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*)

≅ 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 ($\sigma_{\text{KID}=1012}(\text{Titel})$)

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

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

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 ($\rightsquigarrow$ 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

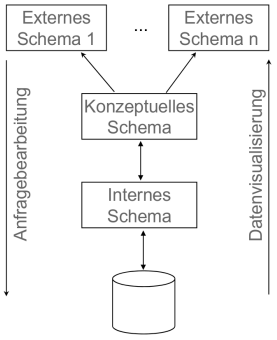
Anfragen deklarativ: Nutzer entscheidet nicht, wie Ergebnis ermittelt wird

Datenunabhängigkeit: DBMS stellt sicher:

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

$\rightsquigarrow$ 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)

$\rightsquigarrow$ **Logische Datenunabhängigkeit**

Datenbankprinzipien – Coddsche 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 (= Isolation)
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?

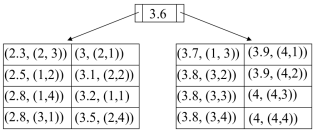
Ähnlichkeitssuche Bilder: Distanz im Merkmalsraum = Maß der Unähnlichkeit

Ziel eines Index: Zahl der zu ladenden Seiten minimieren

Index – B+-tree

= non-clustered primary B+-tree

Beispiel: Student(name, age, gpa, major), B+T für gpa (kleiner=links, größer=rechts, (gpa, (Seite, Eintrag)))



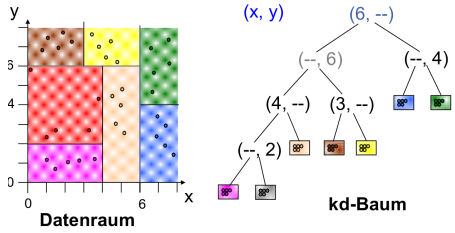
Tom, 20, 3.2, EE	Mary, 24, 3, ECE	Lam, 22, 2.8, ME	Chris, 22, 3.9, CS
Chang, 18, 2.5, CS	James, 24, 3.1, ME	Kathy, 18, 3.8, LS	Vera, 17, 3.9, EE
Bob, 21, 3.7, CS	Chad, 28, 2.3, LS	Kane, 19, 3.8, ME	Louis, 32, 4, LS
Pat, 19, 2.8, EE	Leila, 20, 3.5, LS	Martha, 29, 3.8, CS	Shideh, 16, 4, CS

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

Es müssen nur ein paar kd-Baum-Regionen inspiziert werden, um Resultat zu ermitteln (Abstand zu Region ist untere Schranke)

Implementierung: Priority Queue (Datenobjekte/Baumknoten, sortiert nach Abstand zum Anfragepunkt) initialisiert mit Wurzelknoten; Vorderstes Objekt aufspalten und Teilobjekte einfügen; Ende wenn Punkt vorne in Queue

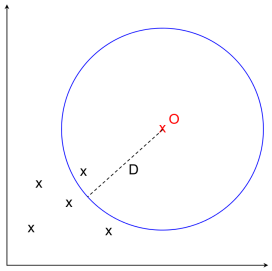
Hier: Baum unbalanciert, Balancierung in Realität für mehrdimensionale Daten

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 – Index-basiert

Punkt ist kein Outlier, wenn k-Abstand < D mit $k = N*(1-p)-1$

Für jeden Punkt:
k-NN Query, dabei stoppen sobald größte noch mögliche k-NN Distanz < D (Baumknoten mit k Objekten und größter Distanz < D)

Viele weitere Ansätze, z.B.
Clustering: Liefert Outlier als Beiprodukt

Clustering – Beispiel Customer Segmentation

Große Kundendatenbank mit Eigenschaften und Käufen

Gesucht: Gruppen von Kunden mit ähnlichem Verhalten finden

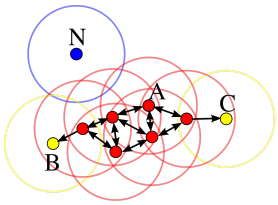
Clustering – DBSCAN

Dichte: Anzahl Objekte pro Volumeneinheit

Dichtes Objekt: mindestens x andere Objekte in Kugel um Objekt mit Radius ϵ (A)

Dichte-erreichbares Objekt: Objekt in ϵ-Umgebung eines dichten Objekts, das selbst nicht dicht ist (B, C)
Clusterrand, Zuordnung zu Clustern ist nichtdeterministisch

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



DBSCAN – Eigenschaften

Komplexität: Lineare, wenn ϵ-Umgebungen vorberechnet wurden (oder mit räumlichem Index in konstanter Zeit bestimmt werden können)
$\rightsquigarrow$ mehrdimensionale Indexstruktur sehr sinnvoll

