Datenbanksysteme 1 | Dbs1

Zusammenfassung

INHALTSVERZEICHNIS

1. UML

TODO: disjoint, overlapping, complete, incomplete, komposition

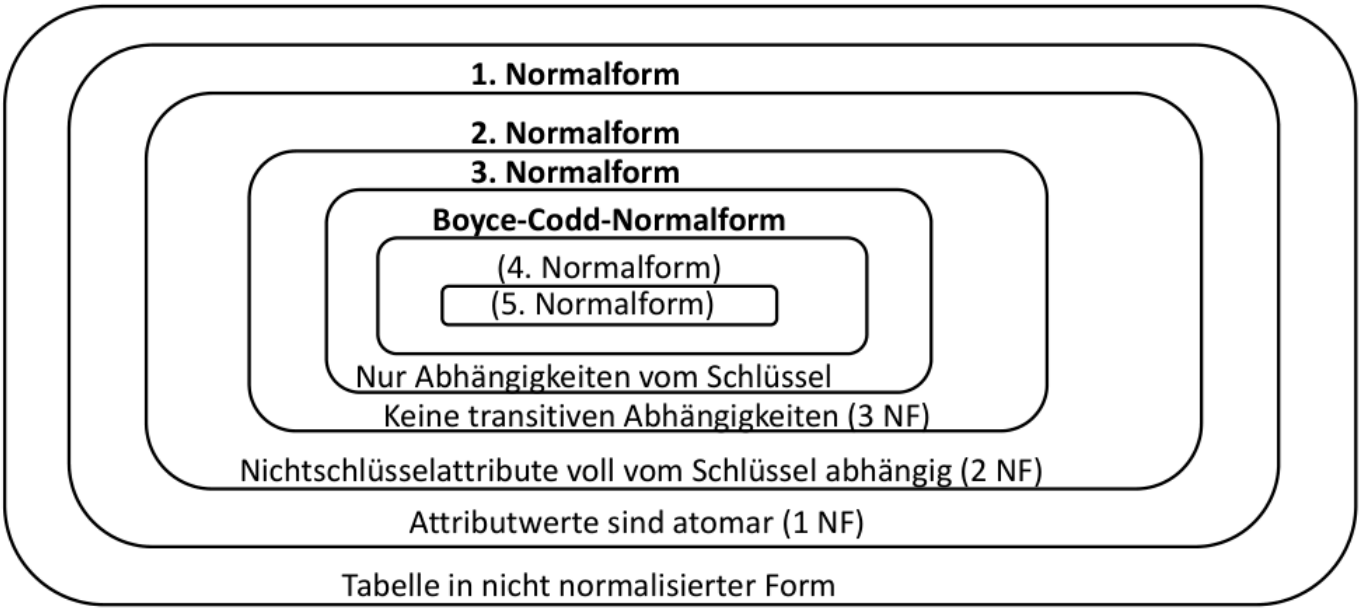
2. ANSI-MODELL

TODO: Folien zusammenfassungen am schluss

3. DATENBANK-ENTWURFSPROZESS

TODO: konzeptionell, ERD, UML Grundbegriff Glossar

4. NORMALFORMEN



TODO: Begriffe Glossar

funktionale abhängigkeit

- B ist voll funktional abhängig von A falls zu jedem wert A genau ein wert B existiert
  - $A \rightarrow B$

4.1. NF1

- attributwerte sind atomar

4.2. NF2

- nichtschlüsselatribute voll vom schlüssel abhängig
- Besteht der PK nur aus einem einzigen Attribut (ist er also nicht zusammengesetzt), so bereits automatisch die 2. NF gegeben.
- Eine Tabelle ist NICHT in 2. NF, wenn sie einen zusammengesetzten Schlüssel hat und mind. ein Nichtschlüsselattribut von nur einem Teil des Schlüssels funktional abhängig ist.

