

## I. EINLEITUNG

### Motivation

50% weniger Aufwand bei Anwendungsentwicklung mit DB  
Ermöglicht neue Anwendungen, die ohne DB zu komplex wären  
Ausfaktorisieren der Verwaltung großer Datenmengen  
ohne Datenbanken

- Daten in Dateien abgelegt, Zugriffsfunktionalität Teil der Anwendung
- Redundanz (in Daten und Funktionalität)
- Programme oft nicht *atomar* (= Programm wird entweder ganz oder gar nicht ausgeführt) – nur bei nicht fehlerfreien Systemen relevant
- Transaktionen* (= Programm oder Kommandofolge) oft nicht *isoliert* (= keine inkonsistenten Zwischenzustände sichtbar) – nur bei mehreren Transaktionen, aber auch bei fehlerfreien Systemen relevant
- Nebenläufigkeit (*concurrency* – paralleler Zugriff auf dieselben Daten) schwer umsetzbar
- Anwendungsentwicklung abhängig von der physischen Repräsentation der Daten (z.B. Datenspeicherung als Tabelle: Reihenfolge Zeilen/Spalten muss bekannt sein)
- Datenschutz (= kein unbefugter Zugriff) nicht gewährleistet
- Datensicherheit (= kein Datenverlust, insb. bei Defekten) nicht gewährleistet

### Relationale Datenbanken

auch RDBMS (*relational database management system*)  
 $\cong$  Menge von Tabellen  
 Relation = Menge von Tupeln = Tabelle

### RDBMS – Terminologie

Relationenschema: **Fett** geschrieben  
Relation: Weitere Einträge der Tabelle  
Tupel: Eine Zeile der Tabelle  
Attribut: Spaltenüberschrift  
Relationenname: Name der Tabelle  
DBS: Datenbanksystem = DBMS + Datenbank(en)  
Schlüssel: Attribut, das nicht doppelt vergeben werden darf  
Fremdschlüssel: Attribut taucht in anderem Relationenschema als Schlüssel auf  
Integritätsbedingungen:

1. **lokal**: Schlüssel in Relationenschema
2. **global**: Fremdschlüssel in Datenbankschema

DB-Schema: = Menge der Relationsschemata + globale Integritätsbedingungen

Sicht (*view*): Häufig vorkommende Datenabfrage, kann mit Sichtnamen als „virtuelle“ Tabelle gespeichert werden

```
create view Cartist as
select NAME, JAHR
from Kuenstler
where LAND == "Kanada"
```

Verwendung wie „normale“ Relation:

```
select * from Cartist where JAHR < 2000
```

Nutzung für Datenschutz: Unterschiedliche Benutzer sehen unterschiedlichen DB-Ausschnitt

### RDBMS – Anfrageoperationen

Selektion: Zeilen (Tupel) wählen ( $\sigma_{KID=1012}(\text{Titel})$ )

Projektion: Spalten (Attribute) wählen ( $\pi_{KID, NAME}(\text{Kuenstler})$ )

Beispiel komplexer Ausdruck:  $\pi_{NAME, ART}(\sigma_{KID=1012}(\text{Titel}))$

Ausgangsrelation:

TITLE ID	NAME	ART	GRÖSSE	KID
102	Neil Young – Heart of Gold	mp3	2.920kb	1012
103	Rammstein – Ich liebe Neil Young	wma	4.234kb	1014
104	Neil Young – Old Man	mp3	3.161kb	1012
105	Neil Young – Four Strong Winds	wma	5.125kb	1012

Ergebnis:

NAME	ART
Neil Young – Heart of Gold	mp3
Neil Young – Old Man	mp3
Neil Young – Four Strong Winds	wma

Weitere Operationen: Verbund (*join*), Vereinigung, Differenz, Durchschnitt, Umbenennung

Operationen beliebig kombinierbar ( $\sim$  Query-Algebra)

### RDBMS – Anfragenoptimierung

Algebraische Ausdrücke äquivalent, Anfrage aber unterschiedlich komplex, z.B.

$\sigma_{\text{Vorname}='Klemens'}(\sigma_{\text{Wohnort}='KA'}(SNUSER))$  vs.  
 $\sigma_{\text{Wohnort}='KA'}(\sigma_{\text{Vorname}='Klemens'}(SNUSER))$

### RDBMS – Physische Datenunabhängigkeit

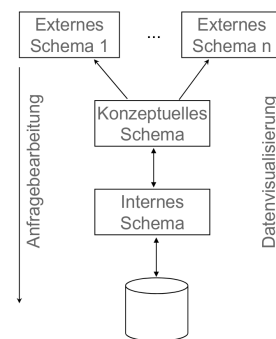
Anfragen deklarativ: Nutzer entscheidet nicht, wie Ergebnis ermittelt wird

Datenunabhängigkeit: DBMS stellt sicher:

1. stabile Anfragenfunktionalität bei physischer Darstellungsänderung
2. Anfrage funktinoiert bei unterschiedlichen Datenbanken (gleiches Schema, unterschiedliche Datenhäufigkeit)

$\sim$  erlaubt höhere Komplexität bei Anwendungsentwicklung

### RDBMS – 3-Ebenen-Architektur



Konzeptionelles Schema: Diskursbereich? Welche Entitäten interessant (bei Studierenden Noten interessant, Hobbies usw. nicht)?

Internes Schema: physische Datenrepräsentation

Externe Schemata: Unterschiedlicher Datenausschnitt für unterschiedliche Nutzer (Datenschutz, Übersichtlichkeit, organisatorische Gründe, Verstecken von Änderungen am konzeptionellen Schema)

$\sim$  **Logische Datenunabhängigkeit**

## Datenbankprinzipien – Codd'sche Regeln

1. Integration: Einheitliche, nichtredundante Datenverwaltung
2. Operationen: Speichern, Suchen, Ändern
3. Katalog: Zugriff auf Datenbankbeschreibungen im data directory
4. Benutzersichten
5. Integritätssicherung: Korrektheit des DB-Inhalts
6. Datenschutz: Ausschluss unauthorisierter Zugriffe
7. Transaktionen: mehrere DB-Operationen als Funktionseinheit (= Atomarität)
8. Synchronisation: parallele Transaktionen koordinieren (= Isolati-on)
9. Datensicherung: Wiederherstellung von Daten nach Systemfehlern

Strengste bekannte Datenbankdefinition

Funktionale Anforderungen (nichtfunktional z.B.: Wie schnell/zuverlässig muss Dienst sein, kurze Antwortzeiten, Zuverlässigkeit, Effizienz, Skalierbarkeit)

### Prüfungsfragen

1. Was ist eine Sicht?
2. Was ist die relationale Algebra? Wozu braucht man sie?
3. Geben Sie Beispiele für Algebra-Ausdrücke an, die nicht identisch, aber äquivalent sind, an.
4. Was leistet der Anfragenoptimierer einer Datenbank?
5. Erklären Sie: Drei-Ebenen-Architektur, physische/logische Datenunabhängigkeit.

## II. CLUSTERING UND AUSREISSER

### Räumliche Indexstrukturen – Motivation

Was ist die nächste Bar, die mein bevorzugtes Bier ausschenkt?

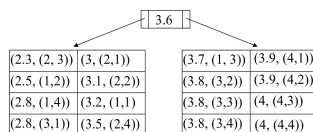
Bereichsanfrage: Wie viele Restaurants gibt es im Stadtzentrum?

Ähnlichkeitssuche Bilder: Distanz im Merkmalsraum = Maß der Unähnlichkeit

### Index – B+-tree

= non-clustered primary B+-tree

Beispiel: Student(name, age, gpa, major), B+-T für gpa (kleiner=links, größer=rechts, (gpa,(x,y)))



Tom, 20, 3.2, EE	Mary, 24, 3, ECE	Lam, 22, 2.8, ME	Chris, 22, 3.9, CS
Chang, 18, 2.5, CS	James, 24, 3.1, ME	Kathy, 18, 3.8, LS	Vera, 17, 3.9, EE
Bob, 21, 3.7, CS	Chad, 28, 2.3, LS	Kane, 19, 3.8, ME	Louis, 32, 4, LS
Pat, 19, 2.8, EE	Leila, 20, 3.5, LS	Martha, 29, 3.8, CS	Shideh, 16, 4, CS

### Index – kd-tree

B+-T löst Bar-Problem nicht wirklich

kd-tree: Splitting für eine Dimension nach der anderen, dann wieder von vorne

Beispiel: Vier Split-Dimensionen



### kd-tree – k-NN

k-NN (= *k-next-neighbour*) := Abstand des *k*-nächsten Nachbarn

Notation:  $E[k\text{-NN}]$

Es müssen nur ein paar Rechtecke inspiziert werden, um Resultat zu ermitteln

Implementierung: Priority Queue (Inhalt Datenobjekte/Baumknoten, sortiert nach Abstand zum Anfragepunkt)

Hier: Baum unbalanciert, Balancierung in Realität für mehrdimensionale Daten

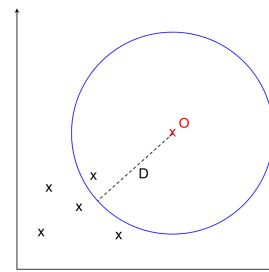
### Outlier

= Element des Datenbestands, das in bestimmter Hinsicht erheblich vom restlichen Datenbestand abweicht

Mögliche Definition:

Objekt *O*, das in Datenbestand *T* enthalten ist ist ein  $DB(p, D)$ -Outlier, wenn der Abstand von *O* zu mindestens *p* Prozent der Objekte in *T* größer ist als *D*.

