# Datenbanken

## 1. SQL:

- **Key-Konstrukt: „*SELECT-FROM-WHERE*“-Klausel:**

| | |
|---|---|
| **SELECT <Distinct> <Attribute/s>** | Wählen Sie einen Satz von Attributen, die im Abfrageergebnis angezeigt werden sollen.<br>Distinct: Duplikate entfernen<br>* (als Attribute): Auswahl aller Datensätze |
| **FROM <Table/s>** | Wählen Sie eine Tabelle, aus der Zeilen entnommen werden sollen |
| **WHERE <Condition/s>** *(optional)* | Zusätzliche Bedingungen, die die Zeilen erfüllen müssen, damit sie im Ergebnis erscheinen. |
| **ORDER BY <Attribute> <asc/desc>** | Sortierung: asc – aufsteigend, desc - absteigend |

- **Aggregatfunktionen:** MIN(), MAX(), SUM(), AVG()
- **Gruppierungen:** GROUP BY <Attribute>
- **Relationen:**
  - **Erstellen:** CREATE [TABLE/VIEW] <name> (<spalte>[,spalte]*) wobei <spalte>::=<attribut-name> <typ> [NOT NULL][PRIMARY KEY/FOREIGN KEY][references <table>]
  - **Attr. Hinzufügen:** ALTER TABLE <name> ADD <attribut-name> <typ>
  - **Löschen:** DROP TABLE <name>
- **Datenoperationen:**

| Einfügen | Löschen | Aktualisieren |
|---|---|---|
| **INSERT: Einfügen von Tupeln**<br>`INSERT INTO <Relationen-Name>[(<Attributname> [, <Attributname>]*)]`<br>`VALUES (<Konstante> [, Konstante]*)`<br>oder<br>`INSERT INTO <Relationen-Name>[(<Attributname> [, <Attributname>]*)]`<br>`SELECT ... FROM ... WHERE ...`<br>▪ Spaltenliste ist optional, wenn alle Spalten angegeben werden | **DELETE: Löschen von Tuplen**<br>`DELETE`<br>`FROM <Relationen-Name>`<br>`[WHERE <Bedingung>]` | **UPDATE: Verändern von Tupeln**<br>`UPDATE <Relationen-Name>`<br>`SET <Attributname> = <Ausdruck> [, <Attributname> = <Ausdruck>]*`<br>`[WHERE <Bedingung>]` |

- **Änderungsverhalten:**
  - Veränderungen des zugehörigen Primärschlüssels werden verbunden
  - Durch „set null“ wird Wert der Verbindung auf null gesetzt
  - **...on update/delete cascade [set null]**
- **Statische Integritätsbedingungen:**
  - 
```
CREATE TABLE Auftrag
            (KName  CHAR(20) references Kunde,
             Ware   VARCHAR(50),
             MENGE  INTEGER check between 1 and 100,
             PRIMARY KEY (Kname, Ware))
```
**check**
- **Aktive Datenbanktechnologie:**
  - **Mittels Triggers:** ECA Prinzip → Event, Codition, Action

**Anlegen eines Triggers**

```
CREATE TRIGGER <trigger-name>
BEFORE | AFTER | INSTEAD OF
INSERT | DELETE | UPDATE [OF (<column>, …)]
ON <table-name> or <view-name>
REFERENCING [OLD AS <name>] [NEW AS <name>]
            [OLD_TABLE AS <name>] [NEW_TABLE AS <name>]]
FOR EACH ROW | FOR EACH STATEMENT
[WHEN (<condition>)]
BEGIN ATOMIC
 <sql-stmt> | <dynamic-compound-sql-stmt>
END
```

Auslösezeitpunkt
Auslöser
Übergangsvariablen
Granularität
Bedingung
Aktion

**Entfernen von Triggern**

- `DROP TRIGGER <trigger-name>`

- **Änderungsanomalien:** INSERT, UPDATE, DELETE

# 2. Modelle:

## • Entity Relation Model (ER Model):

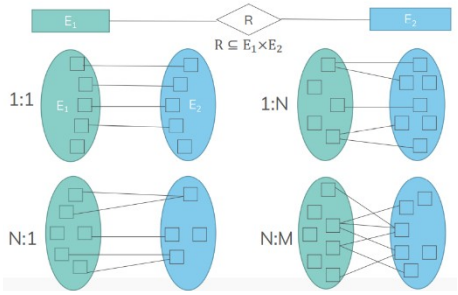
- Unterstrichene Attribute = Schlüsselattribute

