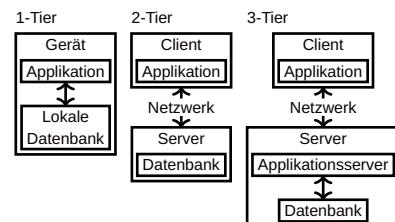
**Glossar**

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)**

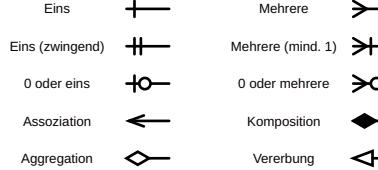
Besteht aus DBMS und Datenbasen

 **DataBase Management System (DBMS)**

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

**ANSI Modell****Logische Ebene:** Logische Struktur der Daten**Interne Ebene:** Speicherstrukturen, Definition durch interne 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 tabellenname (

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**TODo: better examples**

id	full_name	⇒	id	first	last
1	First Last	⇒	1	First	Last

2NF: Nichtschlüsselattr. voll vom Schlüssel abhängig. Ist PK atomar, dann 2NF gegeben

track	title	cd_id	album
1	Turnover	1	Repeater
2	Repeater	1	Repeater

track	cd_id	title	cd	id	album
1	1	Turnover	1	Repeater	
2	1	Repeater	1	Repeater	

**3NF: Keine transitiven Abhängigkeiten**

id	album	interpret	land
1	Repeater	Fugazi	USA
2	Red Medicine	Fugazi	USA

id	album	interpret	cd	id	name	land
1	Repeater	1	1	Fugazi	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_element>
[, ...] ]
[ 'HAVING' <condition> ]
[ 'WINDOW' <window_name> 'AS' [ <>window_definition> ]
[, ...] ]
[ 'UNION' | 'INTERSECT' | 'EXCEPT' ] [ 'ALL' | 'DISTINCT' ]
<select> ]
[ 'ORDER BY' <expression> [ 'ASC' | 'DESC' | 'USING'
<operator> ] [ 'NULLS' { 'FIRST' | 'LAST' } ], ... ]
[ 'LIMIT' { <count> } [ 'ALL' ] ]
[ 'OFFSET' { <start> } [ 'ROW' | 'ROWS' ] ]
<from_item> := <table_name> [ * ] [ [ 'AS' ] <alias> ]
[ <column_alias> [, ...] ]
[ 'LATERAL' ] [ <select> ] [ [ 'AS' ] <alias> ]
[ <column_alias> [, ...] ]
<with_query> := <with_query_name> [ [ <column_name> [, ...] ] ]
[ 'AS' ] [ <select> | <values> | <insert> | <update> | <delete> |
<merge> ]
[ 'USING' <cycle_path_col_name> ]
<join_type> <from_item> { 'ON' } <join_condition> |
'USING' [ <join_column> [, ...] ] [ 'AS' ] <join_using_alias> ]
<from_item> 'CROSS JOIN' <from_item>
<table> <view> <sequence> <database> <function> <schema>

```

**Data Control Language (DCL)**

GRANT kann angewendet werden auf:

```

TABLE COLUMN VIEW SEQUENCE DATABASE FUNCTION SCHEMA
Falls WITH GRANT OPTION: Der Berechtigte kann den Zugriff anderen Usern verteilen. → REVOKE ... CASCADE;
CREATE ROLE u WITH LOGIN PASSWORD '' -- user
GRANT INSERT ON TABLE t TO u WITH GRANT OPTION;
ALTER ROLE u CREATEROLE, CREATEDB, INHERIT;
CREATE ROLE r; -- group
GRANT r TO u; -- put user u in group r
REVOKE CREATE ON SCHEMA s FROM r;
CREATE ROLE u PASSWORD '' IN ROLE r; -- equivalent

```

**Read-only user**