4.3. NF3

- keine transitiven abhängigkeiten

4.4. BCNF

– keine abhängigkeiten vom schlüssel

5. RELATIONALE ALGEBRA

projektion (π)

– $\pi_{(A1,A4)}$ = `SELECT A1,A4 FROM ausgangstabelle;`

selektion (σ)

– $\sigma \Rightarrow (A1 > 30)$ = `SELECT * FROM ausgangstabelle WHERE A1 > 30;`

– sigma steht für =, >, <, !=, <=, >=

kartesisches produkt (CROSS JOIN)

– (R1 x R2) = `SELECT * FROM R1,R2;`

verbund (join). voraussetung attribute r[a] und r[b] sind vereinigungsverträglich (typkompatibel)

– theta-join
  – $R[a\theta b]S$ = `SELECT R.* FROM R JOIN S ON R.R2 $theta$ S.S2;`
  – theta steht für $=, >, <, \neq, \leq, \geq$
– equi join
  – θ ist =
  – `SELECT * FROM R JOIN S ON R.R2 = S.S2;`
– natural join
  – equi join ohne doppelte spalten
  – `SELECT * FROM R NATURAL JOIN S;`
  – on R.R2 = S.S2 kann theoretisch ausgelassen werden, dabei werden implizit gleich heissende columns verglichen
– äussere joins
  – verbindet tupel auch ohne übereinstimmung
– semi-joins
  – «left outer join ohne join»

umbenennung (ρ)

– `SELECT * FROM angestellter AS ang1;`

6. (POSTGRE)SQL

6.1. DDL (DATA DEFINITION LANGUAGE)

Postgres Dokumentation

Definiert Befehle für das Kreieren, Löschen und Modifizieren von Tabellendefinitionen, von externen Sichten (Views), von Constraints und von einigen anderen Datenbankobjekten (Index, Cluster) für die Optimierung der Abbildung von der logischen auf die interne Ebene.

6.1.1. Wichtigste Befehle

Zusätzlich kann `IF (NOT) EXISTS` hinzugefügt werden oder der `CREATE` mit `CREATE OR REPLACE` ausgetauscht werden. Beim `DROP` Befehl kann `CASCADE` hinzugefügt werden.

```
CREATE SCHEMA s;    DROP SCHEMA s;
CREATE DOMAIN d;    DROP DOMAIN d;
CREATE TABLE t;     DROP TABLE t;     ALTER TABLE t;  TRUNCATE TABLE t;
CREATE VIEW v;      DROP VIEW v;
```

```
CREATE INDEX i;     DROP INDEX i;
CREATE DATABASE db; DROP DATABASE db;
```

6.1.2. Beispiel CREATE TABLE

```
CREATE TABLE schema_name.table_name (
  counter SERIAL NOT NULL,
  important INT NOT NULL,
  field INT NOT NULL UNIQUE,
  very_long_name VARCHAR(666) DEFAULT "Based PostgreSQL user",
  age INT CHECK (age > 18),
  reference INT NOT NULL,
  someday DATE DEFATUL SYSDATE,
  another_ref INT FOREIGN KEY REFERENCES other_table.id ON UPDATE RESTRICT,
  PRIMARY KEY (counter, important),
  FOREIGN KEY (reference) REFERENCES third_table.id ON DELETE CASCADE,
  CHECK (field BETWEEN 69 AND 420)
)
```

6.1.3. Datentypen

Postgres Dokumentation

Sinnvolle Konversionen und Rundungen werden implizit durchgeführt.

Begriff	*Bedeutung*
INTEGER/INT	Integer (4 bytes)
BIGINT	Large integer (8 bytes)
SMALLINT	Small integer (2 bytes)
REAL	Single precision floating-point number (4 bytes)
NUMERIC(precision,scale)	Exact numeric of selectable precision
DOUBLE PRECISION	Double precision floating-point number (8 bytes)
SERIAL	Auto-incrementing integer (4 bytes)
BIGSERIAL	Auto-incrementing large integer (8 bytes)
SMALLSERIAL	Auto-incrementing small integer (2 bytes)
CHARACTER/CHAR(size)	Fixed-length, blank-padded string
VARCHAR(size)	Variable-length, non-blank-padded string
TEXT	Variable-length character string
BOOLEAN	Logical Boolean (true/false)
DATE	Calendar date (year, month, day)
TIME	Time of day (no time zone)
TIMESTAMP	Date and time (no time zone)
TIMESTAMP WITH TIME ZONE	Date and time with time zone
INTERVAL	Time interval
JSON	JSON data
UUID	Universally unique identifier

