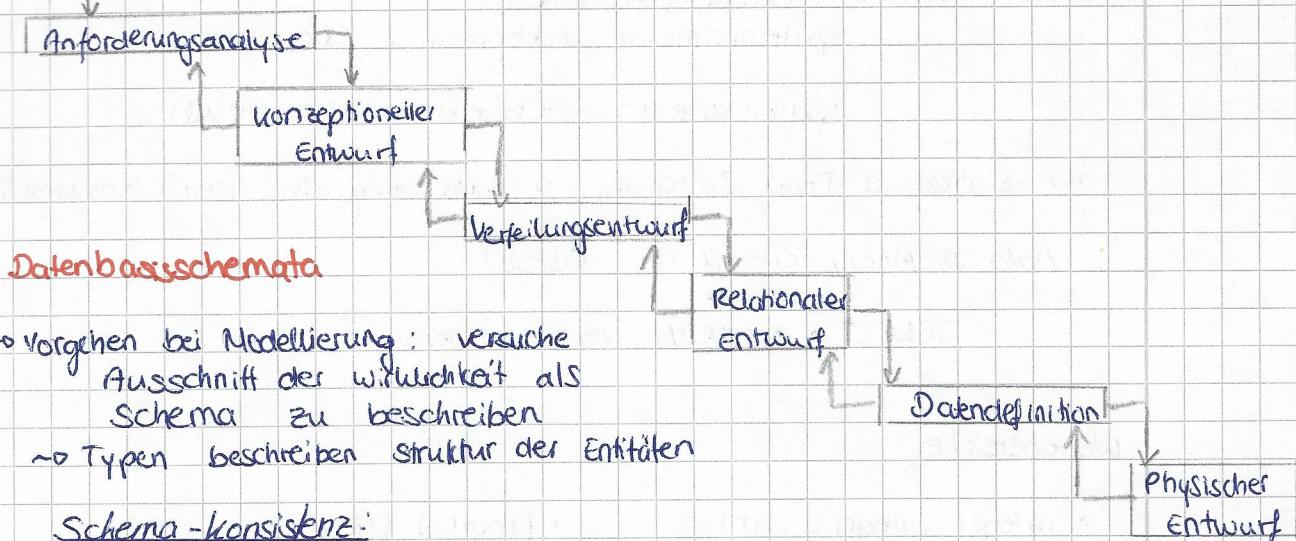


Datenbank - Definitionssprachen

Miniwelt: beschränkte Anwendungswelt; relevanter Weltausschnitt / Diskursbereich

Daten = Modelle (gedankl. Abstraktionen) der Miniwelt

Phasenmodell des DB-Entwurfs



- Vorgehen bei Modellierung: versuche Ausschnitt der Wirklichkeit als Schema zu beschreiben
- Typen beschreiben Struktur der Entitäten

Schemakonsistenz:

Einhaltung vorgegebener Konsistenzbedingungen (muss nicht der Wirklichkeit entsprechen)

- Legitime Zustände durch Ausführen der Operatoren mit Interpretation des Schemas
- vom DBMS überprüfbar

Datenbasiskonsistenz:

Idealfall: volle, aktuelle Übereinstimmung von Datenbasis und Miniwelt

Datenbasis bedeutungstreu, wenn ihre Metadate Elemente Modelle einer gegebenen Miniwelt sind

- schärfste Konsistenzforderung

SQL = standardisierte Sprache für Datenbank-Zugriff (rel. DB)

SQL-DDL = SQL - Data Definition Language → Schemadefinition

→ Definition von Typen / Wertebereichen / Relationenschemata / Integritätsbed.

→ NULL als Attributwert zulässig; Duplikate zulässig (⇒ Relation ist Multimenge; solange kein Schlüsselkonflikt)

Externe Ebene: CREATE VIEW
DROP VIEW

Konzeptuelle Ebene:

CREATE TABLE
ALTER TABLE
DROP TABLE

CREATE DOMAIN
ALTER DOMAIN
DROP DOMAIN

Interne Ebene: CREATE INDEX
ALTER INDEX
DROP INDEX

CREATE TABLE

CREATE TABLE basisrelationenname (
spaltenname₁ wertebereich₁ [NOT NULL],
spaltenname_k wertebereich_k [NOT NULL])

[] = optional

→ Eintrag in Data Dictionary → Vorbereitung der "leeren Basisrelation" in DB

Data Dictionary-Zugriff (mit Oracle):

SELECT * FROM basisrelationenname;

Wertebereiche:

- > integer (integer 4, int)
smallint (integer 2) → float(p) (float)
- > decimal(p,q) → q = Nachkommastellen
numeric (p,q)
- > character(n) (char(n), n=1 char) → strings fester Länge n
- > character varying(n) (varchar(n)) → strings variabler Länge, max. n Zeichen
- > bit(n) / bit varying(n) → analog für Bitfolgen
- > date / time / timestamp → Datums- / Zeit - / Komb. D. z. - Angaben

Integritätsbedingungen:

- Schlüssel kann aus beliebig vielen Attributen bestehen → Primärattribute

Bsp.: CREATE TABLE Titel (
TITLEID integer NOT NULL,
NAME varchar(200),
KID integer,
PRIMARY KEY (TITLEID),
FOREIGN KEY (KID), ← auch mehrere möglich
REFERENCES Künstler(KID) ← Relationen und Attribute wo
Fremdschlüssel als Schlüssel
verkommt

> DEFAULT: Defaultwert für Attribute

> CREATE DOMAIN: Anweisung benutzerdef. Wertebereiche

> CHECK: Überprüfe, dass ...

z.B.

Bsp.: > CREATE DOMAIN Gebiete vorchein (20)
DEFAULT 'Informatik'

CREATE TABLE Vorlesungen (

Studiengang Gebiete)
Attribut Wertebereich

> CREATE TABLE Vorlesungen (

V-NAME varchar(80) NOT NULL, PRIMARY KEY,
SWS smallint, CHECK(SWS ≥ 0),
Semester smallint, CHECK(Semester BETWEEN 1 AND 9))

=> Beliebig komplexe Bedingungen möglich

ALTER TABLE

ALTER TABLE basisrelationenname ADD spaltenname Wertebereich

→ Änderung des Relationschemas im Data Dictionary

(z.B. Hinzufügen eines neuen Attributs)

→ ferner wird bei existierenden Tupeln mit null belegt

auch DEFAULT und CHECK möglich:

... ADD Budget decimal (8,2) DEFAULT 1000, CHECK(Budget ≥ 1000)

oder PRI KEY - Klausur:

... ADD CONSTRAINT u_buey PRIMARY KEY (ISBN)

DROP

DROP spaltenname { RESTRICT | CASCADE }

→ erlaubt Löschen von Attributen, falls

> Fall RESTRICT: keine Sichten und Integritätsbed. mit Hilfe dieses Attribut definiert wurden

> Fall CASCADE: oder mit gleichzeitiger Lösung dieser Sichten und Integritätsbed.

DROP TABLE basisrelationenname { REST. | CASCADE } analog

CREATE INDEX

CREATE [UNIQUE] INDEX indexname ON basisrelationenname
spaltenname_1 ordnung_1,

...
spaltenname_k ordnung_k)

'unique' = Schlüsselbedingung

bsp. CREATE UNIQUE INDEX typ_on_auto (hersteller, modell, baujahr) .

→ Reihenfolge der Spaltennamen = Sortierreihenfolge

SQL - Kern

from - Klammer select * from relationenliste
→ liefert gesamte Relation 'relationenliste'

Kartesisches Produkt bei mehr als einer Relation hinterfrom

self-Join/Selbstverbund: Kart. Prod. einer Tabelle mit sich selbst

▷ Select * from Künstler, Titel $\hat{=}$ Select * from Künstler cross join Titel

Natürlicher Verbung gleiche Attrib. werden nur einmal aufgelistet
→ übersichtlicher, weniger fehleranfällig

Theta-Join: select * from Künstler join Titel on Künstler.kID = Titel.kID
 = Verbindung über Verbindungsbedingungen
 select * from Künstler, Titel where K.kID = T.kID

Orthogonalität: Jeder SFW-Block kann auch hinterfrom eingesetzt werden

Außere Verbinde: outer join übernimmt dangling tuples und füllt mit Nullwerten auf

- > full outer join: in beiden Operanden
 - > left outer join: im linken Operanden
 - > right outer join: im rechten Operanden

Bsp.	links	A	B		rechts	B	C		Nat.	A	B	C
		1	2			3	4		Join	2	3	4
		2	3			4	5					

Verbundtypen



SELECT < \rightarrow FROM
tabelleA A LEFT JOIN
tabelleB B ON A.key = B.key



SELECT < \rightarrow FROM
tabelleA A INNER JOIN
tabelleB B ON A.key = B.key



SELECT < \rightarrow FROM
tabelleA A RIGHT JOIN
tabelleB B ON A.key = B.key



SELECT < \rightarrow FROM tabelleA A
LEFT JOIN tabelleB B ON
A.key = B.key WHERE B.key IS NULL



SELECT < \rightarrow FROM
tabelleA A FULL OUTER JOIN
tabelleB B ON A.key = B.key



SELECT < \rightarrow FROM tabelleA A
RIGHT JOIN tabelleB B ON A.key = B.key
WHERE A.key IS NULL



SELECT < \rightarrow FROM tabelleA A FULL OUTER JOIN
tabelleB B ON A.key = B.key WHERE A.key IS NULL
OR B.key IS NULL

Select - Klammer select [distinct] { attribut | arithm. Ausdruck | aggregat-funkt. } from ...

