Zusammenfassung von Kemper/Eickler, Datenbanksysteme, 5. Auflage, Kapitel 4-14

Geschrieben von Christian Pöcher

Kap 4: Relationale Anfragesprachen

* Anfragesprachen, wie SQL, sind deklarativ.
* Schema definieren: CREATE TABLE ...
* Schema ändern: ALTER TABLE ... [ADD |ALTER COLUMN] ...
* Datensatz einfügen: INSERT INTO ... VALUES ...
* Anfrage: SELECT DISTINCT ... FROM ... WHERE ... ORDER BY ... [DESC|ASC]
* Aggregatfunktionen: AVG, MIN, MAX, SUM, COUNT
* HAVING: Geht nur bei GROUP BY, selektiert ganze Gruppen.
* Mengenoperatoren: UNION, INTERSECT, EXCEPT, IN, NOT IN
* Datensatz ändern: UPDATE ... SET ...
* Datensatz löschen: DELETE FROM ... WHERE ...
* Änderungen werden erst in tempoäre Tabelle oder in
* SQL hat keinen Allquantor, Existenzquantor durch EXISTS
* SQL hat relationales Tupelkalkül als Grundlage
* null bedeutet "korrekter Wert nicht bekannt"
* dreiwertige Logik: true, false, unknown
* vier Arten Joins: cross join, natural join, (inner) join, left/right/full outer join
* SQL nicht Touring-vollständig: Keine Rekursion möglich! Keine transitive Hülle
* Sichten: CREATE VIEW ... AS ...
* Zweck von Sichten: a) Daten für bestimmte Benutzer (un-)zugänglich machen, b) vereinfachte Schreibweise für Anfragen
* Sichten sind update-fähig, wenn a) keine Aggregatfunktionen, DISTINCT, HAVING benutzt werden, b) Schlüssel der Basisrelation enthalten ist und c) nur eine Tabelle verwendet wird.
* Generalisierung kann erreicht werden, wenn Obertyp oder Untertyp durch Views definiert werden.
* Einbettung in Wirtssprachen durch Strings (JDBC) oder Metasprache und Präprozessor (SQLJ)

Zusammenfassung Kapitel 5: Datenintegrität

* Syntaktische Integritätsbedingungen schon durch Tabellendefinition bestimmt, jetzt semantische Integritätsbedingungen
* vier Arten: a) referentielle I., b) statische I. durch CHECK, c) Contraints, d) Trigger
* referentielle Integrität: Fremdschlüssel null oder Fremdschlüssel = Primärschlüssel eines Datensatzes der referenzierten Tabelle
* statische Integrität: CHECK (Note BETWEEN 0.7 AND 5.0)
* CHECK auch ok, wenn Antwort unknown
* Constraints: In Tabellendefinition CONSTRAINT Name CHECK (EXISTS (SELECT ...))
* Trigger: 4 Teile, a) Name, b) Auslöser (BEFORE UPDATE ON ... FOR EACH ROW), c) einschränkende Bedingung (WHEN), d) Prozeduraler Code

Zusammenfassung Kapitel 6: Relationale Entwurfstheorie

* jetzt: Konzeptuelle Feinabstimmung, Güte des Schemas verbessern
* Functional Dependency (FD): alpha --> beta gilt, wenn Werte von beta durch Werte von alpha bestimmt werden.
* FDs stellen eine semantische Konsistenzbedingung dar, d.h. sie müssen durch das DBMS erzwungen werden.
* alpha ist Superschlüssel, wenn alpha --> R, d.h. die Werte von alpha bestimmen alle anderen Attributwerte

