**TODO: glossar****DataBase System (DBS)**

Besteht aus DBMS und Datenbasen

 DataBase Management System (DBMS)Redundanzfreiheit, Datenintegrität, Kapselung, **TODO:**

2 weitere

ANSI Modell**Logische Ebene:** Logische Struktur der Daten**Interne Ebene:** Speicherstrukturen, Definition durch internes Schema (Beziehungen, Tabellen etc.)**Externe Ebene:** Sicht einer Benutzerklasse auf Teilmengen der DB, Definition durch externes Schema**Mapping:** Zwischen den Ebenen ist eine mehr oder weniger komplexe Abbildung notwendig**Relationales Modell**PK sind unterstrichen, FK sind *kursiv*

tabellenname (

```
    id SERIAL PRIMARY KEY,
    grade DECIMAL(2,1) NOT NULL,
    fk INT FOREIGN KEY REFERENCES t2,
    u VARCHAR(9) DEFAULT CURRENT_USER,
);
```

Unified Modeling Language (UML)

→ Assoziation → Komposition

◇ Aggregation → Vererbung

Complete: Alle Subklassen sind definiert**Incomplete:** Zusätzliche Subklassen sind erlaubt**Disjoint:** Ist Instanz von genau einer Unterklasse**Overlapping:** Kann Instanz von mehreren überlappenden Unterklassen sein**Normalisierung****1NF:** Atomare Attributwerte**2NF:** Nichtschlüsselattr. voll vom Schlüssel abhängig**3NF:** Keine transitiven Abhängigkeiten**BCNF:** Nur Abhängigkeiten vom Schlüssel**(Voll-)funktionale Abhängigkeit:** B hängt von A ab, zu jedem Wert von A gibt es genau einen Wert von B ($A \rightarrow B$)**Transitive Abhängigkeit:** B hängt vom Attribut A ab, C hängt von B ab ($A \rightarrow B \wedge B \rightarrow C \Rightarrow A \rightarrow C$)**Denormalisierung:** In geringere NF zurückführen (Verbessert Performance und reduziert Joins-Komplexität)**Anomalien****Einfügeanomalie, Löschanomalie, Änderungsanomalie****Data Definition Language (DDL)**

```
CREATE SCHEMA s;
CREATE TABLE t (id SERIAL PRIMARY KEY,
    name TEXT UNIQUE,
    grade DECIMAL(2,1) NOT NULL,
    fk INT FOREIGN KEY REFERENCES t2.id ON DELETE CASCADE,
    added TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    u VARCHAR(9) DEFAULT CURRENT_USER,
    CHECK (grade between 1 and 6));
ALTER TABLE t2 ADD CONSTRAINT c PRIMARY KEY (a, b);
TRUNCATE/DROP TABLE t;
```

Vererbung**Tabelle pro Sub- und Superklasse:**

```
-- TODO: check if correct
CREATE TABLE sup (id SERIAL PRIMARY KEY, -- 3.a
    name TEXT UNIQUE);
CREATE TABLE sub1 (id SERIAL PRIMARY KEY, age INT);
CREATE TABLE sub2 (id SERIAL PRIMARY KEY);
ALTER TABLE sub1 ADD CONSTRAINT id FOREIGN KEY
    REFERENCES sup (id); -- Auch für sub2
```

Tabelle pro Subklasse: Enthält jeweil. Subklassattribute

```
CREATE TABLE sub1 (id SERIAL PRIMARY KEY, -- 3.b
    name TEXT UNIQUE, age INT);
CREATE TABLE sub2 (id SERIAL PRIMARY KEY,
    name TEXT UNIQUE);
```

Einzige Tabelle für Superklasse: Enthält alle Attribute

```
CREATE TABLE sup (id SERIAL PRIMARY KEY, -- 3.c
    name TEXT UNIQUE, age INT);
```

Datentypen

SMALLINT	INT	INTEGER	BIGINT	REAL	FLOAT
DOUBLE	NUMERIC(precision,scale)	DECIMAL(p,s)			
VARCHAR(size)	TEXT	CHAR(size)	-- fixed size		
DATETIME	DATE	INTERVAL	TIME	BINARY	
CLOB	/Char Large Object*/	BLOB		VARBINARY	

Casting**Implicit TODO**

CAST(5 AS float8) = 5::float8

Data Manipulation Language (DML)

```
FROM -> JOIN -> WHERE -> GROUP BY -> HAVING ->
    SELECT (WINDOW FUNCTIONS) -> ORDER BY -> LIMIT
INSERT INTO t (added, grade) VALUES ('2002-10-10', 1)
    RETURNING id;
```

Views

Resultate werden jedes mal dynamisch queried

CREATE VIEW v (id, u) AS SELECT id, u FROM t;