### Komponenten

durch

| | Entität | Beziehung | Attribut |
|---|---|---|---|
| Set (Schema) | Menge gleichartiger Objekte | Beziehungs-möglichkeit | Merkmal, Eigenschaft |
| Instanz | bestimmtes Objekt | konkrete Beziehung zwischen zwei Objekten | Merkmal eines Objekts |

### Funktionalitäten

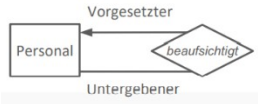



*Gelesen, bspw: „N zu 1“*

- **Erweitertes Model:**
  - **Min, Max Notation:**
    1 Zug besitzt mindestens 1 und höchstens 10
    Wagen. 1 Wagen gehört zu mindestens 1 Zug
    und zu höchstens 1 Zug.


  - **Identifikationsabhängigkeit:**
    Entitäten/Beziehungen, die ohne deren Entitäten nicht existieren kann.

  - **Rollen:**
    Falls ein Entitätstyp mehrmals in einem Beziehungstyp beteiligt ist, wird durch einen Namen angegeben, in welcher Funktion die Entitätstyp jeweils vorkommt.



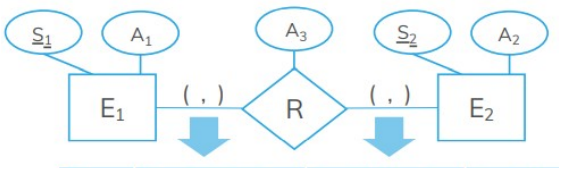

  - **Vererbung:**
    Subklasse/Untertypen: Eigenschaften der Oberklasse & zus. weitere unterschiedliche Attribute und Beziehungen

## • Relationen Modell:

- Mit minimalen Primärschlüssel
- **Vergleich:**

| ER-Modell | Relationales Modell |
|---|---|
| Besitzt 2 Strukturierungskonzepte:<br>▪ Entitätstypen und  <br>▪ Beziehungstypen  | Besitzt nur 1 Strukturierungskonzept: |

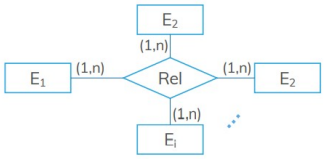
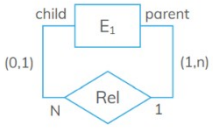
- **ER-Modell → Relationen Modell:**
  1. Übersetzung von Entitäten:
     - Eins-zu-Eins Übersetzung in Relationen
  2. Übersetzung von Attributen:
     - Einfache Attribute:
       - ➢ Direkte Abbildung auf Attribute im Relationenmodell
     - Zusammengesetzte Attribute (z.B.: Adresse):
       - ➢ Komponentenattribute als unabhängige Attribute
       - ➢ Ein Attribut, das die gesamte Information enthält
     - Berechnete Attribute (z.B.: Umsatz = Preis*Verkauf):
       - ➢ Abspeicherung in der Relation und Aktualisierung durch Trigger
     - Mehrwertige Attribute (z.B.: Telefonnummer) :
       - ➢ Zusätzliche Relation mit Fremdschlüssel auf ursprüngliche Relation



**Zusammenfassung Übersetzung**

| Typ | Restriktion $E_1$ | Restriktion $E_2$ | Resultierendes Relationenschema |
|---|---|---|---|
| **1:1** | **(1,1)** | **(1,1)** | **1 Relation mit Attributen von $E_1, E_2 \wedge R$.<br>Primärschlüssel ist Schlüssel von $E_1$ oder $E_2$** |
| | (0,1) | (1,1) | 2 Relationen $E_1$ & $E_2$, wobei<br>- $E_2$ Primärschlüssel von $E_1$ als Fremdschlüssel & alle Attribute von R |
| | (0,1) | (0,1) | 2 Relationen $E_1$ & $E_2$, wobei<br>- $E_1$ Primärschlüssel von $E_2$ als Fremdschlüssel<br>- $E_2$ Primärschlüssel von $E_1$ als Fremdschlüssel & alle Attribute v. R |
| **1:N** | **(0,N)** | **(0,1)** | **2 Relationen $E_1$ & $E_2$, wobei<br>- $E_2$ Primärschlüssel von $E_1$ als Fremdschlüssel & alle Attribute v. R** |
| | (1,N) | (0,1) | 2 Relationen $E_1$ & $E_2$, wobei<br>- $E_1$ Primärschlüssel von $E_2$ als Fremdschlüssel<br>- $E_2$ Primärschlüssel von $E_1$ als Fremdschlüssel & alle Attribute v. R |
| | (0,N) | (1,1) | 2 Relationen $E_1$ & $E_2$, wobei<br>- $E_2$ Primärschlüssel von $E_1$ als Fremdschlüssel & alle Attribute v. R |
| **N:M** | **(0,M)** | **(0,N)** | **3 Relationen $E_1$, $E_2$ & R, wobei<br>- $R$ zus. Primär-schlüssel von $E_1$, $E_2$** |
| | (1,M) | (0,N) | 3 Relationen $E_1$, $E_2$ & R, wobei<br>- $E_1$ Primärschlüssel von $E_2$ als Fremdschlüssel<br>- $R$ Primär-schlüssel von $E_1$, $E_2$ |