Rauschen liefert *mögliche* Outlier (DBSCAN erstellt Vorauswahl)

Hochdimensionale Datenräume – Anomalien

Curse of dimensionality

Sparsity: Raum ist nur dünn mit Punkten besetzt

Hierarchische Datenstrukturen ineffektiv: Es müssen immer alle Blätter betrachtet werden

Keine echten Outlier: bei sehr, sehr vielen Dimensionen ist Abstand zweier Datenobjekte fast gleich dem zweier anderer $\rightsquigarrow$ Outlier-Algorithmen liefern mehr oder weniger zufälliges Objekt

$\rightsquigarrow$ 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

$\rightsquigarrow$ 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 $\rightsquigarrow$ 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

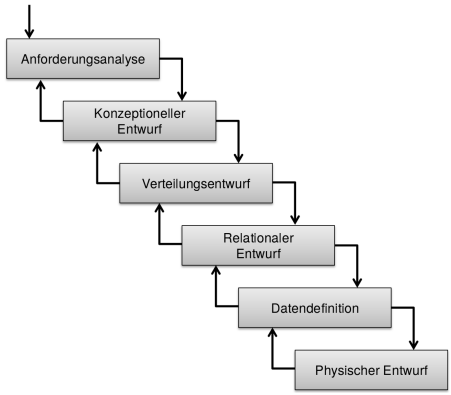
Gewinnung der Konventionen

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

Verzeichnis der vorhandenen Tabellen und Sichten

Selbst wie eine 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)
  add constraint Namekey primary key (Name, Vorname)
```

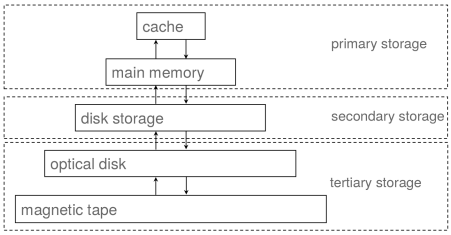
⇝ Änderung Relationsschema im Data Dictionary, existierende Daten werden um **null**-Attribut erweitert

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

⇝ Attribut / Tabelle löschen, dabei gilt:

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

Speicherhierarchie



Index

Für mehrere Attribute möglich

Index für (gpa, name) ≠ Index für (name, gpa)

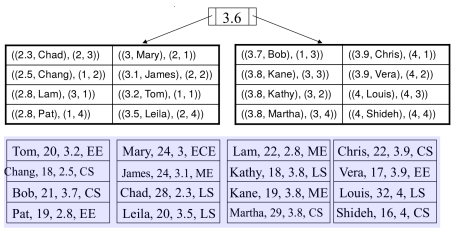
Index kann nachträglich angelegt bzw. gelöscht werden, ohne Daten selbst zu löschen

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

Unique Index zur Simulation von Schlüsselbedingungen



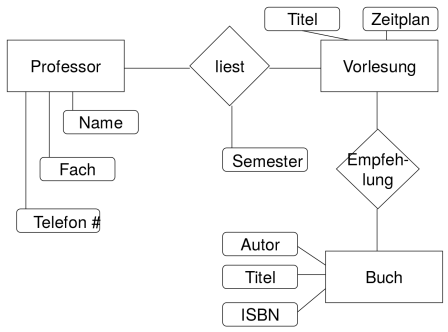
Prüfungsfragen

1. Erläutern Sie anhand eines Anwendungsbeispiels, warum man die Menge der zulässigen Zustände einschränken will.
2. Was ist 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

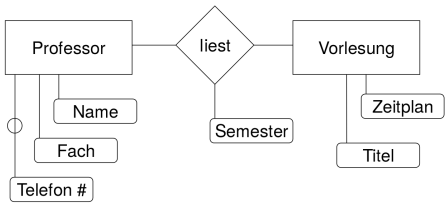
Entity-Relationship-Modelle

Entity: Objekt der Real-/Vorstellungswelt (z.B. Buch)
Relationship: Beziehung zw. Entities (z.B. Schüler hat Buch)
Attribut: Eigenschaft von Entities (z.B. ISBN)



Attribute

Mengenwertig: Durch Doppelrand gekennzeichnet
Optional:



ER – Modellierungskonzepte

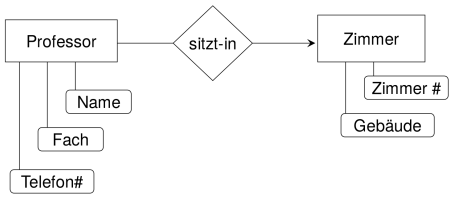
$\mu(D)$: Interpretation von D, mögliche Werte einer Entity-Eig.
$\mu(\texttt{int})$: $\mathbb{Z}$, $\mu(\texttt{string})$: $C^\star$ (Folgen von Zeichen aus C)
$\mu(E)$: Menge der möglichen Entities vom Typ E
σ_i(E): Menge der *aktuellen* Entities vom Typ E in Zustand σ
$\rightsquigarrow$ $\sigma(E) \subseteq \mu(E)$ und $\sigma(E)$ endlich
$\mu(R) = \mu(E_1) \times \cdots \times \mu(E_n)$
$\rightsquigarrow$ Die Menge aller möglichen Ehen ist die Menge aller (Mann,Frau)-Paare.
$\sigma(R) \subseteq \sigma(E_1) \times \cdots \times \sigma(E_n)$
$\rightsquigarrow$ aktuelle Beziehungen nur zwischen aktuellen Entities
Attribut A eines Entity-Typen E ist im Zustand σ eine Abbildung $\sigma(A) : \sigma(E) \to \mu(D)$ (nicht $A : \sigma(E) \to \mu(D)$)
Beziehungsattribute: $\sigma(A) : \sigma(R) \to \mu(D)$ (Beziehung R, Attribut A, möglicher Wertebereich $\mu(D)$)

Mehrstellige Beziehungen

Umwandlung von mehrstelligen Beziehungen in mehrere einstellige Beziehungen i.A. nicht einfach möglich.

Funktionale Beziehungen

Jedem Professor lässt sich ein Zimmer zuordnen, umgekehrt nicht zwingend
Schreibe: $R : E_1 \to E_2$

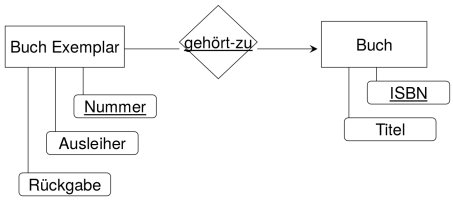


Schlüssel

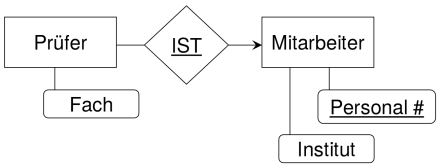
Schlüsselattribute $\{S_1, \ldots, S_k\} \subseteq \{A_1, \ldots, A_m\}$ für Entity-Typ $E(A_1, \ldots, A_m)$
Notation: Schlüssel unterstreichen: $E(\ldots, \underline{S_1}, \ldots, \underline{S_i}, \ldots)$
Schlüssel ist minimal: Wird ein Schlüsselattribut entfernt, so ist das entstehende Tupel nicht mehr eindeutig

Abhängige Entity-Typen

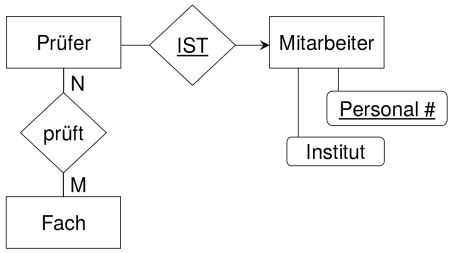
Identifikation über funktionale Beziehung (als Schlüssel)
Bsp: (Exemplar-)Nummer bezieht sich auf jeweiliges Buch



IST-Beziehung



Spezialfall eines abhängigen Entity-Typen (nur Beziehung als Schlüssel)
Vererbung von Attributen (und Werten):
σ(Prüfer) $\subseteq$ σ(Mitarbeiter)

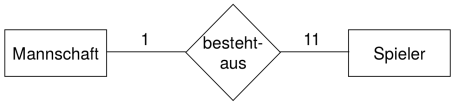


