DATENBANKSYSTEME

Schriftlich, 100 Minuten, handgeschriebenes A4 Blatt + Taschenrechner

Bsp. Tests auf LVA-HP

Vorlesung Folien + Übungs Bsp

SQL + PLSQL + Theorie + Rechenbsp.

SQL: Create, Alter, Sequence, Select, With

PLSQL: Function, Cursor, Trigger

Architektur DBMS: Disk Space Manager, Buffer Manager, Files and Access Methods
Layer, RAID, Index Verfahren: B-Bäume B+-Bäume
Hashing, R-Bäume, erweitertes Hashing

Ballung: Index enthält Daten oder Verweise

Transaktionen: Transaction Manager, Lock Manager, Recovery Manager
BOT, commit, abort, rollback
ACID

Fehlerbehandlung: R1, R2, R3 und R4 Recovery
Ersetzungsstrategien: steal, ¬steal; Zurückschreiben: force, ¬force
⇒ Standard: steal, ¬force
Einbringung: in place, Twin Block, Schattenspeicher
Log: Physische/Logische Protokollierung, WAL, Ringpuffer, LSN

Recovery: Analyse, Redo, Undo Loser (CLR) (R2, R3)
Sicherungspunkte: fuzzy; transaktionskonsistent
aktionskonsistent
unscharf (fuzzy)

Mehrbenutzersynchronisation: 4 mögliche Fehler, Serialisierbarkeit, Historie
Serialisierbarkeitsgraph, Rücksetzbarkeit, Strikte Historien

Sperrprotokolle: Sperrbasierte Synchron: Zwei Phasen Sperren.
Strenges Zwei Phas. Sperren.: Strikt
Konservatives 2PL (Predeclaring)

Deadlock: Erkennung: Time-out, Wartegraph
Vermeidung: Predeclaring, Zeitstempelverfahren: wound-wait
wait-die

Multiple Granularity Locking: Lock Escalation
Top-Down für benötigte Sperren
Bottom-up Freigabe
Insert / Delete Sperren, Phantomproblem

Synchronisationsmethoden: Zeitstempel-basierende Synchronisation
Optimistische Synchronisation
Synchronisation von Sonderstrukturen

JDBC: Connection
Statement: Statisch, Prepared, Callable (executeQuery())
ResultSet: ReadOnly vs Scrollable / Updatable (opt. Int())

Relationale Algebra: σ_p, Π_s, $E_1 \times E_2$, $\rho_v(E_1)$, $\rho_{A \to B}(E)$
$E_1 \cup E_2$, $E_1 \cap E_2$, $E_1 - E_2$
$E_1 \bowtie E_2$, $E_1 \ltimes E_2$, $E_1 \rtimes E_2$, $E_1 \bowtie E_2$

Anfragebearbeitung: Parser, Operator Evaluator, Optimizer, Plan Executor

Parser übersetzt Anfrage in relationale Algebra
Anfrage Optimierer erstellt Auswertungsplan: Logische Opt., Physische Opt.
Plan Executor, Operator Evaluator: Abarbeitung Auswertungsplan

Logische Optimierung:
- Selektionen aufspalten
- Projektionen kaskadieren
- Vertauschen von σ und Π.
-> Selektionen nach unten propagieren
-> Duplikaten vermeiden (kleinere Ergebnisse)
-> Joins erstellen ($\times$ und σ)
-> Projektionen nach unten Propagieren
-> Operationen zusammenfassen

Physische Optimierung: Iteratoren (Schnittstelle für Operator Auswertung)
Pipelining / Materialisierung
Sortieren Hashing

- Sortieren: Two-Way-Merge Sort: Pass 0, Runs
Multi-Way-Merge Sort

- Joins: - Nested Loop Join
- Page oriented + Block Nested Loop Join
- Index Nested Loop Join
- Sort Merge Join
- Hash Join
- Hybrid Hash Join

- Selektion, Projektion, Duplikatelimination, Mengenoperationen
Gruppierung + Aggregat-Funktionen

Kostenmodelle und Tuning: Kostenbasierte Optimierung
- Selektivität
→ Kostenabschätzung: n, b_R, b_S, M_R, M_S, p_R, p_S

n: Satzanzahlen in DB Puffer
b_R, b_S: Seiten für R
M_R, M_S: Anzahl Tupel von R
p_R, p_S: Anzahl Tupel pro Seite

Objektrelationale Datenbanken:
Große Objekte (LOB), User defined Types, Mengenwertige Attribute, Geschachtelte Attribute, Vererbung, Operationen
/ Objekt ID Primärschlüssel

Verteilte Datenbanken: Fragmentierung: horizontal / vertikal
Allokation: mit / ohne Replikation
Transparenz: Fragmentierungstr.
Allokationstr.
Lokale Schema Tr

→ Rekonstruierbarkeit, Vollständigkeit, Disjunktheit

Transaktionskontrolle: Koordinator, Agenten
K: Prepare, A: Ready / Failed, K: Commit / Abort, A: ACK

