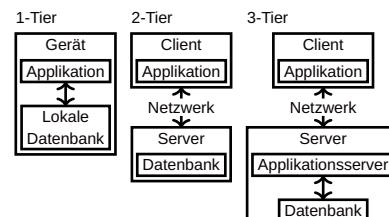
**Glossar**

Term	Definition
Impedance-Mismatch	Diskrepanz zwischen Datenstrukturen auf Applikations- und Datenbankebene
System-/Datenkatalog	Enthält Metadaten über die Datenbankobjekte, z.B. Tabellen und Schemata.
Datenbankschema	Struktur einer Datenbank, die die Organisation der Daten und Beziehungen beschreibt.
Datenbasis	Der physische Speicherort
Surrogate Key	Künstlich generierter PK
Referentielle Integrität	Fremdschlüssel muss zu einem Wert der referenzierten Tabelle oder NULL zeigen
Datenunabhängigkeit	Daten in einer DB ändern können, ohne dass Anwendungen geändert werden müssen
Data Pages	Kleinste Speicher-Dateneinheiten einer DB
Heaps	Unsortierte Datenorganisation
Semantische Integrität	Daten sind nicht nur syntaktisch, sondern auch inhaltlich korrekt, insbesondere nach T
Data dictionary	Zentrale Sammlung von Metadaten über die Daten im DBMS

Datenbankmodelle

Begriff	Bedeutung
Hierarchisch	Daten sind in einer baumartigen Struktur geordnet
Netzwerk	Flexiblere Struktur als hierarchisch, erlaubt mehrere Pfade zwischen Entitäten
Objektorientiert	Speichert Daten und ihr Verhalten in Form von Obj.
Objektrelational	Kombiniert objektorientierte + relationale Prinzipien
Relational	Speichert Daten in Tabellen (Relationen) und verwaltet Beziehungen durch Schlüssel

**DataBase System (DBS)**

Besteht aus DBMS und Datenbasen

 DataBase Management System (DBMS)

- (A) Transaktionen
- (B) Konsistenz
- (C) Benutzerbetrieb
- (D) Große Datensätze
- (E) Sicherheit
- Datentypen
- Abfragesprache
- Backup & Recovery
- Redundanzfreiheit
- Kapselung

ANSI Modell

Logische Ebene: Logische Struktur der Daten

Interne Ebene: Speicherstrukturen, Definition durch internes Schema (Beziehungen, Tabellen etc.)

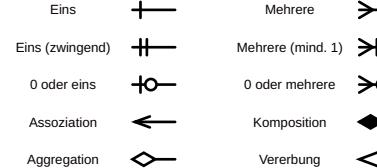
Externe Ebene: Sicht einer Benutzerklasse auf Teilmenge der DB, Definition durch externes Schema

Mapping: Zwischen den Ebenen ist eine mehr oder weniger komplexe Abbildung notwendig

Relationales Modell

PK sind unterstrichen, FK sind *kursiv* tabellename (

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)

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: **track** aufteilen

<i>id</i>	track	<i>id</i>	interpret	<i>titel</i>
1	Fugazi: Song #1	1	Fugazi	Song #1

2NF: Nichtschlüsselattr. voll vom Schlüssel abhängig. Ist PK atomar, dann 2NF gegeben. Im Beispiel sind nicht alle Attribute des PK notwendig, um **album** eindeutig zu identifizieren

track	cd_id	album	titel
1	1	Repeater	Turnover
2	1	Repeater	Song #1

track		cd	
track	cd_id	titel	id
1	1	Turnover	1
2	1	Song #1	Repeater

3NF: Keine transitiven Abhängigkeiten: **land** ist abhängig von **interpret**

<i>id</i>	album	interpret	land
1	Repeater	Fugazi	USA
2	Red Medicine	Fugazi	USA

cd		interpret	
<i>id</i>	album	<i>id</i>	name
1	Repeater	1	Fugazi
2	Red Medicine	1	USA

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 ($A \rightarrow B$)

Teilweise funk. Abh.: B hängt von A ab, aber auch von einem Teil eines zusammengesetzten Schlüssels.

