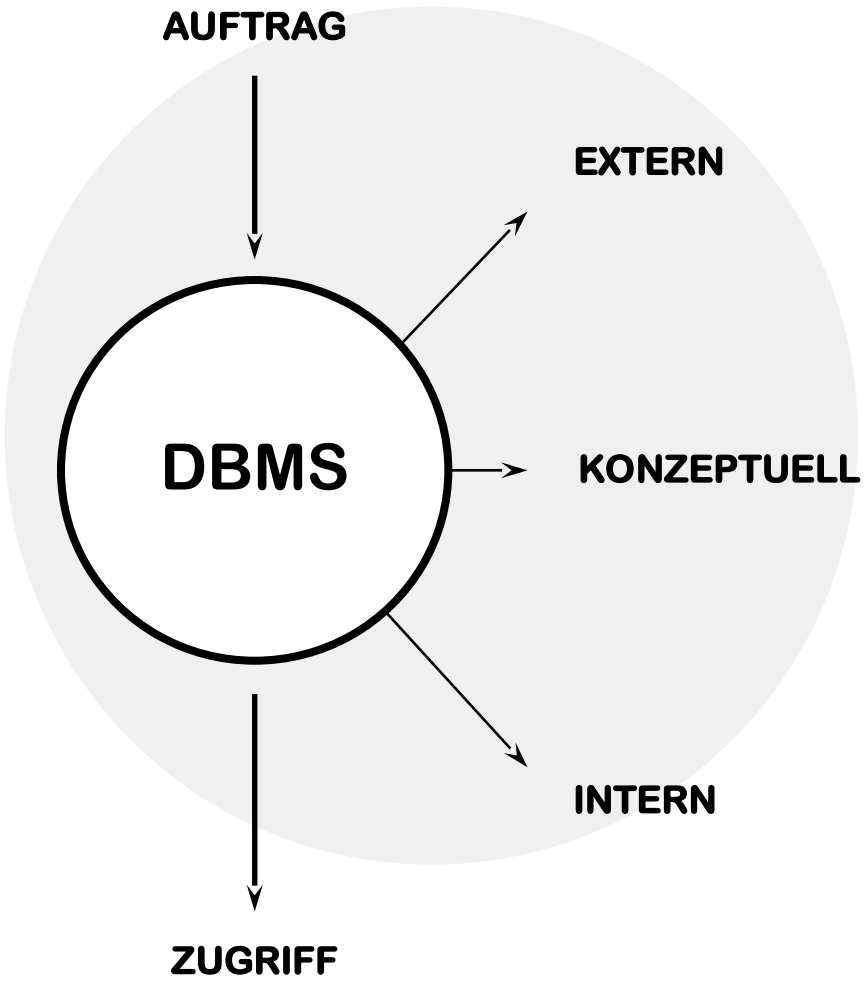
Überblick

- Aufgaben des DBMS
- Schematischer Auftragsablauf
- Zusammenfassung der Auftragsabarbeitung
- Aufgaben des Bindens
- Datenunabhängigkeit und Binden
- Weitere Aufgaben des DBMS

- Aufgaben des DBMS (I) -

Kapselung des Datenbestandes (Datenbank)

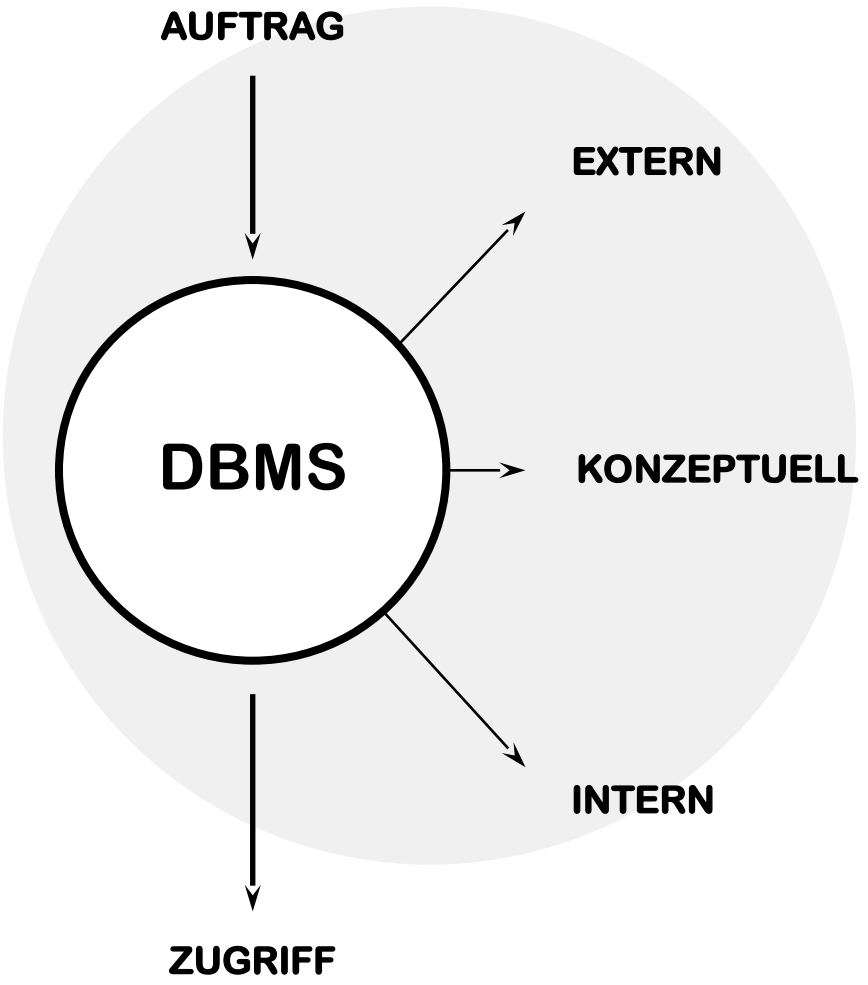
- gegen die Zugriffe der Anwendungen
- Bereitstellung von Operationen zum Datenzugriff und zur Datenveränderung
- Grundlage ist die 3-Schema Architektur nach ANSI/X3/SPARC (siehe Kapitel 2)
  - Externe Ebene
  - Konzeptuelle Ebene
  - Interne Ebene



- Aufgaben des DBMS (II) -

Interpretationsvorschriften zur Durchführung der Aufträge sind

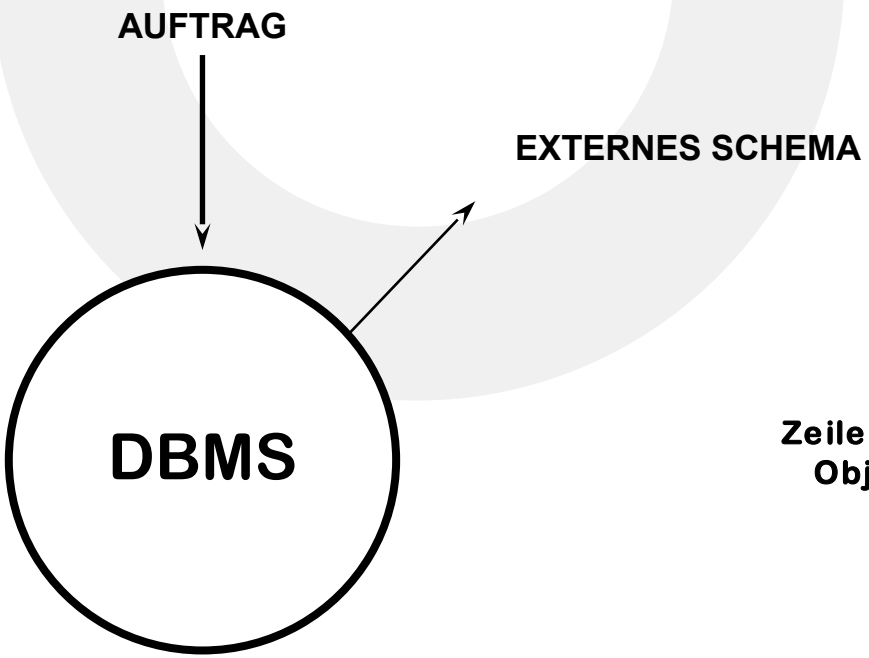
- die drei Ebenen der Datenbank
  - Externe Ebene
  - Konzeptuelle Ebene
  - Interne Ebene
- deren Datenbankmodelle
  - SQL bei relationalen Datenbanksystemen
- deren Schemata
  - konkrete Sichtenstrukturen
  - konkrete Tabellenstrukturen
  - konkrete Speicherstrukturen
  - und den Transformationsregeln zwischen den Ebenen



- Schematischer Auftragsablauf (I) -

Schritte 1 und 2

- das DBMS empfängt von einem Anwendungsprogramm den Auftrag, bestimmte Objekte des EXTERNEN SCHEMAS (Views) zu lesen
  - z.B. die Veranstaltungen, die in Gebäude B stattfinden