Entwurf – Kardinalitäten

An wv. Beziehungen muss Entity teilnehmen? $\rightsquigarrow$ einschränken
Teilnehmerkardinalität: `arbeitet_in(Mitarbeiter[0,1],Raum[0,3])`

1. jeder Mitarbeiter hat einen oder keinen zugeordneten Raum
2. pro Zimmer arbeiten maximal drei Mitarbeiter
3. ein Zimmer kann leerstehen

Standardkardinalität: 1 Mannschaft steht mit 11 Spielern in Bezug
Auch hier Intervallangabe möglich
Speziell: `m:n/1:n/1:1`-Beziehung (Untere Schranke jeweils **0**)



Semantische Beziehungen

Spezialisierung: `Pruefer` Spezialisierung von `Mitarbeiter`
⇝ Vererbung (IST-Beziehung)

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

Generalisierung: Buch oder DVD als Medium
`Medium` ist stets `DVD` oder `Buch`
Aber: Buch muss kein Medium sein.

Aggregierung: `Auto` besteht aus `Motor`, `Karosserie`,...
⇝ Entity aus Instanzen anderer Entity-Typen zusammengesetzt

Sammlung (auch Assoziation): `Team` ist Gruppe von `Person`
⇝ Mengenbildung

EER

= Erweitertes ER-Modell

Übernommen: Werte, Entities, Beziehungen, Attribute, Funktionale Beziehungen, Schlüssel (jetzt ausgefüllter Kreis)

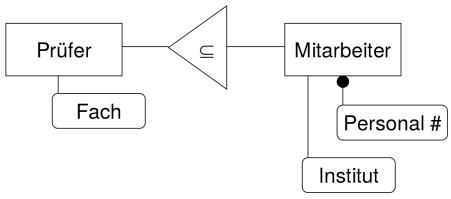
Nicht übernommen: IST-Beziehung – ersetzt durch *Typkonstruktor*

EER – Typkonstruktor

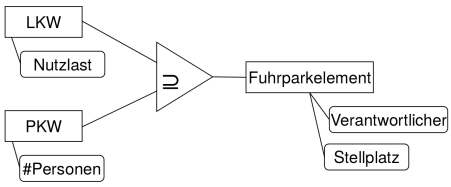
Ermöglicht Spezialisierung, Generalisierung, Partitionierung

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

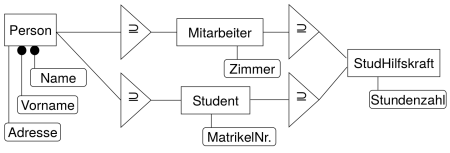
Ausgabetypen mit Spitze verbunden



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

Formalisierung Relationenmodell

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

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

Relationenschema: $R \subseteq U$

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

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; $t_1(\text{Name}) =$ “Andreas” usw.

REL: $\text{REL}(R) = \{r \mid r(R)\}$
Menge aller Relationen über 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 (d 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`

$b : \text{REL}(R) \to \{$ `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 $b(r) =$ `true` für alle $b \in B$

SAT: $\text{SAT}_R(B) = \{r \mid r(\mathcal{R})\}$
Menge aller Relationen über erweitertem Relationenschema (SAT = *satisfy*)

Schlüssel

Schlüssel und Fremdschlüssel einzige Integritätsbedingungen im relationalen Modell

Schlüssel: Minimale identifizierende Attributmenge

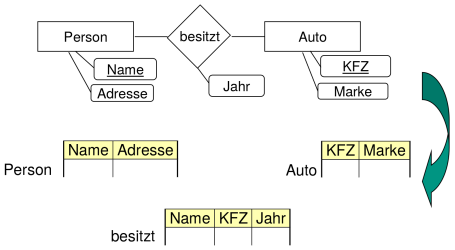
i.A. mehrere Schlüsselkandidaten, ein ausgezeichneter Primärschlüssel

Fremdschlüssel $X(R_1) \to Y(R_2)$:
$\{t(X) \mid t \in r_1\} \subset \{t(Y) \mid t \in r_2\}$ und Y ist Schlüssel von R_2

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

Zu Vermeiden:

Kapazitätserhöhend: relational mehr darstellbar als mit ER

Kapazitätsvermindernd: relational weniger darstellbar als mit ER

Abbildungsregeln

Entity-/Beziehungstypen $\rightsquigarrow$ Relationenschemata
Attribute $\rightsquigarrow$ Attribute Relationenschema
Schlüssel $\rightsquigarrow$ übernehmen

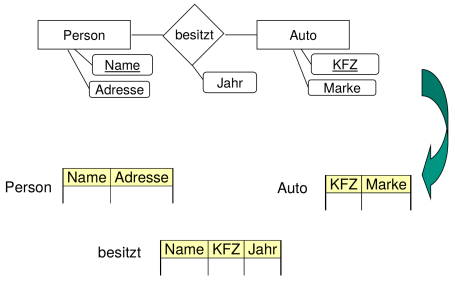
Kardinalitäten $\rightsquigarrow$ Schlüsselwahl

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

Funktionale Abhängigkeiten (FD)

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

Festlegung der FDs a priori beim Schemaentwurf (enthält semantische Information für höhere Konsistenz), nicht hinterher aus dem Datenbestand

Spezialfall Schlüssel X für Relation R: $X \to R$ und X minimal

Transitiv: $X \to Y \to Z \Rightarrow X \to Z$

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

F impliziert f: $F \models f$ (bedeutet $SAT_R(F) \subseteq SAT_R(f)$)

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

Hülle einer Attributmenge X bezuüglich F ist $X_F^* := \{A \mid X \to A \in F^+\}$

Reflexiv: $X \to X$ (und $F \models X \to X$ für alle F, X)

Akkumulativ: $X \to YZ, Z \to VW \Rightarrow X \to YZV$

Projektiv: $X \to YZ \Rightarrow X \to Y$

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

RAP-Algorithmus für das Membership-Problem

Problem: Menge von FDs F. Gilt $X \to Y \in F^+$?

Lösung in linearer Zeit:

1. $X^* := X$ (R-Regel)
2. Erweitere $X^* := X^* \cup Y_1$ für $X_1 \to Y_1$ mit $X_1 \subseteq X^*$ bis X^* stabil (A-Regel)