* alpha ist Kandidatenschlüssel, wenn alpha -*-> R
* beta ist voll funktional von alpha abhängig (alpha -*-> beta), wenn alpha --> beta und alpha nicht mehr verkleinert werden kann.
* F+ Hülle der FDs, Herleitung durch Armstrong Axiome:
  a) Falls beta Teilmenge von alpha, gilt alpha --> beta
  b) Verstärkung: alpha --> beta ==> alpha gamma --> beta gamma
  c) Transitivität
* Mengen von funktionalen Abhängigkeiten sind äquivalent, wenn ihre Hüllen gleich sind. Man schreibt F==G.
* F_c heisst kanonische Überdeckung zu F, wenn
  a) F+_c = F+
  b) Man kann weder alpha noch beta verkleinern, es gibt keine überflüssigen Attribute.
  c) Jede linke Seite in F_c ist eindeutig.
* Berechnung der kanonischen Überdeckung durch
  a) Linksreduktion (Kann man bei FD alpha --> beta ein A von alpha entfernen?)
  b) Rechtsreduktion (B bei beta entfernbar?)
  c) Entferne FDs der Form alpha --> {}
  d) Fasse FDs mit gleicher linken Seite durch Vereinigungsregel zusammen.
* Drei Anomalien:
  a) Update: Wenn Information redundant, dann kann bei Update eines vergessen werden. Zusätzlich höherer Speicher- und CPU-Bedarf
  b) Einfüge: Informationen zweier Entities werden vermischt. Auffüllen mit NULL nötig.
  c) Lösch: unabsichtliches Löschen von Daten wegen Vermischung
* Korrektheitskriterien für Schemazerlegungen:
  a) Verlustlosigkeit: nach Projektion und natürlichem Join muss wieder das gleiche rauskommen.
  b) Abhängigkeitserhaltung: alle FDs müssen auch nach Zerteilung gelten.
* 1NF: Alle Attribute haben atomare Wertebereiche
* 2NF: Jedes Nichtschlüssel-Attribut ist voll funktional abhängig von jedem Kandidatenschlüssel
* 3NF: Für jede FD alpha --> B gilt mind. eine der Bedingungen:
  a) B ist Element von alpha (FD ist trivial)
  b) B ist in einem Kandidatenschlüssel enthalten.
  c) alpha ist Superschlüssel der Relation.
* Berechnung der 3NF:
  1) Berechne kanonische Überdeckung.
  2) Kreiere für jede FD ein Relationsschema.
  3) Wähle Schlüssel.
  4) Eliminiere Schematas, die vollständig in anderem Schema enthalten sind.
* BCNF: Wie 3NF, aber ohne Bedingung b). Nicht abhängigkeitsbewahrend.
* multivalued dependencies (MVD): beta ist mehrwertig abhängig alpha (alpha -->-> beta) gdw. man kann bei geichem alpha-Wert die beta-Werte vertauschen und das resultierende Tupel ist weiter in der Relation.
* Zerlegungen von Relationen mit MVDs: R kann in R1, R2 zerlegt werden, wenn a) R = R1 vereinigt mit R2 und b) R1 geschnitten R2 -->-> R1 oder R1 geschnitten R2 -->-> R2
* alpha -->-> beta ist trivial, wenn beta Teilmenge von alpha oder beta = R - alpha
* 4NF: Für jede MVD gilt: a) MVD ist trivial oder b) alpha ist Superschlüssel für R. Nicht abhängigkeitsbewahrend.

Zusammenfassung Kap 7 Indexstrukturen

* ISAM: Wie Daumenindex. Suchen in log(n), Insert und Delete kann Index komplett nach rechts verschieben. Bild siehe S.206
* B-Baum vom Grad k: Jeder Knoten ausser der Wurzel hat zwischen k und 2k Einträge. Blätter alle gleich weit von Wurzel entfernt. Zugriffe sehr stark logarithmisch.
* B+-Baum: "hohler Baum", Daten nur in Blättern.
* Hashing: Index durch Hashfunktion, z.B. h(x) = x mod p. Problem: Überlaufende Seiten führen zu Ineffizienz.
* Erweiterbares Hashing: h(x) wird binär dargestellt. Prefix als Verzeichniseintrag. Reicht eine Seite nicht mehr aus, wird weitere Stelle zu Prefix hinzugenommen. => doppelte Anzahl

Seiten.
* R-Baum: Boxen um n-dimensional Daten bauen.
* Range Queries können von Hashes nicht gut bearbeitet werden. Bei Exact Match Queries sind sie aber Bäumen überlegen.