- das DBMS holt sich die benötigten Definitionen des entsprechenden Objekttyps aus dem zur Anwendung gehörenden EXTERNEN SCHEMA

Externes Objekt "Lehrort"

LV-Name	Ge-baeude	Raum	Straße
Datenbanken	B	431	Am Schwimmbad
Englisch I	D	123	Blechhammer
Analysis	B	23	Am Schwimmbad
Systemanalyse	C	231	Blechhammer
WWS I	C	250	Blechhammer

Zeilen aus externem Objekt "Lehrort"

LV-Name	Ge-baeude	Raum	Straße
Datenbanken	B	431	Am Schwimmbad
Analysis	B	23	Am Schwimmbad

- Schematischer Auftragsablauf (II) -

Schritt 3

- mit Hilfe der Transformationsregeln EXTERNES / KONZEPTUELLES SCHEMA stellt das DBMS fest, welche konzeptuellen Objekte und Beziehungen benötigt werden

konzeptuelles Objekt "Veranstaltung"

LV-Nr	LV-Name	Ge-baeude	Raum
1238	Datenbanken	B	431
1153	Englisch I	D	123
1352	Analysis	B	23
1543	Systemanalyse	C	231
1421	WWS I	C	250

TRANSFORMATIONSREGELN EXTERN I KONZEPTUELL

Zeilen aus externem Objekt "Lehrort"

LV-Name	Ge-baeude	Raum	Straße
Datenbanken	B	431	Am Schwimmbad
Analysis	B	23	Am Schwimmbad

konzeptuelles Objekt "Gebäude"

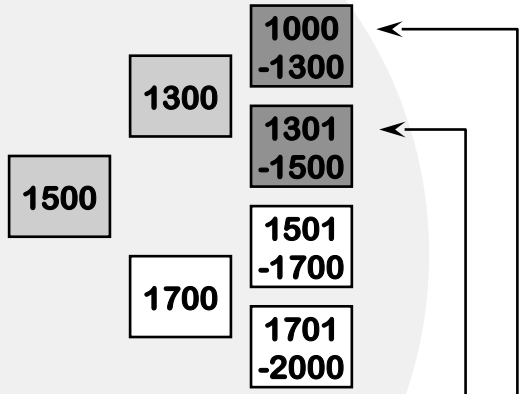
Ge-baeude	Straße
B	Am Schwimmbad
C	Blechhammer
D	Blechhammer

- Schematischer Auftragsablauf (III) -

Schritt 4

- mit Hilfe der Transformationsregeln KONZEPTUELLES / INTERNES SCHEMA stellt das DBMS fest, welche physischen Objekte und Beziehungen zu lesen und welche Zugriffspfade (z.B. Indexe) zu verwenden sind

internes DB-Objekt “Veranstaltung”
z.B. baumartig strukturiert
(vereinfachte Darstellung,
da Suche über LV-Name erfolgen müsste)



Zeilen des konzeptuellen Objekts "Veranstaltung"

LV-Nr	LV-Name	Ge-baeude	Raum
1238	Datenbanken	B	431
1352	Analysis	B	23

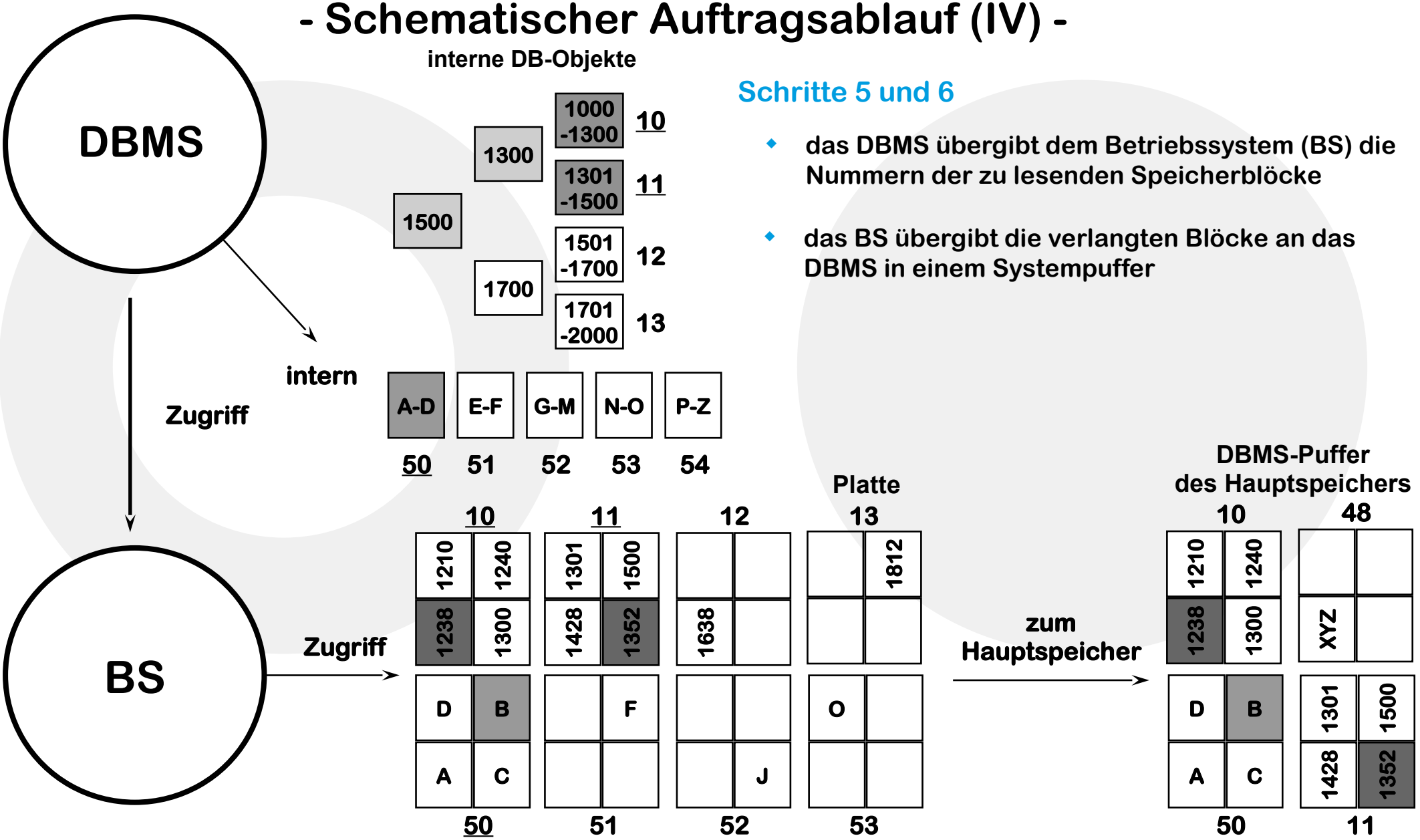
TRANSFORMATIONS
-
REGELN
KONZEPTUELL
/ INTERN

Zeilen aus konzeptuellem Obiekt "Gebäude"