Beispiel: *O* ist Outlier, wenn  $p = 0.6$ , da dann mehr als 60% der Datenobjekte außerhalb des Kreises liegen



### Outlier – Algorithmen

Index-basiert: k-NN für jeden Punkt. Stop, sobald  $k\text{-NN} < D$

Clustering: Liefert Outlier als Beiprodukt

Abstands-basiert

Dichtebasiert

## Clustering – Beispiel

Gegeben: Große Kundendatenbank, enthält Eigenschaften und Käufe

Gesucht: Gruppen von Kunden mit ähnlichem Verhalten finden

## Clustering – DBSCAN

Dichte: Anzahl Objekte pro Volumeneinheit

Dichtes Objekt: mindestens  $x$  andere Objekte in Kugel um Objekt mit Radius  $\epsilon$  (A)

Dichte-erreichbares Objekt: Objekt in  $\epsilon$ -Umgebung eines dichten Objekts, das selbst nicht dicht ist (= Clusterrand, B, C)

Rauschen (*Noise*): Objekte, die von keinem dichten Objekt erreicht werden können (N)



## DBSCAN – Eigenschaften

Komplexität: Lineare, wenn  $\epsilon$ -Umgebungen vorberechnet wurden (oder mit räumlichem Index in konstanter Zeit bestimmt werden können)

→ mehrdimensionale Indexstruktur sehr sinnvoll

Rauschen liefert mögliche Outlier

## Hochdimensionale Datenräume – Anomalien

Sparsity: Raum ist nur dünn mit Punkten besetzt

Hierarchische Datenstrukturen uneffektiv: bei sehr, sehr vielen Dimensionen ist Abstand zweier Datenobjekte fast gleich dem zweier anderer (unter schwachen Annahmen) → keine echten Outlier (Outlier-Algorithmen liefern mehr oder weniger zufälliges Objekt)

→ nur erfolgsversprechende Teilräume nach Ausreißern absuchen

Interessante Cluster sind i.d.R. nicht Cluster in allen Dimensionen

## Outlier – im Höherdimensionalen

Outlier erscheinen als solche nur in Teilräumen

Manche Teilräume ausreißerfrei

Unterschiedlichdimensionale Teilräume enthalten Ausreißer

trivial vs. nichttrivial:

1. **trivial**: Objekt ist in Teilraum bereits Ausreißer
2. **nichttrivial**: Gegenteil

→ Maß für Teilraumrelevanz – wie findet man relevante TR?

## Subspace Search

Exponentiell viele Teilräume  $P(A)$

Auswahl relevanter Teilräume  $RS \subset P(A)$

## HiCS – Prinzip

Attribute korrelieren nicht → Outlier in diesem Raum tendenziell eher trivial

Idee: Suche nach Verletzung statistischer Unabhängigkeit (= **Kontrast**)

## Prüfungsfragen

1. Warum kann man räumliche Anfragen nicht ohne Weiteres auswerten, wenn man für jede Dimension separat einen B-Baum angelegt hat?
2. Wie funktioniert der Algorithmus für die Suche nach den  $k$  nächsten Nachbarn mit Bäumen wie dem kd-Baum?
3. Warum werden bei der NN-Suche nur genau die Knoten inspiziert, deren Zonen die NN-Kugel überlappen?
4. Was ist ein Outlier?
5. Was ist ein Zusammenhang zwischen  $k$ -NN-Suche mit Bäumen wie dem kd-Baum und Outlier-Berechnung?
6. Warum ist die Zuordnung Dichte-erreichbarer Punkte mit DBSCAN nichtdeterministisch?
7. Warum sind hierarchische Datenstrukturen in hochdimensionalen Merkmalsräumen für die  $k$ -NN-Suche nicht das Mittel der Wahl?
8. Was bedeutet *Subspace Search*?
9. Geben Sie die Unterscheidung zwischen trivialen und nichttrivialen Outliern aus der Vorlesung wieder.
10. Was genau bedeutet *Kontrast* im Kontext von HiCS?

## III. DATENBANK-DEFINITIONSSPRACHEN

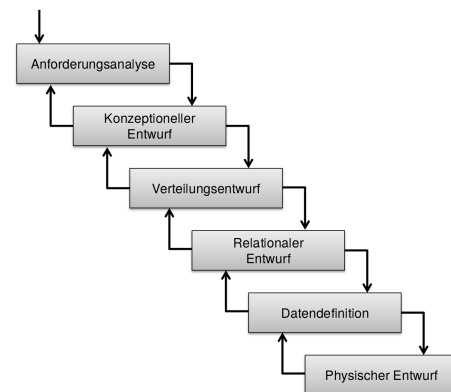
### Gewinnung der Konventionen

Beschränkte Anwendungswelt (= Miniwelt, relevanter Weltausschnitt, Diskursbereich)

Daten: Modelle (gedankliche Abstraktionen) der Miniwelt

Datenbasiskonsistenz: Datenbasis ist bedeutungstreu, wenn ihre Elemente Modelle einer gegebenen Miniwelt sind (schärfste Konsistenzforderung)

### Datenbankentwurf – Phasenmodell



### Datenbankentwurf – Modellierung

Ausschnitt der Wirklichkeit mit Schema beschreiben

Typen = Struktur der Entitäten

Welche Konsistenzbedingungen sind sinnvoll?

Schemakonsistenz: Einhaltung der durch Schema vorgegebenen Konsistenzbedingungen (= von DBMS überprüfbar!)

### SQL

= standardisierte Sprache für DB-Zugriff (relational)

Aspekte:

1. Schemadefinition
2. Datenmanipulation (Einfügen, Löschen, Ändern)
3. Anfragen

## SQL – SQL-DDL

= SQL data definition language

Teilbereich von SQL, der zu tun hat mit Definition von:

1. Typen
2. Wertebereichen
3. Relationsschemata
4. Integritätsbedingungen

## SQL – als Definitionssprache

1. Externe Ebene:

```
{ create | drop } view;
```

2. Konzeptuelle Ebene:

```
{ create | alter | drop } table;
{ create | alter | drop } domain;
```

3. Interne Ebene:

```
{ create | alter | drop } index;
```

## Data Dictionary

= Menge von Tabellen und Sichten

Wie Datenbank aufgebaut

Enthält keine Anwendungsdaten, sondern Struktur-Metadaten

## SQL – Tabelle anlegen

```
create table Kuenstler
(KID integer, NAME varchar(200),
LAND varchar(50) not null, JAHR integer,
primary key (KID))
```

## SQL – Wertebereiche

integer (auch int)

smallint

float(p) (auch float)

decimal(p,q) (auch numeric(p,q), jeweils mit q Nachkommastellen)

character(n) (auch char(n) oder char für  $n = 1$ )

character varying(n) (auch varchar(n), String variabler Länge bis Maximallänge  $n$ )

bit(n) (oder varying(n) analog für Bitfolgen)

date, time, timestamp

## Wertebereiche – Custom

```
create domain Gebiete varchar(20)
default 'Informatik'
```

```
create table Vorlesungen
(Bezeichnung varchar(80) not null, SWS smallint,
Semester smallint, Studiengang Gebiete)
```

## Integritätsbedingungen

Schlüssel kann aus mehreren Attributen bestehen

Fremdschlüssel:

```
create table Titel
(TITLEID integer not null, NAME varchar(200),
KID integer, primary key (TITLEID),
foreign key (KID) references Kuenstler(KID))
```

**default**-Klausel: Standardwert für Attribut

**check**-Klausel: weitere lokale Integritätsbedingungen

```
create table Vorlesungen
(Bezeichnung varchar(80) not null, SWS smallint,
Semester smallint, check(Semester between 1 and 9),
Studiengang Gebiete)
```

## SQL – alter und drop

```
alter table Lehrstuehle
add Budget decimal(8,2)
```

~~ Änderung Relationsschema im Data Dictionary, existierende Daten werden um **null**-Attribut erweitert

```
drop spaltenname { restrict | cascade }
```

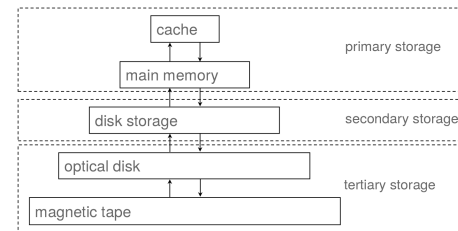
~~ Attribut löschen, falls

1. **restrict**: keine Sichten/Integritätsbedingungen mit diesem Attribut definiert wurden
2. **cascade**: gleichzeitig diese Schichten/Integritätsbedingungen mitgelöscht werden sollen

```
drop table basisrelationenname { restrict | cascade }
```

~~ analog zu Attribut

## Speicherhierarchie



## Index

Für mehrere Attribute möglich

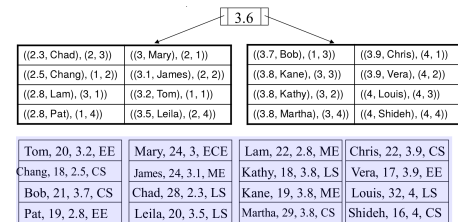
Index für (gpa, name)  $\neq$  Index für (name, gpa)

Index kann nachträglich angelegt bzw. gelöscht werden, ohne Daten selbst zu löschen

Index Bestandteil der physischen Ebene, Index-Definition Teil des internen Schemas

**select name from Student where gpa > 4** liefert Ergebnis unabhängig von Existenz eines Index – wenn vorhanden erhebliche Beschleunigung

**create unique index typ on auto(hersteller, modell, baujahr)** hilft bei Herstellersuche, weniger bei Suche nach Baujahr



## Prüfungsfragen

1. Erläutern Sie anhand eines Anwendungsbeispiels, warum man die Menge der zulässigen Zustände einschränken will.
2. Erläutern Sie: Schema-Konsistenz, Datenbasis-Konsistenz.
3. Was ist ein (DB-)Schema?
4. Was ist das Data Dictionary?
5. Warum sollte man sich die Mühe machen, Integritätsbedingungen als Teil des DB-Schemas zu formulieren?
6. Sind Integritätsbedingungen Bestandteil des internen oder des konzeptuellen Schemas? Begründen Sie Ihre Antwort.
7. Wieso sind Indices Bestandteil des internen und nicht des konzeptuellen Schemas?
8. Geben Sie Beispiele für DB-Features an, die zeigen, dass DB-Systeme physische Datenunabhängigkeit nicht vollständig umsetzen.