Abschließende Projektion, Zielliste

- Bestandteile : - Attribute aus from-Relation
- arithm. Ausdr. über Attributen und Konstanten
- Aggregatfunkt. über Attribute
- distinct: Ergebnismenge statt Multimenge (keine Doppelten)

WHERE - Klammer: Selektionskond./ Verbindlbed.

- Bed. können sein : - = / >= / ... : attribut = 5
- Null-Selektion: attribut is null
- boolesche Ausdrücke mit or, and, not
- quantifizierte Bed. (wenn Argument in Vergleich Menge liefert)

In - Prädikat: ... attribut in (SFW-Block)

Bsp.: Select Titel from Bücher where ISBN in (select ISBN from Empfiehlt)

ungewissheitssélection: '?' -> ein oder beliebig viele Zeichen, '#' -> genau ein Zeichen
attribut wie spezialkonstante (z.B. 'Benjy.Cummings-')

Mengenoperationen

> Vereinigung: union / union all SFW-block 1 union SFW-block 2
Attributkompatibilität muss gegeben sein!

Bsp. select A,B,C from E1
union

select A,C,D from E2

Ergebnis: Attr.namen des linken operieren

union hat Duplikatelimin.
union all nicht

> Mengendifferenz: select... from... EXCEPT (select... from...)

> Mengendurchschnitt: select... from... INTERSECT (select... from...)

Weitere Konstrukte

Operationen auf Wertebereichen: innerhalb von select + where auch skalare Ausdrücke statt Attribute

- numerische Wertebereiche: + / - / * / / ...
- Strings: char-length / Konkatenation / substring / ...
- Datumsarten, Zeitintervalle: current-date / current-time / + / - / ...

z.B. select ISBN, Preis * 1,44 from Bücher

Umbenennung: Attributnamen zuordnen

z.B. select ISBN, Preis * 1,44 as Dollar-Preis from Bücher

Aggregatfunktionen: z.B. select sum(all Preis) from Bücher
select count(distinct PANR) from Prüft

- > count: Anzahl Werte einer Spalte
- > count(*): Anzahl Tupel in einer Relation
- > sum: Summe der Werte einer Spalte
- > avg: arithm. Mittel einer Spalte
- > max/min: Größter/Kleinstes Wert einer Spalte

Argumente:
> Attribut der durch from spezifizierten Relation
> gültiger skalarer Ausdruck

optional vor Argument: distinct oder all (all ist default)
(außer bei count(*))

Nullwerte vor Anwendung aus Wertemenge eliminiert (außer bei count(*))

Group by, having: Syntax: select ... from ... [where ...]

[group by Attributliste]

[having Bedingung]

z.B. select Marke, sum(Anzahl) from Zulassungen group by Marke

having: Selektionsbedingung auf gruppierten Relation, darf Bezug nehmen auf

- Gruppierungsattribute
- beliebige Aggregatfunktion über Nicht-Gruppierungsattribute

z.B. select panr, sum(gehalt) from anstellungen
group by panr
having sum(gehalt) > 10000;

Quantoren, Mengenvergleiche:

all = Allquantor ; any/some = Existenzquantor

Bsp. select panr, immatri.datum from studenten where
Matr. nr. = any (select Matr. nr. from Prüft)

~ Mengenvergleiche nicht damit umsetzbar

Selbst-Verband: Vergleich von Wertemengen / Zählen von Wertemengen

z.B.: alle Bücher mit "U." + "W." als Autor

→ select BA_1.ISBN from Buch-Autor BA_1, Buch-Autor BA_2
where BA_1.ISBN = BA_2.ISBN
and BA_1.Autor = 'U.'
and BA_2.Autor = 'W.'

z.B.: alle Prüfer, die ≥ 2 Studenten geprüft haben

→ select distinct X.PANr from Prüft X, Prüft Y
where X.PANr = Y.PANr and X.Matr.nr <> Y.Matr.nr
↑ ungleich

Order-by-Klausel: Menge von Tupeln → Liste

Bsp.: select Matr.nr., Note from Prüft where V-Bezeichnung = 'DB I'
order by Note asc

asc = aufsteigend
desc = absteigend
a

'Argument nach order-by muss in
Select-Klausel vorkommen!'

Nullwerte:

• Skalare Ausdrücke: Ergebnis null, sobald Nullwert in Berechnung eingeht

and	true	unknown	false	or	t	u	f
true	true	u	f	t	t	t	t
unknown	u	u	f	u	t	u	u
false	f	f	f	f	t	u	f

not

t	f
u	u
f	t

⇒ fast alle vergleiche mit Nullwert ergeben
unknown

Änderungsoperationen

update: update basisrelation set attr-1 = ausdr-1, ..., attr-n = ausdr-n
(where ...)

Bsp.: update Angestellte set Gehalt = Gehalt + 1000
where Gehalt < 5000

delete: delete from basisrelation [where ...]

Bsp.: delete from Ausleihe where Invnr = 4711

insert: insert into basisrelation [(attr-1, ..., attr-n)] optional
values (konst-1, ..., konst-n)

Bsp.: insert into Künstler values (1022, 'R.S.', 'R.G.S')

Warum DB?

- Datenredundanz: Daten mehrfach gespeichert ohne DB
→ Verschwendungen von Speicherplatz
→ "Vergessen" von Änderungen; Inkonsistenz
→ keine zentrale Datenhaltung

Kriterien:

Transaktionen: Programm / Folge von Kommandos, die in Interpreter eingegeben werden

Eigenschaften:

ATOMARITÄT: Ausführung ganz oder gar nicht

ISOLATION: kein inkonsistenten Zwischenzustände sichtbar

→ Mehrere Nutzer / Anwender können parallel auf gleichen Daten arbeiten

physische Datenumabh.: interne Repräsentation der Daten irrelevant

→ Datenschutz (kein unbefugter Zugriff)

→ Datensicherheit (kein ungewollter Datenverlust, insb. bei Defekten)

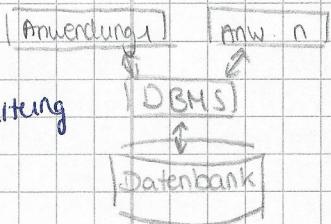
Relationenmodell

Tabellen = "Relationen" → Relation = Menge von Tupeln

DBMS = Datenbank-Management-System → Software zur Datenverwaltung

DBS = Datenbanksystem (DBMS + Datenbank)

→ Daten in Datenbank



Name der Tabelle = Relationenschema

Zeile der Tabelle = Tupel

Spaltenname = Attribut

obere "Titelzeile" = Relationenschema

Schlüssel: eindeutiger Identifizierer

Datenbankschema: Menge von Relationsschemata

Fremdschlüssel: Schlüssel in anderem Relationenschema

Anfrageoperationen / Optimierung

Selektion: Zeile (Tupel) auswählen $\sigma_{Schlüssel} \text{ (Relationenname)}$

Projektion: Spalten (Attribute) auswählen $\pi_{Schlüssel, Attribut} \text{ (Relationenname)}$

→ Schachtelung möglich $\sigma(\pi(\dots))$

Allg. Regel: $\sigma_{A_1 = \text{konst}} (\sigma_{A_2 = \text{konst}} (\text{REL})) \cong \sigma_{A_2 = \text{konst}} (\sigma_{A_1 = \text{konst}} (\text{REL}))$

→ Effizienzfrage → Aufgabe des Anfrage-Optimierers

→ Anfragen deklarativ: Anw. sagt nur, welches Ergebnis, nicht wie

physische Datenunabhängigkeit:

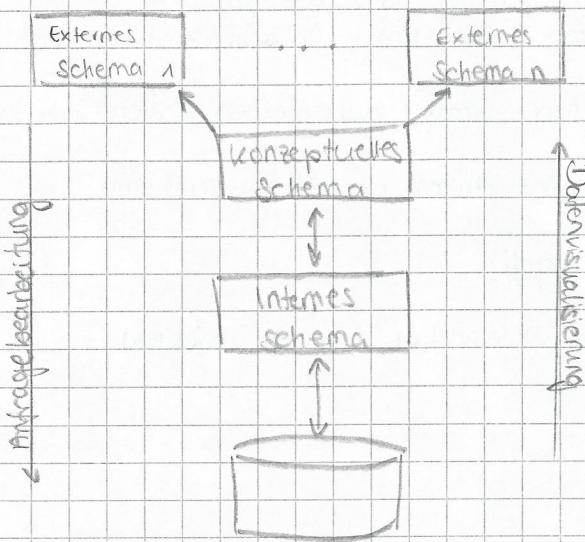
DBMS stellt sicher, dass

> Anfrage funktioniert bei ändernder physischer Darstellung der Daten

> Anfrage in untersch. DBs funktioniert

→ erlaubt höhere Komplexität bei Anwendungsentwicklung

3-Ebenen Architektur



Konzeptionelles Schema:

Was ist mein Diskursbereich?

Welche Entitäten der realen Welt sind relevant?