Zusammenfassung Kap 8 Anfragebearbeitung

* Ziel: Anfrageoptimierung, dazu logische und physische Optimierung
* Regeln zur logischen Optimierung schreiben Anfrage in RA um, z.B. kann Selektion und Kreuzprodukt zu einem Join zusammengefasst werden
* exakte Optimierung so schwer wie das originale Problem, aber Optimierungsheuristiken möglich:
  - Aufbrechen von Selektionen
  - Verschieben der Selektionen nach unten im Baum
  - Zusammenfassen von Selektionen und Kreuzprodukten zu Joins
  - Reihenfolge der Joins, so dass kleine Zwischenergebnisse
  - Einfügen von Projektionen
  - Verschieben der Projektionen nach unten im Baum
* Implementierung von Joins:
  - nested loops
  - seitenorientiertes nested loops
  - Merge Join (zwei Pointer wandern runter)
  - Hash Join
* Durch Kostenmodelle werden Implementierungen ausgewählt und daraus Auswertungspläne erstellt.
* Statistiken werden durch ANALYSE gesammelt
* den Auswertungsplan kriegt man mit EXPLAIN

Zusammenfassung Kap9 Transaktionsverwaltung

* Transaktion: Arbeitseinheit in einer Anwendung
* ACID:
  - Atomicity: Transaktion als kleinste, nicht teilbare Einheit, "ganz oder gar nicht"
  - Constistency: Nach Beendigung von Transaktion ist Datenbasis konsitent
  - Isolation: nebenläufige Transaktionen beeinflussen sich nicht
  - Durability: Wirkungen abgeschlossener Transaktionen bleiben dauerhaft bestehen
* Recovery: kümmert sich um Atomicity und Durability
* Mehrbenutzersynchronisation: kümmert sich um Isolation

Kapitel 10: Fehlerbehandlung
==========

* lokales Undo: treten im normalen Betrieb auf, z.B. durch Fehler im Anwendungsprogramm oder durch ABORT.
* globales Undo: Nach Hauptspeicherverlust nicht abgeschlossene Transaktionen rückgängig machen.
* globales Redo: Nach Hauptspeicherverlust abgeschlossene Transaktionen auf Festplatte schreiben.
* steal/not steal: Pufferspeicher, der durch aktive Transaktionen benötigt ist kann (nicht) ersetzt werden. Macht globales Undo nötig.
* force/not force: Änderungen einer Transaktion werden bei COMMIT (nicht) auf die Festplatte geschrieben. Macht globales Redo nötig.
* Einbringstrategie:
  - update-in-place: Seite auf Festplatte wird überschrieben.
  - Twin-Block: zwei Seiten und Markierung welche Seite aktuell ist.
* Log:
  - LSN: Sequenz Number

  - TransaktionsID
  - PageID
  - PrevLSN: vorheriger Log aus gleicher Transaktion.
* Write Ahead Logging (WAL): Erst Logs schreiben, dann Daten auf Festplatte schreiben.
* Wiederanlauf:
  - Analyse des Logs, Bestimme Winner und Looser
  - Redo von Winnern und Loosern
  - Undo von Loosern

Kapitel 11: Mehrbenutzersynchronisation

* Interleaving von Transaktions kann Wartezeiten überbrücken.
* Lost Update: T1 überschreibt Änderungen von T2.
* Dirty Read: T1 liest Daten von T2 bevor T2 zurückgesetzt wird.
* Phantom Read: T1 erstellt Daten, die von T2 nicht mehr berücksichtigt werden.
* Schedule: Ablauf einer verzahnten Ausführung mehrerer Transaktionen. Bestimme partielle Ordnung für Konfliktoperatioenen.
* Schedules serialisieren: Vertauschen von Operationen, bis Transaktionen nacheinander laufen.
* serialisierbare Schedules: Ein Schedule ist serialisierbar, wenn es äquivalent zu einer seriellen Historie ist.
* Serialisierbarkeitstheorem: Schedule ist serialisierbar, gdw. wenn der Serialisierungsgraph azyklisch ist.
* Rücksetzbare Schedules: Ermöglichen ABORT ohne andere Transaktionen in Mitleidenschaft zu ziehen. Problematisch ist, wenn T1 schreibt und T2 das danach liest.
* Schedules ohne kaskadierendes Rücksetzen: Änderungen von T1 werden erst nach COMMIT freigegeben.
* Scheduleklassen: S. 308
* Locks: Shared/Read, eXclusive/Write
* Two Phase Locking:
  - Jedes Objekt muss vor Benutzung gesperrt werden
  - Keine Transaktion fordert Sperren zweimal an
  - Lock nicht gekriegt -> Warteschlange
  - Erst Wachstumsphase, dann Schrumpfungsphase
* strenges 2PL: Alle Locks werden erst beim Ende der Transaktion freigegeben.
* 2PL garantiert Serialisierbarkeit, strenges 2PL schützt vor kaskadierenden Rollback
* Deadlock-Erkennung:
  - Time-out
  - Wartegraph (Zykel = Deadlock)
  - Preclaim (Alle Locks am Anfang nehmen)
  - Zeitstempel (wound-wait, wait-die)
* Wound-Wait: T1 fordert Lock an, das T2 hat. Wenn T1 älter als T2, verwunde T2. Sonst wartet T1.
* Wait-Die: Wenn T1 älter als T2, wartet T1. Sonst stirbt T1.
* Isolation-Levels:
  - read uncommitted (liest auch uncommitted values)
  - read committed (Problem: non repeatable read)
  - repeatable read
  - serializable