```

-- creating
REVOKE CREATE ON SCHEMA public FROM PUBLIC;
CREATE ROLE u WITH LOGIN ENCRYPTED PASSWORD ''
NOINHERIT; -- don't inherit privileges
GRANT SELECT ON ALL TABLES IN SCHEMA public TO u;
-- read all new tables (also created by others):
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT
SELECT ON TABLES TO u;
-- deleting
REVOKE SELECT ON ALL TABLES IN SCHEMA public FROM u;
ALTER DEFAULT PRIVILEGES IN SCHEMA public
REVOKE SELECT ON TABLES FROM u;
DROP USER u;

```

**Row-Level Security (RLS)**

```

CREATE TABLE exams (
    id SERIAL, -- other fields...
    teacher VARCHAR(60) DEFAULT current_user
);
CREATE POLICY teachers_see_own_exams ON exams
FOR ALL TO PUBLIC USING (teacher = current_user);
ALTER TABLE exams ENABLE ROW LEVEL SECURITY;

```

**Data Definition Language (DDL)**

```

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;

```

**Vererbung****Tabelle pro Sub- und Superklasse:**

```

CREATE TABLE sup ( -- 3.a
    id SERIAL PRIMARY KEY,
    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);
ALTER TABLE sub2 ADD CONSTRAINT id FOREIGN KEY
REFERENCES sup (id);

```

**Tabelle pro Subklasse:** Enthält jeweil. Subklassattribute

```

CREATE TABLE sub1 ( -- 3.b
    id SERIAL PRIMARY KEY,
    name TEXT UNIQUE,
    age INT
);
CREATE TABLE sub2 (
    id SERIAL PRIMARY KEY,
    name TEXT UNIQUE
);

```

**Einzige Tabelle für Superklasse:** Enthält alle Attribute

```

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

```

**Junction Tabellen**

```

CREATE TABLE a_b(
    a: INTEGER REFERENCES a(id),
    b: INTEGER REFERENCES b(id),
    PRIMARY KEY(a, b)
);

```

**Datentypen**

Type	Description
INTEGER/INT	Integer (4 bytes)
BIGINT	Large integer (8 bytes)
SMALLINT	Small integer (2 bytes)
REAL	Single precision float (4 bytes)
Numeric(precision, scale)	Exact numeric of selectable precision Alias for DECIMAL(precision, scale)
DOUBLE PRECISION	Double precision float (8 bytes)
SERIAL	Auto-incrementing integer (4 bytes)
BIGSERIAL	Auto-incrementing large integer (8 bytes)
SMALLSERIAL	Auto-incrementing small integer (2 bytes)
CHARACTER/CHAR(size)	Fixed-length, blank-padded string
VARCHAR(size)	Variable-length, non-blank-padded string
TEXT	Variable-length character string
BOOLEAN	Logical Boolean (true/false)
DATE	Calendar date (year, month, day)
TIME	Time of day (no time zone)
TIMESTAMP	Date and time (no time zone)
TIMESTAMP WITH TIME ZONE	Date and time with time zone
INTERVAL	Time interval
JSON	JSON data
UUID	Universally unique identifier
ARRAY OF base_type	Array of values

```

NUMERIC(4, 2) /* 99.99 */ / NUMERIC(2, 1) /* 9.9 */
VARCHAR(5) /* 'abcde' */ / CHAR(5) /* 'abcde' */

```

**Casting****Explizit**

```

CAST(5 AS float8) = 5::float8
SELECT 'ABCDEFG'::NUMERIC; -- error
SELECT SAFE_CAST('ABCDEFG' AS NUMERIC); -- NULL

```

**Implizit**

```

SELECT 5 + 3.2; -- 5 is cast to 5.0 (numeric)
SELECT 'Number' || 42; -- 42 is cast to '42'
SELECT true AND 1; -- 1 is treated as true
SELECT CURRENT_TIMESTAMP + INTERVAL '1 day'; -- CURRENT_TIMESTAMP to date
SELECT '100'::text + 1; -- '100' is cast to 100

```

**Views**

Resultate werden jedes mal dynamisch queriert