Internes Schema:

physische Repräsentation der Daten

→ Vorgehen beim Entwurf:

erst konzeptionelles, dann internes

Externe Sichten: Untersch. Ausschnitt der Daten für untersch. Benutzer

→ Datenschutz

→ Übersichtlichkeit

→ organisatorische Gründe (Abteilung, Einmischung)

→ Verstecken von Änderungen am Konz. Schema

=> LOGISCHE DATEN UNABHÄNGIGKEIT

Prinzipien für relationale DB: 9 Codicliche Regeln

(1) Integration: einheitliche, nichtredundante Datenverwaltung

(2) Operationen: Speichern, Suchen, Ändern

(3) Katalog: Zugriffe auf Datenbankbeschreibungen im Data Dictionary

(4) Benutzersichten

(5) Integritätssicherung: Korrektheit des Datenbankinhalts

(6) Datenschutz: Ausschluss unbefristeter Zugriffe

(7) Transaktionen: mehrere DB-Operationen als Funktionseinheit (Atomicität)

(8) Synchronisation: Transaktionen (parallele) koordinieren (Isolation)

(9) Datensicherung: Wiederherstellung von Daten nach Systemfehlern

→ funktionale Regeln

nicht-funktionale Ansprüche: kurze Aw-Zeiten, Effizienz, Shalierbarkeit, Zuverlässigkeit

Clustering und Finden von Ausreißern

Kd-Baum:

Daten in Diagramm auftragen → entlang der Dimensionen splitten

→ es entsteht Baum mit splits als Knoten

Problem: Baum nicht balanciert, "wächst nach unten"

K-NN: = k - Nächste Nachbarn

k-Abstand $\epsilon_{[k-NN]}$ = Abstand des k-nächsten Nachbarn

Radius um Anfragepunkt: NN-Distanz / NN-Sphäre

Outlier:

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

Anwendung: Video-/Sicherheitsüberwachung, Marketing, Finanzen, Erkennen von Netzwerlfehlern,

→ Objekt o aus Datenbestand T ist DB(p,D)-Outlier, wenn:
Abstand von o zu mind. p% der Objekte in T größer als D

also wenn k-Abstand < D, wobei $k = N \cdot (1-p) - 1$

zur Umrechnung in absolute Werte

DBSCAN

Objekt dicht: mind. minPts andere Objekte in Kugel um Objekt mit Radius ϵ

dichte-erreichbar: Objekt ist in ϵ -Umgebung eines dichten Objekts, ist aber selbst nicht dicht (Rand des Clusters)

→ Zuordnung dichter Objekte zu Cluster deterministisch
dichte-erreichbare nicht deterministisch

Anomalien

(1) Raum mit Punkten nur dünn besetzt (Sparsity)

(2) hierarchische Datenstrukturen nicht effektiv

Sparsity: Diskretisierung → nur zwei Partitionen pro Dimension
→ zufällige Zelle höchstwahrsch. leer

Annahme: Datenpunkte gleichverteilt

> Wahrscheinlichkeit

Annahme: Datenpunkte gleichverteilt ($d = \text{Dimension}$ Datenpunkte), Datenraum $\mathbb{R}^d = [0,1]^d$

> Wahrscheinlichkeit, dass NN-Distanz von Punkt Q kleiner als r:

$$P[Q, r] = 1 - (1 - \text{Volume}(\text{sphere}^d(Q, r) \cap \mathbb{R}^d))^N$$

> Erwartete NN-Distanz für Punkt Q:

$$E[\text{nn dist}] = \int_0^\infty r \cdot \frac{\partial P[Q, r]}{\partial r} dr$$

> Erwartete NN-Distanz für beliebigen Punkt:

$$E[\text{nn dist}] = \int_{Q \in \mathbb{R}^d} E[Q, \text{nn dist}] dQ$$

- Für große Dimensionen schneidet jede NN-Kugel jeden Block
- Im Baum müssen alle Blätter betrachtet werden
- Curse of dimensionality

Außerdem: Bei vielen Dim. ist Abstand zweier Objekte fast gleich dem zweier anderer Objekte

→ es gibt keine echten Outlier

→ Algorithmen liefern (mehr od. weniger) zufälliges Objekt

=> Nur erfolgsversprechende Teilräume nach Ausreißern absuchen

Triviale / nichttriviale Outlier

Outlier nur in Teilräumen Outlier, manche Teiräume ohne Outlier

Triviale Outlier: Objekt bereits in einem Teilraum als Ausreißer

Nichttr. Outlier: Objekt erst Outlier bei Betrachtung mehrerer Dim.

→ Finde relevante Teiräume mit Outliern

H:CS

Wenn Attribute nicht korreliert sind Ausreißer in diesem Raum tendenziell triviale Outlier.

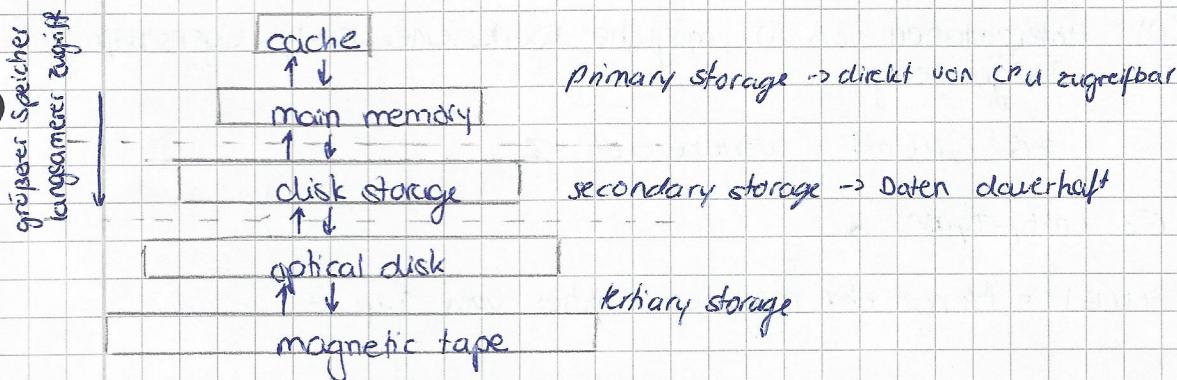
→ Suche nach Verletzungen statistischer Unabhängigkeit (Kontakt)

Fehlerarten:

Miss: Verfahren zur Ausreißerkennung erkennt Ausreißer nicht

False Positive: Nicht-Ausreißer sei Ausreißer

Speicherhierarchie



- Server zur Datenverwaltung verfügt über mehrere Speicher
- Programme in main memory → werden zur Ausführung in Cache/Register geladen
- Daten zur Verarbeitung in Prim. Speicher kopiert

Index: Speicherung der Daten auf Seiten

- Index für mehrere Attribute möglich, aber (gpa, name) ≠ (name, gpa)
- man kann Index nachträglich anlegen / löschen ohne Daten zu löschen
- Bestandteil der physischen Ebene → Definition Bestandteil des internen Schemas

Anfrage: SELECT name FROM Student WHERE gpa > 4.0 AND age = 27
→ liefert Ergebnis, und hängt davon ab, jener Index existiert oder nicht
→ physische Daten unabh.

bei mehreren "WHERE" findet DB überlegene (effizientere) Alternative der Anfrage-Reihenfolge

Datenbankmodelle

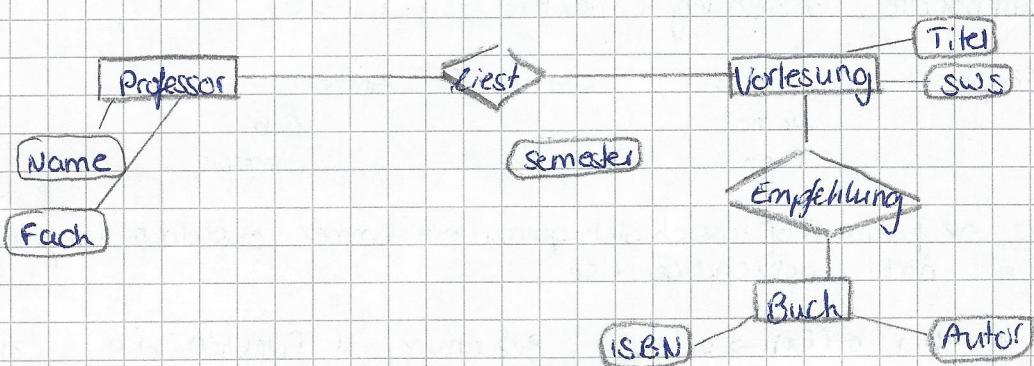
Entity-Relationship - Modelle

Entity: Objekt der realen oder der Vorstellungswelt, über das Infos zu speichern sind (z.B. VL, Buch, Prof)

Relationship: Beziehung zwischen Entities (z.B. Prof hält VL)

Attribut: Eigenschaft von Entities oder Beziehungen (z.B. ISBN eines Buchs)

zB.:



Werte:

- > $\mu(D)$: Interpretation von D , mögliche Werte einer Entity-Eigenschaft
Trägermenge
z.B.: $\mu(\text{int}) = \text{Wertebereich } \mathbb{Z}$

E_1, E_2 Entity-Typen \rightarrow

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

- > $\sigma_i(E)$: Menge der aktuellen Entities vom Typ E in einem Zustand σ_i

Axiome:
 - aktuelle Entities müssen mögliche Elemente sein
 $\rightsquigarrow \sigma(E) \subseteq \sigma_i \mu(E)$
 - $\sigma(E)$ endlich

Beziehungen: Beziehungstypen: Notation n-Stelliger Beziehungstypen:



\rightsquigarrow mögliche Ausprägungen: $\mu(R) = \mu(E_1) \times \dots \times \mu(E_n)$

z.B.