Ge-baeude	Straße
B	Am Schwimmbad

internes DB-Objekt “Gebäude”
z.B. sequentiell strukturiert

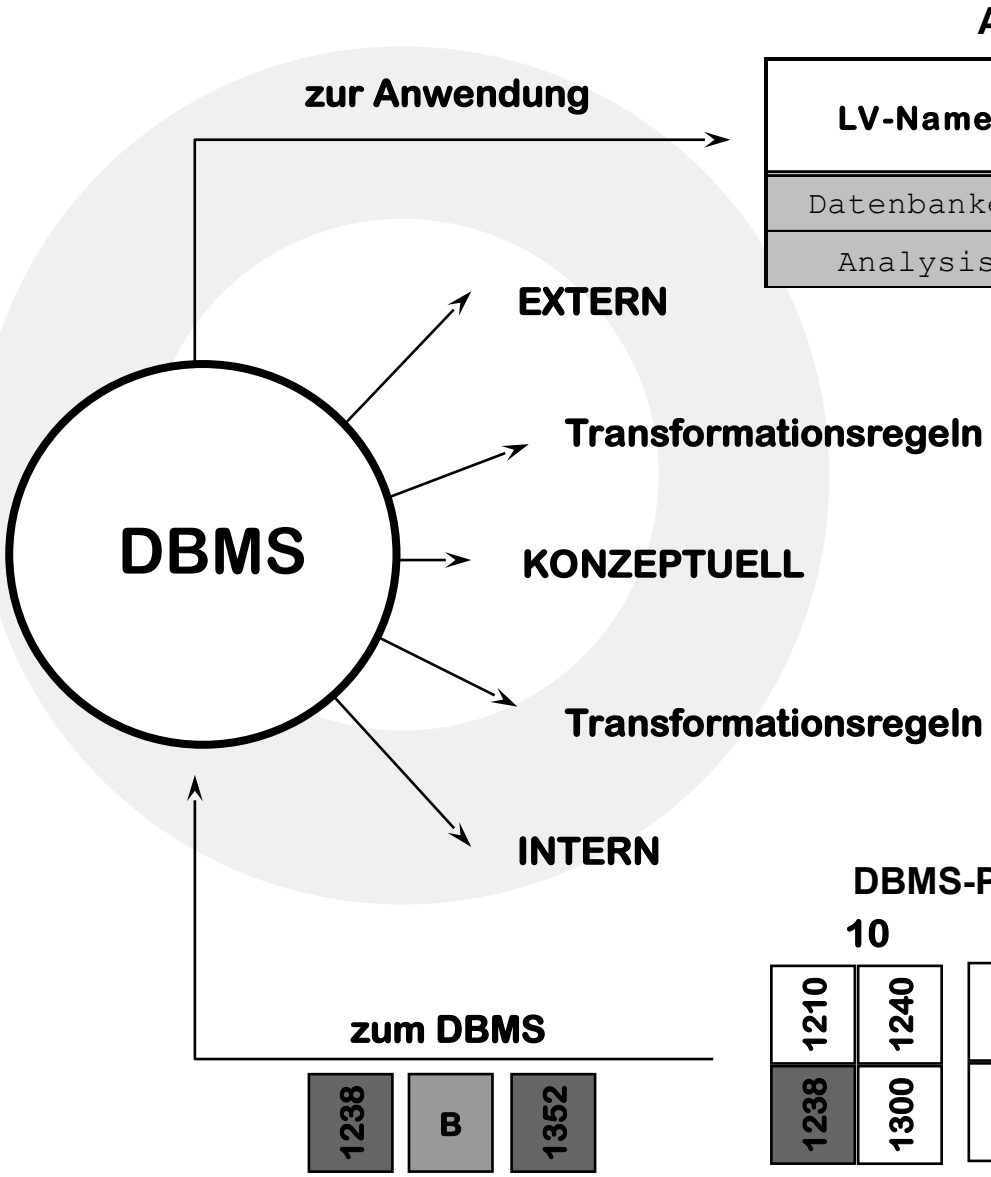
- Schematischer Auftragsablauf (IV) -



Schritte 5 und 6

- das DBMS übergibt dem Betriebssystem (BS) die Nummern der zu lesenden Speicherblöcke
- das BS übergibt die verlangten Blöcke an das DBMS in einem Systempuffer

- Schematischer Auftragsablauf (V) -



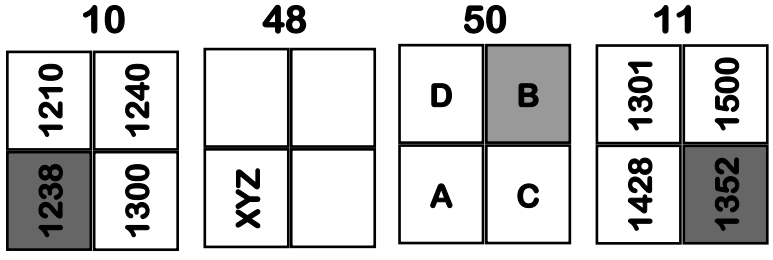
Arbeitsspeicher der Anwendung

LV-Name	Ge-baeude	Raum	Straße
Datenbanken	B	431	Am Schwimmbad
Analysis	B	23	Am Schwimmbad

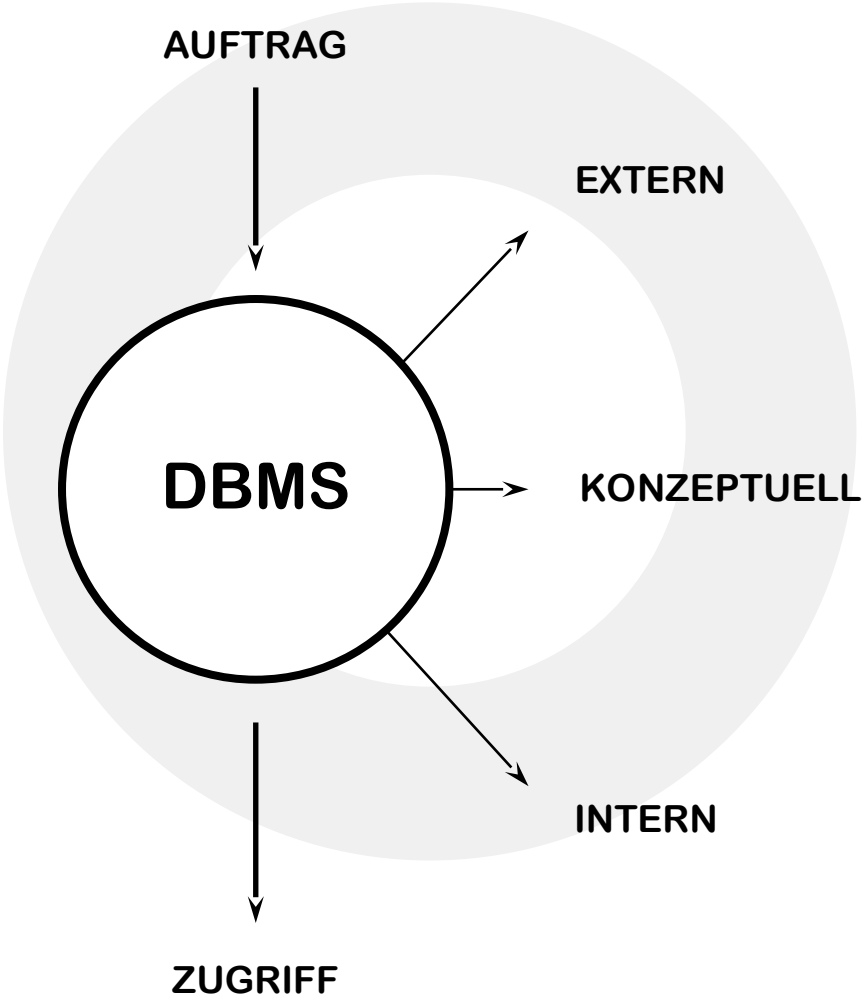
Schritte 7, 8 und 9

- mit Hilfe der Transformationsregeln stellt das DBMS aus den physischen Datensätzen das verlangte EXTERNE Objekt zusammen
- das DBMS übergibt das externe Objekt dem Anwendungsprogramm in seinem Arbeitsspeicher
- das Anwendungsprogramm verarbeitet die vom DBMS übergebenen Daten

DBMS-Puffer des Hauptspeichers



- Zusammenfassung der Auftragsabarbeitung -



Zusammenfassung

- Auftrag an das DBMS
- die Anwendung benötigt bestimmte Zeilen einer externen Anwendungssicht (View)
- das DBMS ermittelt die beteiligten konzeptuellen Objekte (Tabellen)
- das DBMS ermittelt
  - die beteiligten internen Datensätze
  - die Zugriffsmethoden
  - die vom Betriebssystem zum Zugriff benötigten Daten
- das DBMS setzt das externe Objekt zusammen und übergibt es dem Anwendungsprogramm
- eine Weiterverarbeitung und Interpretation der Daten ist nun ausschließlich Sache des Anwendungsprogrammes

- Aufgaben des Bindens -

Ziel: Feste Anbindung von Daten an Anwendungsprogramm

Definition

- sind alle Befehle / Objekte, die sich auf das externe Modell beziehen,
- durch Befehle / Objekte ersetzt, die sich auf das konzeptuelle Modell beziehen
- und sind alle Befehle / Objekte, die sich auf das konzeptuelle Modell beziehen,
- durch Befehle / Objekte ersetzt, die sich auf das interne Modell beziehen,
- dann sind die entsprechenden Daten an das Anwendungsprogramm gebunden

Bedeutung

- Festlegen des Verarbeitungsganges vom Auftrag bis zur Übergabe des Auftragsergebnisses
- Festlegen der Zugriffsmechanismen für den Auftrag

Varianten

- Binden zur Übersetzungszeit einer Anwendung
  - schnell und preiswert (im Sinne von Kosten pro Operation)
  - Compiler & Linker
- Binden zum Zugriffszeitpunkt
  - kostspieliger aber ungleich flexibler
  - Interpreter

- Datenunabhängigkeit und Binden -

Physische Datenunabhängigkeit

- Isolierung der Anwendungen von der physischen Datenorganisation
  - Änderungen in der Speicher- oder Zugriffsorganisation betreffen NICHT die Anwendungen
- erreicht durch Transformationsregeln

Logische Datenunabhängigkeit

- Isolierung der Anwendungen von der konzeptuellen Ebene der Modellierung
  - Änderungen im konzeptuellen Modell betreffen NICHT die Anwendungen
- erreicht durch Transformationsregeln

Statische Datenunabhängigkeit

- Anwendung muss bei Änderungen im konzeptuellen oder internen Modell neu übersetzt und gebunden werden
- erreicht durch Binden zur Übersetzungszeit

