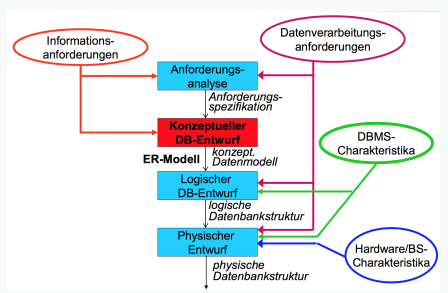


### DB Entwurfsprozess



### Abbildungsregeln

- 1..\* PK von 1 ist FK von \*
- n:m Beziehungstabelle mit zusammengesetztem PK (FK der beiden Tabellen)

### ANSI-3 Ebenenmodell

- externen Ebene: Sicht einer Benutzerklasse auf eine Teilmenge der Datenbank
- konzeptionelle Ebene: logische Struktur der Daten
- interne Ebene: Speicherungsstrukturen

Vorteile: einzelne Bereiche umstrukturieren, ohne Auswirkungen auf restliche Teile

### DDL (Data Definition)

```

Variante 1: Bedingungen formuliert als Column-Constraints
CREATE TABLE Projekt (
  ProjektID NUMBER(4) PRIMARY KEY,
  Bezeichnung VARCHAR2(20) NOT NULL UNIQUE,
  StartDate DATE NOT NULL,
  Status VARCHAR2(10) NOT NULL,
  Projektleiter VARCHAR2(30) REFERENCES Angestellter(PersonID)
);

Variante 2: Bedingungen formuliert als Table-Constraints
CREATE TABLE Projekt (
  ProjektID NUMBER(4),
  Bezeichnung VARCHAR2(20) NOT NULL UNIQUE,
  StartDate DATE NOT NULL,
  Status VARCHAR2(10),
  Projektleiter VARCHAR2(30),
  CONSTRAINT PK_Projekt PRIMARY KEY (ProjektID),
  CONSTRAINT FK_Projekt FOREIGN KEY (Projektleiter) REFERENCES Angestellter(PersonID)
);
    
```

CREATE database/view | ALTER, DROP | ...references a(a) ON DELETE CASCADE/SET NULL

### DML (Data Manipulation)

```

DELETE FROM angestellter WHERE persNr=1100;
INSERT INTO Abteilung (Name, AbtrNr) VALUES ('Entwicklung', 20);
UPDATE test SET name = "a" WHERE id=1;
    
```

### DQL // JOINS

```

SELECT PZ.PersNr, proj.bezeichnung, Zeitanteil, ang.Name FROM projektZuteilung AS PZ
INNER JOIN angestellter AS ang ON ang.PersNr = PZ.PersNr
INNER JOIN projekt AS proj ON PZ.ProjNr=proj.ProjNr
WHERE (PZ.ProjNr=25) OR (PZ.ProjNr=30)
ORDER BY proj.ProjNr, ang.Name;
    
```

### DQL // Subqueries

```

SELECT Name FROM Angestellter A
WHERE EXISTS (
  SELECT * FROM ProjektZuteilung
  WHERE PersNr = A.PersNr );
SELECT ang.Name, Salaer FROM Angestellter as ang INNER JOIN Abteilung as abt ON abt.AbtrNr=ang.AbtrNr WHERE abt.Name='Entwicklung' AND Salaer = (
  SELECT MIN (Salaer) FROM Angestellter as ang INNER JOIN Abteilung as abt ON abt.AbtrNr=ang.AbtrNr WHERE abt.Name='Entwicklung' );
    
```

korreliert = subquery ist abhängig, i.e. nicht alleine ausführbar  
 unkorreliert = subquery unabhängig

### DQL // Aggregatfunktionen

```

SELECT MAX( Salaer ) FROM Angestellter;
SELECT MIN( Salaer ) FROM Angestellter;
SELECT AVG( Salaer ) FROM Angestellter;
SELECT SUM( Salaer ) AS "Salaer-summe" FROM Angestellter;
SELECT name, COUNT(projnr) from projektzuteilung inner join angestellter on projektzuteilung.persnr = angestellter.persnr group by name;
    
```

zur einschränkung nicht WHERE sondern **HAVING!**  
 nur attribute in group by können verwendet werden.  
 ..having count(\*) < 2; ..having name like 'M%'

### Data Control Language (DCL)

```

CREATE ROLE user WITH LOGIN PASSWORD 'pw';
ALTER ROLE user CREATEROLE, CREATEDB;
DROP ROLE AngProj;
    
```

### Index

- Index-Sequential Access Method (ISAM)
  - Daten über Indexspalten asc sortiert
  - + Einfügen/Suchen: schnell
  - aktualisieren: schlecht
- B-Baum (Balanced)
  - für grosse Datenmengen

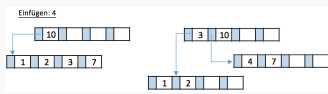
Heap (=Java Linked List)  
 Suchbaum (=Java Tree)

```

CREATE INDEX <IndexName> ON <Table>(attr);
    
```