\rightsquigarrow Menge aller mögl. Ehen ist Menge aller "mögl. Mann"- "mögl. Frau"-Paare

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

= aktuelle Beziehungen nur zwischen aktuellen Entities

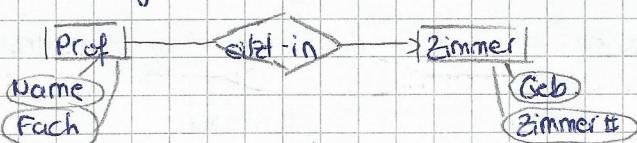
Attribute: Attribut A eines Entity-Typen E ist im Zustand σ eine Abbildung $\sigma(A): \sigma(E) \rightarrow \mu(D)$
 \rightsquigarrow Attributwert abh. vom Zustand
 \rightsquigarrow Attributwert nur für aktuelle Entities

optionales Attribut:
Kreis an Linie

Beziehungsattribute: $\sigma(A): \sigma(R) \rightarrow \mu(D)$
 Textuelle Notation: $E(A_1: D_1, \dots, A_m: D_m)$
 bzw. $R(E_1, \dots, E_n: A_1, \dots, A_p)$

Funktionsbeziehung: Textuell: $R: E_1 \rightarrow E_2$

Graphisch:



\rightsquigarrow Jedem Prof lässt sich genau ein Zimmer zuordnen, umgekehrt nicht notwendigerweise

$\rightsquigarrow \sigma(R): \sigma(E_1) \rightarrow \sigma(E_2)$; Beziehung ist Funktion, die ... zuordnet

Schlüssel: Entity-Typ $E(A_1, \dots, A_m)$

Teilmenge $\{S_1, \dots, S_k\}$ der Attribute = Schlüsselattribute

$\Rightarrow \{S_1, \dots, S_k\} \subseteq \{A_1, \dots, A_m\}$

\Rightarrow In jedem DBzustand identifizieren aktuelle Werte der Schlüsselattribute eindeutig Instanzen des Entity-Typs E :

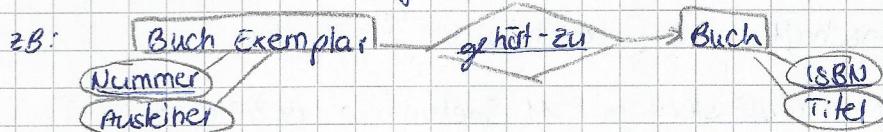
$$\forall e_1, e_2 \in \sigma(E): (\sigma(S_1)(e_1) = \sigma(S_1)(e_2), \dots, \sigma(S_k)(e_1) = \sigma(S_k)(e_2)) \\ \Rightarrow (e_1 = e_2)$$

M Notation: Schlüsselatt. unterstreichen

\Rightarrow Schlüssel ist MINIMAL, d.h. es gibt keine Teilmenge von $\{S_1, \dots, S_k\}$ mit gleichen Eigenschaften

Abhängiger Entity-Typ: Identifikation über funktionale Beziehung

\Rightarrow funktionale Beziehung als Schlüssel



IST-Beziehung:



Jeder Prüfer-Instanz ist genau eine Mitarbeiter-Instanz zugeordnet

\Rightarrow Nur Beziehung ist Schlüssel

d.h. zwei Prüfer, die gleichem Mitarbeiter zugeordnet sind, sind identisch

\Rightarrow "vererbkt" Attribute: Attribute von Mitarbeiter treffen auch auf Prüfer zu

\Rightarrow auch Werte vererben sich

$$\text{d.h. } \sigma(\text{Prüfer}) \subseteq \sigma(\text{Mitarbeiter})$$

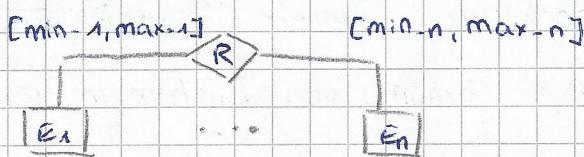
Kardinalitäten

Einschränkung an wie vielen Beziehungen Entität teilnehmen kann/muss

Teilnehmerkardinalität:



Allgemein:



Notation: $R(E_1, \dots, E_n; [min_i, max_i], \dots, E_n)$

Kardinalitätsbedingung: $\min_i \leq |\{e_i | re_0 \circ (R) \wedge r \cdot E_i = e_i\}| \leq \max_i$

$\geq [0, *]$ Standardannahme

\rightarrow totale funktionale Beziehung: $R(E_1[1, 1], E_2)$:
jede Instanz aus E_1 genau einer Instanz aus E_2 zugeordnet

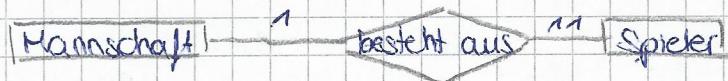
$\rightarrow R(E_1[0, 1], E_2)$: partielle funktionale Beziehung $R: E_1 \rightarrow E_2$
jede Instanz max. einer Instanz aus E_2 zugeordnet

Bsp.: arbeitet_in (Mitarbeiter [0, 1], Raum [0, 3])

- jeder Mitarbeiter in keinem oder einem Raum
- pro Raum 0-3 Mitarbeiter

\rightarrow IST-Beziehung E_1, E_2 : $IST(E_1[1, 1], E_2[0, 1])$

Standardkardinalität:



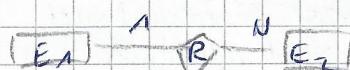
"Eine Mannschaft steht mit 11 Spielern in Beziehung"

Ablesen der Kardinalitätsangabe zu E_1 :

- Instanzen der anderen an Beziehung beteiligten Entitäten fixieren
- Wieviele Instanzen von E_1 stehen hiermit in Verbindung?

! Intervallangaben zulässig!

SK:



TK:



$\rightarrow N \approx *$ $\rightarrow 1$ in Verbindung mit $N \neq$ funktionale Beziehung
 $\wedge N = [0, +]: [0, 1]$

Semantische Beziehung:

Beziehung zw. Entitäten, die unabh. von der Domäne immer wieder auftreten, z.B. IST-Beziehung \rightsquigarrow
nicht Teil des ER-Modells (außer IST)

\rightarrow Spezialisierung: $\hat{=}$ IST-Beziehung

Partitionierung: mehrere disjunkte Entity-Typen

Bsp.: Partition. von Büchern in Lexika und Romane

→ Generalisierung: Entities in allg. Kontext

Bsp. Buch / DVD als Medium → Medium ist stets Buch / DVD
aber Buch muss kein Medium sein.

→ Komplexe Objekte:

Aggregation: Entity aus einzelnen Instanzen anderer Entity-Typen zusammengesetzt,
Bsp. Auto aus Motor, Karosserie, ...

Sammlung /

Assoziation: Mengenbildung; Team als Gruppe von Personen

→ Beziehungen höheren Typs:

- Spez. / Gen. auch für Beziehungstypen
- Beziehungen zw. Beziehungsinstanzen: Beziehungen zweiter und höherer Ordnung

EER-Modell Erweitertes ER-Modell

≡ ER-Modell ohne IST-Beziehung → Typkonstrukt

Typkonstrukt: Modellierungskonzept für IST / Gen. / Part.

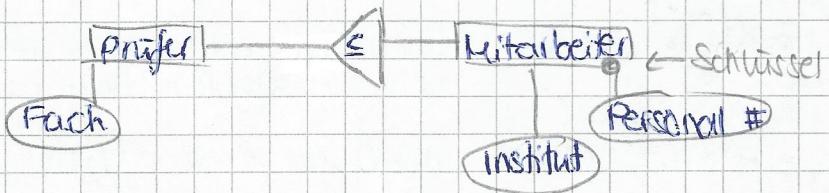
Eingabetypen - verbunden mit Dreiecksbasis

Ausgabetypen - verbunden mit Spitze

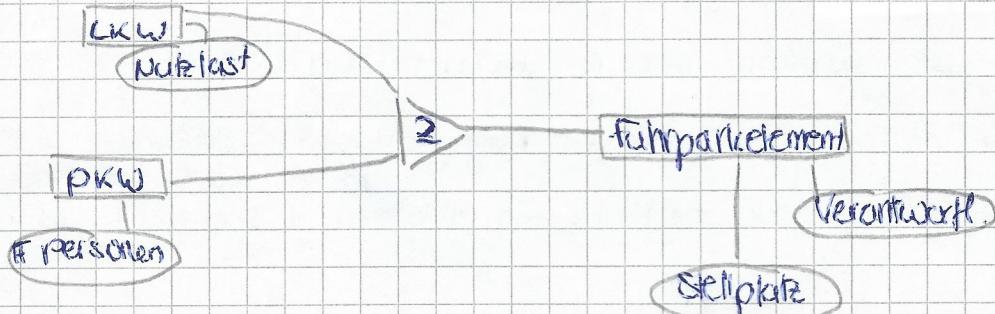
Eingabetypen bei Gen. - speziellere Typen

Eingabetypen bei Spez. / Part. - der / die allg. Typen

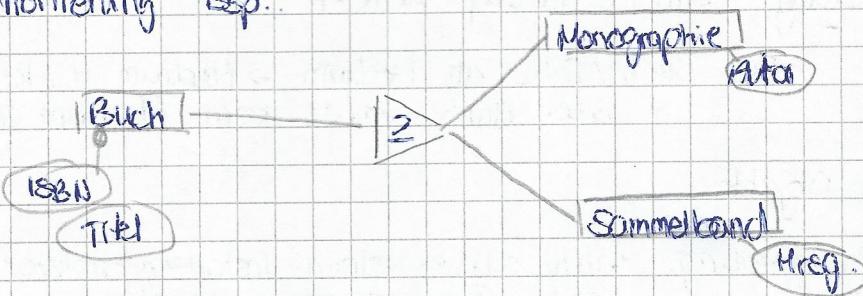
Spezialisierung Bsp.: (IST-Bez.)