Dynamische Datenunabhängigkeit

- Anwendungen sind von allen Änderungen vollkommen unabhängig
- erreicht durch Binden zur Zugriffszeit

- Weitere Aufgaben des DBMS -

Gewährleistung der Daten-Integrität

- Verhütung falscher Eingaben
  - z.B. Typprüfungen bei Dateneingaben durchführen
- Verhütung von Datenverlusten
  - z.B. nach Systemzusammenbruch oder Plattenfehlern
- Koordination konkurrierender Datenzugriffe durch mehrere Benutzer
  - z.B. Benutzer A liest Objekt X während Benutzer B das Objekt X gleichzeitig ändert

Datenschutz

- Zugangskontrollen
  - z.B. Passwortschutz
- Sichtbarkeitskontrollen
  - z.B. Kontrolle benutzervergebener Zugriffsrechte auf Daten

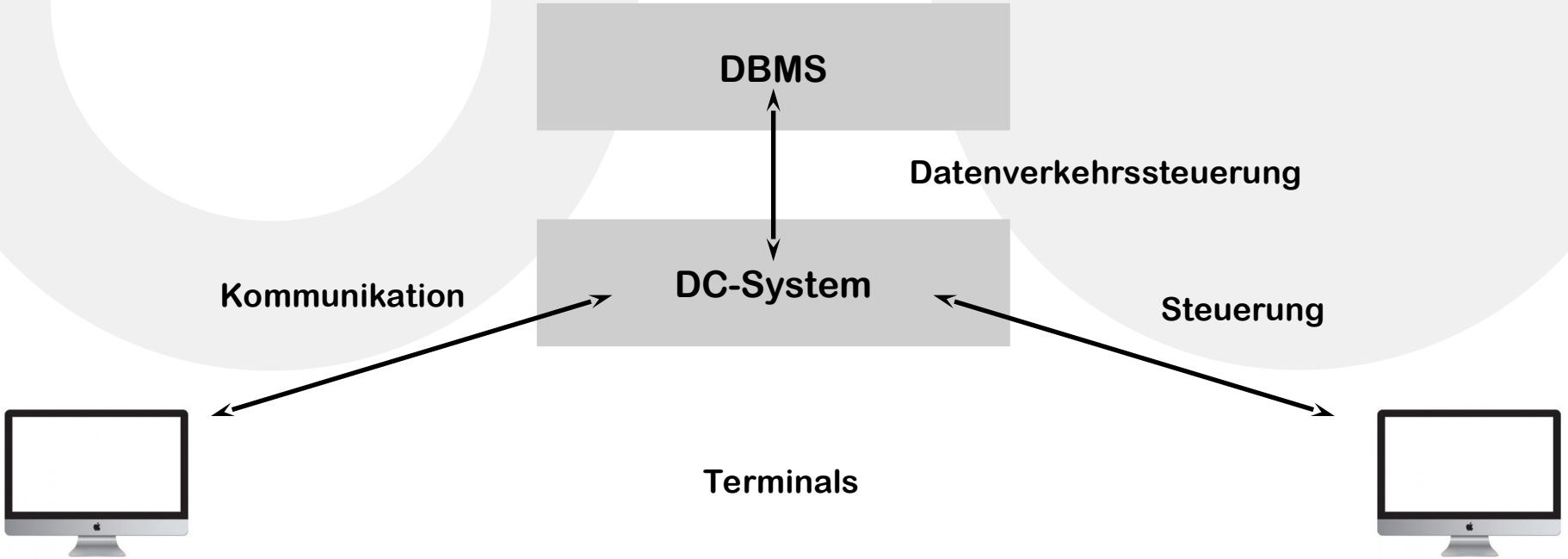
- Aufgaben des DC-Systems -

Übernimmt die Kommunikation mit lokalen oder entfernten Terminals bzw. Clients (z.B.)

- Nachrichtenempfang
- Programmaufruf
- Transaktionssteuerung

Effiziente Steuerung vieler Terminals und hohen Datenaufkommens (z.B.)

- Gewährleistung von Terminalunabhängigkeit
- Formatierungsaufgaben für die Terminals
- Prioritätensteuerung



- Data Dictionary System -

Inhalt

- Die zwölf CODD'schen Regeln
- Datenbankmanagementsystem
- Data Communication System
- Data Dictionary System
- Utilities (System Support Routines)
- Exkurs: Datenbanken in Java - JDBC

Überblick

- Aufgaben des DD-Systems
- Inhalte eines DD-Systems

- Aufgaben des DD-Systems -

Funktion

- Inhaltsverzeichnis
- ein zentraler, für alle Anwender und Programme verbindlicher Katalog
- Beschreibung der Daten
- Angaben zu Beziehungen zwischen den Daten
- Angaben über Zugriffsrechte
- Angaben, welche Programme welche Daten nutzen
- Konsistenzbedingungen
- Beschreibung von Transaktionen (Operationen auf Daten)
- DD-Systeme waren früher als separate Programmpakete erhältlich, in heutigen Systemen aber bereits integriert

- Inhalte eines DD-Systems -

Tabellen oder Sichten des DD-Systems

- **mögliche DD-Inhalte, die mit speziellen DD-Befehlen abgefragt werden können**

Benutzer

Nr	Name
321	Dillo
815	Grau
4711	Schmidt

Rechte

Tabelle	Benutzer	Recht
Fachbereich	4711	RW
Fachbereich	321	R
Fachbereich	815	
Gebaeude	4711	RW
Gebaeude	321	R
Gebaeude	815	

Tabellen-struktur

Tabelle	Attribut	Format
Fachbereich	FB_Nr	numeric(2)
Fachbereich	FB_Name	varchar(30)
Fachbereich	Dekan	char(10)
Gebaeude	Gebaeude	char(5)
Gebaeude	Strasse	varchar(30)
Gebaeude	Nummer	char(5)

- Utilities (System Support Routines) -

Das Angebot an Utilities ist bei kommerziellen Datenbanksystemen sehr unterschiedlich (z.B.)

- Laderoutinen (zum erstmaligen Laden der Datenbank)
- Statistikroutinen
- Fehleranalyse
- Reorganisationsroutinen
- Kopierroutinen
- Archivierungsroutinen

- Datenbanken in Java -

Inhalt

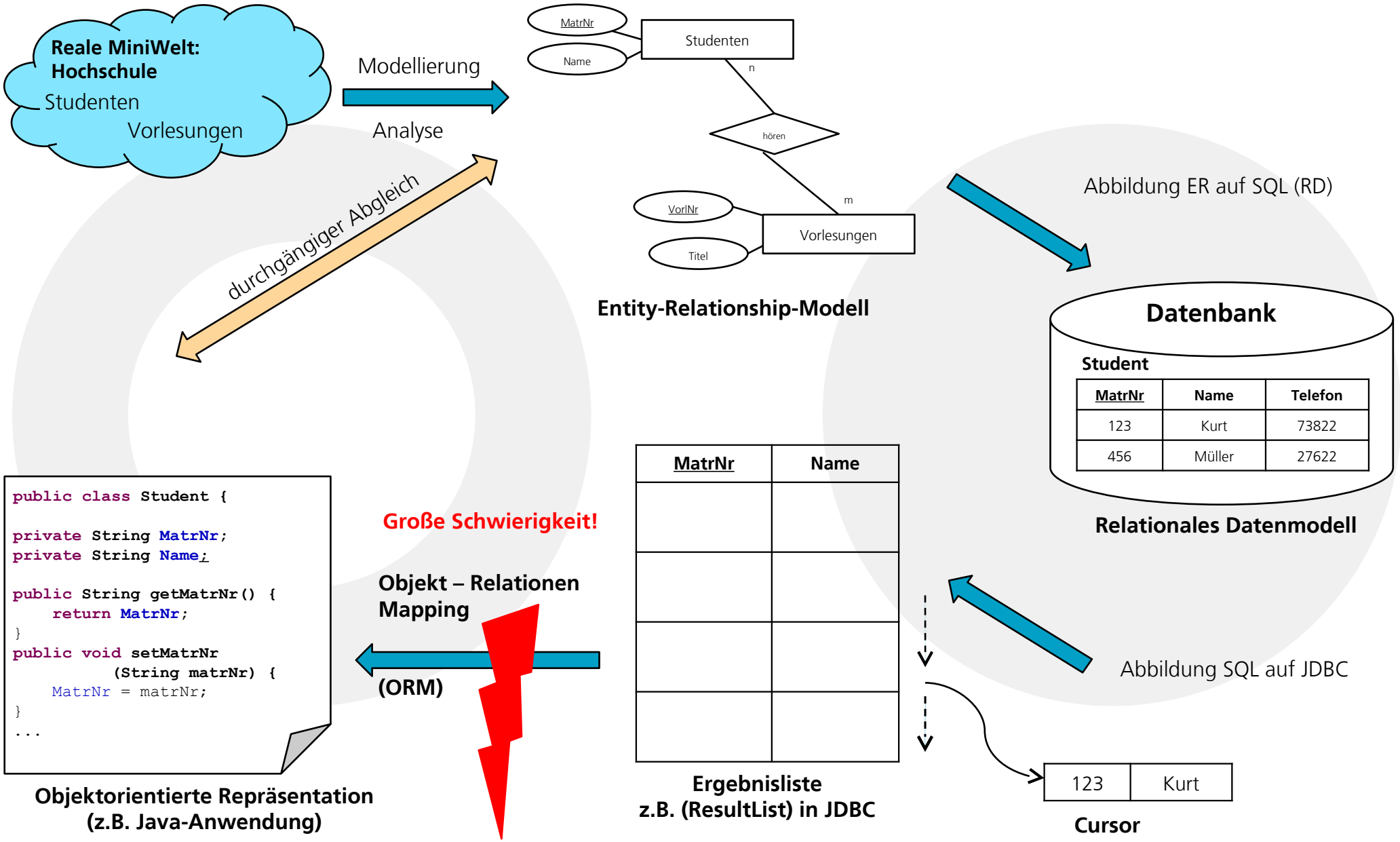
- Die zwölf CODD'schen Regeln
- Datenbankmanagementsystem
- Data Communication System
- Data Dictionary System
- Utilities (System Support Routines)
- Exkurs: Datenbanken in Java
  (Basierend auf Folien von Prof. Alda
  DB 2010-2013, Kapitel 10)