3. Ist $Y \subseteq X^*$, gilt $X \to Y$ (P-Regel)

Redundanzen - Anomalien

Belegen unnötigen Speicherplatz

Widersprüchliche oder fehlende Eingaben (Einfügeanomalie)

Änderungen parallel in allen Vorkommen nötig (Updateanomalie)

Informationen können beim Löschen anderer Inhalte mit verloren gehen (Löschanomalie)

Abhängigkeitstreue

Alle gegebenen Abhängigkeiten sind durch Schlüssel repräsentiert

Genauer: Menge der Abhängigkeiten (FDs) äquivalent zur Menge der Schlüsselabhängigkeiten.

Verbundtreue

Originalrelationen können durch Verbund der Basisrelationen wiedergewonnen werden

Kriterium für zwei Relationen: Dekomposition von X in X_1 und X_2 verbundtreu, wenn $X_1 \cap X_2 \to X_1$ oder $X_1 \cap X_2 \to X_2$

Allgemeines Kriterium: Wenn eine abhängigkeitstreue Dekomposition von R in X_i einen Universalschlüssel erhält (also für ein X_i gilt $X_i \to R$), so ist sie verbundtreu.

Universalrelation

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

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

Entwurfsziel

Relationenschemata, (Fremd-)Schlüssel so wählen, dass

1. alle Anwendungsdaten aus Basisrelation hergeleitet werden können (*Verbundtruee*)
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

Keine **partiellen Abhängigkeiten** eines Nicht-Primattributs von einem möglichen Schlüssel

Auflösen durch Abtrennen der rechten und Kopie der linken Seite

Partielle FD: Nicht-Primattribut hängt voll funktional von einem Teil eines Schlüsselkandidaten ab.

Volle FD: β ist voll funktional abhängig von α, wenn aus α kein Attribut entfernt werden kann, so dass FD immer noch gilt.

Gegenbeispiel: PLZ, Bundesland $\to$ Ort

Dritte Normalform

Keine **transitiven Abhängigkeiten** eines Nicht-Primattributs von einem möglichen Schlüssel

Transitive Abhängigkeit: Schlüssel K bestimmt Attributmenge X funktional, diese wiederum bestimmt Attributmenge Y
(und $X \nrightarrow K, \quad Y \notin KX$)
$\rightsquigarrow$ Transitive Abhängigkeit $K \to X \to Y$

Erreichen durch Abspalten von Y und Kopie von X

3NF impliziert 2NF, da partielle Abhängigkeit Spezialfall von transitiver Abhängigkeit (wähle $X \subsetneq K$)

Boyce-Codd-Normalform

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

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

Zerlegung von $\mathcal{R}$ in $\mathcal{R}_1 = (\alpha \cup \beta), \mathcal{R}_2 = \mathcal{R} - \beta$
$(F \ni f : \alpha \to \beta, \beta$ maximal)

Verbundtreu: $R_1 \cap R_2 = \alpha$ ist Schlüssel von R_1

Aber nicht immer Abhängigkeitstreu: Abhängigkeiten können beim Zerlegen verloren gehen!

Dritte Normalform daher meist ausreichend

Minimalität

Kriterien mit möglichst wenigen Relationenschemata erreichen

Dekomposition

Prinzip: Immer wenn $X \to Y \to Z$ wird Relation zerlegt

Erreicht nur 3NF und Verbundtreue

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

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

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

Wiederholen mit R_2

Probleme: 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 per Definition; Verbundtreue (nur mit Trick), 3NF und Minimalität werden reihenfolgeunabhängig erreicht

Polynomielle Zeitkomplexität

Verfahren:

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

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

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

Trick **Verbundtreue**: Orignal FD-Menge um $R \to \delta$ erweitern

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: *Name* $\twoheadrightarrow$ 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

SQL-Kern

select
Projektionsliste
Attribute, arithmetische Ausdrücke, Aggregatfunktionen
select distinct: keine Dopplungen
Umbenennungen: `select Preis * 1.44 as DollarPreis`

from
Zu verwendende Relationen, Umbenennungen
Orthogonalität: Wiederum SFW-Block möglich
`select * from (select [...]) where [...]`

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

group by
Gruppierung für Aggregatfunktionen

having
Selektionsbedingungen an Gruppen

from - Mehrere Relationen

Bei mehr als einer Relation: **Kartesisches Produkt**
Kommagetrennt oder als expliziter Operator:

