

Inner Join

Zeilen, die in beiden Tabellen
matchen

1	1	Alice	7	LOGIN
---	---	-------	---	-------

Subqueries

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

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;
```

Equi Join

Wie Inner Join

1	1	Alice	7	LOGIN
---	---	-------	---	-------

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

Natural Join

Wie Inner Join aber ohne Duplikate

1	1	Alice	7	LOGIN
---	---	-------	---	-------

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

Semi Join

Nur Zeilen aus a, wobei b matchen muss

1	1	Alice
---	---	-------

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

Anti Join

Nur Zeilen aus a, wobei b nicht matchen darf

1	2	Bob
---	---	-----

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

Left outer Join

Alle Zeilen beider Tabellen, NULL
für b falls kein match

1	1	Alice	7	LOGIN
---	---	-------	---	-------

2	2	Bob		
---	---	-----	--	--

```
SELECT u.*, a.* FROM u LEFT
JOIN a ON u.id = a.uid;
```

Right outer Join

Alle Zeilen beider Tabellen, NULL
für a falls kein match

1	1	Alice	7	LOGIN
---	---	-------	---	-------

3			8	VIEW
---	--	--	---	------

```
SELECT u.*, a.* FROM u RIGHT
JOIN a ON u.id = a.uid;
```

Full outer Join

Alle Zeilen beider Tabellen, NULL
falls kein match

1	1	Alice	7	LOGIN
---	---	-------	---	-------

2	2	Bob		
---	---	-----	--	--

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

Cross Join

Liefert alle möglichen Kombinationen zweier Tabellen.

1	1	Alice	7	LOGIN
---	---	-------	---	-------

2	1	Alice	8	VIEW
---	---	-------	---	------

```
SELECT * FROM u CROSS JOIN a;
```

Union

«Verbindet» zwei SELECT's ohne Duplikate.
Voraussetzung: Spalten müssen ähnliche Datentypen beinhalten. **Union all** ist wie Union, nur mit duplikaten ⇒ Rekursive CTEs

1	Alice
---	-------

2	Bob
---	-----

3	LOGIN
---	-------

4	VIEW
---	------

```
SELECT name FROM u UNION SELECT action FROM a;
```

Lateral Join

Erlaubt Subqueries mit Referenzen zu den anderen Tabellen

1	LOGIN
---	-------

```
SELECT u.* , x.action FROM u JOIN LATERAL
  (SELECT * FROM a WHERE a.uid = u.id)
  AS x ON TRUE;
```

INSERT

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

UPDATE

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

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	Repeatable Read	Serizable
Dirty Write	*	*	*	✗
Dirty Read	✓	✗	✗	✗
Lost Update	✓	✓	✗	✗
Fuzzy Read	✓	✓	✗	✗
Phantom Read	✓	✓	✓	✗
Read Skew	✓	✓	✗	✗
Write Skew	✓	✓	✓	✗
Dauerhaftigkeit	✓	✓	✗	✗
Atomizität	✗	✗	✓	✓

Two-Phase Locking (2PL)

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

Strict 2PL: T geben locks erst nach commit frei

Preclaiming 2PL: Alle Locks werden zu Beginn der T erstellt

Shared Lock: Lesezugriffe (mehrere Transaktionen)

Exclusive Lock: Schreib- & Lesezugriffe (eine Transaktion)

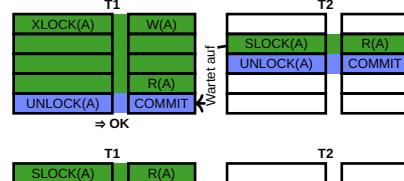
Starvation: T erhält aufgrund von Sperren niemals die Möglichkeit, ihre Arbeit abzuschließen, da immer blockiert wird

Optimistic Lockverfahren

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

Pessimistisches Lockverfahren (Preclaiming 2PL)

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

Growing phase**Shrinking phase****Konfliktpaare:**

R1(x) < W2(x) R2(x) < W1(x)

Konflikt-Serialisierbar:

r1(b)(2)(bw2(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)(2)(bw2(b)(r2(c)r2(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 ist 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.

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 neusten Versionen sehen.

Write-Ahead Log (WAL)

Schreibt Änderungen der T in Log, dann Commit loggen, dann Updates in DB. Kann bei Absturz replayed werden LSN, TxD, PageID, Redo, Undo, PrevLSN

SQL Beispiele

```
CREATE TABLE pferd (
  pnr SERIAL PRIMARY KEY,
  name TEXT,
  alter INT,
  zuechtern INT REFERENCES stall.pn,
  vaterrn INT REFERENCES pferd.pn
);
CREATE TABLE stall (
  zuechtern 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"
```

```
-- Eleganteste anfrage unkorreliert
```

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

```
-- Kürzeste anfrage
```

```
SELECT DISTINCT s.name FROM staelle s
JOIN pferde p ON p.zuechtern = s.zuechtern
JOIN pferde p2 ON p2.pnr = p.vaterrn
WHERE p2.name = 'Hermes';
```

```
-- RECURSIVE CTE
WITH RECURSIVE tens AS (
  SELECT 1 as n
  UNION ALL
  SELECT n+1 FROM tens
) SELECT n FROM tens limit 10;
-- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

B-Baum