Überblick

- Objekt-Relation Mapping – ein Überblick
- JDBC – ein Überblick

Objekt-Relation Mapping – ein Überblick

nicht Klausur relevant



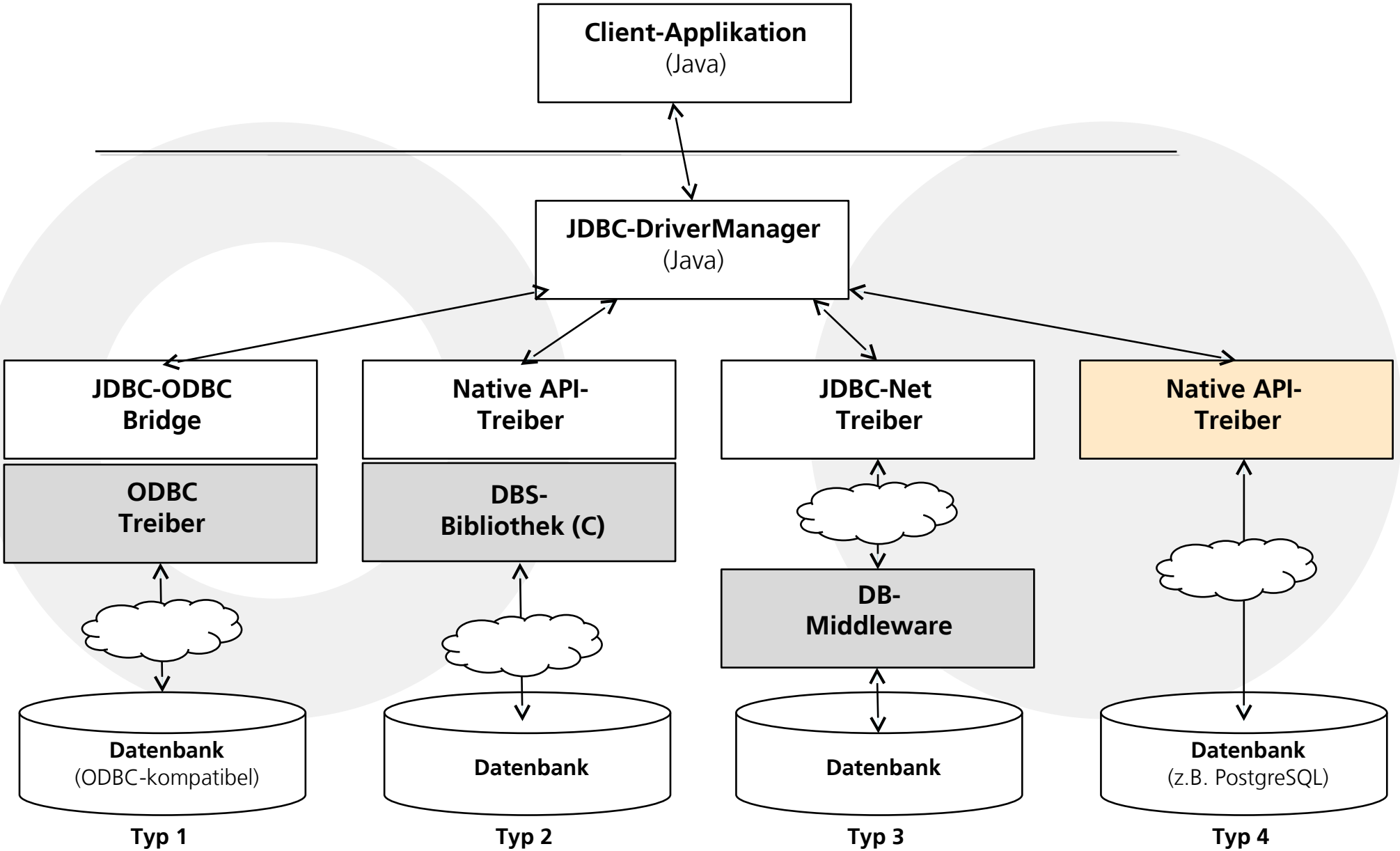
Objekt-Relation Mapping (ORM)

nicht Klausur relevant

- Problem:
  - Defacto-Standard für Datenmodell: Relationales Datenmodell
  - Defacto-Standard für Programmierung: Objektorientiertes Modell
- Beide Welten sind nicht kompatibel (Object-Relational Impedence Mismatch)
- Die Abbildung (Mapping) zwischen diesen beiden Welten ist (immer noch) eine große Herausforderung
- Technologien zur Abbildung einer relationalen Datenstruktur auf eine objektorientierte Struktur nennt man ORM-Technologien
- State-Of-The-Art Technologien (Auswahl):
  - Hibernate (https://www.hibernate.org/)
  - Java Persistence API (JPA) (https://www.oracle.com/technical-resources/articles/java/jpa.html)

Architektur einer JDBC-Applikation mit Treiber-Typen

nicht Klausur relevant



JDBC – ein Überblick

nicht Klausur relevant

- Java Database Connectivity (JDBC) ist ein Framework (Sammlung von Klassen) innerhalb der Java-Plattform, das eine einheitliche Schnittstelle zu relationalen Datenbanken verschiedener Hersteller bietet
  - Standardmäßig ab Java 1.1 integriert
  - Package `java.sql.*`
  - Basiert auf einer Spezifikation (aktuell: JDBC 4.3 für Java 9; JDBC 4.2 für Java 8, JDBC 4.0 für Java 6, sonst JDBC 3.0 für Java 5)
  - https://www.oracle.com/java/technologies/javase/javase-tech-database.html
- Es sind weitere spezielle Treiber-Klassen notwendig, um den Zugriff auf ein gegebenes Datenbanksystem zu realisieren
  - Implementierung der JDBC-Spezifikation
  - Von den meisten DBS-Herstellern angeboten
- Vorbild: ODBC (Open Database Connectivity)

Herausforderung beim Zugriff auf SQL-Tabellen

nicht Klausur relevant

- Wiederholung: Eine SQL-Tabellen entspricht einer mathematischen Relation (eine Menge von Tupeln)

- Problem: Java kann zwar einzelne Tupel (Klasse) darstellen, jedoch keine mengenorientierte Relation
  - Nach-Implementierung möglich, jedoch komplex
  - Seit Java 1.2: Interface `java.util.Set` – aber auch hier nur begrenzte Möglichkeiten zur Auswertung einer Menge

- Ansatz bei JDBC: Relationen werden als abstrakt als Listen realisiert, Zugriff über einen Cursor innerhalb einer Anwendung