Fehler: Absturz Koordinator, Absturz Agent, Verlorene Nachricht

Mehrbenutzersynchronisation: Serialisierbarkeit
Strenges 2 Phasen Sperrprotokoll
→ Lokale / Globale Sperrenverwaltung
Deadlock
→ Timeout, Zentralisierte Erkennung
Dezentralisierte Erkennung

→ Vermeidung: Zeitstempelverfahren
Optimistische Synchron

Replizierte Daten: write all - read one / Qr / Qw
→ Quorum - Consensus Verfahren
- Jede Kopie hat Gewicht: Lesequorum, Schreibquorum
→ $Q_W(A) + Q_W(A) > W(A)$
→ $Q_W(A) + Q_R(A) > W(A)$
→ majority consensus Grundgewicht

```
                                        id DEFAULT nextval('seq_test')
CREATE TABLE test (
    id INTEGER PRIMARY KEY,
    zahl INTEGER CHECK (zahl IN (1,2,3)),
    fk INTEGER,
)

ALTER TABLE test
    ADD FOREIGN KEY (fk) REFERENCES other (id)
        DEFERRABLE INITIALLY DEFERRED;

CREATE SEQUENCE seq_test MINVALUE 1 MAXVALUE 100 NO CYCLE
        INCREMENT BY 2

WITH RECURSIVE categories (parent, child) AS(
    SELECT p AS parent, c AS child
      FROM cat
      WHERE p = 'main'
    UNION ALL
      SELECT c.p AS parent, c.c AS child
        FROM cat c, categories ca
        WHERE c.p = ca.child
)
SELECT * FROM categories
```

```
CREATE OR REPLACE FUNCTION suche (id_i INTEGER)
   RETURNS INTEGER AS $$

  DECLARE
    price_i INTEGER;      price_i tbl.price%TYPE
  BEGIN
    SELECT price INTO price_i FROM tbl WHERE id = id_i
    IF (price_i IS NULL) THEN
      RAISE EXCEPTION 'not found';
    ELSE
      RETURN price
    END IF;
  END

$$ LANGUAGE plpgsql;

IF cond THEN
ELSIF cond THEN
ELE THEN
END IF

PERFORM
IF FOUND

CREATE OR REPLACE FUNCTION t_check() RETURNS trigger AS $$
...

CREATE TRIGGER trig_check BEFORE UPDATE OR INSERT
    ON table FOR EACH ROW EXECUTE PROCEDURE t_check();
```

```
Class.forName("org.postgresql.Driver");
Connection c = DriverManager.getConnection("connectionStr", "user", "pass");

Statement s = c.createStatement;
ResultSet rs = s.executeQuery("SELECT * FROM test");

while (rs.next()){
    int id = rs.getInt("id");
}

rs.close();
s.close();
c.close();

PreparedStatement ps = c.prepareStatement("Select count(*) FROM test
                                           WHERE id = ?");

ps.setInt(1, 10);
ResultSet rs = ps.executeQuery();

c.createStatement(ResultSet.CONCUR_UPDATABLE, ResultSet.TYPE_SCROLL_INSENSITIVE);
rs.updateInt();
rs.updateRow();
rs.deleteRow();
```

$\sigma_P(E)$, $\Pi_S(E)$, $E_1 \times E_2$, $\rho_N(E)$, $\rho_{A \leftarrow B}(E)$, $E_1 \cup E_2$, $E_1 - E_2$

$E_1 \cap E_2$, $E_1 \bowtie E_2$, $E_1 \ltimes E_2$, $E_1 \rtimes E_2$

	NL	S	X	IS	IX
S	✓	✓	-	✓	-
X	✓	-	-	-	-
IS	✓	✓	-	✓	✓
IX	✓	-	-	✓	✓

top-down: zuerst IX sperren

bottom up: zuerst S/X freigeben

m ... Anzahl Seitenrahmen im Puffer
b_R, b_S ... Anzahl der Seiten von R, S
M_R, M_S ... Anzahl Tupel von R, S
p_R, p_S ... Anzahl Tupel pro Seite von R, S

Selektion $\sigma_P(R)$: bei ohne Index, 4 mit B Index, 1 mit Hash Index
+1 bei Referenzen im Index

kein Schlüssel: ungeballter Index $M_R \cdot sel_{A=p}$
geballter Index $b_R \cdot sel_{A=p}$

Sortieren: $2b_R \cdot (1 + \lceil \log_{m-1}(\lceil b_R/m \rceil) \rceil)$
m ... Länge der Level 0 Runs

Nested Loop Join: $b_R + M_R \cdot b_S$ R ... kleinere Relation

Pagewise Nested Loop Join: $b_R + b_R \cdot b_S$ ideal

Block Nested Loop Join: $b_R + k + \lceil b_R/(m-k-1) \rceil \cdot (b_S - k)$ k ... $= 1$
für 1 in R nur 1 in S: $b_R + M_R \cdot c$

Index Nested Loop Join: geballt: $b_R + M_R \cdot (c + b_S \cdot sel_{RS})$ $c \in [1,5]$ / je nach Index
ungeballt: $b_R + M_R \cdot (c + M_S \cdot sel_{RS})$

