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 (⇝ Query-Algebra)

RDBMS – Andragenoptimierung

Algebraische Ausdrücke äquivalent, Abfrage 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

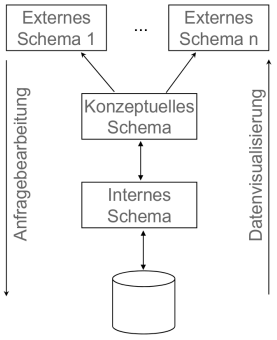
Anfragen deklarativ: Nutzer entscheidet nicht, wie Ergebnis ermittelt wird

Datenunabhängigkeit: DBMS stellt sicher:

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

⇝ 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)

⇝ **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?)

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?

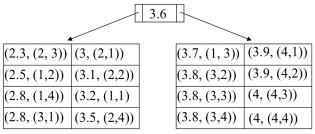
Bereichsanfrage: Wie viele Restaurants gibt es im Stadtzentrum?

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)))



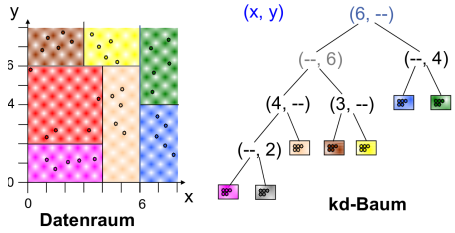
Tom, 20, 3.2, EE	Mary, 24, 3, ECE	Lam, 22, 2.8, ME	Chris, 22, 3.9, CS
Chang, 18, 2.5, CS	James, 24, 3.1, ME	Kathy, 18, 3.8, LS	Vera, 17, 3.9, EE
Bob, 21, 3.7, CS	Chad, 28, 2.3, LS	Kane, 19, 3.8, ME	Louis, 32, 4, LS
Pat, 19, 2.8, EE	Leila, 20, 3.5, LS	Martha, 29, 3.8, CS	Shideh, 16, 4, CS

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-NN]

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

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

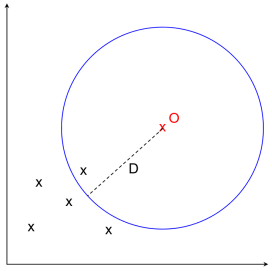
Hier: Baum unbalanciert, Balancierung in Realität für mehrdimensionale Daten

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-NN<D

Clustering: Liefert Outliner als Beiprodukt

Abstandsbasiert

Dichtebasiert

Clustering – Beispiel

Gegeben: Große Kundendatenbank, enthält Eigenschaften und Käufe

Gesucht: Gruppen von Kunden mit ähnlichem Verhalten finden

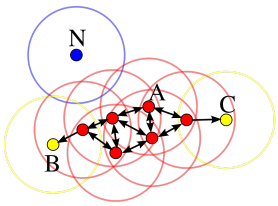
Clustering – DBSCAN

Dichte: Anzahl Objekte pro Volumeneinheit

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

Dichte-erreichbares Objekt: Objekt in ϵ-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 ϵ-Umgebungen vorberechnet wurden (oder mit räumlichem Index in konstanter Zeit bestimmt werden können)
$\rightsquigarrow$ 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) $\rightsquigarrow$ keine echten Outliner (Outliner-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

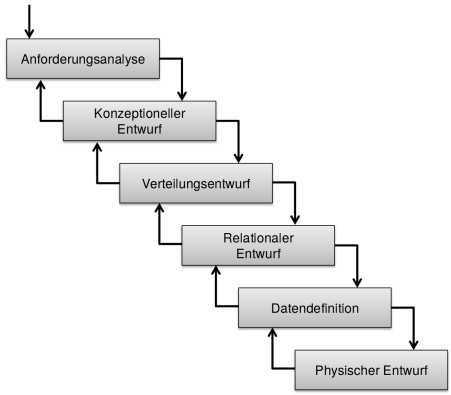
Gewinnung der Konventionen

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

= 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 }
```

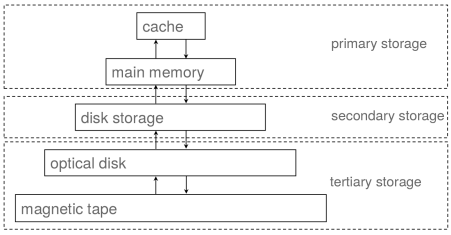
⇝ Attribut löschen, falls

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

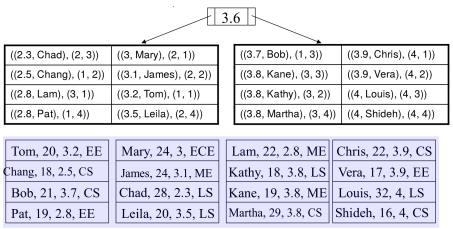
Index für (gpa, name) ≠ Index für (name, gpa)

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 dür DB-Features an, die zeigen, dass DB-Systeme physische Datenunabhängigkeit nicht vollständig umsetzen.