```

CREATE VIEW v (id, u) AS SELECT id, u FROM t;
-- complex query
CREATE VIEW cheap_restaurant_view AS
WITH big_restaurant AS (
    SELECT * FROM restaurant
    WHERE anzahl_plaetze > 20
)
SELECT r.name AS restaurant_name, s.name,
MIN(g.preis) AS cheap_gericht
FROM big_restaurant r
LEFT JOIN skigebiet s ON (r.skigebiet_id = s.id)
LEFT JOIN menukarte m ON (r.id = m.restaurant_id)
LEFT JOIN menu_gericht mg ON (m.id = mg.menu_id)
LEFT JOIN gericht g ON (g.id = mg.gericht_id)
WHERE ist_tagesmenue = true
GROUP BY r.id, s.id, restaurant_name
HAVING MIN(g.preis) > 3
ORDER BY cheap_gericht;

```

**Updatable View**Views sind updatable wenn diese Kriterien erfüllt sind:

- Single base table
- Keine aggregate, DISTINCT, GROUP BY, oder HAVING Klauseln
- Alle Spalten müssen zur originalen Tabelle direkt gemappt werden können

**Materialized View**

Speichert resultat auf Disk

```

CREATE MATERIALIZED VIEW mv AS SELECT * FROM t;
REFRESH MATERIALIZED VIEW mv; -- refresh results

```

**Temporäre Tabellen**

```

CREATE TEMPORARY TABLE temp_products (
    id SERIAL PRIMARY KEY,
    product_name TEXT
);

```

```

INSERT INTO temp_products (product_name) VALUES
('Product A'), ('Product B'), ('Product C');
SELECT ts.product_name, ts.quantity FROM
temp_sales ts JOIN temp_products tp ON
ts.product_name = tp.product_name;

```

**Data Manipulation Language (DML)**

```

FROM -> JOIN -> WHERE -> GROUP BY -> HAVING ->
SELECT (WINDOW FUNCTIONS) -> ORDER BY -> LIMIT

```

**Common Table Expressions (CTE)**

- Erlauben die zeilenweise Ausgabe
- Erlauben Abfragen quasi als Parameter
- Können rekursiv sein

```

-- normal
WITH cte AS (SELECT * FROM t) SELECT * FROM cte;
WITH tmp(id, name) AS (SELECT id, name FROM t)
    SELECT id, name FROM tmptable;
-- recursive
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;

```

**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(v,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
		9	4	LOGIN

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

**Inner Join**

Zeilen, die in beiden Tabellen  
matchen

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

**Equi Join**

Wie Inner Join

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

**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;**

**TODO:****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
4			9	LOGIN

```
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		
3			8	VIEW
4			9	LOGIN

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

**Cross Join**

Liefert alle möglichen Kombinationen zweier Tabellen.

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

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;
```

1	Alice
2	Bob
3	LOGIN
4	VIEW

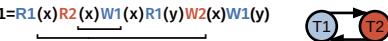
**READ UNCOMMITTED:** Lesezugriffe nicht synchronisiert (keine Read-lock), Read ignoriert jegliche Sperren

**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

**Serieller Scheduler:** Führt Transaktionen am Stück aus  
Nicht serialisierbar:

**Konfliktpaare:**

R1(x) < W2(x)      R2(x) < W1(x)

**Konflikt-Serialisierbar:**

r1(b)2(b)w2(b)2(c)r2(d)w3(a)r4(d)3(b)w4(d)r5(c)r5(a)w4(c)

**Konflikt-Äquivalenter serieller Scheduler:**

r1(b)r2(b)w2(b)2(c)r2(d)w3(a)r3(b)r5(c)r5(a)r4(d)w4(d)w4(c)

Begriff	Bedeutung
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 regxit 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 sich selbst «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.

**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 aufgetreten → 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, PagId, Redo, Undo, PrevLSN****SQL Beispiele**

```
CREATE TABLE pferd (
  pnr SERIAL PRIMARY KEY,
  name TEXT,
  alter INT,
  zuechternr INT REFERENCES stall.pk,
  veternnr 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.veternnr
  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.veternnr
WHERE p2.name = 'Hermes'
```

--

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

--

**B-Baum**