6.1.4. Contraints

Postgres Dokumentation

Begriff	*Bedeutung*
PRIMARY KEY	Attribut ist Primärschlüssel und damit «UNIQUE» und «NOT NULL»
NOT NULL	Attributwerte müssen immer einen definierter Wert haben (default ist NULL)
UNIQUE	Attributwerte aller Tupels der Tabelle müssen eindeutig sein. UNIQUE in Kombination mit NOT NULL definiert einen Sekundärschlüssel. UNIQUE in Kombination mit NULL bedeutet, dass alle Attributwerte ungleich NULL eindeutig sein müssen.
CHECK	erlaubt die Definition von weiteren Einschränkungen (siehe Beispiele)
DEFAULT	setzt einen Defaultwert, dieser gilt wenn beim Einfügen eines Tupels mit INSERT kein Attributwert angegeben wird.
REFERENCES	Attribut ist Fremdschlüssel

6.1.5. ALTER TABLE

Postgres Dokumentation

Sollte bei foreign Keys bevorzugt werden, da somit rekursive Referenzen einfacher umgesetzt werden können.

```
ALTER TABLE table_name
  ADD CONSTRAINT constraint_name
  CHECK (counter < 1337);

ALTER TABLE other_table
  ADD CONSTRAINT fk_that
  FOREIGN KEY (that) REFERENCES table_name (counter);
```

6.1.6. Referentielle Integrität

Der Fremdschlüssel in einer abhängigen Tabelle muss als Wert entweder einen aktuellen Wert des Schlüssels der referenzierten Tabelle (1:n) oder NULL (1c:n) aufweisen. Diese referentielle Integritätsbedingung kann bei Einfüge-, Lösch- und Änderungsoperationen verletzt werden und sollte daher vom DBMS überprüft werden.

Trigger:

```
ON UPDATE
ON DELETE
```

Aktion:

```
CASCADE
RESTRICT
SET DEFAULT
SET NULL
```

Siehe «Beispiel CREATE TABLE» (Seite 5) .

6.1.7. Index

Ein Index ist eine Hilfsdatenstruktur, die zu einem gegebenen Attributwert die Adressen der Tupel mit diesem Attributwert liefert.

TODO: Beispiele

6.1.8. Schema

Postgres Dokumentation

Ein Schema ist ein Menge von DB-Objekten, welche zu einer logischen Datenbank gehören. Ein Schema kann folgende Objekte beinhalten: Tabellen, Sichten, Zusicherungen (Assertions), Berechtigungen etc.

TODO: Beispiele

6.1.9. Permissions

Postgres Dokumentation

TODO: Beschreibung, Beispiele

6.2. DML (DATA MANIPULATION LANGUAGE)

Postgres Dokumentation

6.2.1. Execution order

TODO:

6.2.2. INSERT

Postgres Dokumentation

```
INSERT INTO table_name (important, field, reference) VALUES (3, 99, 7);

INSERT INTO other_table VALUES (20, "Goodbye world.");
```

6.2.2.1. INSERT … SELECT

```
INSERT INTO combined_table
  SELECT t.age, t.very_long_name FROM table_name AS t
  INNER JOIN other_table AS o ON t.reference = o.id;
```

6.2.3. UPDATE

Postgres Dokumentation

Mit dem Update-Befehl können bestehende Tupel in der Datenbank modifiziert werden.

```
UPDATE table_name
  SET field = field * 2
  WHERE field IN (71,73,74);
```

6.2.4. DELETE

Postgres Dokumentation

```
DELETE FROM table_name
  WHERE field > 300;
```

6.2.5. SELECT

Postgres Dokumentation

```
SELECT field FROM table_name;

SELECT important, CONCAT(very_long_name, age) AS personal_info
  FROM table_name
  WHERE counter > 21
  GROUP BY personal_info
  ORDER BY age, counter DESC
  LIMIT 77;
```