Sort Merge Join: R sortieren + S sortieren + $b_R + b_S$ (Join on Schlüssel)

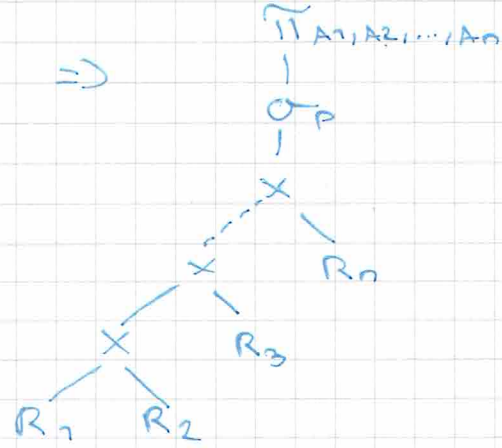
Hash Join: $3 \cdot (b_R + b_S)$

Hybrid Hash Join: $(3 - 2k/n) \cdot (b_R + b_S)$ k ... Anzahl Buckets im Puffer
$(b_R + b_S)$ ← eine Relation passt in Puffer

Kanonische Übersetzung:

```
SELECT A1, A2, ..., An
FROM   R1, R2, ..., Rn
WHERE  P
```

=>



```
CREATE SEQUENCE seq_tbl MINVALUE 10
    MAXVALUE 1000 INCREMENT BY 10 NO CYCLE;

CREATE TABLE tbl (
  id1 INTEGER PRIMARY KEY DEFAULT nextval('seq_tbl'),

  fk INTEGER REFERENCES other(id);

  val VARCHAR(5) CHECK (val IN ('ab','cd'))
);

ALTER TABLE name
    ADD FOREIGN KEY (fk) REFERENCES
      tbl(id) DEFERRABLE INITIALLY DEFERRED;

WITH RECURSIVE call(par,chi) AS (
  SELECT p AS par, c AS chi
    FROM calls
    WHERE p='main'
  UNION ALL
  SELECT c.p AS par, c.c as chi
    FROM call t, calls c
    WHERE c.p = t.chi
)
```

```
CREATE OR REPLACE FUNCTION f_t() RETURNS
    trigger AS $$
DECLARE
  tbl tbl.ROWTYPE
BEGIN
  IF EXISTS (SELECT....)
    THEN
      RAISE EXCEPTION 'Hotel';
  ELSE
      RAISE NOTICE 'yo';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER t BEFORE INSERT OR
  UPDATE ON tbl FOREACH ROW
    EXECUTE PROCEDURE f_t();
```

```
Class.forName("org.postgresql.Driver");
Connection c = DriverManager.getConnection(
    "connect str","user","pass");

Statement s = c.createStatement();
PreparedStatement ps = c.prepareStatement(
    "SELECT * FROM tbl where id = ?");

ResultSet rs_S = s.executeQuery("SELECT
    id from other");

while (rs_S.next()){
  ps.setInt(1, rs_S.getInt("id"));
  ps.executeUpdate();
}

c.createStatement(ResultSet.CONCUR_UPDATABLE,
    ResultSet.TYPE_SCROLL_INSENSITIVE);

rs.updateInt();
rs.updateRow();
rs.deleteRow();

rs_S.close();
ps.close();
s.close();
c.close();
```

	NL	S	X	IS	IX
S	✓	✓	–	✓	–
X	✓	–	–	–	–
IS	✓	✓	–	✓	✓
IX	✓	–	–	✓	✓

m ... Seitenanzahl im Puffer
b_R, b_S ... Seiten von R, S
M_R, M_S ... Tupel von R, S
p_R, p_S ... Tupel pro Seite von R, S

Selektion:
ohne Index: b_R
B-Index: 4 } +1 bei Rückgriffen
Hash-Index: 1

unsortierter Index: $M_R \cdot sel_{A=p}$
sortierter Index: $b_R \cdot sel_{A=p}$

Sortieren:
$2 \cdot b_R \cdot (1 + \lceil \log_{m-1}(\lceil b_R/m \rceil) \rceil)$
Länge durch 0 Runs

Naiver Loop Join
$b_R + M_R \cdot b_S$

Primitiver NL Join (optimal)
$b_R + b_R \cdot b_S$

Block NL Join: k=1, R ... kleinere Rel.
$b_R + k + \lceil b_R/(m-k-1) \rceil \cdot (b_S - k)$

Index NL Join
für T in R nur 1 in S: $b_R + M_R \cdot c$
sortiert: $b_R + M_R \cdot (c + b_S \cdot sel_{RS})$
unsortiert: $b_R + M_R \cdot (c + M_S \cdot sel_{RS})$
$c \in [1,5]$ ... x nach Seite

Sort Merge Join
R sortieren + S sortieren + $(b_R + b_S)$

Hash Join
$3 \cdot (b_R + b_S)$

Hybrid Hash Join
eine Relation passt in Puffer: $(b_R + b_S)$
sonst: $(3 - 2k/n) \cdot (b_R + b_S)$
Bruchteil von S im Puffer
Anzahl Buckets von R und S

Niklas Natter