## Spezialfälle

| Beziehungen zwischen mehr als zwei Relationen | Rekursiver Beziehung |
|---|---|
| R enth. Primärschlüssel von A&B $(A \times B \rightarrow C)$ und alle Schlüssel von C als Fremdschlüssel   | R enth. Primärschlüssel von E zweifach mit untersch. Namen   |

- **Universalrelation:** Eine Relation mit allen Attributen, dadurch u.a. mit NULL-Werten
- **Horizontale Partitionierung:** Jede Entität wird eine separate Relation inkl. seinen Attributen
- **Vertikale Partitionierung:** Entitäten werden zu Relationen mit Verknüpfung dieser unterein.

  Vererbungshierarchie entfällt → Einsatz v. Fremdschlüsseln

# 3. Relationale Algebra:

- **Basisoperationen:**
  - **Projektion -** $\pi_{A'}(R) = \{\pi_{A'}(R) \mid t \in R\}$**:**
    - ➢ Auswahl bestimmter Spalten aus Relation (R) und Ausgabe dessen als neue Relation, ohne Duplikate
    - **SQL:** „DISTINCT“
    - **Bsp:** $\pi_{Name,Ort}(Studenten)$ -> alle Attr. Name,Ort aus Tabelle Studenten
  - **Selektion -** $\sigma_P(R) = \{t \mid t \in R \wedge P(t)\}$**:**
    - ➢ Auswahl bestimmter Zeilen aus Relation (R) und Ausgabe dessen als neue Relation, ohne Duplikate
    - **SQL:** „WHERE“
    - Zusammensetzung P:
      -> Operanden: Konstanten oder Name eines Attributs
      -> Vergleichsoperatoren: ¿, ≠, <, ≤, >, ≥
      -> Boolesche Operatoren: ∧ , ∨ , ¬
    - **Bsp:** $\sigma_{Name='Schmidt'}(Studenten)$
  - **Umbenennung:** $\rho_{¿\,alias_{name}\,of\,R>¿(R)¿}$
  - **Division:** $R/S := \pi_{A-B}(R) - \pi_{A-B}\left(\left(\pi_{A-B}(R) \times S\right) - R\right)$
  - **Verbund (Join):**
    - ➢ Auswahl bestimmter Tupel aus dem kartesischen Produkt RxS
    - Gibt es kein gemeinsames Attribut, so ist das Ergebnis das kartesische Produkt
    - Bsp. „Left Outer Join“:
      Alle Tupel der linken Relation R, die keinen Join-Partner in der rechten Relation S haben, werden trotzdem ausgegeben, bekommen aber entspr. null
    - Bsp. „Full Outer Join“:
      Ausgegeben werden alle Tupel von R & S, die die keinen Join-Partner haben, erhalten entspr. Die Werte null

- Bsp. „Semi Join“: Enthält alle Tupel der Relation R, die die Join-Bedingung mit S erfüll.