6.2.5.1. Prädikate (WHERE)

Postgres Dokumentation

```
BETWEEN ... AND ...
IN (..., ...)
LIKE '___%'          -- 3 chars or more
LIKE '%asd'          -- ending with "asd"
AND
```

```
OR
IS (NOT) NULL
```

6.2.5.2. Aggregatfunktionen

Postgres Dokumentation

```
MAX()
MIN()
AVG()
SUM()
COUNT()
```

6.2.5.3. Gruppierung (GROUP BY and HAVING)

Postgres Dokumentation

GROUP BY teilt die Resultattabelle in Gruppen auf, die in der GROUP BY - Spalte gleiche Werte aufweisen. NULL-Werte einer GROUP-BY Spalte werden als separate Gruppe behandelt.

Die ***HAVING*** Klausel kann nur nach einer GROUP-BY Klausel stehen. Sie erlaubt die Auswahl von Zeilen, die durch die Anwendung der GROUP BY Bedingung entstehen (analog der WHERE-Klausel). Die Bedingung der HAVING-Klausel muss mit einer Funktion beginnen, welche in der SELECT-Klausel vorkommen muss.

```
SELECT very_long_name, age, COUNT(*)
  FROM table_name
  GROUP BY age, very_long_name
  HAVING COUNT(*) > 2;
```

6.2.5.4. Join

Postgres Dokumentation , Visuelle Beispiele Atlassian , Visuelle Beispiele CodeCademy

Die Join-Operation verbindet Tabellen über Spalten mit dem gleichen Datentyp. Damit können Beziehungen zwischen den Tabellen (realisiert über Fremdschlüssel) aufgelöst werden.

6.2.5.4.1. CROSS JOIN

Gibt das kartesische Produkt der beiden Tabellen zurück, d.h. jede Zeile aus der ersten Tabelle wird mit jeder Zeile aus der zweiten Tabelle kombiniert.

```
SELECT a.*, b.*
  FROM a
  CROSS JOIN b;
```

6.2.5.4.2. INNER JOIN

Gibt nur die Zeilen zurück, die in beiden Tabellen übereinstimmen.

```
SELECT a.*, b.*
  FROM a
  INNER JOIN b ON a.id = b.id;
```

6.2.5.4.3. LEFT JOIN

Gibt alle Zeilen aus der linken Tabelle zurück und die übereinstimmenden Zeilen aus der rechten Tabelle. Wenn es keine Übereinstimmung gibt, werden NULL-Werte für die rechte Tabelle zurückgegeben.

6.2.5.4.4. RIGHT JOIN

Wie left join, nur umgekehrt.

6.2.5.4.5. FULL OUTER JOIN

Gibt alle Zeilen zurück, die in einer der beiden Tabellen vorhanden sind. Wenn es keine Übereinstimmung gibt, werden NULL-Werte für die Tabelle zurückgegeben, in der keine Übereinstimmung gefunden wurde.

6.2.5.4.6. JOIN LATERAL

Ein spezieller Typ von Join, der es ermöglicht, eine Unterabfrage zu verwenden, die auf Zeilen der äußeren Abfrage basiert. Dies ist nützlich für komplexe Abfragen, bei denen die Ergebnisse einer Unterabfrage von den Ergebnissen der äußeren Abfrage abhängen.

```
SELECT a.*, b.*
  FROM table_a AS a,
    JOIN LATERAL (SELECT * FROM table_b WHERE table_b.id = a.id) AS b;
```

6.2.5.5. Mengenoperationen

Postgres Dokumentation

Jede dieser Operationen kann mit `ALL` postfixed werden, um die Duplikate beizubehalten.