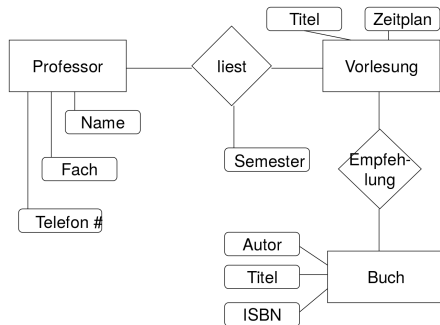
## IV. DATENBANKMODELLE FÜR DEN ENTWURF

### Entity-Relationship-Modelle

**Entity:** Objekt der Real-/Vorstellungswelt (z.B. Buch)

**Relationship:** Beziehung zw. Entities (z.B. Schüler hat Buch)

**Attribut:** Eigenschaft von Entities (z.B. ISBN)



### ER – Modellierungskonzepte

$\mu(D)$ : Interpretation von  $D$ , mögliche Werte einer Entity-Eig.

$\mu(\text{int})$ : Wertebereich  $\mathbb{Z}$

$\mu(\text{string})$ : Wertebereich  $C^*$  (Folgen von Zeichen aus  $C$ )

$\mu(E)$ : Menge der möglichen Entities vom Typ  $E$

$\sigma_i(E)$ : Menge der *aktuellen* Entities vom Typ  $E$  in Zustand  $\sigma_i$  (Index  $i$  weglassen, wenn eindeutig)

1.  $\sigma(E) \subseteq \mu(E)$
2.  $\sigma(E)$  endlich

$\mu(R) = \mu(E_1) \times \dots \times \mu(E_n)$

$\rightsquigarrow$  Die Menge aller möglichen Ehen ist die Menge aller (Mann,Frau)-Paare.

$\sigma(R) \subseteq \sigma(E_1) \times \dots \times \sigma(E_n)$

$\rightsquigarrow$  aktuelle Beziehungen nur zwischen aktuellen Entities

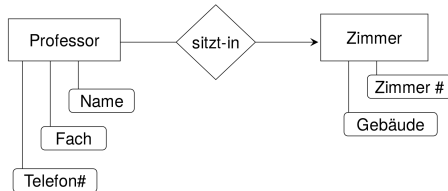
Attribut  $A$  eines Entity-Typen  $E$  ist im Zustand  $\sigma$  eine Abbildung  $\sigma(A) : \sigma(E) \rightarrow \mu(D)$  (nicht  $A : \sigma(E) \rightarrow \mu(D)$ )

Beziehungsattribute:  $\sigma(A) : \sigma(R) \rightarrow \mu(D)$  (Beziehung  $R$ , Attribut  $A$ , möglicher Wertebereich  $\mu(D)$ )

### Funktionale Beziehungen

Jedem Professor lässt sich ein Zimmer zuordnen, umgekehrt nicht zwingend

Schreibe:  $R : E_1 \rightarrow E_2$



### Schlüssel

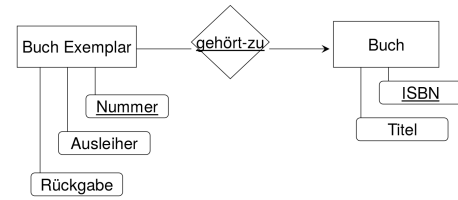
Schlüsselattribute  $\{S_1, \dots, S_k\} \subseteq \{A_1, \dots, A_m\}$  für Entity-Typ  $E(A_1, \dots, A_m)$

Notation: Schlüssel unterstreichen:  $E(\dots, \underline{S_1}, \dots, \underline{S_i}, \dots)$

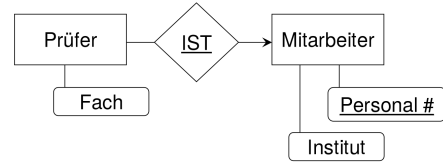
Schlüssel ist minimal: Wird ein Schlüsselattribut entfernt, so ist das entstehende Tupel nicht mehr eindeutig

### Abhängige Entity-Typen

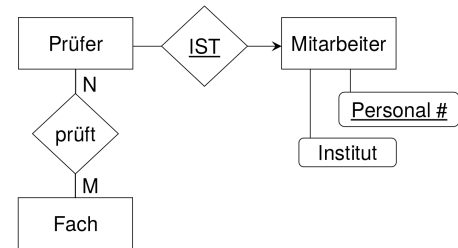
Identifikation: Funktionale Beziehung



### IST-Beziehung



Vererbung von Attributen (und Werten):  
 $\sigma(\text{Prüfer}) \subseteq \sigma(\text{Mitarbeiter})$



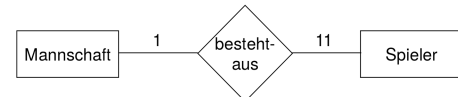
### Entwurf – Kardinalitäten

An wv. Beziehungen muss Entity teilnehmen?  $\rightsquigarrow$  einschränken

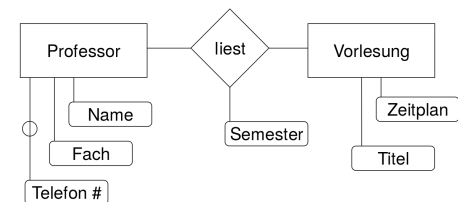
Teilnehmerkardinalität: arbeitet\_in(Mitarbeiter  $[0, 1]$ , Raum  $[0, 3]$ )

1. jeder Mitarbeiter hat einen zugeordneten Raum, aber einige Mitarbeiter haben kein Arbeitszimmer
2. pro Zimmer arbeiten maximal drei Mitarbeiter
3. ein Zimmer kann leerstehen

Standardkardinalität: 1 Mannschaft steht mit 11 Spielern in Bezug  
speziell: m:n/1:n/1:1-Beziehung



### Optionale Attribute



## Semantische Beziehungen

Spezialisierung: Prüfer Spezialisierung von Mitarbeiter  
 $\rightsquigarrow$  Vererbung

Partitionierung: Spezialfall der Spezialisierung, mehrere *disjunkte* Entity-Typen (z.B. Partitionierung von Buch in Monographie und Sammelband)

Generalisierung: Medium ist stets DVD oder Buch  
 $\rightsquigarrow$  Abstrakte Klasse Medium

Aggregation: Auto besteht aus Motor, Karosserie,...

$\rightsquigarrow$  Entity aus Instanzen anderer Entity-Typen zusammengesetzt

Sammlung (auch Assoziation): Team ist Gruppe von Person  
 $\rightsquigarrow$  Mengenbildung

### EER

= Erweitertes ER-Modell

Übernommen: Werte, Entities, Beziehungen, Attribute, Funktionale Beziehungen, Schlüssel

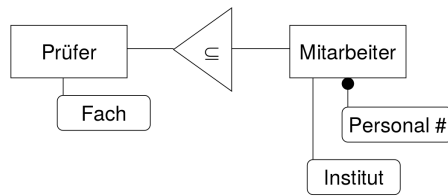
Nicht übernommen: IST-Beziehung – ersetzt durch *Typkonstruktor*

### EER – Typkonstruktor

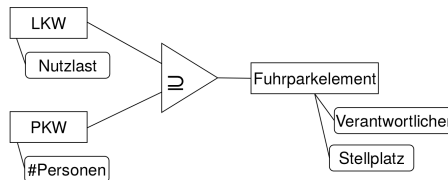
Ersetzt Spezialisierung, Generalisierung, Partitionierung

Eingabetypen mit Dreiecksbasis verbunden (bei Generalisierung spezielle Typen, bei Spezialisierung/Partitionierung allgemeine Typen)

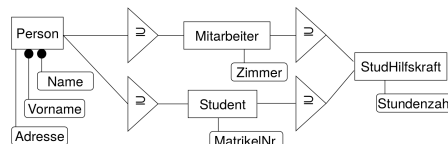
Ausgabetyphen mit Spitze verbunden



Spezialisierung



Generalisierung



Mehrfache Spezialisierung

### Prüfungsfragen

1. Wie ist die Semantik von Datenmodellen definiert?
2. Geben Sie ein Beispiel für mehrstellige Beziehungen an und erläutern Sie, warum der Sachverhalt mit mehreren zweistelligen Beziehungen nicht korrekt darstellbar wäre.
3. Welche semantischen Beziehungen aus dem EER-Kontext kennen Sie? Erläutern Sie die Unterschiede und geben Sie jeweils ein Beispiel an.

