

# 1 Grundlagen

- Datenmodell: Strukturen (Typdefinitionen + Extension, Abstrakte Obj.), Operationen (Änderungen von Strukturen, Vereinigung, Durchschnitt), Randbedingungen (Regeln denen die Struktur gehorchen muss)
- inhärente (logische: Bei Mengen keine Duplikate) und explizite (benutzerspezifizierte: ) Randbedingungen
- Konzeptuelle Modelle: UML und ER
- Logische DB Modelle: Relationales DBmodell und OODBmodell
- Physische Datenstrukturen: B\*-Baum, Hashing
- ER: Entities (Rechtecke), Relations (Rauten) + Attribute (Kreise) und Rollen (Text an Ausgangskanten einer Relation). Relationen können auch Attribute haben, schwache Entities.
- Warum Datenbanken? Redundanz und Inkonsistenz, beschränkte Zugriffsmöglichkeiten, Mehrbenutzerbetrieb, Verlust von Daten, Integritätsverletzungen, Sicherheitsprobleme, hohe Entwicklungskosten.
- DB-Entwurf: Anforderungsanalyse, konzeptueller und logischer Entwurf, Implementierung, Validierung und Akzeptanztest, Betrieb
- isA: partiell oder total, disjunkt (Pfeile auf Spezialisierung) oder nicht disjunkt (Pfeile auf Generalisierung)
- Entity-Typ vs. Attribut: Entity-Typ, falls das Konzept selber Eigenschaften hat oder mehrfach im Modell auftaucht
- Generalisierung/Spezialisierung vs. Attribut: Generalisierung, falls das Unterkonzept zusätzliche Attribute oder Bezeichnungen hat
- zusammengesetztes Attribut vs. Entity: zusammengesetzt, falls der Name des Attributs eine spezielle Bedeutung hat. Entity-Typ, falls mehrfach verwendet und die Information gemeinsam genutzt wird

# 2 Relationales Modell

- $dom(A)$ : Wertebereich von A
- Attribut: Spalte einer Tabelle
- Tupel: Zeile einer Tabelle
- Wann welche ER nach RM Übersetzung: disjunkt oder nicht disjunkt, partiell oder total, viele oder wenige Attribute
- zwei Abfragemöglichkeiten: relationale Algebra oder Relationenkalkül (Domänenkalkül, Tupelkalkül)
- Selektion:  $\sigma_{\text{Bedingung}}$
- natural Join:  $r \bowtie s$ , Auswahl der Zeilen von  $r$  (an die die passende Zeile von  $s$  angehängt wird), die eine existierende Schlüsselrelation nach  $s$  haben
- Projektion:  $\Pi_{\text{Spalten}}$
- Umbenennung:  $\rho_{B \leftarrow A}, A \text{ AS } B$
- Theta-Join: Kreuzprodukt
- Semi-Join:  $r \ltimes s$ , wie der natural Join, nur dass die passende Zeile von  $s$  diesmal nicht an die jeweilige Zeile von  $r$  angehängt wird
- UNION: Ergebnistypen müssen gleich sein, Duplikate werden eliminiert (bei UNION ALL nicht)