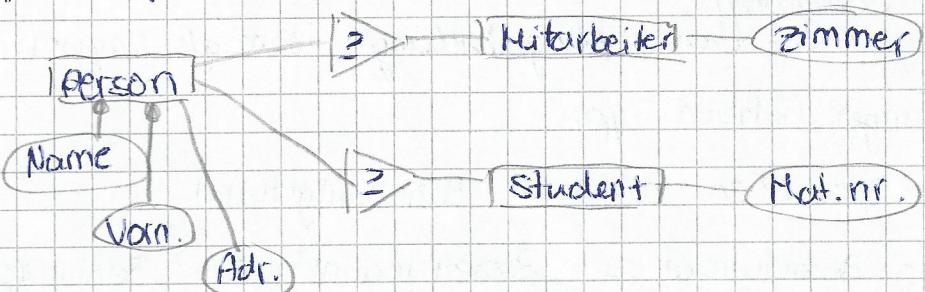
Generalisierung Bsp.:



Partitionierung Bsp.:



Mehrfache SPEZ. Bsp.:



Formalisierung Relationenmodell

- > $U = \text{Universum}$: nichtleere, endliche Menge
- > $A \in U$: Attribut
- > $D_i \in \{D_1, \dots, D_m\}$: Menge nichtleerer, endl. Mengen \rightsquigarrow jeder D_i : Domäne
- > total def. Funktion dom: $U \rightarrow D$
- > dom(A) = Domäne von A ; $w \in \text{dom}(A)$: Attributwert für A
z.B. $U = \{\text{Name}, \text{Alter}, \text{Haarfarbe}, \dots\}$
- > $\{D_1, \dots, D_m\} = \{\{1, 2, 3, \dots\}, \{\text{wahr}, \text{falsch}\}, \{\text{schwarz}, \text{rot}, \text{blond}\}, \dots\}$
- > $\text{dom}(\text{Haut}) = \{1, 2, 3, \dots\}$; schwarz mögl. Attributwert für Haarfarbe
- > $R \subseteq U$: Relationenschema \rightsquigarrow wähle diejenigen Attribute des Universums, die für aktuellen Anwendungsfall sinnvoll
- > Tupel t in $R = \{A_1, \dots, A_n\}$ ist Abbildung $t: R \rightarrow \bigcup_{i=1}^n D_i$
 \rightsquigarrow Zugriff auf Wert eines Tupels: $t_{A_i}(A) \in \text{dom}(A)$
- > Relation r über $R = \{A_1, \dots, A_n\}$ ist endl. Menge von Tupeln $\rightsquigarrow r \subseteq R^n$
- > Menge aller Relationen über R: $\text{REL}(R) := \{r | r(R)\}$
- > $S = \{R_1, \dots, R_p\}$: Datenbankschema = Menge von Relationschemas
- > Datenbank über S: $d(S) = \text{Menge von Relationen } d = \{r_1, \dots, r_p\}$ und $r_i(R_i)$

Lokale Integritätsbedingung

Abbildung aller mögl. Relationen zu einem Schema auf true oder false

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

Erweitertes Relationenschema $R = (B, B)$

$\rightsquigarrow r(R) : r \text{ ist Relation von } R \text{ und } b(r) = \text{true } \forall b \in B$

$\Rightarrow \text{SAT}_R(B) = \{r \mid r(R)\}$ Menge aller Relationen über erw. Rel.schemata R

Identifizierende Attributmenge:

$$K = \{B_1, \dots, B_n\} \subseteq R : \forall t_1, t_2 \in r : t_1 \neq t_2 \Rightarrow \exists B \in K : t_1(B) \neq t_2(B)$$

Schlüssel = minimale id. Attributmenge

Primattribut = Element eines Schlüssels

Primärschlüssel = ausgezeichneter Schlüssel

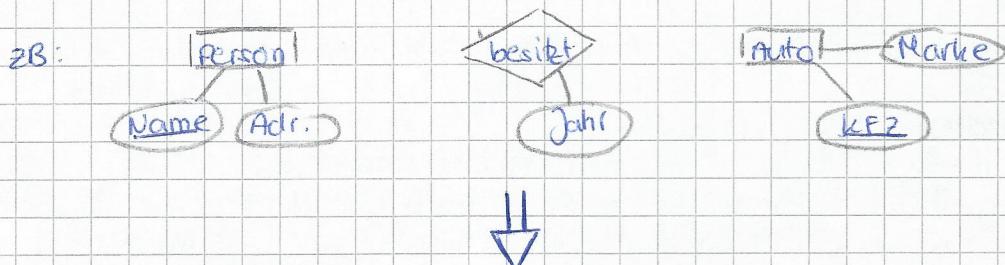
Fremdschlüssel = $X(R_1) \rightarrow Y(R_2)$

\rightsquigarrow Schlüssel + Fremdschlüssel einzige Integritätsbed. im relationalen Modell

Datenbankentwurf

Abbildungen von Modellen, konkret: ER-Modell \rightarrow Relationalmodell

ZIEL: Kapazitätserhaltende Abbildung (genauso viele Instanzen in beiden Fällen darstellbar)



Person	Name	Adr.

Auto	KFZ	Marke

besitz	Name	KFZ	Jahr

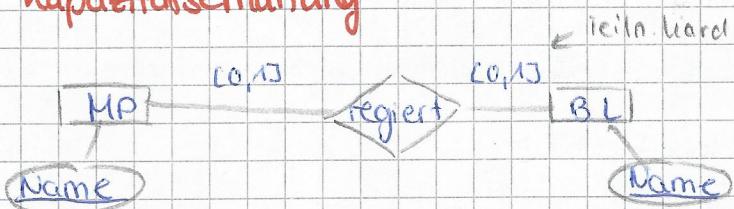
Integritätsbedingungen

Schlüssel kann aus beliebig vielen Attributen bestehen, : A. gibt es mehrere Schlüsselkandidaten

Notation: $K = \{ \{ \text{Vorname}, \text{Name}, \text{Strasse} \}, \{ \text{Name}, \text{Strasse}, \text{Alter} \}, \dots \}$

Kapazitätserhaltung

Erhöhung:



teiln. Kard.

Abbildung von regiert:

$$K = \{ \{ NPName \} \}$$

| Kap. erhöhend | \geq

es gibt MP, die nicht an
bez. teilnehmen!

$$K = \{ \{ NPName \}, \{ BWName \} \}$$

Kap. erhaltencl ✓

vermind.



$$K = \{ \{ Name \} \}$$

| Kap. vermindert | \leq

eine pers. kann nur ein
Aktionär sein, darf
aber mehr

$$K = \{ \{ Name, WKZ \} \}$$

Kap. erhaltencl ✓

Abbildung auf relationales Modell

- Entity-Typen, Beziehungstypen \rightarrow Relationschemata
 - \rightsquigarrow Schlüssel werden übernommen
- Kardinalitäten der Beziehungen \rightarrow Wahl der Schlüssel
- Fremdschlüsse entstehen bei Abb. von Beziehungen \rightsquigarrow setzen Linie von Bez. zu Entity

Auswahl der Schlüssel: (bin. Bez.)

- m:n - Bez.: beide Primärschlüsse werden Schlüssel
- 1:n - Bez.: Primärschlüssel der n-Seite (standardl.; bei funktionaler Notation Seite ohne Pfeilspitze) wird Schlüssel
- 1:1 - Bez.: Beide Primärschlüsse werden je ein Schlüssel, einer wird Primärschlüssel

ER-Konzept
Entity-Typ E;
Attribute von E;
Prim. Schlüssel p;

Abb. auf in rel. Konzept |
Relationschema R;
Attribute von R;
Prim. Schlüssel p;

E₁, E₂ Ent. -Typen
P₁, P₂ der. Prim. Schl.
1:n : E₂ ist n-Seite
IST : E₁ Spez. E.-Typ

Beziehungstyp
klassen Attribute
1:n
1:1
m:n
IST-Bez.

Relationschema, Attribut p₁, p₂:
weitere Attribute

p₂ wird Prim. Schl. der Beziehung
p₁, p₂ werden Schlüssel der Beziehung
p₁ v p₂ wird Prim. Schl. der Bez.
R₁ erhält zusätzl. Schlüssel p₂ \leftarrow Prim. Schl. des allg. Ent.-Typs

Verschmelzen von Relationenschemata

zwingende Beziehungen (1..-J):

- 1:n - Bez.: Entity-Relationschema der n-Seite kann/sollte in das Relationschema der Bez. integriert werden

- 1:1 - Bez.: Beide Ent.-Rel. Schemata trennen sollten in Rel.sch. der Bez. integriert werden

1:1 Bez: Bsp.



→ drei Rel.sch. werden angelegt:

- Prof mit Attributen PANr, Stufe
- Lehrstühle mit Attribut BCZ, Plansstellen
- Hat-Lehrst. mit Prim. schl. der bet. Entity-Typen, hier: PANr, BCZ

→ Verschmelzen nur sinnvoll wenn Beziehungskeilnahme zwingend

Relationaler Datenbankentwurf

Verbund: Relat. 1 \bowtie Relat. 2 \rightarrow join

allg.: $R = R_1 \bowtie R_2 \bowtie R_3 \dots \bowtie R_n$ = Universalrelation von R_1, \dots, R_n

PANr	PLZ	PLZ	ort	ort	BL

dangling tuples: Tupel, die keinen Partner finden werden eliminiert

Universalschlüssel: Schlüssel der Universalrelation

Funktionale Abhängigkeit = FA = FD

Def.: FA einer Relation zwischen Attributmengen X und Y ($X \rightarrow Y$), wenn in jedem Tupel der Relation der Attributwert unter den X-Komponenten den Attributwert unter den Y-Komponenten festlegt.