## V. RELATIONENENTWURF

### Integritätsbedingungen

Schlüssel und Fremdschlüssel einzige Integritätsbedingungen im relationalen Modell

### Formalisierung Relationenmodell

Universum  $U$ : nichtleere endliche Menge  $U$   
 (z.B.  $U = \{\text{Name, Alter, Haarfarbe, ...}\}$ )

Attribut:  $A \in U$

Domäne  $D \in \{D_1, \dots, D_m\}$ : endliche, nichtleere Menge  
 (z.B.  $D_1 = \{1, 2, 3, \dots\}$ ,  $D_2 = \{\text{schwarz, rot, blond}\}$ )

Attributwert:  $\text{dom} : U \rightarrow D$ : total definierte Funktion,  $\text{dom}(A)$   
 Domäne von  $A$ ,  $w \in \text{dom}(A)$  Attributwert für  $A$   
 (z.B.  $\text{dom}(\text{Haarfarbe}) = \{\text{schwarz, rot, blond}\}$ )

Relationenschema:  $R \subseteq U$

Tupel ( $t$  in  $R = \{A_1, \dots, A_n\}$ ):  $t : R \rightarrow \bigcup_{i=1}^n D_i$

Relation ( $r$  über  $R = \{A_1, \dots, A_n\}$ ): endliche Menge von Tupeln  
 Notation:  $r(R)$  (Relation  $r$ , Relationenschema  $R$ )

r	Name	Alter	Haarfarbe
	Andreas	43	blond
	Gunter	42	blond
	Michael	25	schwarz

Beispiel:

$R = \{\text{Alter, Haarfarbe, Name}\}$

$r$  besteht aus Tupeln  $t_1, t_2, t_3$ ;  $t_1(\text{Name}) = \text{"Andreas"}$  usw.

REL:  $\text{REL}(R) = \{r \mid r(R)\}$

Menge aller  $r$ , die Relation von  $R$  sind

( $r$  oben:  $r \in \text{REL}(\{\text{Name, Alter, Haarfarbe}\})$ ,  
 aber  $r \notin \text{REL}(\{\text{Name, Vorname}\})$ )

Datenbankschema:  $S = \{R_1, \dots, R_p\}$

Menge von Relationenschemata

Datenbank (über  $S$ ): Menge von Relationen

$d = \{r_1, \dots, r_p\}$  und  $r_i(R_i)$

$d(S)$  Datenbank  $d$  über  $S$

### Lokale Integritätsbedingung

= Abbildung aller möglichen Relationen zu einem Schema auf true oder false

$b : \{r \mid r(R)\} \rightarrow \{\text{true, false}\} \ (b \in B)$

Erweitertes Relationenschema:  $\mathcal{R} = (R, B)$

Abkürzung:

$r(R)$  –  $r$  ist Relation von  $R$

$r(\mathcal{R})$  –  $r$  ist Relation von  $R$ , und  $b(r) = \text{true}$  für alle  $b \in B$

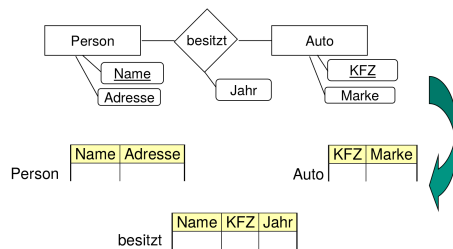
SAT:  $\text{SAT}_R(B) = \{r \mid r(\mathcal{R})\}$

Menge aller Relationen über erweitertem Relationenschema  
 (SAT = *satisfy*)

### Prüfungsfragen

1. Wie definieren wir
  - (a) Relation,
  - (b) Relationenschema,
  - (c) Integritätsbedingung?

## VI. ABBILDEN - ER ZU RELATIONAL



### Abbildungsziel

Kapazitätserhaltende Abbildung: In beiden Fällen gleich viele Instanzen darstellbar

Kapazitätserhöhende Abbildung: relational mehr darstellbar als mit ER

Kapazitätsvermindernde Abbildung: relational weniger darstellbar als mit ER

### Abbildungsregeln

Entity-/Beziehungstypen  $\rightsquigarrow$  Relationenschemata

Attribute  $\rightsquigarrow$  Attribute Relationenschema

Schlüssel  $\rightsquigarrow$  übernehmen

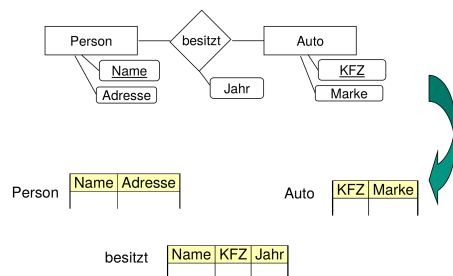
Kardinalitäten  $\rightsquigarrow$  Schlüsselwahl

Ggf. Relationenschemata und Entity-/Beziehungstypen verschmelzen

Einführung neuer Fremdschlüsselbedingungen

1. Teil der Schema-Definition
2. Entstehen bei Abbildung von Relationships
3. Ersetzen Linie von Relationship zu Entity

Beziehungstyp  $\rightsquigarrow$  Relationenschema mit Attributen des Beziehungstyps und Primärschlüssel der beteiligten Entity-Typen



### Prüfungsfragen

1. Warum gibt es im ER-Modell keine Fremdschlüssel?
2. Was bedeutet "kapazitätserhaltende Abbildung"? Geben Sie Beispiele.
3. Wiedergabe der unterschiedlichen Beziehungsabbildungen (1:1, 1:n, m:n)
4. In welchen Fällen lässt sich das Schema optimieren? Was bedeutet Optimierung hier?
5. Wie lassen sich mengenwertige Attribute abbilden?
6. Warum ist Abbildung der folgenden Konstrukte vom ER-Modell ins Relationenmodell problematisch? Rekursive Beziehungen, Partitionierung, Generalis.

## VII. RELATIONALER DATENBANKENTWURF

### Universalrelation

Universalrelation (von  $R_1, \dots, R_n$ ):  $R = R_1 \bowtie \dots \bowtie R_n$

Universalschlüssel: Schlüssel der Universalrelation

Beispiel:  $R_1, R_2, R_3$ :

PANr	PLZ	PLZ	Ort	Ort	Bundesland

$R_1 \bowtie R_2 \bowtie R_3$ :

PANr	PLZ	Ort	Bundesland

### Funktionale Relation

In Relation  $R(X, Y)$  ist  $Y$  von  $X$  funktional abhängig (schreibe  $X \rightarrow Y$ ), falls zu jedem  $X$ -Wert genau ein  $Y$ -Wert gehört (z.B. Buchtitel, ISBN-Nummer oder Stadt, Bundesland)

$\rightsquigarrow$  "X bestimmt Y"

$F$ : Menge von FDs (*functional dependencies*),  $f \in F$  einzelne FD

$F$  impliziert  $f$ :  $F \models f$

Hülle:  $F_R^+ = \{f \mid (f \text{ FD über } R) \wedge F \models f\}$

Transitiv:  $PLZ \rightarrow Ort \rightarrow Bundesland$

$\rightsquigarrow PLZ \rightarrow Bundesland$

Projektiv:  $ISBN \rightarrow Autor \text{ Verlag}$

$\rightsquigarrow ISBN \rightarrow Autor$

Akkumulativ:  $ISBN \rightarrow Verlag \text{ Autor, Autor} \rightarrow Straße \text{ Ort}$

$\rightsquigarrow ISBN \rightarrow Verlag \text{ Autor Straße}$

Äquivalente FD-Mengen (Überdeckungen):  $F \equiv G$  falls  $F^+ = G^+$

### Anomalien

VNr	Bez	PANr	Name	Büro
123	Datenbanksysteme	321	Böhm	367
456	Datenhaltung in der Cloud	321	Böhm	367
789	Workflow-Management	432	Mülle	370

Updateanomalie: Büro von Böhm ändert sich

$\rightsquigarrow$  Änderung mehrerer Einträge

$\rightsquigarrow$  Aufwendig, fehleranfällig. Wie vermeiden?

Einfügeanomalie: Neuer Dozent ohne VL (NULL-Werte)

$\rightsquigarrow$  Was wenn VNr Schlüssel?

Löschanomalie: Mülle hält Workflow nicht mehr

$\rightsquigarrow$  Tupel löschen  $\rightsquigarrow$  Müßel-Information verloren

### Abhängigkeitstreue

Beispiel: (InvNr, Titel, ISBN, Autor)

oder (InvNr, Titel, ISBN), (ISBN, Autor)

oder (InvNr, Titel, ISBN), (ISBN, Autor)?