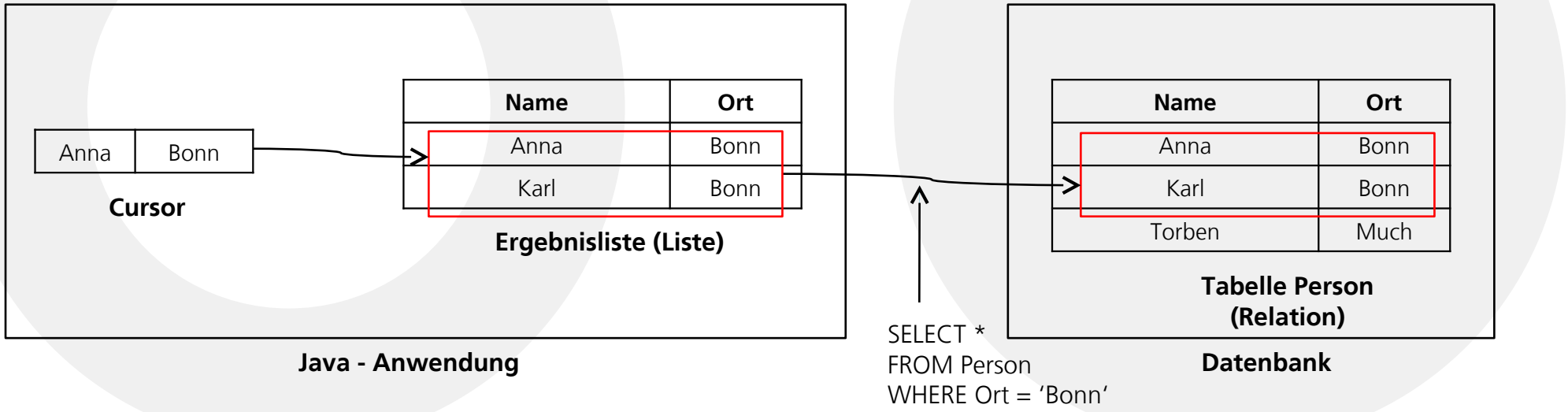
- Implementierung des DECLARE CURSOR Anweisung bei SQL.
  - Weitere Infos: http://www.postgresql.org/docs/9.0/static/plpgsql-cursors.html

Ansatz bei JDBC – Darstellung Ergebnislisten

nicht Klausur relevant

- Ansatz bei JDBC: Relationen werden abstrakt als Listen realisiert, die eine Sicht einer Tabelle aus einer Relationalen Datenbank darstellen

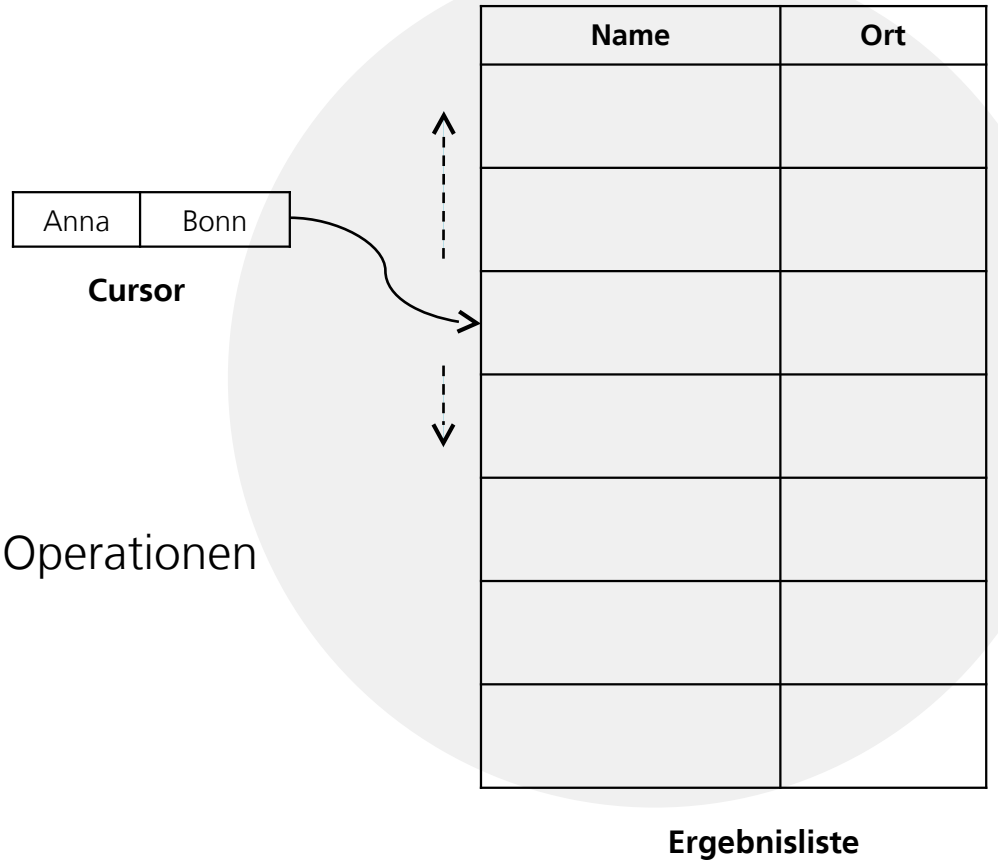
- Navigation über die Liste mit Hilfe eines Cursors.



Cursor in JDBC

nicht Klausur relevant

- Ein Cursor ist ein Iterator über eine Liste von Tupel, d.h. ein Zeiger auf ein Tupel einer Liste, der vor- und zurückgesetzt (seit JDBC 3.0) werden kann



- Über einen Cursor sind alle grundlegenden Operationen möglich:
  - Auslesen eines Tupel
  - Löschen eines Tupel
  - Verändern eines Tupel

Aufbau einer JDBC-Anwendung

nicht Klausur relevant

- Eine Verbindung mit dem DBS sowie Abfragen usw. werden aus der Java-Anwendung heraus initialisiert (mit vier Klassen aus `java.sql.*`):

- Die Kommunikation mit dem Datenbanksystem erfolgt durch Treiber-Klassen, die die Schnittstellen der JDBC-Spezifikation implementieren

Klasse (java.sql.*)	Inhalt
DriverManager	Einstiegspunkt, in dem die notwendigen Treiber-Klassen registriert und Verbindungen zur Datenbank aufgebaut werden;
Connection	repräsentiert eine Datenbankverbindung
Statement (PreparedStatement)	ermöglicht die Ausführung von SQL-Anweisungen über eine gegebene Verbindung (Anweisungsobjekt)
ResultSet	verwaltet die Ergebnisse einer Anfrage in Form einer Liste (Ergebnisobjekt)

Aufbau einer JDBC-Verbindung

nicht Klausur relevant

```
import java.sql.*;
import java.util.*;

public class SimpleClient {

 public static void main(String[] args) throws SQLException {

  DriverManager.registerDriver( new org.postgresql.Driver() );
  String url = "jdbc:postgresql://dumbo.inf.h-brs.de/demouser";
  Properties props = new Properties();
  props.setProperty("user","demouser");
  props.setProperty("password","demouser");

  Connection conn = DriverManager.getConnection(url, props);

  Statement st;
  st = conn.createStatement();
  ResultSet rs = st.executeQuery("SELECT * FROM bierschema.angebot");

  while (rs.next() ){
    System.out.println( "Kneipe: " + rs.getString( „kneipe" )); }
  }
}
```

JDBC Treiber registrieren

nicht Klausur relevant

- Treiber müssen von den Herstellern der Datenbanksysteme (ORACLE, IBM, PostgreSQL) bezogen werden (.*jar* File)
- Setzen des Treibers in der CLASSPATH Umgebungsvariable
- Reinladen des Treibers in Java:

```
DriverManager.registerDriver( new org.postgresql.Driver() );
```

- Alternative: Angabe des Treibers während des Starts des Java-Programms mit dem Parameter –D (CLASSPATH muss auch hier gesetzt sein!)

```
java -Djdbc.drivers = org.postgresql.Driver SimpleClient
```

Verbindungsaufbau

nicht Klausur relevant

- Übergabe der Netzwerkadresse der Datenbank als URL (Uniform Resource Locator). Syntax:

  `url = jdbc:postgresql:// host : port / database`

- Parameter:
  - *host*: Host-Name des Datenbankservers. Hier: `dumbo.inf.h-brs.de`
  - *port*: Port-Number, unter der der Server „horcht“. Hier: `5432` (optional)
  - *database*: Bezeichung der Datenbank. Hier: `userid`!

- Verbindung:

```
Properties props = new Properties();
  props.setProperty("user","demouser");
  props.setProperty("password","demouser");

  Connection conn = DriverManager.getConnection(url, props);
```

Ausführung einer DML Anweisung über JDBC

nicht Klausur relevant

- Anweisungsobjekt (Statement) erzeugen:

```
Statement st = conn.createStatement();
```

- DML-basierte Abfragen auf eine Datenbank werden mit der Methode `executeQuery( String sql )` ausgeführt
- Konkreter SQL-String mit der genauen Abfragen muss übergeben werden
- Rückgabe in ein `ResultSet`-Objekt:

```
ResultSet rs = st.executeQuery("SELECT * FROM bierschema.angebot");
```

Ausführung einer DDL Anweisung über JDBC

nicht Klausur relevant

- DDL-Anweisungen zur Erzeugung von Tabellen usw. (CREATE TABLE, ALTER TABLE) werden über die Methode `execute( String sql )` ausgeführt. Beispiel:

```
boolean check = st.execute("CREATE TABLE person ( name varchar(80) );
");
```

- DDL-Anweisungen (INSERT, UPDATE, DELETE) werden über die Methode `int executeUpdate( String sql )` ausgeführt. Beispiel:

```
int rows = st.executeUpdate("DELETE FROM angebot WHERE
kneipe='Rheinlust';");
```

- Der Rückgabewert (rows) liefert die Anzahl der betroffenen Zeilen

Weitere Parameter für ein Statement (Auswahl)

nicht Klausur relevant

- Über die Methode `setQueryTimeout ( int secs )` kann ein Timeout festgelegt werden
  - Default-Wert: 0 (unbegrenztes Warten)
  - Ein Überschreiten eines gesetzten Limits wird durch eine Exception (java.sql.SQLException) signalisiert

- Über die Methode `setMaxRows( int max )` wird die maximale Anzahl von Tupeln im Anfrageergebnis (ResultSet) festgelegt
  - Default-Wert: 0 (keine Einschränkung)

Ermittlung der Ergebnisse aus einem ResultSet

nicht Klausur relevant

- Ein `ResultSet`-Obekt verwaltet einen internen Cursor, der einen tupelweisen Zugriff auf die Elemente der Menge erlaubt.