Index lohnt sich für : Schlüssel, Häufiger Vergl. mit Konstanten (zb Jahr = 2000)

### B-Tree einfügen



### Window Functions

```
//alle namen mit salär&differenz zum nächsten
```

```
SELECT name, salaer, (salaer - lead(salaer, 1) OVER(ORDER BY salaer desc)) AS "differenz" FROM angestellter ORDER BY 2 DESC LIMIT 5;
```

```
//alle Vor-&Nachnamen mit ihrer Anzahl
```

```
SELECT nachname, vorname, COUNT(*) OVER (PARTITION BY vorname) AS Anzahl FROM person;
```

Funktionen, die auf ein „Daten-Fenster“ (d.h. umgebende Tupel bezogen auf die aktuelle Zeile) angewendet werden. Ähnlich wie AggregatsF. aber Zeilen behalten separaten Informationsgehalt

### Common Table Expressions (CTE)

```
WITH angestelltemitprojekten AS (
    SELECT a.name, proj.bezeichnung
    FROM angestellter a
    JOIN projzut pz ON a.persnr=pz.persnr
    JOIN projekt proj ON pz.projnr=proj.projnr
)
SELECT * FROM angestelltemitprojekten;
```

CTE's ("WITH" Queries) ermöglichen Definition von Hilfs-Queries in bzw. vor einer grösseren Query

### Views

```
CREATE VIEW AngPublic (Persnr, Name, Tel, Wohnort) AS
SELECT Persnr, Name, Tel, Wohnort FROM Angestellter;
SELECT * FROM AngPublic ORDER BY Name;
DROP VIEW AngPublic;
```

virtuelle Tabelle, basierend auf anderen Tabellen/Views.  
werden mit Select-Anweisung definiert  
Nutzen: Datenkapselung, Benutzeranpassung(Vereinfachung), Datenschutz

### Transactions

```
BEGIN ISOLATION LEVEL
SERIALIZABLE;
SAVEPOINT one;
ROLLBACK TO one;
COMMIT;
```

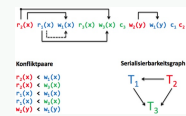
Isolationslvl:  
Read Uncommitted = read nicht synch.  
Read Committed = read nur kurz synch.  
Repeatable Read = zugedr. rows sind synch.  
Serializable = vollstä. Isolation

### Isolationsfehler

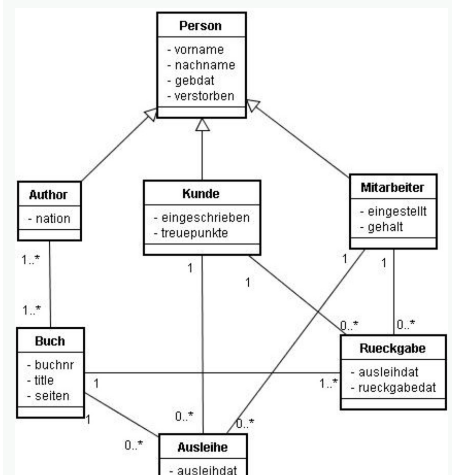
Isolation Level	Dirty Read	Fuzzy Read	Phantom Read
READ UNCOMMITTED	Möglich	Möglich	Möglich
READ COMMITTED	Nicht möglich	Möglich	Möglich
REPEATABLE READ	Nicht möglich	Nicht möglich	Möglich
SERIALIZABLE	Nicht möglich	Nicht möglich	Nicht möglich

Dirty = Lese Daten von anderer nicht committed Transaktion  
Fuzzy = Lese gleiche Daten mehrmals --> andere Werte  
Phantom = Select entdeckte plötzlich neue/gelöschte Rows

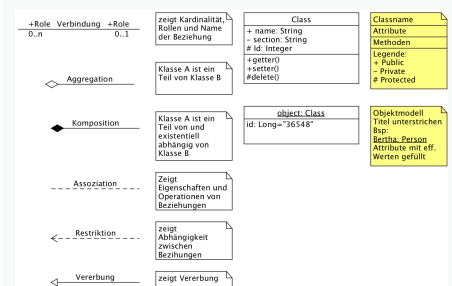
### Serialisierbarkeitsgraph



### BibliotheksDB



### UML



### Locking

2-Phase Locking: Sobald die Transaktion ein Lock freigegeben hat, darf sie keine weiteren Locks beziehen  
Strict 2-Phase Locking: Alle gehaltenen Sperrern werden erst nach Ende der Transaktion freigegeben



### Locking (cont)

xlock Exclusive Lock für schreibe-/lesezugriff

slock Shared Lock für lesezugriff

wenn slock(x) vergeben, muss andere Transaktion mit xlock(x) warten

### Anforderungen Datenbank

Redundanzfreiheit

Datenintegrität Datenkonsistenz

Datensicherheit

Datenschutz

Datenunabhängigkeit

DBMS Funktionen: Transaktionen, Mehrbenutzerbetrieb, Sicherheit, Backup/Recovery

### ACID-Kriterien

<b>A</b>	Atomicity	Eine Transaktion wird entweder vollständig oder gar nicht ausgeführt wird
<b>C</b>	Consistency	Eine Transaktion führt die Daten von einem konsistenten Zustand in einen anderen über
<b>I</b>	Isolation	Eine Transaktion soll so ausgeführt werden, als sei sie von anderen isoliert
<b>D</b>	Durability	Alle Änderungen einer Transaktion sind persistent und gehen nicht durch Fehler verloren