6.2.5.5.1. UNION

Kombiniert die Ergebnisse zweier oder mehrerer SELECT-Abfragen und gibt alle Zeilen zurück, ausschließlich Duplikate.

```
SELECT r1, r2
FROM A
UNION ALL
SELECT r1, r2
FROM B;
```

6.2.5.5.2. INTERSECT

Gibt die gemeinsamen Zeilen aus zwei SELECT-Abfragen zurück. Das Ergebnis enthält nur die Zeilen, die in beiden Abfragen vorhanden sind.

6.2.5.5.3. EXCEPT (MINUS)

Gibt die Zeilen aus der ersten SELECT-Abfrage zurück, die nicht in der zweiten SELECT-Abfrage vorhanden sind.

6.2.5.6. Unterabfragen

Postgres Dokumentation

Eine Unterabfrage darf nur einen Spaltennamen oder einen Ausdruck und keine ORDER BY - Klausel enthalten.

Verschachteltes Beispiel:

```
SELECT name
  FROM mitarbeiter
  WHERE id IN
  (SELECT mitarbeiterNr
    FROM projektZuteilung
    WHERE projNr IN
    (SELECT projNr
      FROM projekt INNER JOIN mitarbeiter
      ON projekt.projLeiter = mitarbeiter.id
      WHERE name = 'Kropotkin, Peter'
    )
  );
```

6.2.5.6.1. =

Vergleicht den Wert einer Spalte mit dem Ergebnis einer Unterabfrage. Unterabfrage darf nur einen Wert zurückliefern.

```
SELECT *
  FROM mitarbeiter
  WHERE gehalt = (SELECT MAX(gehalt) FROM mitarbeiter);
```

6.2.5.6.2. IN

Überprüft, ob der Wert einer Spalte in der Menge der Ergebnisse einer Unterabfrage enthalten ist.

```
SELECT *
  FROM bestellungen
  WHERE kundenID IN (SELECT id FROM kunden WHERE land = 'CH');
```

6.2.5.6.3. EXISTS

Überprüft, ob eine Unterabfrage mindestens eine Zeile zurückgibt.

```
SELECT name
  FROM mitarbeiter
  WHERE EXISTS
  (SELECT * FROM projektZuteilung WHERE mitarbeiterNr = mitarbeiter.id);
```

6.2.5.6.4. ANY

Überprüft, ob der Wert einer Spalte irgendeinen Wert in der Menge der Ergebnisse einer Unterabfrage erfüllt.

```
SELECT *
  FROM mitarbeiter
  WHERE gehalt > ANY (SELECT gehalt FROM manager);
```

6.2.5.6.5. ALL

Überprüft, ob der Wert einer Spalte alle Werte in der Menge der Ergebnisse einer Unterabfrage erfüllt.

6.2.5.7. Window functions

Postgres Dokumentation

Window Functions sind spezielle SQL-Funktionen, die Berechnungen über eine Menge von Zeilen durchführen, die mit der aktuellen Zeile in Beziehung stehen.

```
SELECT
  mitarbeiter,
  gehalt,
  RANK() OVER (ORDER BY gehalt DESC) as gehaltsrang
FROM
  mitarbeitertabelle;

SELECT
  persnr,
  name,
  LAG(salaer, 1) OVER (
    PARTITION BY abtnr
  ORDER BY
    salaer DESC
  ) - salaer AS differenz
FROM
  angestellter;
```

6.2.5.7.1. Funktionen

Postgres Dokumentation

Begriff	*Bedeutung*
RANK()	Vergibt Rangpositionen mit Berücksichtigung von Gleichständen. Bei mehreren gleichen Werten erhalten diese den gleichen Rang, und die nächste Position wird übersprungen

Begriff	*Bedeutung*
ROW_NUMBER()	Weist jeder Zeile eine eindeutige fortlaufende Nummer zu. Auch bei gleichen Werten erhält jede Zeile eine unterschiedliche Nummer
LAG(value, offset)	Greift auf den Wert einer vorherigen Zeile im Fenster zu
LEAD(value, offset)	Greift auf den Wert einer nachfolgenden Zeile im Fenster zu
FIRST_VALUE(value)	Liefert den ersten Wert in der definierten Fenstermenge. Nützlich für Vergleiche mit dem Anfangswert einer Partition