- Navigation erfolgt mit der Methode **`boolean next()`**:
  - Liefert so lange **`true`**, wie das Weitersetzen des Cursors erfolgreich war, und damit anzeigt, dass noch weitere Tupel in der Relation vorhanden sind.

```
while ( rs.next() ){

  // Verarbeitung der einzelnen Tupel
  // Spaltenzugriff...
}
```

Zugriff auf die einzelnen Spalten

nicht Klausur relevant

- Nach Positionierung des internen Cursors des ResultSet-Objekts kann der Zugriff auf die Spaltenwerte über getXXX Methoden erfolgen
  - (XXX = Datentyp einer Spalte, muss hier bekannt sein)

- Zugriff über Spaltenindex (ab Wert ‚1' aufsteigend!):

```
String kneipe = rs.getString( 1 );
```

- Zugriff über Spaltenname (falls bekannt):

```
String = kneipe rs.getString( „kneipe“ );
```

Typabbildung von SQL nach Java

nicht Klausur relevant

SQL	get-Methode	Java-Typ
SMALLINT	getShort	short
INTEGER	getInt	int
BIGINT	getLong	long
NUMERIC	getBigDecimal	BigDecimal
DECIMAL	getBigDecimal	BigDecimal
FLOAT	getDouble	double
REAL	getFloat	float
DOUBLE PRECISION	getDouble	double
CHAR	getString	String
VARCHAR	getString	String
BIT	getBoolean	boolean
VARYING BIT	getBytes	byte[]
DATE	getDate	Date
TIME	getTime	Time
TIMESTAMP	getTimestamp	Timestamp

Generische Methode getObject() bei ResultSet

nicht Klausur relevant

- Mittels der Methode `Object getObject( int collindex )` bzw. `Object getObject( String collname )` wird ein Java-Objekt entsprechend dem SQL-Typ der spezifizierten Spalte zurückgeliefert

- Generischer Zugriff möglich durch Casting:

```
String kneipe = (String) rs.getObject( "kneipe" );
```

Behandlung von NULL-Werten

nicht Klausur relevant

- Bei normalen Objekten: Direkte Überprüfung auf NULL möglich:

```
Object ob = rs.getObject("ID");
if ( ob == null ){
  System.out.println("Null-Wert!");
}
```

- Die Überprüfung, ob ein numerischer Attributwert in der DB auf NULL gesetzt wurde, ist nicht so einfach
  - Abhängig vom DB-Anbieter werden unterschiedliche Werte gesetzt
  - Alternative: Methode `wasNull()`

```
int ID = rs.getInt("ID");
if( rs.wasNull() )
 System.out.println("ID ist auf null gesetzt");
else
 System.out.println("Aktuelle ID: " + ID);
```

Parametrisierte Anweisungen (PreparedStatement)

nicht Klausur relevant

- PreparedStatement kapselt eine vorkompilierte Anweisung, die bereits während der Erzeugung des Objekts zum Datenbanksystem gesendet wird

- Laufzeitvorteile wenn Anweisungen mehrfach *parametrisiert* ausgeführt werden müssen (z.B. mehrfache INSERTs oder Anfragen)

```
private void queryPrepared() throws SQLException {
  ...
  Connection conn = DriverManager.getConnection(...);

  PreparedStatement stmt = conn.prepareStatement("INSERT
        INTO studenten VALUES (?, ?, ?, ?)");

  stmt.setString(1, "Karl");
  stmt.setString(2, "Müller");
  stmt.setString(3, "Bonn");
  stmt.setInt(4, 21);

  stmt.executeUpdate();
  // kann beliebig oft wiederholt werden
```

Parametrisierte Anweisungen (II)

nicht Klausur relevant

- NULL-Werte werden über die Methode `setNull( int paramIdx, int nullType )` übermittelt. Angabe der Typkonstante aus `java.sql.Types`

```
private void queryPrepared() throws SQLException {
  ...
  Connection conn = DriverManager.getConnection(...);

  PreparedStatement stmt = conn.prepareStatement("INSERT
        INTO studenten VALUES (?, ?, ?, ?)");

  stmt.setString(1, "Karl");
  stmt.setString(2, "Müller");
  stmt.setString(3, "Bonn");
  stmt.setInt(4, 21);

  stmt.executeUpdate();
  // kann beliebig oft wiederholt werden
}
```

Parametrisierte Anweisungen (III)

nicht Klausur relevant

- Ausführung von parametrisierten Abfragen mit der Methode `executeQuery()`. Parametrisierung nur von Attributswerten möglich:

```
...
PreparedStatement stmt2 =
       conn.prepareStatement("SELECT * from studenten WHERE name = ? ");

stmt2.setString(1, "Maier");
ResultSet rs = stmt2.executeQuery();

while (rs.next() ){
   System.out.println( "Name: " + rs.getString( "name" ));
}
...

}
```

Typabbildung von Java nach SQL

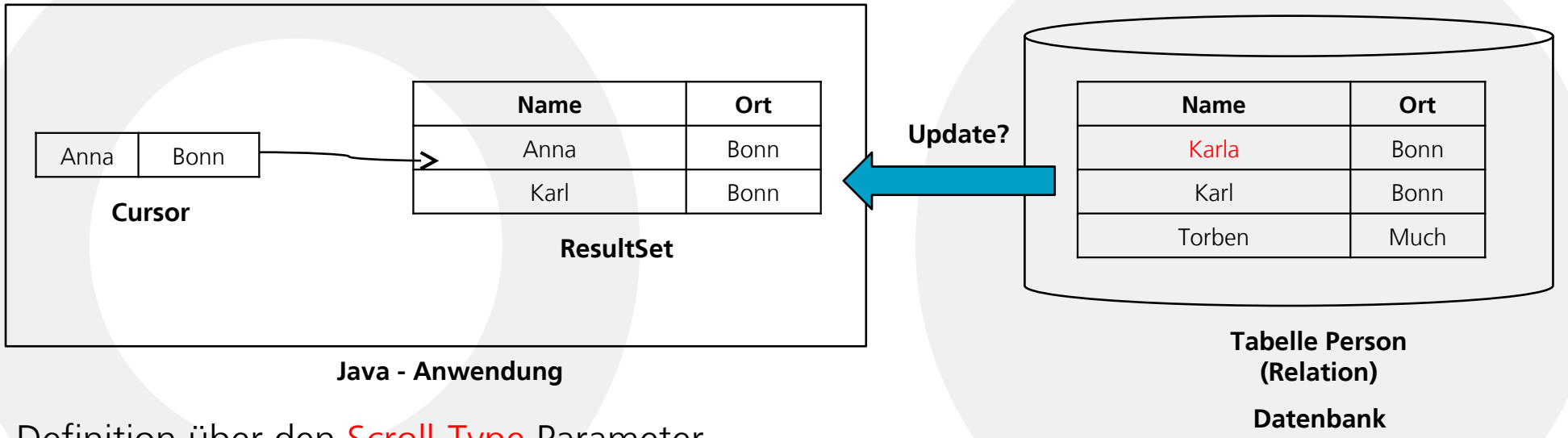
nicht Klausur relevant

Java-Typ	set-Methode	SQL
byte	setByte	SMALLINT
short	setShort	SMALLINT
int	setInt	INTEGER
long	setLong	BIGINT
BigDecimal	setBigDecimal	NUMERIC
float	setFloat	REAL
double	setDouble	DOUBLE
String	setString	VARCHAR
boolean	setBoolean	BIT
byte[]	setBytes	VARYING BIT
Date	setDate	DATE
Time	setTime	TIME
Timestamp	setTimestamp	TIMESTAMP

Erweiterte ResultSets – ScrollType (also Änderungstyp)

nicht Klausur relevant

- `ResultSets` können weiter parametrisiert werden, um das Änderungs-Verhalten der Liste festzulegen, falls es auf Seiten der Datenbank zu Änderungen kommt



- Definition über den Scroll-Type Parameter
  - Festlegung zudem, ob man in einer Liste vorwärts und rückwärts navigieren (scrollen) kann

Erweiterte ResultSets - ScrollType

nicht Klausur relevant

- ScrollTypen (statische Klassen-Variablen aus der Klasse `ResultSet`)
  - TYPE_SCROLL_INSENSITIVE
    - ResultSet wird nach Änderung des Datenbestandes *nicht* geändert
    - Statische Sicht (`ResultList` ist reine Kopie)
    - vorwärts- und rückwärts scrollbar (navigierbar)
  - TYPE_SCROLL_SENSITIVE
    - ResultSet wird nach der Änderung des Datenbestandes geändert
    - Dynamische Sicht (`ResultList` ist mit Datenbank synchronisiert)
    - vorwärts- und rückwärts scrollbar
  - TYPE_FORWARD_ONLY (Default)
    - ResultSet wird nach Änderung des Datenbestandes nicht geändert
    - Statische Sicht
    - nur vorwärts scrollbar

Wie werden Updates aus einer DB in ein ResultSet gebracht?