Anschaulich: In Relation $R(x,y)$ ist Y von X funktional abh., falls zu jedem Wert von X genau ein Wert von Y gehört

$x \rightarrow y$
"X bestimmt Y"

X, Y können jeweils mehrere Attribute sein, z.B. ISBN, Inv.nr. \rightarrow Titel

- Festlegen der FDs Bestandteil des Schemadefinitions, beschreiben Anwendungsszenario genauer
→ ergeben sich nicht aus Datenbestand

Schlüssel: Spezialfall der FD. Schlüssel X liegt vor, wenn für Rel. schema R die FD $X \rightarrow R$ gilt + X ist minimal

→ DBMSe unterstützen Überprüfung von FDs: Allg. nicht
⇒ Ziel des OB-Entwurfs: FDs in Schlüsselabh. umformen ohne semantische Information zu verlieren

Ableitung: $F = \text{Menge von FDs}$
 $f = \text{einzelne FD}$

Gilt für f über R $\text{SAT}_R(f) \subseteq \text{SAT}_R(F)$, dann impliziert F die FD f
 $F \subseteq f$

Hülle: $F_R^+ := \{f \mid (f \text{ FD über } R) \wedge F \subseteq f\}$ | Überdeckung: F äquivalent zu G ($F \equiv G$)
 $\Leftrightarrow F^+ = G^+$

Ableitungsregeln:

R Reflexivität $\{x\} \Rightarrow x \rightarrow x$

A Akkumulation $\{x \rightarrow y, z \rightarrow vw\} \Rightarrow x \rightarrow yzv$

P Projektivität $\{x \rightarrow y\} \Rightarrow x \rightarrow y$

Regeln sind

- gültig
- vollständig
- unabhängig

Membership-Problem: $x \rightarrow y \in F^+ ?$

→ Hülle einer Attributmenge X bzgl. F : $X_F^* := \{A \mid X \rightarrow A \in F^+\}$

A einzelnes Attribut
 X kann Menge sein

ZB. $F = \{PNU \rightarrow PLZ, PLZ \rightarrow \text{stadt}, \text{stadt} \rightarrow \text{Land}\}$

$$\{\text{stadt}\}_F^* = \{\text{stadt}, \text{Land}\}$$

$$\{PLZ, \text{stadt}\}_F^* = \{PLZ, \text{stadt}, \text{Land}\}$$

→ Löse " $x \rightarrow y \in F^+ ?$ " durch " $y \subseteq X_F^* ?$ " (lineare Zeit)

RAP-Algorithmus

- (1) bestimme X , setze $X^* = X$ (R-Regel für X)
- (2) Gibt es FD $f_1 = X_1 \rightarrow Y_1 \in F$ mit $X_1 \subseteq X^* ?$
- (3) wenn ja: vergrößere X^* gemäß $X^* = X^* \cup Y_1$ (A-Regel)
- (4) Schritt 2+3 solange bis X^* stabil → Hülle
- (5) $Y \subseteq X^* ? \rightarrow X \rightarrow Y \in F^+ \text{ (P-Regel)}$

Updateanomalie:

Ein Wert der mehrfach auftaucht verändert sich \rightarrow kindern mehrere
 Einträge nötig
 \rightsquigarrow aufwändig
 \rightsquigarrow fehleranfällig

Einfügeanomalie:

Neuer Eintrag in Tabelle mit Null-Wert \rightsquigarrow was wenn Schlüssel?

Löscharomatie:

Eintrag wird gelöscht \rightarrow alle Infos aus Tupel werden gelöscht, auch
 wenn manche noch interessant, aber nicht in aktueller Relation

Transformationseigenschaften

Abhängigkeitstreue: Menge der erfassten Abhängigkeiten äquivalent zur
 Menge der im System darstellbaren Abhängigkeiten
 (etwa (Fremd-)Schlüssel)

Formal:

$S = \{(R_1, K_1), \dots, (R_p, K_p)\}$ (lokal erweitertes DB-Schema)

F Menge von Abh.

S charakterisiert vollst. F (ist abh. k. t. bzgl. F)

\Leftrightarrow

$$F \equiv \{K \rightarrow R_i \mid (R_i, K_i) \in S, K \in K_i\}$$

Verbundtreue: Originalrelation soll aus zerlegten Relationen mit natürlichen
 Verbund zurückgewonnen werden können

Dekomposition einer Attributmenge X in X_1, \dots, X_p mit $X = \bigcup_{i=1}^p X_i$
 heißt Verbundtreu (\bowtie -frei) bzgl. einer Menge von Abh. G über
 $X \Leftrightarrow \forall r \in \text{SAT}_X(G) : \pi_{X_1}(r) \bowtie \dots \bowtie \pi_{X_p}(r) = r$ gilt

Kriterium für Verbundtreue: G Menge funkt. Abh., Dekomp. von X in X_1, \dots, X_p abh. zu G

Dann: $\exists i \in \{1, \dots, p\} : X_i \rightarrow X \in G^\perp \Rightarrow$ Dek. von X in X_1, \dots, X_p ist verbundfrei bzgl. G

Minimale Teilmenge von X : universalschlüssel (da es ges. Menge X funktional bestimmt)

Anschaulich: eine Relation muss Universalschlüssel enthalten

KURZ:

\Rightarrow Abhängigkeitstreue = alle geg. Abhängigkeiten sind durch Schlüssel repräsentiert

\Rightarrow Verbundtreue = Originalrelationen können durch Verbund der Basisrelationen
 wieder gewonnen werden

Schema-Eigenschaften

Normalformen

Erste Normalform (1NF): nur atomare Attribute in Relationschema
d.h. zusammengesetzte/mengenwertige Attribute (a_1, b_3) nicht erlaubt

ZB:

InvNr	ISBN	Autor	InvNr	ISBN	Autor
0007	3-125	J. Bond	0007	3-125	J. Bond
1201	3-111	{Heuer, Schöll}	1201	3-111	Heuer
~> keine 1NF			1201	3-111	Schöll

\Rightarrow

~> 1NF

Zweite Normalform (2NF): Jedes Nicht-Primattribut voll funktional abhängig von allen ganzen Schlüsseln (d.h. nicht nur von einem Teil des Schlüssels)
anspruch: partelle Abhängigkeit

β voll funkt. abh. von α , wenn aus α kein Attribut entfernt werden kann;
so dass FD immer noch gilt

Relation ist in 2NF \Leftrightarrow (1) ist in 1NF und

(2) \forall -Attribut $\alpha \in$ Relation gilt:

- α ist Teil eines Schlüsselkandid. od.
- α von Schlüsselkandid. abh. aber α nicht von echten Teilmenge eines Schl. kandid. abhängig

~> jede Relation in 1NF, deren

Schl. kandid. aus je einem Attribut
bestehen sind in 2NF

\rightarrow 2NF erreichen: Elimination der rechten Seite der part. abh. und Kopie der linken Seite

Dritte Normalform (3NF): Relations in 3NF \Leftrightarrow Relation.s. in 2NF und
kein Nichtschlüsselattr. von Schlüssel. transiv abhängig

Transitive Abh.: Schlüssel K bestimmt Attributmenge X funktional, diese
aber auch Attributmenge Y $K \rightarrow X \rightarrow Y$

~> Nichtschl. attr. darf nicht von Menge aus Nichtschl. attr. abh. sein,
sondern nur von Primärschlüssel

\rightarrow 3NF erreichen: Elimination von Y , Kopie von X

Boyce-Codd-Normalform (BCNF): Relat. sch. R mit FDs F in BCNF,
wenn für jede FD $\alpha \rightarrow \beta$ gilt
(1) $\beta \subseteq \alpha$ (triviale Abh.) oder
(2) α Schlüssel von R

~> Dekomposition notwendig

Übersicht:

- > 1NF nur atomare Attribute
- > 2NF keine part. Abh. eines Nicht-Primattr. von einem Schlüssel
- > 3NF keine transitive Abh. eines Nicht-Primattr. von einem Schlüssel
- > Minimalität min. Anzahl von Rel. Schemata, die die anderen Eigenschaften erfüllt

Dekomposition

Prinzip: Immer wenn trans. Abh. $X \rightarrow Y \rightarrow Z$ wird Relation, an dieser Stelle 'zerlegt'

Normalisierungsschritt: Falls $K \rightarrow X \rightarrow Y$, aus R Attr. menge Y eliminieren und mit X in neues Rel. schema stecken
 - $R = (R, K)$ und F über R geben
 - falls R in 3NF ist: fertig
 - sonst: Schritt wdh.

Bsp: $U = \{\text{PANr, PLZ, Ort, Land, Stadt}\}$
 $F = \{\text{PANr} \rightarrow \text{PLZ}, \text{PLZ} \rightarrow \text{Ort}, \text{Ort} \rightarrow \text{Land}, \text{Land} \rightarrow \text{Stadt}\}$

$$(U, K(F)) = (\{\text{PANr, PLZ, Ort, Land, Stadt}\}, \{\text{PANr}\})$$

' $K \rightarrow X \rightarrow Y$ ' jetzt: , PANR \rightarrow Land \rightarrow Stadt'

Neue Rel.: $R_1 = \{\text{Land, Stadt}\}$
 $R_2 = \{\text{PANr, PLZ, Ort, Land}\}$

wdh. mit $R_2 \rightsquigarrow$ nächste trans. FD z.B. PANR \rightarrow Ort \rightarrow Land

- ~ ④ 3NF als Ergebnis, verbündet
- ⑤ keine abh. neue, Minimalität
- ⑥ NP vollst. (Schlüsselsuche)
- ⑦ Reihenfolgenabhängigkeit ~ FDs können verloren gehen

Syntheseverfahren

→ erreicht 3NF + Minimalität, reihenfolgenunabh.