Abhängigkeitstreue: Alle gegebenen Abhängigkeiten sind durch Schlüssel repräsentiert

### Verbundtreue

Originalrelationen können durch Verbund der Basisrelationen wiedergewonnen werden



## Entwurfsziel

Relationenschemata, (Fremd-)Schlüssel so wählen, dass

1. alle Anwendungsdaten aus Basisrelation hergeleitet werden können (*Verbundtreue*)
2. nur semantisch sinnvolle und konsistente Anwendungsdaten dargestellt werden können (*Abhängigkeitstreue*)
3. möglichst nicht-redundante Daten

## Erste Normalform

Nur atomare Attribute in Relationenschemata

## Zweite Normalform

Volle FD:  $\beta$  ist voll funktional abhängig von  $\alpha$ , wenn aus  $\alpha$  kein Attribut entfernt werden kann, so dass FD immer noch gilt.

Gegenbeispiel: PLZ, Bundesland  $\rightarrow$  Ort

Partielle FD: liegt vor, wenn ein Nicht-Primattribut voll funktional von einem Teil eines Schlüsselkandidaten abhängt

Zweite NF: keine partiellen Abhängigkeiten  
 $\rightsquigarrow$  Durch Struktur der Abhängigkeiten Redundanzen entdecken

## Dritte Normalform

Transitive Abhängigkeit: Schlüssel  $K$  bestimmt Attributmenge  $X$  funktional, ist selber aber auch Attributmenge  $Y$   
 $\rightsquigarrow$  transitive Abhängigkeit  $\rightarrow X \rightarrow Y$

dritte NF: Keine transitiven Abhängigkeiten zwischen einem möglichen Schlüssel und weiteren nicht-Primattributen

Erreichen durch Elimination von  $Y$  und Kopie von  $X$

3NF impliziert 2NF, da partielle Abhängigkeit Spezialfall von transitiver Abhängigkeit

## Boyce-Codd-Normalform

Relationenschema  $\mathcal{R}$  mit FDs  $F$  ist in BCNF, wenn für jede FD  $\alpha \rightarrow \beta$  eine der folgenden Bedingungen gilt:

1.  $\beta \subseteq \alpha$  (triviale Abhängigkeit)
2.  $\alpha$  Schlüssel von  $\mathcal{R}$  (oder Obermenge eines Schlüssels von  $\mathcal{R}$ )

liefert Zerlegung von  $\mathcal{R}_i$  in  $\mathcal{R}_{i1} = (\alpha \cup \beta)$ ,  $\mathcal{R}_{i2} = \mathcal{R}_i - (\alpha \cup \beta)$   
 $(F \ni f : \alpha \rightarrow \beta, \beta \text{ maximal})$

## Minimalität

Ziel: Kriterien mit möglichst wenigen Relationenschemata erreichen

## Dekomposition

Prinzip: Immer wenn  $X \rightarrow Y \rightarrow Z$  wird Relation zerlegt erreicht nur 3NF und Verbundtreue

Normalisierung: Falls  $K \rightarrow X \rightarrow Y$ , dann  $Y$  aus  $R$  entfernen und mit  $X$  in neues Relationenschema stecken

Beispiel:  $U = \{\text{PANr, PLZ, Ort, Land, Staat}\}$ ,  
 $F = \{\text{PANr} \rightarrow \text{PLZ}, \text{PLZ} \rightarrow \text{Ort}, \text{Ort} \rightarrow \text{Land}, \text{Land} \rightarrow \text{Staat}\}$   
 $\rightsquigarrow (U, K(F)) = (\{\text{PANr, PLZ, Ort, Land, Staat}\}, \{\{\text{PANr}\}\})$   
 Betrachte  $\text{PANr} \rightarrow \text{Land} \rightarrow \text{Staat}$ . Neue Relationen:

1.  $R_1 = \{\text{Land, Staat}\}$
2.  $R_2 = \{\text{PANr, PLZ, Ort, Land}\}$

Wiederholen mit  $R_2$

Vorteile: 3NF, Verbundtreue

Nachteile: Keine Abhängigkeitstreue, keine Minimalität, reihenfolgeabhängig, NP-vollständig (Schlüsselsuche)

## Syntheseverfahren

Prinzip: Synthese formt Original-FD-Menge  $F$  in Menge von Schlüsselabhängigkeiten  $G$  so um, dass  $F \equiv G$

Abhängigkeitstreue integriert

3NF und Minimalität werden reihenfolgeunabhängig erreicht

polynomielle Zeitkomplexität

Übersicht:

1. Redundanzen eliminieren:  
Entfernen unnötiger FDs und Attribute ( $f$  überflüssig wenn  $F \equiv F - \{f\}$ , überflüssige Attribute später)
2. FDs zu Äquivalenzklassen zusammenfassen:  
FDs in selber Klasse, wenn sie äquivalente linke Seiten haben  $\rightsquigarrow$  ein Relationenschema pro Äquivalenzklasse

Beispiel:  $F = \{A \rightarrow B, AB \rightarrow C, A \rightarrow C, B \rightarrow A, C \rightarrow E\}$

1. Redundante FDs:  $A \rightarrow C$   
Stand:  $F' = \{A \rightarrow B, AB \rightarrow C, B \rightarrow A, C \rightarrow E\}$
2. Überflüssige Attribute:  $B$  in  $AB \rightarrow C$   
Stand:  $F'' = \{A \rightarrow B, A \rightarrow C, B \rightarrow A, C \rightarrow E\}$   
 $\underbrace{\hspace{10em}}_{\text{Äquivalenzklasse}}$
3. Ergebnis Relationenschema:  
 $(ABC, \{\{A\}, \{B\}\}), (CE, \{\{C\}\})$

## Mehrwertige Abhängigkeiten

Mehrwertige Abhängigkeit (*multi value dependency, MVD*):

Jeder Wert des abhängigen Attributes kommt in Kombination mit allen Werten der anderen Attribute vor

Redundanzbehaftet

Beispiel:

Kurs	Buch	Dozent
AHA	Silberschatz	John D
AHA	Nederpelt	John D
AHA	Silberschatz	William M
AHA	Nederpelt	William M

Neues Buch: für jeden Dozenten anlegen  $\rightsquigarrow$  MVD

## Vierte Normalform

Beispiel: Relation mit Attributen *Name, Neffe, Hobby*

Es gelte MVD:  $\text{Name} \twoheadrightarrow \text{Neffe}$

Wenn

(Heinrich, Martin, Autos) und (Heinrich, Thomas, Basteln)

$\in r$ , dann auch

(Heinrich, Martin, Basteln) und (Heinrich, Thomas, Autos)

Formal:  $r$  genügt MVD  $X \twoheadrightarrow Y \Leftrightarrow$

$\forall t_1, t_2 \in r : [(t_1 \neq t_2 \wedge t_1(X) = t_2(X))$

$\Rightarrow \exists t_3 \in r : t_3(X) = t_1(X) \wedge t_3(Y) = t_1(Y) \wedge t_3(Z) = t_2(Z)]$

4NF: solche MVDs aufspalten

Trivial, wenn keine weiteren Attribute im zugehörigen Schema

## Prüfungsfragen

1. Erläutern Sie die folgenden Begriffe: Redundanz, Funktionale Abhängigkeit, Normalform, Verbundtreue, Abhängigkeitstreue, Minimalität.
2. Erläutern Sie die Aussage: "Funktionale Abhängigkeiten beinhalten semantische Informationen."
3. Welche Anomalien kennen Sie? Erläutern Sie für jede dieser Anomalien, warum Sie störend ist.
4. Warum braucht man für Verbundtreue Kriterien, für Abhängigkeitstreue jedoch scheinbar nicht?
5. Welche Normalformen kennen Sie? Sagen Sie umgangssprachlich, wie sie definiert sind.



## VIII. RELATIONALE DATENBANKSPRACHEN

### Aggregatfunktionen

Prinzip: Berechnung eines Werts aus Werten eines Attributs

Join (natural): Kartesisches Produkt zweier Relationen

Weitere in Standard SQL: count(), sum(), min(), max(), avg()

### SQL-Kern

select

Projektionsliste,  
arithmetische Operationen und Aggregatfunktionen  
**select distinct:** keine Dopplungen

from

zu verwendende Relationen, ggf. Umbenennungen

where

Selektions- und Verbundbedingungen  
geschachtelte Anfragen (wieder SFW-Block)

group by

Gruppierung für Aggregatfunktionen

having

Selektionsbedingungen an Gruppen

### Self-Join

Kartesisches Produkt einer Tabelle mit selbst

Beispiel:

```
select * from SNUser eins, SNUser zwei
where eins.Alter < zwei.Alter
```

Vierspaltiges Ergebnis:

```
eins.Name, eins.Vorname,
zwei.Name, zwei.Vorname
```

### Kartesisches Produkt

Verbunde als explizite Operatoren:

```
select * from Kuenstler cross join Titel
```

### Natürlicher Verbund

Oft besser als herkömmliche Formulierung, weil

1. übersichtlicher
2. weniger fehleranfällig (man vergisst leicht Attribut, wenn man alle aufzählen muss)

```
select * from Kuenstler natural join Titel
```

### Theta-Join

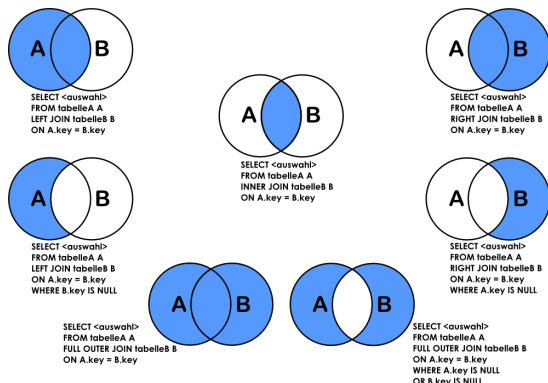
Verbund über Verbundbedingungen

```
select * from Kuenstler
join Titel on Kuenstler.KID = Titel.KID
```

