Relationale Algebra

Selektion

$\sigma_{A='abc'}(S)$

Projektion

$\pi_{\text{Attribut1, Attribut2, ...}}(R)$

Bei Bags werden keine Duplikate entfernt

Umbenennung

$\rho_{S(A',B')}(R(A,B))$

Kreuzprodukt $R \times S$

Natürlicher Verbund (Join) $R \bowtie S$

Wenn keine übereinstimmende Spalten Crossjoin

Theta-Join $R \bowtie_{A<B} S$

Nur bei diesem Join Tabellennamen behalten (z.B. R.A und S.A) Logik operatoren: oder = $\vee$, und = $\wedge$

Vereinigung $R \cup S$

Bei Bags nimmt man die grössere Anzahl

Bag concatenation $R \sqcup S$

Man nimmt die Summe

Durchschnitt $R \cap S$

Bei Bags nimmt man die kleinere Anzahl

Differenz $R \setminus S$

Bei Bags nimmt man die Differenz

Duplikat-Elimination $\delta(R)$

Outer-Join

- Full $⟗(R,S)$
- Left $⟕(R,S)$ (behält alle linken Einträge)
- Right $⟖(R,S)$

Erweiterte Projektion $\pi_{3 \cdot A \to X, B}(R)$

Entity Relationship-Diagramme

Keys

- Unterstrichen → PK
- Umkreist → K
- Mehrere Umkreist → Zusammengesetzter K
- Mehrere Umkreist &/ Unterstrichen → Zusammengesetzter PK

Beziehungen

- A mit PK X
- B mit PK Y
- 1 zu 1 → (X), (Y)
- 1 zu m → (Y)
- m zu m → (X, Y)
- ID: Hat selbst noch anderen PK. PK + Referenz = weiterer K (komplexe attribute)
- ISA: Andere Tabelle aber gleicher PK (spezialfall)
- zusammengesetzte Beziehung: weitere beziehung darf nur existieren, wenn diese existiert. zb hatgeliefert->liefert

SQL

DDL

Erzeugen einer Datenbank

```
CREATE SCHEMA <dbname> [AUTHORIZATION <userName>];
DROP SCHEMA <dbName> [CASCADE];
```

Domain

```
CREATE DOMAIN <domainName> AS dataType<domain)] [<attributeConstraintDef>];
CREATE DOMAIN pk_type AS char(9);
DROP DOMAIN <domainName>;
```

Create table

```
CREATE TABLE <tablename> (
    <name> <type> [NOT NULL] [PRIMARY KEY] [UNIQUE],
    [CONSTRAINT <name>] PRIMARY KEY (<fields>),
    [CONSTRAINT <name>] UNIQUE (<fields>),
    [CONSTRAINT <name>] FOREIGN KEY (<fields>) REFERENCES <table>(<fields>),
    [CONSTRAINT <name>] CHECK (x > 0),
    [CONSTRAINT <name>] DEFAULT SYSDATETIME();
);
CREATE TABLE Besucher2 (LIKE Besucher)
ALTER TABLE A ALTER <field> DROP <constraint>;
DROP TABLE <tabelName> [CASCADE | RESTRICT];
```

Foreign Key-Trigger

```
CREATE TABLE <tablename> (
    ...
    FOREIGN KEY (...) ON UPDATE/DELETE NO ACTION / SET NULL / SET DEFAULT / CASCADE
)
```

Alter Table

```
ALTER TABLE <tableName> <action> ;
<action> ::= "ADD" <column> | "DROP" <column> | "ALTER" <column> | "ADD" <constraint> | "DROP"
<constraint>
```

DML

Insert

```
INSERT INTO <tableName> [(<attributes>)] VALUES (<values>)
```

```
INSERT INTO Besucher2 (SELECT * FROM Besucher WHERE name LIKE '%a%');
```

Update

```
UPDATE <tabelName> SET a = b, c = d WHERE ...;
```

Delete

```
DELETE FROM <tabelName> WHERE ...;
```

Queries

```
SELECT ... FROM (SELECT ... FROM ...) AS x
SELECT ... WHERE EXISTS (SELECT ...)
SELECT ... WHERE x IN (1, 2, 3)
SELECT ... WHERE y [NOT] IN (SELECT ...)
SELECT ... WHERE frequenz > [ALL | ANY] (SELECT frequenz FROM ...)
SELECT ... FROM A UNION [ALL | DISTINCT] SELECT ... FROM B
```