Kapitel 12: Sicherheitsaspekte

* drei Arten:
  - Identifikation
  - Zugriffskontrolle
  - Auditing (Logs angucken)
* Discretionary Access Control (DAC): Quintupel (Objekte, Benutzer, Rechte, Filterprädikat, Weitergabeflag)

* Mandatory Access Control (MAC): Jeder User hat ein Clearance-Level. Objekte haben ein Classification-Level.
* Multilevel Datenbanken:
  - Jedes Attribut und das ganze Tupel zusätzlich haben Classification.
  - Polyinstatierung: Tupel darf mit mehreren Sicherheitseinstufungen vorkommen.

Zusammenfassung Kapitel 13: OODB

* Schwachpunkte relationaler DBs:
  - Anwendungsobjekt wird auf unterschiedliche Relationen segmentiert.
  - Künstliche Schlüsselattribute
  - Kein Verhalten
  - Impedance Mismatch: Mengenorientiert vs. Satzorientiert
* Vorteile OO-Modellierung:
  - Information Hiding: Operationen ausführbar ohne Kenntnis der Repräsentation der Daten
  - Transformationen entfallen (Mengenorientiert vs. Satzorientiert)
  - Objektidentität statt künstlicher Schlüssel
* Drei Bestandteile eines Objektes:
  - Identität (OID)
  - Typ
  - Wert
* Drei Bestandteile eines Objekttyps:
  - Strukturbeschreibung (Attribute)
  - Verhaltensbeschreibung (Operationen)
  - Generalisierungs-/Spezialisierungsbeziehungen
* Nachteile von "identity through contents":
  - Objekte mit gleichem Wert müssen nicht identisch sein.
  - künstliche Schlüssel müssen vom Benutzer gewartet werden, obwohl keine Anwendungssemantik.
  - Schlüssel dürfen nicht verändert werden
* OIDs = Objekt-Referenzen
* Beziehungen in ODMG über RELATIONSHIP und INVERSE

```
CLASS Professoren {
        RELATIONSHIP Räume residiertIn INVERSE Räume::beherrbergt;
};
```

* Soll eine Seite N-wertig sein definiert man

```
RELATIONSHIP set <Vorlesungen> liest inverse Vorlesungen::gelesenVon
```

* INVERSE garantiert Symmetrie
* Extent ist die Menge aller Instanzen eines Objekts
* Vererbung möglich wo Substituierbarkeit gegeben ist:
  Eine Untertyp-Instanz ist überall dort einsetzbar, wo eine Obertyp-Instanz gefordert ist.
* dynamisches Binden: erst zur Laufzeit wird bestimmt welche Implementierung einer Operation gewählt wird.

Zusammenfassung Kapitel 14: Objektrelationale DBs

* Erweiterungen:
  - Large Objects
    # CLOBs: Character Large Objects
    # BLOBs: Binary Large Objects
  - Mengenwertige Attribute
  - Geschachtelte Relationen
  - User Defined Types (UDTs)
    # wertbasierter Typ
    # row Types
  - OID/Referenzen/Pfadausdrücke
  - Vererbung

- Operationen
* Locator: Pointer auf LOBs, Operationen werden nur logisch ausgeführt
* Vergleich bei UDTs nur mit gleichen Typen möglich
* Vererbung durch CREATE TYPE ProfessorenTyp UNDER AngestelltenTyp ...