Verfahren: (1) Eliminiere Redundanzen durch Entfernen redundanter / überflüssiger FDs + Attribute

(2) Fasse FDs zu "Äquivalenzklasse" zusammen: FDs in eine Klasse, die gleiche/äquivalente linke Seiten haben
 ~ pro Äqu. Kl. ein Rel. schema

z.B. $F = \{A \rightarrow B, AB \rightarrow C, A \rightarrow C, B \rightarrow A, C \rightarrow E\}$ ~ redundante FD: $A \rightarrow C$

nach (1): $F' = \{A \rightarrow B, AB \rightarrow C, B \rightarrow A, C \rightarrow E\}$ ~ überfl. Attr. B in $AB \rightarrow C$

result. FDs: $\underbrace{A \rightarrow B, A \rightarrow C, B \rightarrow A, C \rightarrow E}_{\text{Äq. Klasse}}$

nach (2): $(ABC, \{\text{A}\}, \{\text{B}\})$, $(CE, \{C\})$

Mehrwertige Abhängigkeiten = Multi-value Dependency (MVD)

Jeder Wert des abh. Attributes kommt in Kombination mit allen Werten der anderen Attribute vor
→ Rechenlast behaftet

Bsp.: Attr. A,B,C , MVD $A \Rightarrow\Rightarrow B$ und
 $(a_1, b_1, c_1) \in R + (a_2, b_2, c_2) \in R$
dann auch $(a_1, b_1, c_2) \in R + (a_2, b_2, c_1) \in R$

Vierte Normalform (4NF): Kleine MVOs zulassen → Relation aufspalten

→ MVD $X \Rightarrow\Rightarrow Y$ trivial, wenn keine weiteren Attribute im Rel. schema.

Nebenläufigkeit, Transaktionen

Transaktion: Kernkonzept zur Sicherstellung korrekter Funktionsweise im Mehrbenutzerbetrieb
Folge von Lese-, Schreiboperationen, die DB von konsistentem in geänderten Zustand überführen

es gilt: ACID Prinzip

Attomicity: Transaktion wird vollst. oder nicht ausgeführt

Consistency: vor und nach Transakt. gelten sämtliche Integritätsbed.

Isolation: Nutzer hat Eindruck, er arbeite allein auf DB

Durability: Ergebnis wird dauerhaft gespeichert

Synchronisationskomponente stellt Konsistenz bei vielen gleichzeitigen Zugriffen sicher

Nebenläufigkeit soll transparent sein, "Illusion", dass nur ein Nutzer ist

Serielle Ausführung:

- kein Synchron.-aufwand
- DB nach jeder Transakt. konsistent

ABER: extreme Wartezeiten, schlechte Ausnutzung der Ressourcen

Unkontrollierte nicht-serielle Ausführung:

Probleme: lost updates, inkonsistente Lesezugriffe, Dirty Reads, Phantome

Lost Update: Programme überschreiben gegenseitig Werte, ohne diese zu speichern

Dirty Read: Reads von uncommitteten Updates

Non-repeatable-Reads: inkonsist. Lesezugr.; Progr. liest Datenobj. mehr als einmal. sieht Änderung, die anderes Prgr. durchgeführt hat

Phantom: Berechnung von Änderungen auf veralteten Werten

Transaktion = partielle Ordnung:

- vollst. Reihenfolge der Op. muss nicht bekannt sein
- erzeugt Freiheitsgrade beim anordnen

Konflikt: Zwei Op. p, q konfliktieren := p, q greifen auf gleiches Datenobj. zu, und p oder q ist Schreiboperation

Transaktion - formale Def.

Transaktion ist partielle Ordnung mit Ordnungsrelation \leq , so dass gilt:

- (1) $T \subseteq \{r[x], w[x] \mid x \text{ ist Datenobj.}\} \cup \{a, c\}$ $a = \text{abort}$ $c = \text{commit}$
- (2) $a \in T \Leftrightarrow c \in T$
- (3) $t = \text{oder } a \text{ für jede andere Op. gilt: } p \in T : p \leq t$
- (4) wenn $r[x], w[x] \in T$, dann $r[x] < w[x]$ oder $w[x] < r[x]$

$$\text{z.B. } r[x] \rightarrow w[x] \xrightarrow[c]{} \\ r[y] \rightarrow w[y]$$

ist TA

$$r[x] \rightarrow w[y] \xrightarrow[c]{} \\ r[y] \rightarrow w[x] \xrightarrow[c]$$

ist keine TA

Histories

Ausführung der Op. mehrerer TAs, die nebelaufig ablaufen

$$T = \{T_1, \dots, T_n\} \text{ Menge von TAs}$$

Vollständige Historie H über T := partielle Ordnung mit Ordnungsbez. \leq_H , so dass

- (1) $H = \bigcup_{i=1}^n T_i \rightarrow \text{keine Op. vergessen}$

- (2) $\leq_H \supseteq \bigcup_{i=1}^n \leq_i \rightarrow \text{partielle Ordnung aller beteiligten Transakt. enthalten}$

$p, q \in H$ konfliktieren $\Rightarrow p \leq_H q$ oder $q \leq_H p$

Historie: Präfix einer vollst. Historie

$$\text{Bsp.: Gegeben } T_1 \quad r_1[x] \rightarrow w_1[x] \xrightarrow[c]{} \\ r_1[y] \rightarrow w_1[y]$$

vollst. H.:

$$r_2[x] \rightarrow w_2[x] \rightarrow c_2 \\ r_1[x] \rightarrow w_1[x] \xrightarrow[c_1]{} \\ r_1[y] \rightarrow w_1[y] \xrightarrow[c]$$

✓

keine H.:

$$r_2[x] \rightarrow w_2[x] \rightarrow c_2 \\ \text{fehlt: } \nearrow \rightarrow \\ r_1[x] \rightarrow w_1[x] \xrightarrow[c_1]{} \\ r_1[y] \rightarrow w_1[y] \xrightarrow[c]$$

X

H., keine vollst. H.:

$$r_2[x] \\ r_1[x] \rightarrow w_1[x] \xrightarrow[c_1]{} \\ r_1[y] \rightarrow w_1[y] \xrightarrow[c]$$

✓

! History muss nicht korrekt sein, kann z.B. lost Updates enthalten!

Serialisierbarkeit - Äquivalenz der committed projection

Committed Projection: einer History H : C(H)

resultiert aus TAs, in dem alle Ops, die nicht committed sind gelöscht werden

H serialisierbar, wenn C(H) zu schneller History H_s äquivalent

prefix commit-closed: Abschlossenheitseigenschaft: Effekt gilt auch für jeden Präfix der C(H)

Äquivalenz von Histories:

Konflikt-Äquivalenz: H, H' konflikt-äquivalent wenn

(1) gleiche TAs, gleiche Ops

(2) gleiche Ordnung konfliktierender Ops, d.h. gleiche Konfliktrelationen conf()

> Konflikt relation:

bestimme für alle Paare von Transakt. in History konflikt. Ops

→ Zugriff auf gleiches Datendobj. durch untersch. TA

→ mind. eine der Ops ist Schreibzugriff

z.B. $H = r_1[x] \rightarrow r_1[y] \rightarrow w_2[x] \rightarrow w_1[y] \rightarrow r_2[z] \rightarrow w_1[x] \rightarrow w_2[y] \rightarrow c_1 \rightarrow c_2$

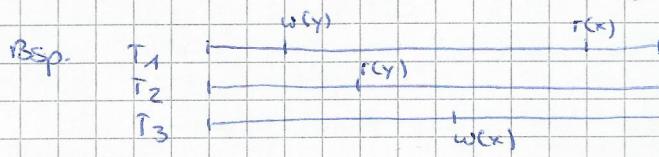
$$\text{conf}(H) = \{(r_1, c_2), (w_2, x), (w_1, x), (r_1, y), (w_2, y), (w_1, y), (w_2, y)\}$$

Serialisierbarkeitsgraph = Abh. keitsgraph

Test, ob ein Schedule (TAs zusammen mit Ops / History serialisierbar)

Knoten = TAs, Kante von T_i nach T_j wenn (op_i[..], op_j[..]) ∈ Konfliktrel.