- Die Bag Concatenation ⊔ entspricht UNION ALL
- Der Durchschnitt ∩ entspricht INTERSECT ALL
- Die Differenz \ entspricht EXCEPT ALL
- Default ist DISTINCT

Join

```
SELECT ... FROM A, B WHERE ...
SELECT ... FROM A JOIN B ON a.x = b.y
SELECT ... FROM A CROSS JOIN B
SELECT ... FROM A [NATURAL] [LEFT | RIGHT | FULL] OUTER JOIN B
```

String

```
WHERE name BETWEEN 'C%' AND 'E%' ;
WHERE name LIKE '%a%';
WHERE name LIKE 'A____'
```

Order By

- default asc
- Lexikographische Ordnung: <a,b,c> < <x,y,z> → (a<x) OR (a=x AND b<y) OR (a=x AND b=y AND c<z)

```
SELECT *
FROM Besucher
WHERE (
    (Name = 'Meier' AND Vorname >= 'Hans')
    OR
    (Name > 'Meier' AND Name < 'Schmid')
    OR
    (Name='Schmid' AND Vorname <= 'Joseph')
);
```

Aggregationen

- COUNT. COUNT(attribut) → alle bei denen attribut nicht NULL ist
- MAX
- MIN

- SUM. sum(menge * preis) möglich
- AVG
- SUM und AVG ignorieren NULL, COUNT nicht

Group By: Es gibt eine Gruppe mit allen NULL-Werten

```
SELECT ... FROM A WHERE ... GROUP BY x HAVING SUM(y) > 10
```

Case

```
SELECT x, y, CASE
    WHEN <Bedingung1> THEN <Wert1>
    {WHEN <Bedingung2> THEN <Wert2>}
    [ELSE <Wert3>]
END
```

Reihenfolge der Ausführung

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

Views

```
CREATE VIEW A AS
SELECT ...
```

Common table expressions (CTEs)

```
WITH cte_name1 AS (
    SELECT ...
), cte_name2 AS (SELECT ... FROM cte_name1)
SELECT ...
```

Integrität

Durch DB sichergestellt:

- Bereichsintegrität
- Entitätsintegrität (PK muss eindeutig sein)
- Referentielle Integrität (FK muss existieren)

Durch Constraints sichergestellt:

- UNIQUE
- CHECK
- DEFAULT

PL/pgSQL

Funktion

```
{CREATE|ALTER|DROP} FUNCTION myFunction(x integer) RETURNS void AS
$body$
    DECLARE
```

```
        -- Deklarationsblock
        -- Der DECLARE Abschnitt ist optional
        KNr INTEGER; -- ist jetzt NULL
        ProdNr INTEGER := 0;
        UserID users.UserID%TYPE;
        Zeile users%ROWTYPE;
        name RECORD; -- zeile ohne typ
        ...
    BEGIN
        RAISE NOTICE 'Test';
         ...
    EXCEPTION
        -- Ausnahmeverarbeitung
        -- Der EXCEPTION Abschnitt ist optional
    END;
$body$
LANGUAGE plpgsql;

SELECT myFunction();
DROP FUNCTION myFunction();
```

Stored Procedure

```
CREATE PROCEDURE insert_data(a integer, b integer)
LANGUAGE SQL
AS $$
    INSERT INTO table_x VALUES(a);
    INSERT INTO table_x VALUES(b);
$$;

CALL insert_data(1,2);
```

Kontrollstrukturen

- `IF … [ELSE | ELSIF …]`
- `CASE … WHEN … ELSE … END`
- `LOOP … EXIT – WHILE … LOOP … END LOOP`
- `FOR … IN … LOOP … END LOOP`

Cursor

```
CREATE OR REPLACE FUNCTION zeige_alle_besucher_namen()
RETURNS VOID AS $$
    DECLARE
        rec_besucher record;
        b_namen CURSOR FOR SELECT Name,Vorname FROM Besucher;
    BEGIN
        OPEN b_namen;
        LOOP FETCH b_namen INTO rec_besucher;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE 'Name: % Vorname: % ',rec_besucher.Name, rec_besucher.Vorname;
        END LOOP;
        CLOSE b_namen;
    END;
$$
LANGUAGE plpgsql;
```