Voraussetzung für Verlässlichkeit von Systemen und Transaktionen

### Relationale Schreibweise

Tabellenname (id INTEGER PK, name TEXT(20) NOT NULL, abteilung NOT NULL REFERENCES abteilung);

PK attribute unterstreichen

FK attribute kursiv oder gestrichelt unterstreichen

### Normalformen

1.NF Attributwerte atomar

2. NF Nichtschlüsselattribut von Schlüssel voll funktional abhängig  
Attribut muss vom ganzen Schlüssel abhängen nicht nur von Teilen  
i.e. PK aus 1 Attribut --> immer 2.NF

{Autor} -> Adresse von Autor funktional abhängig, lesen: "bestimmt eindeutig"

3. NF kein Nichtschlüsselattribut von Schlüssel transitiv abhängig

Nutzen: Redundanzen erkennen & Anomalien (Einfüge-, Lösch-, Änderungs-) verhindern

Determinante: min. Attributmenge, von der andere Attr. funktional abhängen  
zb ISBN | Ausleiher | Autor --> Determinante ISBN

### NF Bsp

```

A -> B A -> C A -> D
A -> B, C A -> D
Schlüsselattribut A, B, C, D
Nichtschlüsselattribut B, C, D
transitiv: A -> B, C, D

```

### Relationale Algebra

o	Selektion	Wählt gesuchte Zeilen/Tupel aus	R[alter>18]																
π	Projektion	Wählt gesuchte Spalten/Attribute aus <th>R[name]</th>	R[name]																
⋈	Natural Join	Verbindet angegebene Tabellen automatisch über gleichnamige rows	<table border="1"> <tr> <td>1</td> <td>2</td> <td>3</td> <td>4</td> </tr> <tr> <td>1</td> <td>2</td> <td>3</td> <td>4</td> </tr> <tr> <td>1</td> <td>2</td> <td>3</td> <td>4</td> </tr> <tr> <td>1</td> <td>2</td> <td>3</td> <td>4</td> </tr> </table>	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1	2	3	4																
1	2	3	4																
1	2	3	4																
1	2	3	4																

### JDBC

```

try (Connection connection = DriverManager.getConnection(
    "jdbc:postgresql:mydb", username, password)) {
    try (Statement statement = connection.createStatement()) {
        ResultSet resultSet = statement.executeQuery("SELECT * FROM Account");
        while (resultSet.next()) {
            String owner = resultSet.getString("Owner");
            int balance = resultSet.getInt("Balance");
            System.out.println(owner + "\t" + balance);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

### JDBC Isolationslevel

```

try {
    connection.setAutoCommit(false);
    connection.setTransactionIsolation(
        Connection.TRANSACTION_SERIALIZABLE);
    // Statement...
    connection.commit();
} catch (SQLException exception) {
    connection.rollback();
    throw exception;
}

```

### SQL Injections / Prepared Statement

```

void newAccount(String name) throws SQLException {
    getDBStatement().executeUpdate(
        "INSERT INTO Account VALUES('" + name + "', 0)");
}
}
Problem: User input -> "; DROP TABLE Account; --"
Lösung:
PreparedStatement statement = connection.prepareStatement(
    "UPDATE Account SET Balance = Balance + ? WHERE Owner = ?");
statement.setInt(1, 100);
statement.setString(2, "Bob");
statement.execute();
}

```

Problemlösung: 1) prepared statements verwenden 2) Benutzerrechte möglichst einschränken 3) SQL steuerzeichen escapen (aus ' ; --> \' \ ;)

### JDBC update

```

void transfer(Connection connection,
    String from, String to, int amount) throws SQLException {
    try (Statement statement = connection.createStatement()) {
        statement.executeUpdate(
            "UPDATE Account SET Balance = " + amount +
            " WHERE Owner = '" + from + "'");
        statement.executeUpdate(
            "UPDATE Account SET Balance = " + amount +
            " WHERE Owner = '" + to + "'");
    }
}

```

### JDBC MetaData

```

ResultSet resultSet = statement.executeQuery(
    ("SELECT * FROM " + tableName));

ResultSetMetaData rsMeta = resultSet.getMetaData();
for (int col = 1; col <= rsMeta.getColumnCount(); col++) {
    String name = rsMeta.getColumnName(col);
    System.out.println(name + "\t");
}

```

connection.getMetaData() => DatabaseMetaData, gibt Infos über DB (Produktenamen, Driver, unterstützte Datentypen...)



By **tarinya**  
[cheatography.com/tarinya/](https://cheatography.com/tarinya/)

Published 31st January, 2016.  
Last updated 7th January, 2016.  
Page 3 of 3.

Sponsored by **Readable.com**  
Measure your website readability!  
<https://readable.com>