**DataBase System**

Besteht aus DBMS und Datenbasen

 DataBase Management System

Redundanzfreiheit, Datenintegrität, Kapselung

ANSI Modell**Logische Ebene:** Logische Struktur der Daten**Interne Ebene:** Speicherstrukturen, Definition durch internes Schema (Beziehungen, Tabellen etc.)**Externe Ebene:** Sicht einer Benutzerklasse auf Teilmenge der DB, Definition durch externes Schema**Mapping:** Zwischen den Ebenen ist eine mehr oder weniger komplexe Abbildung notwendig**Relationale Schreibweise****TODO****Unified Modeling Language**

→ 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:**TODO****Transitive Abhängigkeit:****TODO****Denormalisierung:****TODO**

Einfügeanomalie, Löschanomalie, Änderungsanomalie

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);
```

Einige Tabelle für Superklasse: Enthält alle Attribute

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

Data Definition Language

```
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;
```

Data Manipulation Language

```
INSERT INTO t (added, grade) VALUES ('2002-10-10', 1)
RETURNING id;
```

Views

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

Updatable View**TODO****Materialized View****TODO****Row-Level Security (RLS)****TODO****Temporäre Tabellen****TODO****Data Control Language**

```
CREATE ROLE r WITH LOGIN PASSWORD ''
GRANT INSERT ON TABLE t TO r;
REVOKE CREATE ON SCHEMA s FROM r;
ALTER ROLE r CREATEROLE, CREATEDB, INHERIT;
GRANT r TO user_name;
-- read all future created tables
ALTER DEFAULT PRIVILEGES IN SCHEMA s GRANT SELECT ON
TABLES TO readonlyuser;
CREATE POLICY p ON t FOR ALL TO PUBLIC USING (u =
current_user);
ALTER TABLE t ENABLE ROW LEVEL SECURITY;
```

Common Table Expressions

```
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, =
```

JOIN

```
SELECT a.* , b.* FROM a INNER JOIN b ON a.id = b.id;
SELECT y.* , x.* FROM t AS y JOIN LATERAL
  (SELECT * FROM t2 WHERE t2.id = y.id) AS x;
```

Inner Join**TODO****Equi Join****TODO****Natural Join****TODO****Semi Join****TODO****Anti Join****TODO****Left outer Join****TODO****Right outer Join****TODO****Full outer Join****TODO****Lateral Join****TODO****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))
INCLUDE (added);
CREATE INDEX i ON t (grade) WHERE grade > 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

```
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

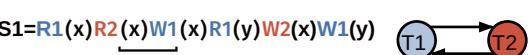
	Read Committed	Uncommitted	Read Committed	Repeatable Read	Serializable
Dirty Write	+		+	-	x
Dirty Read	x	+	x	x	x
Lost Update	+	+	x	x	x
Fuzzy Read	+	+	x	x	x
Phantom Read	+	+	+	x	x
Read Skew	+	+	x	x	x
Write Skew	+	+	+	+	-

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

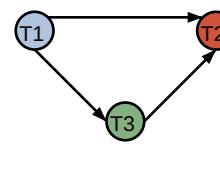
Dirty Read: Lese Daten von nicht committed T's**Fuzzy Read:** Versch. Werte beim mehrmaligen Lesen gleicher Daten (da durch andere T geändert)**Phantom Read:** Neue/Gelöschte Rows einer anderen T**Read Skew:** Daten lesen, die sich während der T ändern**Write Skew:** Mehrere T lesen Daten und Ändern 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 Serialisierbar	Keine Deadlocks	Keine Cascading Rollbacks	Keine Konflikt-Rollbacks	Hohe Parallelität	Realistisch (ohne Voranalyse)
Two-Phase Locking	+	x	x	+	x	x
Strict 2PL	+	x	+	+	x	+
Precalming 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; $\rho_{a \leftarrow R}$ SELECT * FROM R AS a; $R \times S$ SELECT * FROM R,S; $R \bowtie S$ 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:** $S1=R1(x)R2(x)W1(x)R1(y)W2(x)W1(y)$ **Conflict serializable (serialisierbar):**

<i>T</i> ₁	<i>T</i> ₂	<i>T</i> ₃
R(x)		
R(y)		
	R(x)	
	R(z)	
W(y)		
CT		
		R(y)
		R(z)
		W(y)
		CT
	W(x)	
	W(z)	
CT		



Backup

TODO

MVCC

TODO

2PL

TODO

Write-Ahead Log

TODO: LSN, Taid, PageID, Redo, Undo, PrevLSN

Dreiwertige Logik (cursed)

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

B-Baum