→ Schedule serialisierbar, wenn Abh. k. gr. zyklusfrei



zykl. → serialisierbar
Serialisierungsreihenfolge: T₃ < T₁ < T₂

Probleme:

- Ansatz nicht praktikabel: Serialisierbarkeit nur im Nachhinein überprüfbar

- Fehlertoleranz: H = r₁[c₁] → w₁[x] → r₂[x] → c₂ → w₂[c₂] → c₂

Abbruch T₁ führt zu Abbruch T₂

Rücksetzbarkeitsklassen

Liest-von-Relation: T_i liest x von T_j (\Rightarrow T_j x vorher geschrieben und kein abort
 $w_j[x] < r_i[x]$ $c_j < r_i[x]$)

Rücksetzbar: (RC)

Commit für T_i erst erlauben, wenn alle TAs T_j, von denen T_i liest committed haben
 $w_1[x] \rightarrow r_2[x] \rightarrow w_2[y] \rightarrow c_1 \rightarrow c_2$

formal: $\forall (T_i, T_j) (i \neq j): T_i \text{ liest } x \text{ von } T_j \text{ und } c_i \in H: c_j < r_i[x]$

Vermeiden kostspieliger Abbrüche (ACA)

nur lesen von committed TAs: $w_1[x] \rightarrow c_1 \rightarrow r_2[x] \rightarrow w_2[y] \rightarrow c_2$

formal: $\forall (T_i, T_j) (i \neq j): T_i \text{ liest } x \text{ von } T_j \text{ und } c_i \in H: c_j < r_i[x]$

Striktheit (SI)

Beim Schreiben von x darauf warten, dass alle TAs, die x verändert haben committed/aborted sind
 formal: wenn $w_i(x) < 0 \wedge c_i < i+j$:
 entweder $c_j \leq 0; c_j < 0; c_j < 0; c_j < 0$
 → kein Objekt x einer noch nicht beendeten TA darf gelesen/überschrieben werden

Rollback, Restart, Locking

Rollback = Rückgängig machen einer TA

Restart = meist in Verbindung mit rollback & restart

Locking: lock mögl. für jedes Datenobj.; jede Art von Op:

$rL[x]$: read lock auf x

$wL[x]$: write lock auf x

→ erfordert zwingend unlock $rL[x]$ | $wL[x]$ bzw. $u[x]$

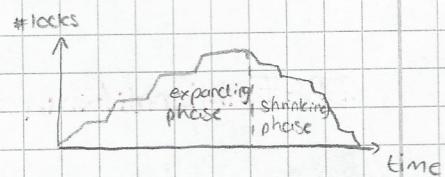
Locking rules:

- $wL[x]$ nur nach $wL[x]$ der entspr. TA
- $rL[x]$ nach $rL[x]$ oder $wL[x]$
- nur locks auf nicht gesperrten Obj.
- nur Verschärfung von $rL[x]$ durch $wL[x]$ möglich
- nach $u[x]$ durch T; darf T; x nicht mehr sperren
- vor commit müssen alle Sperrungen aufgehoben werden

→ Problem: Dead-, Livelocks werden nicht verhindert, sichert keine Konfliktserialisierbarkeit zu

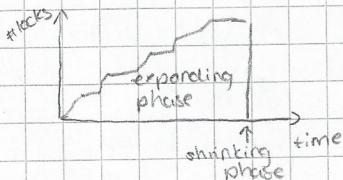
Zwei-Phasen-Sperprotokoll (2PL)

- Phase in der locks genommen werden
- Phase in der locks freigegeben werden
- stellt Serialisierbarkeit sicher
- verhindert keine Deadlocks



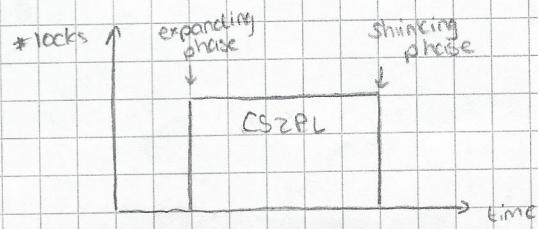
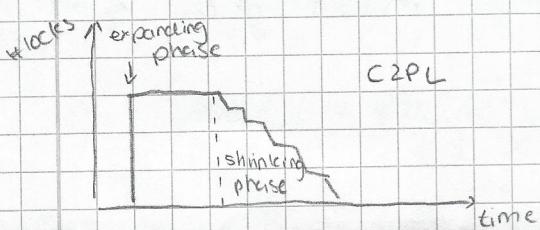
Strenge Zwei-Phasen-Sperprotokoll (SZ2PL)

- verhindert kaskadierende Abbrüche durch unlock erst am Ende der TA
- weniger Histories mögl., keine Deadlock-vermeidung



Konservativer 2PL (C2PL)

- atomare Anforderungsphase
- verhindert Deadlocks
- weniger Histories mögl., kein ACA
- Ops. müssen zuvor bekannt sein



Konsistenz in Cloud-basierten Systemen

Fragmentierung, Reparation

- Verteilung:
- last auf mehrere Knoten verteilt
 - höhere Lokalität des Zugriffs \rightarrow Beschleunigung
 - höhere Ausfallsicherheit

Koordination vs. Serialisierbarkeit:

- Ablaufungsentschl. müssen widerspruchsfrei sein \rightarrow ein Knoten muss sie fallen
- Skalierbarkeit schwierig/unmöglich - ein Knoten muss alle Transakt. kennen (zur Verwaltung)

CAP Theorem: 'consistency', 'availability', 'partition tolerance'

- Konsistenz: 'single system images': zeigt stets letzte Version des Datenobj., egal auf welchem Knoten
- CAP Theorem \approx wenn Netzwerkpartitionen möglich, sind hohe Verfügbarkeit und Konsistenz des Datenbestands i. A. unvereinbar

Eventual Consistency:

Wenn ab bestimmten Zeitpunkt keine Updates mehr, dann werden irgendwann alle Lesezugriffe auf ein Datenobj. den gleichen Wert zurückliefern

\rightarrow keine Safety („nothing bad happens“) nur Liveness („es gibt Fortschritt“)

Kompensation:

- Konsistenzverletzungen ziehen i. A. Kompensationen nach sich (Entschuldigung),
 \hookrightarrow Prozesse müssen entworfen werden (Entschuldigungsdesign) Präsent, ...)

\Rightarrow klassische starke Konsistenz im verteilten Fall (z.B. Cloud) teuer
bzw. nicht erreichbar
 \hookrightarrow schwächere Konsistenzbegriffe oft ausreichend

DB-Funktionalität in der Cloud

Binäre Operationen: Implementierung: tupelweiser Vgl. der Input-Mengen

Nested-Loop: für jedes Tupel der externen Relation S Iteration über innere Relation R

Merge-basiert: Iteriere über R und S, beide sortiert, Tupel für Tupel. in Sortierrichtungsfolge

Nested-Loop-Join: Relation S muss wiederholt aus Sekundärspeicher geholt werden \rightarrow teuer ($O(ns \cdot nr)$)

\rightarrow Block-Nested-Loop-Join: Blöcke von Tupel in Speicher holen $O(br \cdot bs)$

Merge-basiert: $x := R \cap S$, sortiere R, S nach x

(1) $tr(x) < ts(x)$, lies nächstes tr $\in R$

(2) $tr(x) > ts(x)$, lies nächstes ts $\in S$

(3) $tr(x) = ts(x)$, merge tr mit ts und allen Nachfolgern von ts mit gleichem x-Wert

alle Tupel gleiches x: $O(nr \cdot ns)$

x-Schlüssel in Rad. S: $O(nr \log nr + ns \log ns)$

Rel. sortiert: $O(nr + ns)$

Histogramme: zeigt Häufigkeit der Werte

→ Kompression: Einteilen der Dimension in vorgegebene Anz. Intervalle = Buckets

→ nützlich bei Anfrage auf ein Attribut

Blockierende / nichtblock. Op.:

Op. blockiert = Ergebnis des Op. muss vor Ausführung des nachfolgenden Op.s vollst. berechnet sein

Zugriffe:

synchron = Zugriff innerhalb einer Transaktion

asynchron = mehrere Transaktionen

Replikation: mehrere Kopien der Datenobj.

⊕ bessere Lese-Leistung, höhere Verfügbarkeit beim Lesen

⊖ umgekehrt beim Schreiben

Service-level Agreements: Vereinbarung zw. Client und Server, die Ausführung des Dienstes betreffend

Zustände: zustandslos, zustandsbehaftet

Quorum: Konsistenz mit Quorum-basiertem Protokoll:

Lesen: Lies Mind. Anzahl von Versionen R, nimm aktuellste

Schreiben: aktualisiere Mind. Anzahl von Kopien W

↪ jede Kopie hat Versionsnummer

üblich: $R + W > N$

Synchroner Zugriff

- muss Spez. für write für jede Kopie bekommen
- Koord. Knoten: kümmert sich um Spez., führt Buch
- ⊕ warten auf langsamsten Knoten, kein Fortschritt bei Ausfall des Knotens, keine Skalierung

↔ asynchroner Zugriff

- ⊕ höhere Verfügbarkeit
- ⊖ weniger Konsistenz

Eventual Consistency: jedes Update muss bei allen Replikaten ankommen

Scale-out: soll möglich sein ohne Auswirkungen auf System/Betrieb
↪ Hinzunahme neuer Knoten

Symmetrie: keine Knoten mit Sonderrolle / besonderer Verantwortlichkeiten

Heterogenität: Knoten mit untersch. Fähigkeiten / Leistungsfähigk. sind Norm

Chord: P2P-System, Datenstruktur: Identifier circle, chord ring

→ jedes Datenobj. hat Schlüssel (m Bits)

→ weise Schlüssel k dem (im Uhrz. Sinn) nächsten Knoten zu

→ Suche: finger table: Verweise zu anderen Schlüsseln

↪ log. Aufwand

Vector Clocks

liste von (knoten, zähler) - Paaren , eine liste pro Version

Partielle Ordnung: A ist Vorgänger von Version B, wenn

- zähler für jeden Prozess im Zeitstempel (A) ist gleich zähler im Zeitstempel (B)
- und mind. einer dieser Zähler strikt kleiner

→ Quorum-basiert: i. Alg. Konsistenz im verteilten Fall, Skalieren nicht

Scale Independence

Anfrage scale independent = Laufzeitverhalten unabh. von DB Größe

Klassifikation von Anfragen gemäß Aufwand

Klasse 1: konstant, z.B. Nutzer-ID

Klasse 2: beschränkt, z.B. explizite Begrenzung (max. 5000 Freunde)

Klasse 3: linear / sublinear, z.B. Ausgabe aller Kunden

Klasse 4: super-linear, z.B. Clustering-Algo

Ziel: Klasse 1+2