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

BNF

```
<select> := [ 'WITH' [ 'RECURSIVE' ] <with_query> [, ...] ]
SELECT' [ 'ALL' | 'DISTINCT' [ 'ON' [ <expression> [, ...] ] ] ]
[ { * | <expression> [ [ 'AS' ] <output_name> ] ] [, ...] ]
[ 'FROM' <from_item> [, ...] ]
[ 'WHERE' <condition> ]
[ 'GROUP BY' [ 'ALL' | 'DISTINCT' ] <grouping_elem> [, ...] ]
[ 'HAVING' <condition> ]
[ 'WINDOW' <window_name> 'AS' [ <>window_def> ] [, ...] ]
[ 'INTERVAL' [ 'INTERSECT' | 'EXCEPT' ] [ 'ALL' | 'DISTINCT' ]
<select> ]
[ 'ORDER BY' <expression> [ 'ASC' | 'DESC' ] [ 'USING' <op> ] ]
[ 'NULLS' { 'FIRST' | 'LAST' } [, ...] ]
[ 'LIMIT' [ <count> | 'ALL' ] ]
[ 'OFFSET' <start> [ 'ROW' | 'ROWS' ] ]
);

<from> := <table> [ * ] [ [ 'AS' ] <alias> [ ( <col_alias>
[, ...] ) ]
[ 'LATERAL' ] ( <select> ) [ [ 'AS' ] <alias> [ ( <col_alias>
[, ...] ) ]
[ <with_query_name> [ [ 'AS' ] <alias> [ ( <col_alias>
[, ...] ) ]
[ <from_item> <join_type> <from_item> [ 'ON' <join_condition> ]
[ 'USING' [ <join_column> [, ...] ] ] [ 'AS' ] ( <select>
[ <values> | <insert> | <update> | <delete> | <merge> ]
[ 'USING' <cycle_path_col_name> ]
);

<with> := <name> [ ( <col_name> [, ...] ) ] [ 'AS' ] ( <select>
[ <values> | <insert> | <update> | <delete> | <merge> ]
[ 'USING' <cycle_path_col_name> ]
);

<data> := <table> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<database> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
<constraint> [ <column> [, ...] ]
<table> [ <column> [, ...] ]
<view> [ <column> [, ...] ]
<sequence> [ <column> [, ...] ]
<function> [ <column> [, ...] ]
<schema> [ <column> [, ...] ]
&lt
```

Georgiy Shevoroshkin**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(n), NTILE(n), LEAD(v,o), ROW_NUMBER()
```

INSERT

```
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, =
```

users (u)

id	name	id	uid	action
1	Alice	7	1	LOGIN
2	Bob	8	2	VIEW

INFO: FK **uid** in den Query-Resultaten unten aus Platzgründen ausgelassen

Cross Join

Liefert alle möglichen Kombinationen zweier Tabellen.

1	1	Alice	7	LOGIN
2	1	Alice	8	VIEW
3	1	Alice	9	LOGIN
4	2	Bob	7	LOGIN
5	2	Bob	8	VIEW
6	2	Bob	9	LOGIN

Union

«Verbindet» zwei SELECT's ohne Duplikate.
Voraussetzung: Spalten müssen ähnliche Datentypen beinhalten

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

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
```

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)

Dreiwertige Logik (cursed)

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

INDEX

	B-Tree	Hash	BRIN	ISAM
Gleichheitsabfragen	✓	✓	✗	✓
Range Queries	✓	✗	✓	✗
Sortierte Daten	✓	✗	✓	✓
Grosse Tabellen	*	bei =	✓	✓
Häufig abfragen	✓	*	✓	✗
Direkter zugriff über PK	✓	✓	✗	*
Überlaufseiten	✓	✓	✗	✓

```
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

1 Alice	Read Uncommitted	Read Committed	Repeatable Read	Serizable
Dirty Write	*	*	*	✗
Dirty Read	✓	✗	✗	✗
Lost Update	✓	✓	✗	✗
Fuzzy Read	✓	✓	✗	✗
Phantom Read	✓	✓	✓	✗
Read Skew	✓	✓	✗	✗
Write Skew	✓	✓	✓	*
Dauerhaftigkeit	✓	✓	✗	✗
Atomizität	✗	✗	✓	✓

* 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

	Serialisierbar	Deadlocks	Cascading RollB.	Konflikt-RollB.	Hohe Parallelität	Realistisch
Two-Phase Locking	✓	✓	✓	*	✗	✗
Strict 2PL	✓	✓	*	✗	✗	✓
Preclaiming 2PL	✓	*	✗	✗	✗	✗
Validation-based	✓	*	✓	✓	✓	✓
Timestamp-based	*	*	✓	✓	✓	✓
Snapshot Isolation	✗	*	✗	✓	✓	✓
SSI	✓	*	✗	✓	✓	✓

* Deadlock in PSQL mit Snapshot Isolation

SQL Beispiel

```
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;
```

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

;

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.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

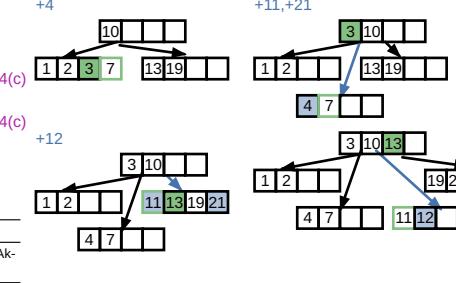
JOIN pferde p ON p.zuechternr = s.zuechternr

WHERE EXISTS (

SELECT vaternr FROM pferde p2

WHERE p2.pnr = p.vaternr AND p2.name = 'Hermes'

);

B-Baum**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, mehrere 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, TailD, PageID, Redo, Undo, PrevLSN**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,

);