Beispiel: (AutoModell, AutoPreis), (BootModell, BootPreis)

Kunde will Boot und Auto, aber Boot soll billiger sein als Auto  
 $\leadsto \text{Auto} \bowtie_{\text{AutoPreis} > \text{BootPreis}} \text{Boot}$

### Outer, Left, Right Join



where

Trivial: **where** Buecher.Titel = "Titel"

Verbundbedingung (alternativ):

```
select Buecher.Titel, Buecher_Stichwort.Stichwort
from Buecher, Buecher_Stichwort
where Buecher.ISBN = Buecher_Stichwort.ISBN
```

**like:** Ungewissheitsselektion (RegEx)

**where** attribut **like** spezialkonstante

**in: where** ISBN **in** (select ISBN from Empfiehlt)

### Mengen – Vereinigung

```
select A, B, C from R1 union
select A, C, D from R2
```

**union:** Duplikate werden eliminiert

**union all:** Duplikate werden behalten

### Mengen – Differenz

Alle Mitarbeiter, die keine Studierenden sind:

```
select PANr from Mitarbeiter
EXCEPT select PANr from Studenten
```

### Mengen – Durchschnitt

Alle Mitarbeiter, die auch Studenten sind:

```
select PANr from Mitarbeiter
INTERSECT select PANr from Studenten
```

### Umbenennung

```
select ISBN, Preis * 1.44 as DollarPreis
from BuchVersionen
```

### Grouping

Marke	Datum	Bundesland	Anzahl
BMW	07.01.1994	Hessen	28
BMW	08.01.1994	Bayern	37
BMW	07.01.1994	Saarland	41
Opel	07.01.1994	Hessen	48
Opel	08.01.1994	Bayern	62
Opel	08.01.1994	Saarland	5
Opel	09.01.1994	Saarland	95
Audi	07.01.1994	Hessen	55
Audi	08.01.1994	Bayern	52
Audi	09.01.1994	Bayern	27
Audi	10.01.1994	Bayern	62

```
select Marke, sum(Anzahl)
from Zulassungen
group by Marke
```

```
select Marke, max(Anzahl)
from Zulassungen
group by Marke
```

**having-Bedingung:**

```
select PANr, sum(Entlohnung)
from anstellungen
group by PANr
having sum(entlohnung) > 10000
```

### Quantoren

**any/some:**

```
select PANr, ImmaDatum
from Studenten
where MatNr = any (select MatNr from Prueft)
```

**all:**

```
select Name from Kunde, Bestellung
where Kunde.id = Bestellung.KundeID
and bestellwert > ALL (SELECT avg(bestellwert)
from Bestellung group by KundeID)
```

## Sortieren

**order by**-Klausel:

```
select MatNr, Note from Prueft
  where V_Bez = 'DBS'
  order by Note asc
```

alternativ: **desc**

## Nullwerte

Vergleiche mit Nullwert: **unknown** statt **true** oder **false**  
 $\leadsto A = A$  keine Tautologie!

## Update

```
update relation set attribut1 = wert, ...
  [where bedingung]
```

## Delete

```
delete from relation [where bedingung]
```

## Insert

```
insert into Kuenstler(KID, NAME, LAND, JAHR)
  values (1022, 'Raul-Seixas',
         1945, 'Brasilien')
```

### Prüfungsfragen

1. Formulieren diverser (komplexer) SQL-Anfragen
2. Vorgegebene geschachtelte Anfrage als nicht-geschachtelte schreiben
3. Welche Join-Varianten kennen Sie?
4. Geben Sie ein Beispiel an, in dem ein Self-Join sinnvoll ist.
5. Was ist der Zusammenhang zwischen Vereinigung und Outer Join?
6. Was ist eine Umbenennung im SQL-Kontext? Wann wird sie gebraucht?
7. Geben Sie ein sinnvolles Beispiel für eine Anfrage an, die eine having-Klausel hat.
8. Geben Sie ein Beispiel für eine Anfrage mit einer having-Klausel an, bei der man
  - (a) die Klausel durch eine where-Klausel ersetzen kann,
  - (b) das nicht kann.
9. Erläutern Sie, warum im SQL-Kontext " $A=A$ " keine Tautologie ist.

## IX. NEBENLÄUFIGKEIT, TRANSAKTIONEN

### Transaktion

Partiell geordnete Folge von Lese- und Schreibzugriffen auf Datenobjekte (mit Commit oder Abort am Ende)

ACID Eigenschaften:

Atomicity: Entweder alles oder gar nichts ausführen

Consistency: Integritätsbedingungen bleiben erhalten

Isolation: Nutzer hat Eindruck, er wäre alleine

Durability: Änderungen sollen dauerhaft sein

### Synchronisation

Viele Nutzer sollen Daten gleichzeitig lesen und schreiben können

$\leadsto$  Konsistenz sicherstellen  $\leadsto$  **Synchronisationskomponente**

Serielle Ausführung:

- + Konsistenz immer gewährleistet
- extreme Wartezeiten, schlechte Ressourcenausnutzung

### Unkontrollierte nicht-serielle Ausführung: Probleme

#### Lost Update

Programm  $T_1$  transferiert 300 EUR von Konto  $A$  nach  $B$ ,

Programm  $T_2$  schreibt Konto  $A$  3% Zinsen gut

$\leadsto$  Zinsen aus  $S_5$  von  $T_2$  verloren, weil  $T_1$  in  $S_6$  überschreibt

Schritt	$T_1$	$T_2$
1	Read(A, a1)	
2	a1 := a1-300	
3		Read(A, a2)
4		a2 := a2 *1.03
5		Write(A, a2)
6	Write(A, a1)	
7	Read(B, b1)	
8	b1 := b1 + 300	
9	Write(B, b1)	

#### Dirty Read

Commit, Abort

$T_2$  schreibt Zinsen gut basierend auf einem Wert, der nicht zu einem konsistenten Zustand gehört, denn später erfolgt Abort von  $T_1$

Schritt	$T_1$	$T_2$
1	Read(A, a1)	
2	a1 := a1-300	
3	Write(A, a1)	
4		Read(A, a2)
5		a2 := a2 *1.03
6		Write(A, a2)
7		<b>commit</b>
8	Read(B, b1)	
9	...	
10	<b>abort</b>	

### Non-Repeatable Reads

Programm liest Datenobjekt mehr als einmal und sieht dabei Änderung durch anderes Programm

Schritt	$T_1$	$T_2$
1	Read(A, a1)	
2	a1 := a1-300	
3	Write(A, a1)	
4		Read(A, a2)
5		a2 := a2 *1.03
6		Write(A, a2)
7	Read(A, a3)	
8	...	

### Phantom

Berechnung von Änderung auf veralteten Werten

### Konflikt

Zwei Operationen  $p, q$  konfliktieren

$\Leftrightarrow p, q$  greifen auf selbes Datenobjekt zu und  $p$  oder  $q$  ist Schreiboperation

In einer Transaktion müssen konfliktierende Operationen **geordnet sein** (andere nicht zwingend)

## Histories

Vollständige Historie: Menge von Transaktionen und Ausführungsordnung (nebenläufige Verzahnung, Ordnung konfligierender Operationen zwischen Transaktionen)

Historie: Präfix einer vollständigen Historie

Committed Projection ( $C(H)$ ):  $H$  nach Entfernen aller nicht-committierten Operationen

Eine Eigenschaft von Histories ist **prefix commit closed**  
 $\Leftrightarrow (H \text{ erfüllt Eigenschaft} \Rightarrow C(H') \text{ erfüllt Eigenschaft})$

## Konfliktäquivalenz (CSR)

$H, H'$  (Konflikt-)Äquivalent, wenn

1. gleiche Transaktionen, gleiche Operationen
2. gleiche Ordnung konfligierender Operationen (gleiche Konfliktrelation)

## Serialisierbarkeit

$H$  serialisierbar  $\Leftrightarrow C(H) \equiv H_S$  (serielle History)

Serialisierbarkeitsgraph (Abhängigkeitsgraph):

Knoten = Transaktionen

(gerichtete) Kante von  $T_1$  nach  $T_2$  wenn  $op_1$  und  $op_2$  konfligieren und  $op_1 < op_2$

**Theorem**: Schedule ist serialisierbar, wenn entsprechender Abhängigkeitsgraph zyklfrei ist

Konflikt-Serialisierbarkeit ist prefix commit-closed

**Ansatz nicht praktikabel**:

1. Serialisierbarkeit nur im Nachhinein überprüfbar
2. Administrativer Overhead zu hoch: Abhängigkeiten zu bereits terminierten Transaktionen berücksichtigen

## Rücksetzbarkeitsklassen

Rücksetzbar (RC): Commit für  $T_j$  erst erlaubt, wenn alle  $T_i$  von denen  $T_j$  liest, committed sind (Abort darf Semantik von bereits committierten Transaktionen nicht verändern).

Avoid cascading aborts (ACA): Nur Objekte von bereits committierten Transaktionen lesen.