```
select * from Kuenstler K, Titel T
select * from Kuenstler cross join Titel
```

Natürlicher Verbund

Automatischer Equi-Join auf allen übereinstimmenden Spalten, diese erscheinen nur ein mal in der Ergebnisrelation

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

Theta-Join

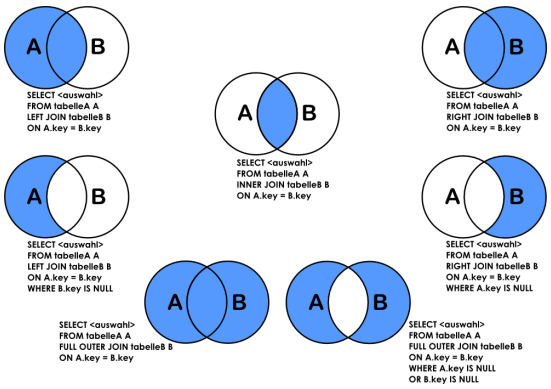
Verbund über Verbundsbedingungen

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

Beispiel: Auto $\bowtie_{AutoPreis>BootPreis}$ Boot

Outer, Left, Right Join

Dangling Tuples übernehmen und mit Nullwerten füllen



Self-Join

Kartesisches Produkt einer Tabelle mit selbst

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

Vierspaltiges Ergebnis:

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

Anwendungen: Vergleichen oder Zählen von Wertemengen

where

Konstanten-Selektion:
`select * from Buecher where Buecher.Titel = "Titel"`
Verbundbedingung bei Cross-Join (Attribut-Selektion):

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

like: Ungewissheitsselektion (`where name like "C\%"`)
'%' – Beliebig viele Zeichen '_' – Genau ein Zeichen
and, **or**, **not**, is **null**
in: `where ISBN in (select ISBN from Empfiehlt)`

Verzahnt geschachtelte Anfragen

In der inneren Anfrage Attribute aus der äußeren verwenden