Trigger

```
-- Funktion für Trigger erzeugen
CREATE OR REPLACE FUNCTION myFunction()
```

```
RETURNS TRIGGER
LANGUAGE PLPGSQL AS $$
    BEGIN
        IF
            NEW.Strasse <> OLD.Strasse
        THEN
            INSERT INTO Adressaenderung(name, vorname, strasse_alt, strasse_neu, geaendert_am)
VALUES(OLD.Name,OLD.Vorname,OLD.Strasse,NEW.Strasse,now());
        END IF;
        RETURN NEW;
    END;
$$

{CREATE|ALTER|DROP} TRIGGER
{BEFORE|AFTER|INSTEAD OF} {UPDATE|DELETE}
ON <tabelle>
FOR EACH ROW
EXECUTE PROCEDURE myFunction();
```

Indexe

```
CREATE INDEX <field_idx> ON <table>(<field>);
EXPLAIN SELECT ...
```

- Implementierung kann variieren
  - B-Bäume (dünne oder dichte bäume)
  - Bitmap
  - Hashen
- UPDATE, INSERT, DELETE langsamer
- SELECT schneller
- Index hilft: `WHERE start_year = 2000`
- Index hilft nicht: `WHERE original_title LIKE '%Star%'`
- Indexiert werden sollen:
  1. Attribute, die oft abgefragt werden, sollten indiziert werden.
  2. Fremdschlüssel sollten indexiert werden, insbesondere dann, wenn über «Primär-Fremdschlüssel» gejoint wird (was häufig der Fall ist).
  3. Attribute über die oft gejoint wird; wenn über mehrere Attribute gejoint wird dann muss ein zusammengesetzter Index verwendet werden.
  4. Attribute mit niedriger Kardinalität (Extrembeispiele: Geschlecht, Ja/Nein- Flags u.ä.) sollten nicht indexiert werden (es gibt dafür spezielle Indexstrukturen, hier aber nicht behandelt).

Transaktionen

ACID

1. Atomarität (Atomicity) / Unit of Work: Zusammengehörige Folge von Lese- und Schreibzugriffen (in sich geschlossene „Arbeitseinheit“), muss als Ganzes entweder erfolgreich abgeschlossen (Commit) oder rückgängig gemacht werden können (Rollback).
2. Consistency / Konsistenz: Alle Operationen hinterlassen die Datenbank in einem korrekten Zustand.
3. Isolation / Nebenläufigkeit (Concurrency): „Gleichzeitiger“ Zugriff mehrerer Benutzer ermöglichen, so dass diese Transaktionen keinen unerwünschten Einfluss aufeinander haben (bei nur einer CPU auch möglich → Gleichzeitigkeit wird durch Zeitscheibenverfahren „simuliert“).
4. Dauerhaftigkeit (Durability) / Recovery: Automatische Behandlung von Ausnahmesituationen (Fehlern) und schneller (möglichst automatischer) Wiederanlauf nach schwerwiegenden Fehlern. Wiederherstellung (Rollforward) verlorener Daten und Rücksetzten (Rollback) fehlerhafter Daten

Probleme

1. Lost-Update: Überschreiben bereits getätigter Updates (nie tolerierbar)
2. Dirty-Read: Lesen von Veränderungen noch nicht abgeschlossener Transaktionen
3. Non-Repeatable-Read: Lesen von zwischenzeitlich von anderen Transaktionen durchgeführten Veränderungen
4. Phantom-Read: Lesen von zwischenzeitlichen Veränderungen, die von anderen Transaktionen durchgeführt worden sind

Lösungen

- Liberaler Scheduler lässt Konflikte zu und versucht sie aufzulösen, falls sie entstehen. Grenzfall: Keine Transaktion ist erfolgreich.
- Konservativer Scheduler nimmt Wartezeiten in Kauf zb. mit Sperrverfahren. Grenzfall: Keine Parallelisierung.

Fehlerbehandlung

- Logging
- Recovery