Striktheit (ST): Objekte von noch nicht committierten Transaktionen dürfen weder gelesen noch überschrieben werden (ermöglicht einfache Implementierung des Rücksetzens)

## Locking

Lock für jedes Datenobjekt und jede Operationsart  
 Notation:  $rl_i[x]$ ,  $wl_i[x]$

Aber: Sperrdisziplin alleine reicht für Korrektheit nicht aus!

### Zwei-Phasen-Sperrprotokoll (2PL):

1. Locks werden hinzugenommen
2. Locks werden freigegeben

$\leadsto$  Serialisierbarkeit sichergestellt

Deadlocks sowie kaskadierende Abbrüche weiterhin möglich

### Strenges Zwei-Phasen-Sperrprotokoll (S2PL):

Atomare Freigabephase am Ende der Transaktion

$\leadsto$  Zusätzlich ACA: Vermeidung kaskadierender Abbrüche

### Konservatives Zwei-Phasen-Sperrprotokoll (C2PL):

Atomare Anforderungsphase zu Beginn der Transaktion

$\leadsto$  Zusätzlich: Vermeidet Deadlocks

### CS2PL:

Kombination aus streng und konservativ: Atomare Anforderungs- und atomare Freigabephase

$\leadsto$  Serialisierbarkeit, ACA, Deadlockfreiheit

**Aber**: Jede Einschränkung schränkt auch die Zahl der möglichen Histories ein und verringert damit den möglichen Grad der Parallelität!

## Prüfungsfragen

1. Was ist Isolation? Was ist der Zusammenhang zwischen Isolation und Serialisierbarkeit?
2. Welche Probleme können bei unkontrollierter nebenläufiger Ausführung von Transaktionen auftreten?
3. Beispiele für Lost Updates, Non-Repeatable Reads usw. angeben, die bestimmte Bedingungen erfüllen
4. Warum ist es wichtig, dass unser Korrektheitskriterium für Histories prefix commit closed ist? Erklären Sie, warum Konflikt-Serialisierbarkeit prefix commit closed ist.
5. Ist eine gegebene History serialisierbar/recoverable/cascadeless?
6. Haben zwei Konflikt-äquivalente Histories stets die gleichen Reads-from-Beziehungen?
7. Warum verwendet man in der Regel nicht den Serialisierbarkeitsgraphen, um Serialisierbarkeit sicherzustellen?
8. Bei Deadlocks wird in der Regel eine Transaktion zurückgesetzt. Kann es vorkommen, dass die gleiche Transaktion mehrmals/beliebig oft zurückgesetzt wird? Wenn ja, was kann man jeweils dagegen tun?
9. Geben Sie ein Beispiel für eine serialisierbare Ausführung, bestehend aus drei Transaktionen, mit folgender Eigenschaft an: Die zeitliche Reihenfolge der Commits ist  $c_1$  vor  $c_2$  vor  $c_3$ , die der äquivalenten seriellen Ausführung jedoch  $c_3$  vor  $c_2$  vor  $c_1$ .
10. Um einen Deadlock aufzulösen muss eine der beteiligten Transaktionen zurückgesetzt werden. Welche Kriterien sind Ihres Erachtens nach sinnvoll, um diese Auswahl zu treffen?

## X. CLOUDSYSTEME – KONSISTENZ

### Verteilung

Vorteile (scheinbar):

1. Leselastverteilung
2. Beschleunigung (durch höhere Lokalität)
3. Höhere Ausfallsicherheit

Nachteile:

1. Transaktionen müssen auf Knoten gleich angeordnet sein
2. Widerspruchsfreie Anordnungsentscheidungen nötig für Konfliktfreiheit  $\leadsto$  schlechte Skalierbarkeit
3. Für Konsistenz müssen alle Knoten verfügbar sein  $\leadsto$  geringere Ausfallsicherheit

$\leadsto$  Netzwerkpartitionierung

**CAP-Theorem**: Wenn Netzwerkpartitionierung möglich, dann sind hohe Verfügbarkeit und Datenbestandskonsistenz unvereinbar

### Eventual Consistency

“Wenn ab Zeitpunkt keine Änderungen mehr, dann werden irgendwann alle Lesezugriffe gleichen Wert zurückliefern”

Alternativ: “... dann werden irgendwann alle Lesezugriffe zuletzt geschriebenen Wert zurückliefern”

Beispiel (social network): Netzwerkpartition

Starke Konsistenz: Vorübergehend keine Postings möglich  
 Eventual Consistency: User kann Posting schreiben, Fol-lower sehen es sobald möglich

## Prüfungsfragen

1. Geben Sie die Probleme mit dem klassischen, starken Konsistenzbegriff im verteilten Fall wieder.
2. Bekommt man mit *eventual consistency* irgendeine Form von Sicherheit? Begründen Sie Ihre Antwort.
3. Warum kann man im Bank-Kontext in manchen Situationen doch auf starke, klassische Konsistenz verzichten?
4. Geben Sie ein weiteres Beispiel für eine Folge von Operationen, deren Anordnung egal ist.

## XI. CLOUDSYSTEME – FUNKTIONALITÄT

### Was ändert sich in der Cloud?

- Physischer Entwurf muss automatisch erfolgen
- Obligatorische Datenverteilung
- Anfrageauswertung in Gegenwart anderer Anfragen  
→ entsprechende Planung
- Unterschiedliche QoS-Vereinbarungen mit unterschiedlichen Dienstnehmern
- Plötzliche extreme Zunahme von Zugriffen eines Dienstnehmers i.A. nicht vorhersehbar  
→ Infrastruktur sollte damit umgehen können
- Secure Storage*: Verschlüsselung der Daten, trotzdem soll Dienstanbieter möglichst großen Teil der Anfrageauswertung übernehmen

### Relationale Algebra

- Projektion: Optimierung: bei vielen Projektionen hintereinander reicht die zuletzt ausgeführte auch allein:  
 $\pi[\text{KName}](\pi[\text{KName}, \text{Land}](\text{Kuenstler})) \rightsquigarrow \pi[\text{KName}](\text{Kuenstler})$
- Selektion: Optimierung: Selektionen lassen sich beliebig vertauschen, manchmal auch Projektion und Selektion
- Verbund: Kommutativ, Assoziativ
  - Nested-Loop Join: Teuer, da pro Eintrag links über alle rechten Einträge iteriert wird
  - Merge Join: Beide Relationen sortieren, dann Eintrag für Eintrag

### Blockierende/Nichtblockierende Operatoren

- Operator blockiert  $\Leftrightarrow$  Ergebnis des Operators muss vor Ausführung des nachfolgenden vollständig berechnet sein (z.B. Sort-Operator)

### Histogramme

- Zeigt Auftrittshäufigkeit eines Intervalls
- Equi-Width-Histogramm: Breite aller Buckets gleich
- Equi-Depth-Histogramm: Auftrittshäufigkeit aller Buckets gleich
- Nützlich bei ein-Attribut-Anfragen, sonst nicht so:
- Mehrdimensionale Histogramme schwer konstruierbar und wartbar, Anzahl Attributkombinationen exponentiell wachsend zur Anzahl der Attribute

### Synchroner und asynchroner Zugriff

- Synchron: innerhalb einer Transaktion
- Asynchron: mehrere Transaktionen

### Service-Level Agreements

- Vereinbarung zwischen Client und Server bzgl. Dienstaufführung  
"Antwort innerhalb von 300ms für 99,9% der Aufrufe bei 500 Zugriffen pro Sekunde"

### Zustände

- Zustandslos: z.B. Umrechnungsdienst
- Zustandsbehaftet: z.B. Ausführung Geschäftsprozess

### Quorum

- Szenario: Replikation mit  $n$  Knoten  
→ Wie Konsistenz sicherstellen? Was, wenn nicht alle Knoten verfügbar?
- Quorum Consensus:
  - Lesen: Lese Mindestanzahl von Versionen ( $R$ ), nehme aktuelle
  - Schreiben: Aktualisiere Mindestanzahl von Kopien ( $W$ )
  - Für Lesen/Schreiben muss eine festgelegte Anzahl an Knoten zustimmen. Forderungen ( $N$  = Anzahl Knoten):

1.  $Q_R + Q_W > N$
2.  $2Q_W > N$

### P2P

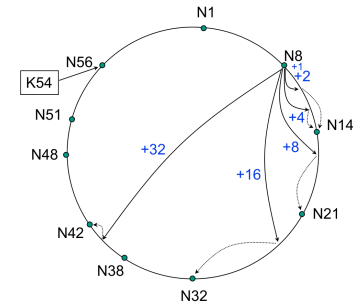
peer to peer-Systeme:

- Jeder Knoten für Ausschnitt des Schlüsselraums verantwortlich
- Verwaltung von (Schlüssel, Wert)-Paaren
- (put, get)-Interface
- Zu Größe des Schlüsselraums logarithmischer Suchaufwand

Beispiel: Chord

Zentrale Datenstruktur: *identifier circle*, *chord ring*

- Suche: Jeder Knoten hat *finger table*,  $i$ -ter Eintrag von Knoten  $n$ :  $\text{successor}(n + 2^{i-1})$  ( $m$  Anzahl Bits)



Replikation über *chained replication*: Schlüssel nicht nur bei einem Knoten, sondern auch bei  $k$  Nachfolgern einfügen

Heterogenität: Knoten können unterschiedlich leistungsstark sein (ggf. unterschiedliche Zuständigkeitsbereiche, unterschiedliche Last)

### Dynamo

- Key-Value-Store
- get-/put-Interface
- Objekte BLOBs  $\rightsquigarrow$  kein DB-Schema  $\rightsquigarrow$  Interpretieren nötig
- Keine Isolation  $\rightsquigarrow$  keine totale Konsistenz
- Schreibzugriff jeweils nur für ein Objekt

Problem	Technik	Vorteil
Partitionierung	Consistent Hashing	Skalierbarkeit, inkrementell
Hohe Verfügbarkeit für das Schreiben	Vector Clocks mit Abgleich beim Lesen	
Umgang mit vorübergehenden Ausfällen	Sloppy Quorum mit hinted handoff	Hohe Verfügbarkeit und Dauerhaftigkeit
Recovery	Anti-Entropy	Synchronisation läuft im Hintergrund ab.
Erkennen von Ausfällen	Gossip-basierte Protokolle	Deckt Anforderung 'Symmetrie' ab.

