



Georgiy Shevoroshkin

```

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**

Note: in postgres gibt es keine geschachtelten T.

**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 ...; -- transaction
SET SESSION CHARACTERISTICS AS TRANSACTION
ISOLATION LEVEL ...; -- 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 Un-committed	Read Committed	Repeata-ble Read	Seria-lizable
Dirty Write	*	*	*	x
Dirty Read	x		x	x
Lost Update	x	x	x	x
Fuzzy Read	x	x	x	x
Phantom Read	x	x	x	x
Read Skew	x	x	x	x
Write Skew	x	x	x	*

\* Nur in SQL92 möglich, PSQL &gt;= 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

	Serial-sierbar	Dead-locks	Cas-cading Roll.	Kon-flikt-Roll.	Hohe Paral-lelität	Realis-tisch
Two-Phase Locking	✓	✓	✓	x	x	x
Strict 2PL	✓	✓	x	x	x	✓
Precalim-ing 2PL	✓	x	x	x	x	x
Validation-based	✓	x	✓	✓	✓	✓
Timestamp-based	x	✓	✓	✓	✓	✓
Snapshot Isolati-on	x	*	x	✓	✓	✓
SSI	✓	*	x	✓	✓	✓

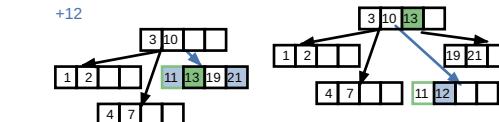
\* Deadlock in PSQL mit Snapshot Isolation

**SQL Beispiel**

```

BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

```

**Relationale Algebra**

```

πR1,R4(R) SELECT R1,R4 FROM R; (Projektion)
σR1>30(R) SELECT * FROM R WHERE R1 > 30; (Selektion)
ρa ← R SELECT * FROM R AS a; (Umbenennung/Alias)
R × S SELECT * FROM R,S; (Kartesisches Produkt)
R ⋈A=B S SELECT * FROM R JOIN S ON R.A=S.B; (Verbund)

```

**Serialisierbarkeit**

Shared Lock: Schreib- &amp; Lesezugriffe (eine Transaktion)

Exclusive Lock: Lesezugriffe (mehrere Transaktionen)

Serieller Schedule: Führt Transaktionen am Stück aus

Nicht serialisierbar:

$$S1 = R1(x) \xrightarrow{\text{W1}(x)} R1(y) \xrightarrow{\text{W2}(x)} R1(y)$$
**Konfliktpaare:** $R1(x) < W2(x)$        $R2(x) < W1(x)$ **Konflikt-Serialisierbar:** $r1(b)r2(b)r2(c)r2(d)r3(a)r4(d)r3(b)r4(d)r5(c)r5(a)r4(c)$ **Konflikt-Äquivalenter serieller Schedule:** $r1(b)r2(b)r2(c)r2(d)r3(a)r3(b)r5(c)r5(a)r4(d)r4(d)r4(c)$ **Vollständiges Backup**

Exakte Kopie der ganzen DB

**Inkrementelles Backup**

Sichert nur die seit dem letzten Backup geänderten Daten.

**Logisches Backup (SQL Dump)**

Blockiert keine T. Für mittelgroße Datenmengen, interkompatibel mit neuen PG-Versionen und anderen Maschinen.

**Physisches Backup (File System)**

Datenbank muss gestoppt werden, schneller als logisches Backup, passt nur zu derselben «Major Version» von PG.

**Multi-Version Concurrency Control (MVCC)**

Ermöglicht es, mehreren T gleichzeitig zu laufen. Bei jeder Änderung wird eine neue Version der Daten erstellt. Leser sehen die älteren Versionen, während Schreiber die neuesten Versionen sehen.

**Two-Phase Locking (2PL)****TODO: example**

Stellt Isolation der T sicher

- 1) Growing Phase: Die T. kann neue Locks erwerben, jedoch keine freigeben
- 2) Shrinking Phase: Locks können freigegeben werden, aber keine neuen mehr erworben werden

**Write-Ahead Log (WAL)**

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

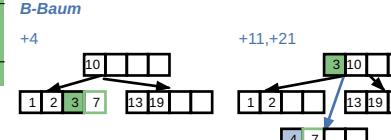
LSN, TaID, PageID, Redo, Undo, PrevLSN

**Dreiwertige Logik (cursed)**

```

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

```

**B-Baum****SQL Beispiele**

```

CREATE TABLE pferd (
    pnr SERIAL PRIMARY KEY,
    name TEXT,
    alter INT,
    zuechternr INT REFERENCES stall.pk,
    vaternr INT REFERENCES pferd.pk
);

CREATE TABLE stall (
    zuechternr SERIAL PRIMARY KEY,
    name TEXT,
    plz INT,
    ort TEXT,
    strasse TEXT
);

```

-- Welche Züchter haben in ihren Ställen mindestens 1 Kind von dem Vater mit Namen "Hermes"

-- Elegantere anfrage unkorreliert

```

SELECT s.name FROM staelle s
WHERE s.zuechternr IN (
    SELECT p.zuechternr
    FROM pferde p
    JOIN pferde p2 ON p2.pnr = p.vaternr
    WHERE p2.name = 'Hermes'
);

```

-- Kürzeste anfrage

```

SELECT DISTINCT s.name FROM staelle s
JOIN pferde p ON p.zuechternr = s.zuechternr
JOIN pferde p2 ON p2.pnr = p.vaternr
WHERE p2.name = 'Hermes';

```

--

```

SELECT DISTINCT s.name FROM staelle s
JOIN pferde p ON p.zuechternr = s.zuechternr
WHERE EXISTS (
    SELECT vaternr FROM pferde p2
    WHERE p2.pnr = p.vaternr AND p2.name = 'Hermes'
);

```