```
select Nachname from Personen
where 1.0 in (select Note from Prueft
        where PANr = Personen.PANr)
```

Exists: Test, ob Ergebnis der inneren Anfrage nicht leer ist

```
select ISBN from Buch
where exists (select * from Ausleihe
        where Invnr = Buch.Invnr)
```

Mengenoperationen

Übernimmt Attributnamen des linken Operanden
Vereinigung

```
select A, B, C from R1 union [all]
        select A, C, D from R2
```

union all: Duplikate werden behalten
Differenz: **except**
Durchschnitt: **intersect**

Aggregatfunktionen

Prinzip: Berechnung eines Werts aus Werten eines Attributs
`sum([all / distinct] Attributname)`
Standard SQL: **count**(), **sum**(), **min**(), **max**(), **avg**()
Speziell: **count**(*)
Modifikatoren: **all** / **distinct** (Voreinstellung: all)

Group by

Gruppierung G: Für gleiche G-Werte werden Resttupel in Relation gesammelt, darauf dann Aggregatfunktionen angewendet

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

Wichtig: Jedes Select-Attribut muss entweder Gruppiert oder Aggregiert werden!
having: Bedingung auf gruppierter Relation

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

```
select Matrikelnr from Pruefung
        group by Matrikelnr
        having avg(Note) < (select avg(Note) from Pruefung)
```

Quantoren

any/some (äquivalent):

```
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)
```

Aber: Anwendbarkeit eingeschränkt, z.B. kein Vergleich auf Mengengleichheit

order by

Menge von Tupeln $\rightsquigarrow$ Sortierte Liste

```
select MatNr, Note from Prueft
       where V_Bez = 'DBS'
       order by Note [asc / desc], MatNr
```

Aufsteigend (**asc**, Standard) oder Absteigend (**desc**)

Wichtig: Sortier-Attribut(e) müssen in Select vorkommen!
Denn: Sortierung wird auf das Ergebnis der vorherigen SFW-Anfrage angewendet.

Nullwerte

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

Deshalb **nicht gleich**: `select * from Person` und
`select * from Person where Name = Name`
(Letzteres eliminiert Tupel mit Name = null)

Änderungsoperationen

Insert

```
insert into relation [(attribut1, ...)]
       values (wert1, ...)
```

Auch SQL-Anfragen als Wert möglich.