- Tupelkalkül (TK):  $\{t|P(t)\}$  zB  $\{p|p \in \text{Professoren} \wedge p.\text{Rang} = "C4"\}$ ,  $\exists$  und  $\forall$  sind erlaubt
- Domäne einer Formel: Menge aller Konstanten und aller Attributwerte, die die Formel referenziert
- TK sicher: Wenn das Ergebnis des Ausdrucks eine Teilmenge der Domäne ist
- Domänenkalkül (DK):  $\{[v_1, v_2, \dots, v_n]|P(v_1, v_2, \dots, v_n)\}$ ,  $v_1, \dots, v_n$  sind die einzigen freien Variablen in  $P$
- DK sicher: **3 Regeln**
- Armstrong Axiome (korrekt und vollständig):
  - Reflexivität:  $Y \subseteq X \Rightarrow X \rightarrow Y$
  - Verstärkung:  $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$
  - Transitivität:  $X \rightarrow Y \wedge Y \rightarrow Z \Rightarrow X \rightarrow Z$
- RAP-Regeln: 3 (4) Ableitungen der Armstrong-Axiome (Reflexivität, Akkumulation, Projektivität)
- 1NF: keine mehrwertigen Attribute, keine zusammengesetzten Attribute
- 2NF: jedes nicht primäre Attribut ist jeweils vom ganzen Schlüssel abhängig, nicht nur von einem Teil
- 3NF: Jedes Nichtschlüsselattribut ist von keinem Schlüsselkandidaten transitiv abhängig. Für jede FD  $\alpha \rightarrow B$  muss eine der folgenden Regeln gelten:
  1.  $B \in \alpha$ , die FD ist trivial
  2.  $B$  ist prim (ist in einem Kandidatenschlüssel enthalten)
  3.  $\alpha$  ist Superschlüssel
- BCNF: Jedes Attribut ist direkt abhängig von jedem Schlüssel (jede Determinante ist ein Superschlüssel). Eine der beiden folgenden Regeln muss für jede FD  $\alpha \rightarrow \beta$  gelten:
  1.  $\beta \subseteq \alpha$  (triviale Abhängigkeit)
  2.  $\alpha$  ist Superschlüssel
- Syntheseargorithmus: Erst Linksreduktion (ist zB bei der FD  $AC \rightarrow D$  das  $D$  nur durch  $A$  oder  $C$  determiniert?), dann Rechtsreduktion (komme ich mit  $AC \rightarrow \emptyset$  auch auf  $D$ ?), dann Elimination von FD mit leerer Menge und zum Schluss die Regeln mit den selben Determinanten zusammenfassen. Mit den finalen FDs kann das neue Relationenschema aufgebaut werden.
- Dekompositionsalgorithmus:  $Z = \{R\}$ . Solange es noch ein Relationenschema  $R_i \in Z$  gibt, das nicht in BCNF ist (siehe Punkt BCNF), führe folgende Schritte durch:
  1. Zerlege  $R_i$  in  $R_{i1} := \alpha \cup \beta$  und  $R_{i2} := R_i - \beta$
  2. Ersetze  $R_i$  in  $Z$  durch  $R_{i1}$  und  $R_{i2}$
 Übrig bleibt eine Menge von BCNF-Relationen

### 3 Verteilte DBMS

- Fragment als kleinste Einheit, auf die zugegriffen wird, zB  $A_1 = \{r_1, r_2, r_3, r_6, r_7\}$ ,  $A_2 = \{r_4, r_5, r_6, r_7\}$ , dann sind  $R_1 = \{r_1, r_2, r_3\}$ ,  $R_2 = \{r_4, r_5\}$ ,  $R_3 = \{r_6, r_7\}$  die Fragmente
- horizontal: Zerlegung in disjunkte Tupelmengen
- vertikal: Attribute mit gleichen Zugriffsmustern werden zusammengefasst
- Read-Only-Anwendungen: vier Transparenzstufen:
  - Fragmentierungstransparenz** VDBMS kümmert sich um alles
  - Allokationstransparenz** Nutzer muss die Fragmentierung kennen
  - Lokale-Schema-Transparenz** Nutzer muss Fragmentierung und Station kennen
  - keine Transparenz** wie Lokale-Schema-Transparenz mit zusätzlicher Kenntnis der lokalen DBMS aller Standorte