### Dynamo – Vector Clocks

- Ziel: eventual consistency
- Liste von (Knoten, Zähler)-Paaren (eine Liste pro Version)  $\rightsquigarrow$  Erfassung der Zusammenhänge zwischen Versionen
- Quorum-basierte Techniken  $\rightsquigarrow$  Inkonsistenzen vermeiden
- vector clock-basierte Techniken  $\rightsquigarrow$  Inkonsistenzen erkennen und auflösen
- Unterschiedliche Knoten können Schreiboperationen absetzen  
→ Differenzierung
- Version 1 ist Vorgänger von Version 2, wenn jeder Zähler in List von V1 einen kleineren Wert hat als in der von V2
- Update (put) muss festlegen, welche Version aktualisiert werden soll
- Get gibt i.A. mehrere Versionen zurück

## Scale Independence

Anfrage ist *scale-independent*

↪ Laufzeitverhalten unabhängig von DB-Größe

Anfragenklassifikation nach Aufwand:

1. Klasse I (konstant):  
z.B. ID-basierter Zugriff, **LIMIT**-beschränkte Anfragen
2. Klasse II (beschränkt):  
Explizite Begrenzung liegt vor  
Als Kardinalität im erweiterten DB-Schema darstellbar
3. Klasse III ((sub-)linear):  
z.B. Ausgabe aller Kunden/Produkte
4. Klasse IV (superlinear):  
z.B. Clustering-Algo, der Self-Join der zugrundeliegenden Relation ausführt

↪ **PIQL** (*performance insightful query language*) - Scale Independent durch Erweiterungen und Beschränkungen der Anfragesprache

## Ergebnisgröße

Wie bestimmte Größe des Anfrageergebnisses garantieren?

↪ **LIMIT**, Pagination, Berücksichtigung von Fremdschlüsselbeziehungen, Erweiterung DB-Schema um Kardinalitäten

## Physische Optimierung

Zwei Arten von physischen Operatoren:

1. *remote operator*: Zugriffe auf key-value store und elementare Verarbeitungsschritte
2. Client-seitige Operatoren für Query-Logik

Remote Operator: Muss explizite Beschränkung der Größe des Zwischenergebnisses enthalten (i.A. stop-Operator in Operator-Darstellung)

Remote-Operatoren:

1. **IndexScan**: Prädikat muss zusammenhängendem Ausschnitt des indexierten Wertebereichs entsprechen, "Sort" muss Sortierreihenfolge des Index sein
2. **IndexForeignKeyJoin**: Beschränkung durch Fremdschlüsseleigenschaft ↪ kein logischer Stop-Operator, linker Teilausdruck enthält Fremdschlüssel
3. **SortedIndexJoin**: Bei Sortierung des Inputs nach Join Key lässt sich aus limit hint-Begrenzung der Anzahl an Datenobjekten pro Schlüssel ableiten

## SLO Compliance-Vorhersage

SLO = *service-level objectives*

Größenbeschränkung Zwischenergebnisse noch keine Garantie für insgesamt beschränkten Aufwand

Wenn anliegende Last sehr groß kann IndexScan-Ausführung beliebig lange dauern

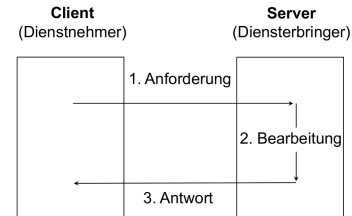
Lookup über Zufallsverteilung (Parameter Tupelgröße, Anzahl erwarteter Tupel)

## Prüfungsfragen

1. Was für Möglichkeiten kennen Sie, den Join zu implementieren? Welche Komplexität haben sie?
2. Welche Möglichkeiten kennen Sie, den Aufwand, den eine Anfrage verursacht, zu reduzieren/begrenzen?

## XII. ANWENDUNGSENTWICKLUNG

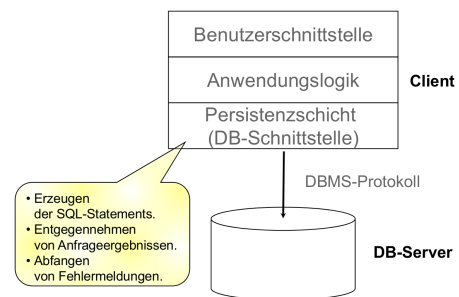
### Client-Server-Architektur



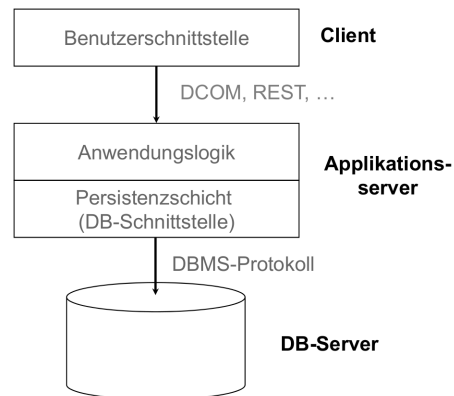
Erfordert

1. Kenntnis über angebotene Dienste
2. Protokoll zur Regelung der Interaktion

### Zwei Schichten-Architektur



### Drei Schichten-Architektur



### Anwendungslogik

Anwendungslogik: Algorithmen, die anwendungsspezifisches Wissen beinhalten

Personal-DB enthält Mitarbeiter-Daten

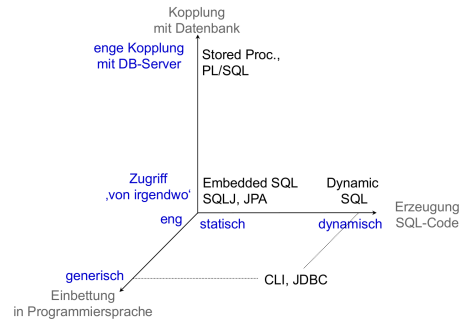
↪ Anwendung: schlägt Teamleiter für konkrete Projekte vor  
↪ Bedeutsamkeit der Fähigkeiten usw. Anwendungsteil

### Cursor-Konzept

Cursor ≡ Iterator

Programmiersprachen: einzelne Datenobjekte als zugrundeliegende Struktur

## Programmiersprachenanbindung



## Prepared Statements

Reduzieren Ausführungszeit, da bereits vorab kompiliert

```
PreparedStatement updateSales =
    con.prepareStatement('UPDATE COFFEES
    SET SALES = ? WHERE COF_NAME LIKE ?');
```

```
updateSales.setInt(1,75);
```

## Gespeicherte Prozeduren

In DB-Server verwaltete und ausgeführte Software-Module in Form von Prozeduren/Funktionen

Aufruf aus Anwendungen/Anfragen heraus

→ Weniger Kontextwechsel in Anwendung

## Variablen und Typen

```
DECLARE preis NUMBER;
```

Stellt sicher, dass Attributtyp in DB identisch zu Typ in Programm ist

## Kontrollfluss

```
DECLARE
    a NUMBER;
    b NUMBER;
BEGIN
    SELECT e, f INTO a, b
    FROM T1 WHERE e > 1;
    IF b = 1 THEN
        INSERT INTO T1 VALUES(b, a);
    ELSE
        INSERT INTO T1 VALUES(b+10, a+10);
    END IF;
END;
```

## Performance Anti-Patterns

Excessive Dynamic Allocation:

Häufige unnötige Objekterstellung/-zerstörung derselben Klasse

The Stifle:

Unpassende DB-Schnittstellennutzung

Circuitous Treasure Hunt:

Abfrage von Relation A, damit Relation B abfragen,...

Sisyphus DB Retrieval:

Riesige Datenmenge abfragen, obwohl nur wenige Einträge nötig

Spaghetti Query:

Mehrere Informationsbedürfnisse in einer Anfrage

Insufficient Caching:

Zu wenig Caching

Wrong Caching Strategy:

Falsche Objekte werden in Cache abgelegt

## Prüfungsfragen

1. Erläutern Sie die Dimensionen des Raums der Möglichkeiten des Zugriffs auf Datenbanken aus Anwendungen heraus.
2. Erläutern Sie die Begriffe
  - (a) Anwendungslogik,
  - (b) Cursor,
  - (c) Call-Level Interface,
  - (d) Host-Variablen.
3. Kann man mit Embedded SQL sicherstellen, dass keine Schema-spezifischen Fehler auftreten? Wenn ja, wie geht es?
4. Was sind die Vorteile von Stored Procedures? Erläutern Sie das Konzept.