```
insert into Kunde (select LName, LAdr, 0 from Lieferant)
```

Update

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

Delete

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

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 Bespiel 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
$\rightsquigarrow$ Konsistenz sicherstellen $\rightsquigarrow$ **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
$\rightsquigarrow$ Zinsen aus S_5 von T_2 verloren, weil T_1 in S_6 überschreibt

Schritt	T1	T2
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	T1	T2
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		**commit**
8	Read(B, b1)	
9	...	
10	**abort**	

Non-Repeatable Reads

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

Schritt	T1	T2
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 konfligieren
$\Leftrightarrow$ p, q greifen auf selbes Datenobjekt zu und p oder q ist Schreiboperation

In einer Transaktion müssen konfligierende 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

Commited Projection ($C(H)$): H nach Entfernen aller nicht-committeten Operationen

Eine Eigenschaft von Histories ist **prefix commit closed**
$\Leftrightarrow$ (H erfüllt Eigenschaft $\Rightarrow$ $C(H')$ 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 zykelfrei 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 committeten Transaktionen nicht verändern).

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

Striktheit (ST): Objekte von noch nicht committeten 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

$\rightsquigarrow$ Serialisierbarkeit sichergestellt
Deadlocks sowie kaskadierende Abbrüche weiterhin möglich

Strenges Zwei-Phasen-Sperrprotokoll (S2PL):

Atomare Freigabephase am Ende der Transaktion
$\rightsquigarrow$ Zusätzlich ACA: Vermeidung kaskadierender Abbrüche

Konservatives Zwei-Phasen-Sperrprotokoll (C2PL):

Atomare Anforderungsphase zu Beginn der Transaktion
$\rightsquigarrow$ Zusätzlich: Vermeidet Deadlocks

CS2PL:

Kombination aus streng und konservativ: Atomare Anforderungs- und atomare Freigabephase
$\rightsquigarrow$ 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 $\rightsquigarrow$ schlechte Skalierbarkeit
3. Für Konsistenz müssen alle Knoten verfügbar sein $\rightsquigarrow$ geringere Ausfallsicherheit

$\rightsquigarrow$ 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, Follower sehen es sobald möglich

Prüfungsfragen

1. Geben Sie die Probleme mit dem klassischen, starken Konsistenzbegriff im verteilten Fall wieder.
2. Bekommt man mit *eventual consitency* 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 π: Optimierung: bei vielen Projektionen hintereinander reicht die zuletzt ausgeführte auch allein:
π`[KName]`(π`[KName, Land](Kuenstler)`) ⇝ π`[KName](Kuenstler)`

Selektion σ: Optimierung: Selektionen lassen sich beliebig vertauschen, manchmal auch Projektion und Selektion

Verbund $\bowtie$: Kommutativ, Assoziativ (Aber: Ausführungsreihenfolge kann erhebliche Performance-Unterschiede erzeugen)
Nested-Loop Join: Teuer ($O(n*m)$), da pro Eintrag links über alle rechten Einträge iteriert wird.
Besser: **Bock-Nested-Loop Join** (Arbeitsspeicher ausnutzen)
Merge Join: Beide Relationen sortieren, dann Eintrag für Eintrag Merge-Technik anwenden (linear wenn X Schlüssel)

Logische vs. physische Operatoren

DBS enthält meist mehrere pysische Operatoren und Implementierungen für den gleichen logsichen Operator

DBS sucht selbst den optimalen pysischen Operator heraus

Pysische Operatoren können dabei mehrere logische Operatoren zusammenfassen

Blockierende/Nichtblockierende Operatoren

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

Histogramme

Zeigt Auftrittshäufigkeit eines Intervalls (Bucket)