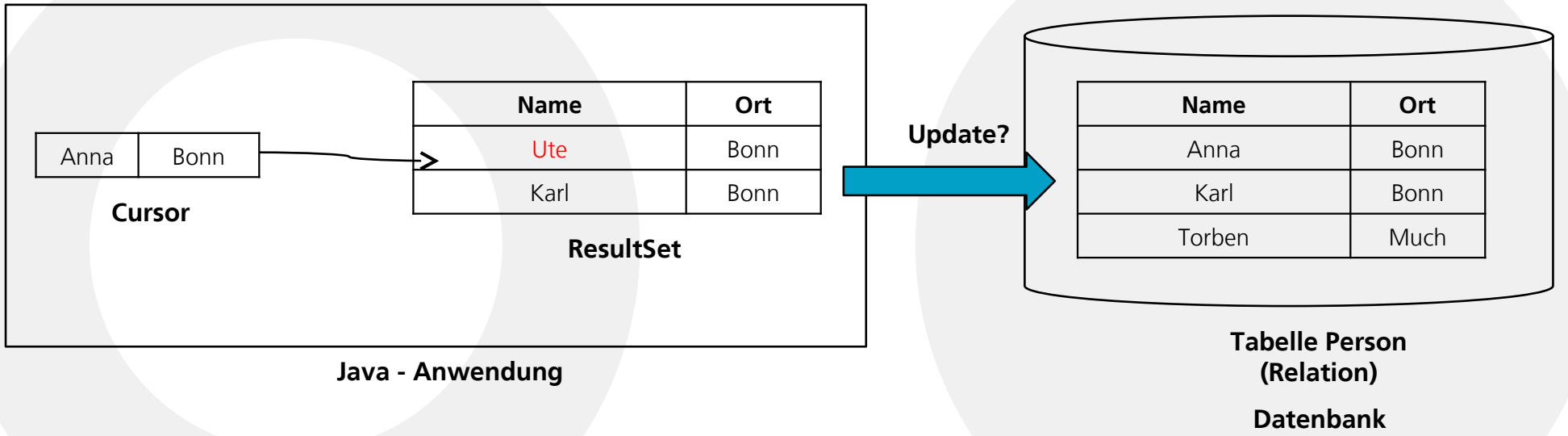
nicht Klausur relevant

- Übernahme der neuen Tupelwerte beim Scrollen (z.B. mit `next()`)
- Explizite Angabe eines Refreshs (Methode `refreshRow()`)
- Überprüfung, ob sich ein Tupel an der aktuellen Cursorposition nach der initialen Erstellung bzw. nach dem letzten Update geändert hat:
  - `boolean rowDeleted()`
  - `boolean rowInserted()`
  - `boolean rowUpdated()`
- Aus-Implementierung und somit Funktionalität der Methoden oft unterschiedlich und abhängig vom Datenbanksystem

Erweiterte ResultSets - Synchronisation

nicht Klausur relevant

- `ResultSets` können weiter parametrisiert werden, um festzulegen, ob ein `ResultSet` geändert werden darf und ob diese Änderungen an eine Datenbank weitergeleitet werden dürfen (Synchronisationsverhalten)



- Definition durch Synchronisations-Typen

Implementierung eines ResultSets - Beispiele

nicht Klausur relevant

- Implementierung eines `ResultSet`-Objekts, das änderbar und insensitiv ist:

```
Statement st;
st = conn.createStatement( ResultSet.TYPE_SCROLL_INSENSITIVE,
                           ResultSet.CONCUR_UPDATABLE );

ResultSet rs = st.executeQuery(„SELECT * FROM angebot ) ;
```

- Implementierung eines `ResultSet`-Objekts, das nicht änderbar und nur vorwärtsnavigierbar ist:

```
Statement st;
st = conn.createStatement( ResultSet.TYPE_FORWARD_ONLY,
                           ResultSet.CONCUR_READ_ONLY );
// oder:
st = conn.createStatement();

ResultSet rs = st.executeQuery(„SELECT * FROM angebot ) ;
```

Methoden zur Navigation innerhalb eines ResultSet

nicht Klausur relevant

Methoden (aus ResultSet)	Cursor-Position
`beforeFirst()`	Vor dem ersten Tupel
`first()`	Erstes Tupel
`afterLast()`	Nach dem letzten Tupel
`last()`	Letztes Tupel
`next()`	Nächstes Tupel
`previous()`	Vorheriges Tupel
`absolute( int zeile )`	Tupels auf Indexposition p
`relative( int zeile )`	Tupels auf Indexposition (aktuelle Position + p)

- Bestimmung der aktuellen Position des Cursors:
  - `int getRow()`
  - `boolean isLast(), boolean isFirst(), boolean isAfterLast(), boolean isBeforeFirst()`

Aktualisierung von Tupel in einem ResultSet

nicht Klausur relevant

- Ähnlich wie bei einem VIEW in SQL sollten `ResultSets` nur dann geändert werden, falls folgende Eigenschaften vorhanden:
  - Die Anfrage beinhaltet nur eine Relation und kein JOIN-Operationen
  - Alle Attributswerte zum Primärschlüssel werden in das Ergebnis übernommen (zur eindeutigen Identifikation)

- Änderungen werden auf das Tupel an der aktuellen Cursorposition mit der Methode `updateXXX()` durchgeführt (XXX = Typ der Spalte)
- Definitive Änderung in DB erst durch Methode `updateRow()`:

```
ResultSet rs = st.executeQuery("SELECT * from \"Department\"");

rs.first();
rs.updateString("Name", "FB Informatik");
rs.updateRow();

System.out.println("Tupel aktualisiert");
```

Hinzufügen von Tupel in einem ResultSet

nicht Klausur relevant

- Das Einfügen von Tupeln erfolgt an einer speziellen Einfügeposition, die mit `moveToInsertRow()` erreicht wird.
- Das „Einfügetupel“ wird über die `updateXXX()` Methoden eingefügt.
- Definitive Änderung in DB erst durch Methode `insertRow()`:

```
ResultSet rs = st.executeQuery("SELECT * from \"Department\"");

rs.first();
rs.moveToInsertRow();
rs.updateString("Name", "Fachbereich Journalismus");
rs.updateInt( "ID", 5 );
rs.insertRow();

System.out.println("Tupel eingefügt");
```

Löschen von Tupel in einem ResultSet

nicht Klausur relevant

- Für das Löschen von Tupel muss der Cursor entsprechend positioniert werden.
- Löschen aus ResultSet sowie definitives Löschen aus DB erfolgt durch die Methode `deleteRow()`:

```
ResultSet rs = st.executeQuery("SELECT * from \"Department\"");

// Löschen des Eintrags des FB 5
while (rs.next() ){
   int idInt = rs.getInt("ID");
   if (idInt == 5) {
       rs.deleteRow();
   }
}

System.out.println("Tupel Gelöscht");
```

Schließen einer Verbindung

nicht Klausur relevant

- Nachdem eine Anweisung ausgeführt wurde und die Ergebnisse ermittelt wurden, sollten damit verbundene Ressourcen (Client wie DB-seitig) durch Aufruf der close-Methode freigegeben werden:
  - `ResultSet`
  - `Statement`
  - `Connection`
- Falls nicht: Verbindung wird automatisch durch Garbage Collector geschlossen (evt. verzögert)

```
public class SimpleClient {

 public static void main(String[] args) throws SQLException {
   ....

   rs.close()
   st.close();
   conn.close();
 }
```

Metadaten über ein ResultSet

nicht Klausur relevant

- Ein Objekt der Klasse `ResultSetMetaData` liefert übergreifende Informationen über ein `ResultSet`
- Abrufbar über ein ResultSet über die Methode `getMetaData()`

- Fokus auf Eigenschaften von Spalten:
  - `int getColumnCount();`
  - `String getColumnName( int col );`
  - `String getColumnTypeName( int col );`
  - u.a.

- Weitere Infos:
  - Tabellenname, Schemaname (nicht immer unterstützt)

- Was fehlt: die Anzahl der Tupel im `ResultSet`
  - Kann man trotzdem über die Cursor-Technik rausfinden (→ Selbststudium)