## Operationen

Dresden Database Research Group

### Basisoperationen

- Projektion

$$\pi_{A'}(t) = (A'_1(t), \ldots, A'_m(t))$$

- Selektion

$$\sigma_P(R) = \{t | t \epsilon R \wedge P(t)\}$$

- Kartesische Produkt

$R \times S = \{(a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m) \mid (a_1, a_2, \ldots, a_n) \in R \wedge (b_1, b_2, \ldots, b_m) \in S\}$

- Mengenoperationen
  - Union $R \cup S := \{r \mid r \in R \text{ oder } r \in S\}$
  - Differenz $R - S := \{r \mid r \in R \text{ und } r \notin S\}$

### Abgeleitete Operationen

- Durchschnitt

$R \cap S := \{r \mid r \in R \text{ und } r \in S\}$

- Division

$$R \div S = \{t \mid \exists x, y \in R \; \forall y \in S\}$$

- Join
  - Natürlicher Verbund

$$R \bowtie S := \pi_{i_{k+1}, i_{k+2}, \ldots, i_{r+s}}(\sigma_{R.A1=S.B1 \wedge \ldots \wedge R.Ak=S.Bk}(R \times S))$$

  - Theta-Join

$$R \bowtie_{i \theta j} S := \sigma_{A_i \Theta B_j}(R \times S)$$

  - Halbverbund $R \ltimes S := \pi_R(R \bowtie S)$

# 4. Funktionale Abhängigkeiten:

- **Definitionen:**

**Definition: Funktionale Abhängigkeit (FD)**

- Seien A und B Attributmengen des Relationenschemas RS (A,B $\subseteq$ RS).
- B ist von A funktional abhängig geschrieben A→B gdw.
  Für alle real möglichen Relationen r(RS) zu jedem Wert in A genau ein Wert in B gehört:
  $\mathbf{A \rightarrow B \Leftrightarrow \forall t_1, t_2 \in r(RS) : t_1[A] = t_2[A] \Rightarrow t_1[B] = t_2[B]}$
- Sprechweisen
  - A bestimmt B (funktional)
  - A identifiziert oder impliziert B

➔ Zu jedem B gibt es genau ein A

**Definition: Volle / Partielle Funktionale Abhängigkeit**

- Eine funktionale Abhängigkeit X→Y heißt voll, wenn es keine echte Teilmenge Z ⊂ X gibt, so dass gilt: Z→Y (oder **∀A∈X: X-{A} → Y**)
  - Alle Attribute in X sind für die funktionale Abhängigkeit notwendig, es darf keines weggelassen werden
  - Die Determinate X ist also irreduzibel
- Gibt es eine solche Teilmenge, dann heißt X→Y partielle Abhängigkeit.

- **Superschlüssel:**
  Ist $A \subseteq R$, wenn gilt $A \rightarrow RS$
  → *Menge von Attributen, die die Datensätze einer Relation eindeutig identifiziert*
- **Kandidatenschlüssel:**
  $(\kappa \subseteq R) \rightarrow R$ *(Vollständigkeit)*, es gibt kein $\kappa' \subset \kappa$, sodass $\kappa' \rightarrow R$ *(Minimalität)*
  → ***Minimale*** *Menge von Attributen, die die Datensätze einer Relation eindeutig identifiziert*
- **Normalformen:**
  - **1. NF:** alle Attribute sind nur atomare Werte, die nicht weiter zerlegbar sind
  - **2. NF:** zus. Jedes Attribut ist von jedem KS voll funkt. abh.

**Bsp:**

| R.-Nr. | Datum | Name | Vorname | Straße | Hnr. | PLZ | Ort | Artikel | Anzahl | Preis | Währung |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 187 | 01.01.2012 | Mustermann | Max | Musterstr. | 1 | 12345 | Musterort | Bleistift | 5 | 1,00 | Euro |

1.NF

-> 2.NF:

**Rechnung**

| R.-Nr. | Datum | Knr. |
|---|---|---|
| 187 | 01.01.2012 | 007 |

**Kunde**

| Knr. | Name | Vorname | Straße | Hnr. | PLZ | Ort |
|---|---|---|---|---|---|---|
| 007 | Mustermann | Max | Musterstr. | 1 | 12345 | Musterort |

**Rechnungsposition**

| R.-P.-Nr. | R.-Nr. | Art.-Nr. | Anzahl |
|---|---|---|---|
| 1 | 187 | 69 | 5 |

**Artikel**

| Art.-Nr. | Artikel | Preis |
|---|---|---|
| 69 | Bleistift | 1,00 |

- **3. NF:** Wenn kein Nichtschlüsselattribut *(=Attribute, die sich nicht als Schlüssel für eine andere/neue Relation eignen)* vom KS abhängig ist.
  → D.h. Beseitigung von Nicht-Schlüsselattributen

| Kunde | | | | | | |
|---|---|---|---|---|---|---|
| Knr. | Name | Vorname | Straße | Hnr. | PLZ | Ort |
| 007 | Mustermann | Max | Musterstr. | 1 | 12345 | Musterort |