## 4 RDBMS

- optimal nested loop: Vorauswahl und dann Vergleich mit der Auswahl anstatt mit der gesamten Menge
- Join hat exponentiell wachsenden Berechnungsaufwand, Semi-Join nur linearen
- Quantgraphen: Abfrage durch Semi-Joins berechenbar? Knoten: EACH, SOME und Allquantor des Tupelkalküls, ungerichteter Graph zyklfrei: gutartig (sonst böseartig)
- B\*-Bäume: Werte nur in den Blättern (Referenzen auf Werte), sonst nur Verzweigungsinformationen
- Synchronisationsprobleme: Lost Update (zwei Nutzer schreiben eine Variable), Dirty Read ( $w_1(x), r_2(x), a_1$ ), Phantom Problem (Nutzer 1 arbeitet mit mehreren Variablen, Nutzer 2 schreibt eine dieser Variablen neu, Nutzer 1 arbeitet weiterhin mit dem alten Wert)
- ACID:
  - Atomicity** alle oder keine Operation wird bei einer Transaktion durchgeführt
  - Consistency** vor und nach der Transaktion ist die DB konsistent
  - Isolation** isolierte Ausführung einer Transaktion
  - Durability** Transaktion erfolgreich  $\rightarrow$  Effekte müssen fortbestehen, auch nach einem Hardwaredefekt
- $conf(s)$ : alle Paare von Operationen auf einer Variable von verschiedenen Transaktionen, bei denen mindestens eine Operation eine Schreiboperation ist (bereinigt von Abbrüchen)
- Konfliktgraph: Graph von  $conf(s)$ , wenn zyklfrei (ungerichtet), dann ist  $s$  (Schema) in CSR (konfliktserialisierbar)
- Transactions:
  - RC** (Recoverability) Keine Transaktion wird freigegeben, bis alle Transaktionen, von denen sie gelesen hat, auch freigegeben wurden.
  - ACA** (Avoid Cascading Aborts) Eine Transaktion darf nur von bereits erfolgreich abgeschlossenen Transaktionen lesen.
  - ST** (Strict) Ein Objekt darf weder gelesen noch überschrieben werden (von einer fremden Transaktion), bis die Transaktion, die es zum letzten Mal geschrieben hat, abgeschlossen ist. Vermeidet w-r und w-w-Konflikte.
  - RG** (Rigorous) Ein Schedule ist rigoros, falls er strikt ist und kein Objekt überschrieben wird, bis alle Transaktionen, die das Objekt gelesen haben, abgeschlossen sind. Vermeidet zusätzlich r-w-Konflikte.
- $RG \subset ST \subset ACA \subset RC$
- Sperren:
  - 2PL** nach dem ersten  $ou_i$  kommt kein  $ql_i$  mit  $o, q \in \{r, w\}$ , nicht dead-lock-frei
  - konservatives 2PL** als erstes alle Sperren setzen
  - striktes 2PL** (S2PL) alle Schreibsperren bis EOT halten
  - starkes 2PL** (SS2PL) alle Sperren bis EOT halten
  - Nachteile von 2PL-Protokollen** wenige große Sperren  $\rightarrow$  viele Konflikte, mehrere kleine Sperren  $\rightarrow$  hohe Kosten
- MGL: jede Transition wählt selbständig eine geeignete Sperr-Granularität (Baumstruktur)
- Recovery:
  - comittete Transaktionen vor einem Fehler brauchen ein REDO, falls sie nicht im stabilen Speicher sind
  - noch aktive Transaktionen vor einem Fehler brauchen ein UNDO, falls einige Ergebnisse schon auf der Platte sind

## 5 OODBMS und ORDBMS

- Grenzen des relationalen Modells: primitive Typen, keine ADTs, nur atomare Attribute, keine algorithmische Vollständigkeit
- Segmentierungsproblem: ein logisches Objekt wird auf viele Relationen verteilt → teure Join(t)s, viel Redundanz  
Redundanz, komplexe Abfragen, keine Operation auf den Objekten (zB rotate)
- Objekte bestehen aus der Sicht des Programmierers aus zwei Dimensionen: Struktur und Verhalten
- Multimenge: MULTISSET
- Typen: CREATE TYPE xy AS (attribut1 VARCHAR(20), attribut2 INTEGER) METHOD methode1 (var1 INTEGER);
- Methoden: CREATE METHOD methode1 (var1 INT) FOR xy BEGIN ... END (DB-abhängig)
- Vererbung: CREATE TYPE xz UNDER XY (attribut3 INTEGER); (auch mit Tabellen möglich: TYPE → TABLE)
- Objekt-ID vom System erzeugt (SYSTEM GENERATED) oder vom User angegeben (DERIVED)
- Attribute als Referenzen auf andere Tabellen: ab REF (abType) SCOPE (abTable)
- **RDF?**
- XPath: Abfragesprache, um Teile eines XML-Dokuments zu adressieren
- XQuery: FLWR als zentrale Rolle, beinhaltet XPath
- XSLT: Extensible Stylesheet Language for Transformation
- DTD: **fehlt**
- XML Schema: **fehlt**