Updatable View

Views sind updatable wenn diese Kriterien erfüllt sind:

- Single base table
- Keine aggregate, DISTINCT, GROUP BY, oder HAVING Klauseln
- Alle Spalten müssen zur originalen Tabelle direkt ge-mappt werden können

Materialized View

Speichert resultat auf Disk

```
CREATE MATERIALIZED VIEW mv AS SELECT * FROM t;
REFRESH MATERIALIZED VIEW mv; -- refresh results
```

Row-Level Security (RLS)

```
CREATE TABLE exams (id SERIAL, -- other fields...
    teacher VARCHAR(60) DEFAULT current_user);
CREATE POLICY teachers_see_own_exams ON exams
    FOR ALL TO PUBLIC USING (teacher = current_user);
ALTER TABLE exams ENABLE ROW LEVEL SECURITY;
```

Temporäre Tabellen**TODO****Data Control Language (DCL)**

```
CREATE ROLE u WITH LOGIN PASSWORD ''; -- user
GRANT INSERT ON TABLE t TO u;
ALTER ROLE u CREATEROLE, CREATEDB, INHERIT;
CREATE ROLE r; -- group
GRANT r TO u; -- put user u in group r
REVOKE CREATE ON SCHEMA s FROM r;
```

Read-only user

```
-- creating
REVOKE CREATE ON SCHEMA public FROM PUBLIC;
CREATE ROLE u WITH LOGIN ENCRYPTED PASSWORD ''
    NOINHERIT; -- don't inherit privileges
GRANT SELECT ON ALL TABLES IN SCHEMA public TO u;
-- read all new tables (also created by others):
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT
    SELECT ON TABLES TO u;
-- deleting
REVOKE SELECT ON ALL TABLES IN SCHEMA public FROM u;
ALTER DEFAULT PRIVILEGES IN SCHEMA public
    REVOKE SELECT ON TABLES FROM u;
DROP USER u;
```

Common Table Expressions (CTE)

```
-- normal
WITH cte AS (SELECT * FROM t) SELECT * FROM cte;
WITH tmp(id, name) AS (SELECT id, name FROM t)
    SELECT id, name FROM tmptable;
-- recursive
WITH RECURSIVE q AS (SELECT * FROM t WHERE grade>1
    UNION ALL SELECT * FROM t INNER JOIN q ON
        q.u = t.name) SELECT id as 'ID' FROM q;
```

Window Functions

```
SELECT id, RANK() OVER
    (ORDER BY grade DESC) as r FROM t;
SELECT id, u, LAG(name, 1) OVER
    (PARTITION BY fk ORDER BY id DESC) FROM t;
-- PERCENT/DENSE_RANK(), FIRST_VALUE(v), LAST_VALUE(n)
-- NTH_VALUE(v,n), NTILE(n), LEAD(v,o), ROW_NUMBER()
```

Subqueries

```
SELECT * FROM t WHERE grade > ANY (SELECT g FROM t2);
SELECT * FROM t WHERE EXISTS (SELECT g FROM t2);
-- ALL, ANY, IN, EXISTS, =
```

Inner Join

Zeilen, die in beiden Tabellen matchen

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

Equi Join

Wie Inner Join

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

Natural Join

Wie Inner Join aber ohne Duplikate

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

Semi Join

Nur Zeilen aus a, wobei b matchen muss

```
SELECT a.* FROM a WHERE EXISTS
    (SELECT 1 FROM b WHERE a.id = b.id);
```

Anti Join

Nur Zeilen aus a, wobei b nicht matchen darf

```
SELECT a.* FROM a WHERE NOT EXISTS
    (SELECT 1 FROM b WHERE a.id = b.id);
```

Left outer Join

Alle Zeilen beider Tabellen, NULL für b falls kein match

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

Right outer Join

Alle Zeilen beider Tabellen, NULL für a falls kein match

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

Full outer Join

Alle Zeilen beider Tabellen, NULL falls kein match

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

Lateral Join

Join, der Subqueries erlaubt

```
SELECT x.* ,y.* FROM a AS x JOIN LATERAL
    (SELECT * FROM b WHERE b.id = y.id) AS y ON TRUE;
```

GROUP BY

```
SELECT id, COUNT(*) FROM t
    GROUP BY grade, id HAVING COUNT(*) > 2;
```

WHERE

```
BETWEEN 1 AND 5; LIKE '_%'; AND; IS (NOT) NULL
IN (1, 5) ; LIKE '%asd'; OR ;
```

INDEX

```
CREATE INDEX i ON t /*USING BTREE*/ (grade, UPPER(u));
CREATE INDEX j ON t (fk) INCLUDE (added) WHERE fk > 4;
DROP INDEX i;
```