6.2.5.7.2. OVER Klausel

Begriff	*Bedeutung*
ORDER BY	Sortiert die Zeilen innerhalb des Fensters nach einem oder mehreren Spalten. Bestimmt die Reihenfolge, in der Berechnungen für Window Functions durchgeführt werden
PARTITION BY	Teilt das Resultset in Partitionen, auf die Window Functions separat angewendet werden. Ermöglicht Berechnungen innerhalb definierter Gruppen, ohne die Gesamtergebnismenge zu ändern

6.3. VIEWS

Eine View ist eine virtuelle Tabelle, welche auf eine oder mehrere Tabellen oder Views abgebildet wird. Die Abbildung wird mit einer Select-Anweisung definiert. Die Daten der View werden erst zur Ausführungszeit aus den darunter liegenden Tabellendaten hergeleitet.

Die Views erlauben es, dass verschiedene Benutzer die Daten unterschiedlich strukturiert sehen. Mit Views kann die Formulierung von Abfragen, die sich über mehrere Tabellen erstrecken, verein- facht werden. Views erlauben einen wirksamen Zugriffsschutz, da es möglich ist, Spalten der da- runterliegenden Tabellen auszublenden.

```
CREATE VIEW mitarbeiter_public (id, name, tel) AS
  SELECT id, name, tel
  FROM mitarbeiter;
```

6.3.1. Common table expressions

TODO:

6.3.1.1. Recursive

A recursive CTE references itself to return subsets of data until all results are retrieved.

6.4. FUNKTIONEN

```
CREATE PROCEDURE
EXEC
```

6.5. DCL (DATA CONTROL LANGUAGE)

6.5.1. Benutzerverwaltung

Postgres Dokumentation Postgres Dokumentation

ROLE: Oberbegriff für Benutzer (***USER***) oder Gruppe (***GROUP***).

```
CREATE ROLE user_name
WITH LOGIN PASSWORD 'password';

DROP ROLE user_name;
```

6.5.2. Berechtigungen

Postgres Dokumentation

6.5.2.1. System

```
GRANT INSERT ON TABLE table_name TO user_name;
REVOKE INSERT ON TABLE table_name TO user_name;
```

6.5.2.2. Objekt

```
ALTER ROLE user_name CREATEROLE, CREATEDB, INHERIT;
```

6.5.3. Gruppen

```
CREATE ROLE manager;
GRANT SELECT ON table_name TO manager;
GRANT manager TO user_name;
```

6.5.4. Read-only user

```
-- creating
REVOKE CREATE ON SCHEMA public FROM PUBLIC;
CREATE ROLE readonlyuser WITH LOGIN ENCRYPTED PASSWORD 'readonlyuser' NOINHERIT;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO readonlyuser;
-- assign permissions to read all newly tables created in the future (by others):
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON TABLES TO readonlyuser;
-- deleting
REVOKE SELECT ON ALL TABLES IN SCHEMA public FROM readonlyuser;
ALTER DEFAULT PRIVILEGES IN SCHEMA public
  REVOKE SELECT ON TABLES FROM readonlyuser;
DROP USER readonlyuser;
```

6.5.5. RBAC (Role-Based Access Control)

Ist das, was oben erklärt wird.

6.5.6. RLS (Row-Level Security)

– Check-Constraint auf Tabelle pro Rolle (sozusagen ein zusätzliches WHERE)
– (Tipp: Aufpassen bei TRUNCATE, REFERENCES und VIEWS)

```
CREATE TABLE exams (
  id SERIAL,
  student_name TEXT,
  grade DECIMAL(2,1),
  added TIMESTAMP DEFAULT current_timestamp,
  teacher_pguser VARCHAR(60) DEFAULT current_user
);

CREATE POLICY policy_teachers_see_own_exams ON exams
FOR ALL
TO PUBLIC
USING (teacher_pguser = current_user);

ALTER TABLE exams
ENABLE ROW LEVEL SECURITY;
```

6.6. PREPARED STATEMENTS