

Georgiy Shevoroshkin

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

UPDATE

```
UPDATE t SET grade = grade+1, name=' ' WHERE id = 1;
```

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 ;
```

Aggregatfunktionen

COUNT ; SUM ; MIN ; MAX ; AVG

Weitere Funktionen

```
COALESCE(a1, a2, ...); -- returns first non-null arg
```

INDEX

	B-Tree	Hash	BRIN	ISAM
Gleichheitsabfragen	✓	✓	X	✓
Range Queries	✓	X	✓	X
Sortierte Daten	✓	X	✓	✓
Grosse Tabellen	*	*	✓	✓

* Hash: Nur bei Gleichheitsabfragen

```
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
```

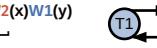
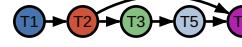
READ UNCOMMITTED: Lesezugriffe nicht synchronisiert (keine Read-lock). Read ignoriert jegliche Sperren**READ COMMITTED:** Lesezugriffe nur kurz/temporär synchronisiert (default), setzt für gesamte T Write-Lock, Read-lock nur kurzfristig**REPEATABLE READ:** Einzelne Zugriffe ROWS sind synchronisiert, Read und Write Lock für die gesamte T**SERIALIZABLE:** Vollständige Isolation nach ACID

	Read Un-committed	Read Com-mitted	Repeata-ble Read	Seria-lizable
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

```
BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
UPDATE accounts SET balance = balance - 100.00
  WHERE name = 'Alice';
SAVEPOINT my_savepoint;
UPDATE accounts SET balance = balance + 100.00
  WHERE name = 'Bob';
ROLLBACK TO my_savepoint;
UPDATE accounts SET balance = balance + 100.00
  WHERE name = 'Wally';
COMMIT;
```

Relationale Algebra $\pi_{R1,R4}(R)$ SELECT R1,R4 FROM R; (Projektion) $\sigma_{R1>30}(R)$ SELECT * FROM R WHERE R1 > 30; (Selektion) $p_a \leftarrow R$ SELECT * FROM R AS a; (Umbenennung/Alias) $R \times S$ SELECT * FROM R,S; (Kartesisches Produkt) $R \bowtie S$ SELECT * FROM R JOIN S ON R.A=S.B; (Verbund)**Serialisierbarkeit****S shared Lock:** Schreib- & Lesezugriffe (eine Transaktion)**E X clusive Lock:** Lesezugriffe (mehrere Transaktionen)**Starvation:** T erhält aufgrund von Sperren niemals die Möglichkeit, ihre Arbeit abzuschließen, da T immer blockiert wird**Serieller Schedule:** Führt Transaktionen am Stück aus**Nicht serialisierbar:** $S1=R1(x)R2(x)W1(x)R1(y)W2(x)W1(y)$ **Konfliktpaare:** $R1(x) < W2(x)$ $R2(x) < W1(x)$ **Konflikt-Serialisierbar:** $r1(b)r2(b)w2(b)r2(c)w2(d)w3(a)r4(d)r3(b)w4(d)r5(c)r5(a)w4(c)$ **Konflikt-Äquivalenter serieller Schedule:** $r1(b)r2(b)w2(b)r2(c)w2(d)w3(a)r3(b)r5(c)r5(a)r4(d)w4(d)w4(c)$ **Begriff****Seriell** Alle T in einem Schedule sind geordnet**Konfliktäquivalent** Reihenfolge aller Paare von konfigurierenden Aktionen ist in beiden Schedules gleich**Konfliktserialisierbar** Ein S backslash a registriert konfliktäquivalent zu einem seriellen S**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.

Physikalisches 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

Optimistisches Lockverfahren

T operieren ohne anfängliche Sperren. Überprüfen am Ende falls Konflikte auftreten → Änderungen zurücksetzen.

Pessimistisches Lockverfahren

T fordern sofort Sperren an, damit andere T nicht gleichzeitig auf dieselben Daten zugreifen oder diese ändern.

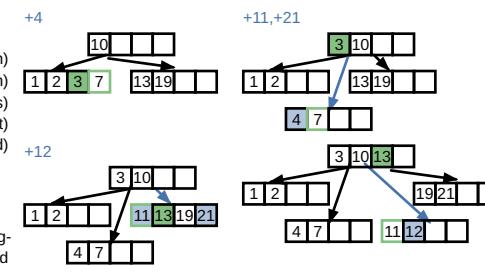
Write-Ahead Log (WAL)

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 p.pferdnr = p2.pferdnr
  WHERE p2.vaternr = 'Hermes'
);
```

);

-- Kürzeste Anfrage

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);

--

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

);