Transaktionen**Atomicity:** Vollständig oder gar nicht**Consistency:** Konsistenter Zustand bleibt erhalten**Isolation:** Transaktion ist von anderen T isoliert**Durability:** Änderungen sind persistent

```
BEGIN; SAVEPOINT s;
COMMIT; ROLLBACK /*TO SAVEPOINT s*/;
```

Isolation

Georgiy Shevoroshkin

SET TRANSACTION ISOLATION LEVEL ...; -- for transaction
 SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION
 LEVEL ...; -- for session

READ UNCOMMITTED: Lesezugriffe nicht synchronisiert (keine Read-lock). Read ignoriert jegliche Sperren
READ COMMITTED: Lesezugriffe nur kurz/temporär synchronisiert (default), setzt für gesamte T Write-Lock, Read-lock nur kurzfristig

REPEATABLE READ: Einzelne Zugriffe ROWS sind synchronisiert, Read und Write Lock für die gesamte T

SERIALIZABLE: Vollständige Isolation nach ACID

	Read Uncommitted	Read Committed	Repeatable Read	Serializable
Dirty Write	*	*	*	X
Dirty Read	✓	X	X	X
Lost Update	✓	✓	X	X
Fuzzy Read	✓	✓	X	X
Phantom Read	✓	✓	✓	X
Read Skew	✓	✓	X	X
Write Skew	✓	✓	✓	*

* Nur in SQL92 möglich, PSQL >= 9.1 verhindert dies

Dirty Read: Lese Daten von nicht committed T's

Fuzzy Read: Versch. Werte beim mehrmaligen Lesen gleicher Daten (da durch andere T geändert)

Phantom Read: Neue/Gelöschte Rows einer anderen T

Read Skew: Daten lesen, die sich während der T ändern

Write Skew: Mehrere T lesen Daten und Ändern sie

Deadlock: Mehrere T blockieren sich, da sie auf die gleiche Ressource warten

Cascading Rollback: T schlägt fehl und alle davon abhängigen T müssen ebenfalls zurückgerollt werden

	Garantiert Serializable	Keine Deadlocks	Keine Cascading Rollbacks	Keine Konflikt-Rollbacks	Hohe Parallelität	Realistisch (ohne Voranalyse)
Two-Phase Locking	✓	X	X	✓	X	X
Strict 2PL	✓	X	✓	✓	X	✓
Precalimining 2PL	✓	✓	✓	✓	X	X
Validation-based	✓	✓	X	X	✓	✓
Timestamp-based	✓	✓	X	X	✓	✓
Snapshot Isolation	X	*	✓	X	✓	✓
SSI	✓	*	✓	X	✓	✓

* Deadlock in PSQL mit Snapshot Isolation

Relationale Algebra

$\pi_{R1,R4}(R)$ SELECT R1,R4 FROM R;

$\sigma_{R1>30}(R)$ SELECT * FROM R WHERE R1 > 30;

$p_a \leftarrow R$ SELECT * FROM R AS a;

$R \times S$ SELECT * FROM R,S;

$R \bowtie_{A=B} S$ SELECT * FROM R JOIN S ON R.A=S.B;

Serialisierbarkeit

Shared Lock: Schreib- & Lesezugriffe (eine Transaktion)

Exclusive Lock: Lesezugriffe (mehrere Transaktionen)

Serieller Schedule: Führt Transaktionen am Stück aus

Nicht serialisierbar:

$S1=R1(x)R2(x)\underline{W1(x)}R1(y)W2(x)W1(y)$

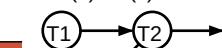


Konflikt-Serialisierbar:

r1(b) r2(b) w2(b) r2(c) r2(d) w3(a) r4(d) r3(b) w4(d) r5(c)
 r5(a) w4(c)

Konflikt-Äquivalenter serieller Schedule:

r1(b) r2(b) w2(b) r2(c) r2(d) w3(a) r3(b) r5(c) r5(a) r4(d)
 w4(d) w4(c)



Backup

TODO

Vollständiges Backup

TODO

Inkrementelles Backup

TODO

Multi-Version Concurrency Control (MVCC)

Ermöglicht es, mehreren T gleichzeitig zu laufen

TODO

Two-Phase Locking (2PL)

Stellt Isolation der T sicher

TODO

Write-Ahead Log (WAL)

Schreibt Änderungen der T in Log, dann Commit loggen, dann Updates in DB. Kann bei Absturz replayed werden

LSN, TaID, PageID, Redo, Undo, PrevLSN

Dreiwertige Logik (cursed)

```
SELECT NULL IS NULL; -- true
SELECT NULL = NULL; -- [null]
```

B-Baum