**Bsp:**

| Kunde | | | | | |
|---|---|---|---|---|---|
| Knr. | Name | Vorname | Straße | Hnr. | PLZ |
| 007 | Mustermann | Max | Musterstr. | 1 | 12345 |

| Postleitzahl | |
|---|---|
| PLZ | Ort |
| 12345 | Musterort |

2.NF ->
3.NF:

- **Berechnung Kandidatenschlüssel:**
  - Mittels Armstron-Axiome
  - **Armstron-Regeln:**

| **Reflexivität:** | **Vestärkung:** | **Transitivität:** |
|---|---|---|
| $Sei\, B \subseteq A \Longrightarrow A \to B$ | $A \to B \Longrightarrow A \vee C \to B \vee$ | $A \to B, B \to C \Longrightarrow A \to C$ |
| **Vereinigungsregel:** | **Dekomp.sregel:** | **Pseudoregel:** |
| $A \to B, A \to C \Longrightarrow A \to B \vee C$ | inv.Vereinig.reg. | $A \to B, B \vee C \to D \Longrightarrow A \vee C \to D$ |

- **Zerlegung eines Relationschemas:**
  - Relation wird in Teilrelationen gesplittet
  - **Verlustlosigkeit:**
    - Ausprägung der Ursprungsrelation muss rekonstruierbar sein
    - Es gilt: $R = \pi_{RS1}(R) \bowtie \pi_{RS2}(R) \rightarrow$ über Verbund rekonstruierbar
  - **Abhängigkeitserhaltung:**
    - Funkt. Abgängigkeiten der Ursprungsrelation muss Teilschemas übertragbar sein
    - Es gilt: $F_{RS}^{+} ¿= (F_{RS\,1} \vee \ldots \vee F_{RSn})^{+} ¿$

- **3NF-Synthesealgorithmus:**
  - zerlegt Schema verlustlos und abhängigkeitsbewahrend in 3.NF
  1. Kanonische Überdeckung berechnen

**Schritt 1: Linksreduktion**
- Führe für jede FD (A→B)∈F die Linksreduktion durch, indem für alle X∈A überprüft wird, ob das Attribut X überflüssig ist, d.h. ob gilt

B ⊂ Hülle(F, A – {X})

Ist dies der Fall, ersetze A→B durch A – {X}→B

**Schritt 2: Rechtsreduktion**
- Führe für jede verbliebene FD (A→B)∈F die Rechtsreduktion durch, indem für alle Y∈B überprüft wird, ob das Attribut Y überflüssig ist, d.h. ob gilt:

Y∈ Hülle(F – (A→B) ∪ (A→B – {Y}), A)

Ist dies der Fall, ersetze A→B durch A→B – {Y}

**Schritt 3: Entferne die FDs der Form**

A→∅

**Schritt 4: Ersetze alle FDs der Form**

$A \to B_1, A \to B_2, \ldots, A \to B_k$ durch $A \to B_1 \cup B_2 \cup \ldots \cup B_k$

2. Führe für jede FD (A→B)∈$F_C$ folgende Anweisungen aus:
   Erzeuge ein Relationenschema $R_A$ und ordne $R_A$ die FDs $F_A$ = {C→D∈$F_C$ | C∪D⊆$R_A$} zu.
3. Falls alle der in Schritt 2 erzeugten Schemata keinen Schlüsselkandidaten des ursprünglichen Schemas R enthalten, so erzeuge zusätzlich eine Relation mit dem Schema $R_K$ = K und $F_K$ = ∅, wobei K ein Schlüsselkandidat von R ist.
4. Eliminiere die Schemata $R_A$, die in einem anderen Schema enthalten sind.