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. Dienstausführung
"Antwort innserhalb von 300ms für 99,9% der Aufrufe bei 500 Zugriffen pro Sekunde"

Quorum Consensus

Szenario: Replikation mit n Knoten
⇝ Wie strenge Konsistenz beim Schreiben sicherstellen? Was, wenn nicht alle Knoten verfügbar?

Lesen: Lese Mindestanzahl von Versionen (R), nehme aktuelle

Schreiben: Aktualisiere Mindestanzahl von Kopien (W)

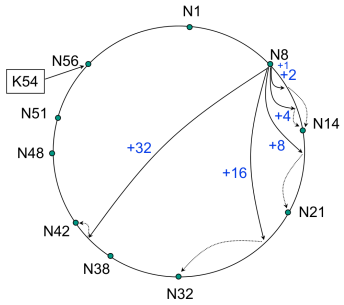
Jede Kopie erhält Versionsnummer

Üblich ist $Q_R + Q_W > N$

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*
Schlüssel k gehört zum im Uhrzeigersinn nächsten Knoten
Einfaches Hinzufügen / Entfernen von Knoten möglich
Suche: Jeder Knoten hat *finger table*, i-ter Eintrag von Knoten n: 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)

Umrechnen von Anwendungs- in Systemschlüssel, um Last zu verteilen (gleich / ungleich, evtl. auf mehrere Positionen)

Dynamo

Key-Value-Store

get-/put-Interface

Objekte BLOBs ⇝ kein DB-Schema ⇝ Interpretieren nötig

Keine Isolation ⇝ 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) ⇝ Erfassung der Zusammenhönge zwischen Versionen

Quorum-basierte Techniken ⇝ Inkonsistenzen vermeiden

Vector-Clock-basierte Techniken ⇝ Inkonsistenzen erkennen und auflösen

Unterschiedliche Knoten können Schreiboperationen absetzen
⇝ Eine Liste von (Knote, Zähler)-Paaren pro Version

Version 1 ist Vorgänger von Version 2, wenn jeder Zähler in Liste 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

Kombination mit **Sloppy Quorum**: $Q_R + Q_W < N$

Datenbanktechnologie auf Dynamo

Dynamo kein DBS im klassischen Sinn: Niedrigere Schnittstelle für Anwendungsentwicklung

Aber: Bessere nichtfunktionale Eigenschaften

Im Folgenden: Ansätze für DBS 'On Top of' Dynamo

Scale Independence

Anfrage ist *scale-independent*
⇝ Laufzeitverhalten unabhängig von DB-Größe

Anfragenklassifikation nach Aufwand:

1. Klasse I (konstant):
   z.B. Schlüssel-Zugriff, `LIMIT`-beschränkt, Paginierung
   Join auf Fremdschlüssel
2. Klasse II (beschränkt):
   Explizite Begrenzung liegt vor
   Als Kardinalität im erweiterten DB-Schema darstellbar
3. Klasse III (linear / sublinear):
   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

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 (und damit der Ausführungsdauer) des Zwischenergebnisses enthalten (i.A. `dataStop`-Operator; Fehlermeldung und Nichtausführung wenn dies nicht der Fall ist)

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-Begrenzuung der Anzahl an Datenobjekten pro Schlüssel ableiten

SLO Compliance-Vorhersage

SLO = *serivce-level objectives*

Größenbeschränkung Zwischenergebnisse noch keine Garantie für insgesamt beschränkten Aufwand

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

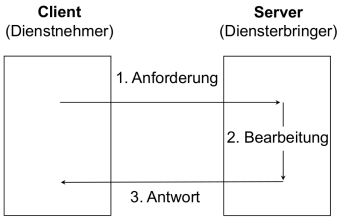
Histogramm-Lookup über Zufallsverteilung (Tupelgröße, Anzahl erwarteter Tupel)

Prüfungsfragen

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

XII. ANWENDUNGSENTWICKLUNG

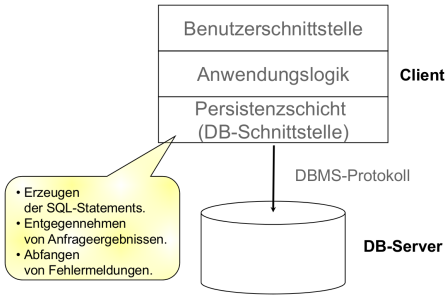
Client-Server-Architektur



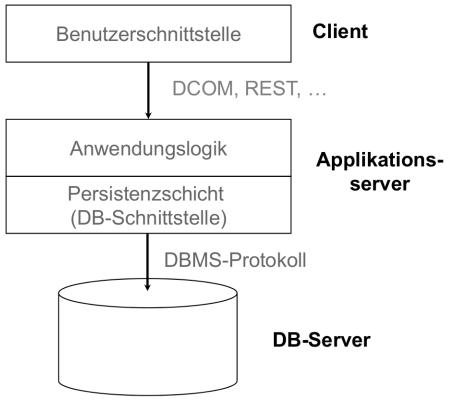
Erfordert

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

Zwei Schichten-Architektur



Drei Schichten-Architektur



Anwendungslogik

Anwendungslogik: Algorithmen, die anwendungsspezifisches Wissen beinhalten

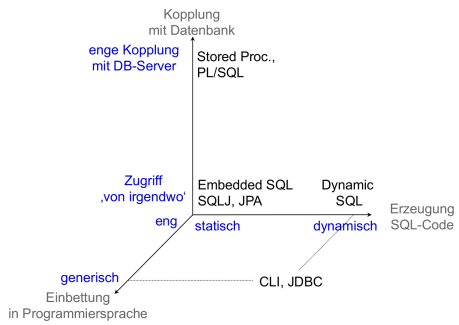
Personal-DB entählt Mitarbeiter-Daten
⇝ Anwendung: schlägt Teamleiter für konkrete Projekte vor
⇝ Bedeutsamkeit der Fähigkeiten usw. Anwendungsteil

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 ?');

upcdateSales.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;
.
run;
```

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.