# 5. Transaktionsverarbeitung:

- **Isolationssteuerung:**

**SQL-Anweisung zum Setzen der Konsistenzebene**

```
SET TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED| READ COMMITTED |
REPEATABLE READ | SERIALIZABLE }
```

| Isolation Level | Lost Update möglich? | Dirty Read möglich? | Unrepeatable Read möglich? | Phantom Read möglich? |
|---|---|---|---|---|
| Read Uncommitted | Nein | Ja | Ja | Ja |
| Read Committed | Nein | Nein | Ja | Ja |
| Repeatable Read | Nein | Nein | Nein | Ja |
| Serializable | Nein | Nein | Nein | Nein |

- **Operationen:**

| **BOT *(Begin of Transaction)*** | **Commit** | **Abort** |
|---|---|---|
| Kennzeichnet den Beginn einer Transaktion | Erfolgreiche Beendigung einer Transaktion & dauerhafte Einbringung aller Änderungen in die Datenbasis | Selbstabbruch der Transaktion und Zurücksetzen der Datenbasis in Zustand vor Beginn der Transaktion |

- **Eigenschaft – ACID:**

| **Atomarität** | **Konsistenzerhaltung** | **Isolation** | **Dauerhaftigkeit** |
|---|---|---|---|
| Transaktion wird entweder ganz oder garnicht ausgeführt | Erfolgreiche TA garantiert konsistente DB | Isolierter Ablauf von TAs | Persistierung der Ergebnisse einer erfolgr. TA |

- **Anomalien *(durch Ablauf gleichzeitiger TAs)*:**

| | |
|---|---|
| ▪ **Lost Updates:** Konkurrierendes Verändern eines Datenelementes<br>➔ Mehrere TAs verändern 1 Datenelement | ▪ **Non-repeatable read:** Während der Verarbeitung von T2 wird der Datenbestand durch T1 verändert |
| ▪ **Dirty Read:** Abhängigkeiten von nicht freigegebenen Änderungen *(-> Rollback/Abort)*<br>➔ TA 1 verändert Datenelement, welches noch nicht commited ist, während TA 2 darauf zugreift -> TA 2 liest noch alten Wert | ▪ **Phantom-Problem:** „Non-repeatable read“ auf höherer Ebene |

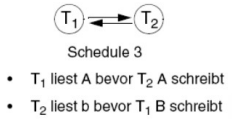
- **Synchronisation:**
  - **TA $T_i$ setzt sich aus folgenden Operationen zusammen:**

    Leseop.: $r_i$ | Schreibop.: $w_i$ | Abbruch: $a_i$ | Commit: $c_i$

  - **Ausführungsplan/Historie:**
    - **Konfliktoperationen:**
      - ➢ Zwei Operationen stehen in Konflikt miteinander, wenn beide auf dem gleichen Datenobjekt arbeiten und mindestens eine davon eine Schreiboperation ist

        $read_i[A] <_H write_j[A]$
        $write_i[A] <_H read_j[A]$
        $write_i[A] <_H write_j[A]$

    -




*Serialisierbar: wenn kein*

**Serialisierbarkeitsgraph:**

- **Serialisierbarkeit - sequenzieller Ablaufplan:**
  - Beschränkung der „Parallelität“ auf erlaubte Verarbeitungsreihenfolge
  - Schedule von Transaktionen ist serialisierbar, wenn er zu irgendeinem seriellen Ablauf der in ihm enthaltenen Transaktionen äquivalent ist

- korrekt und frei von Anomalien

- **Weitere, wünschenswerte Eigenschaften:**

| Rücksetzbarkeit | Vermeidung Kaskadierendes Rücksetzen | Striktheit |
|---|---|---|
| Abgeschlossene TAs dürfen nicht mehr zurückgesetzt werden<br>➔ TA wird erst nach der Freigabe aller TAs, von denen sie gelesen hat, freigegeben | Jede TA darf nur Ergebnisse von abgeschlossenen TA sehen | Objekte, die von einer TA T verändert wurden, dürfen nicht von anderen TA verändert werden, bevor die TA T abgeschlossen ist |

- **Datenbank Scheduler:**
  - ➢ Programm, welches eingehenden Operationen ordnet und für eine serialisierbare und rücksetzbare Historie sorgt
  - **Strategien:**
    - Pessimistischer: Verzögerung d. entgegengenommenen Operation
    - Optimistischer: Schickt entgegengenommenen Operation mögl. Schnell zur Ausführung
  - **Pessimistische Synchronisation – mittels Sperren:**
    - ➢ Für exklusiven Zugriff auf Datenobjekte
    - Zentrale Protokollierung des Sperrmodus jedes Datenobjektes
    - Vor/Während und bis zum Ende einer Transaktion
    - **Arten:** X (Schreibsperre), S/R (Lesesperre)
    - Serialisierbarkeit durch **2-Phasen Sperrprotokoll** gewährleistet:
      - ➢ Jeweils 1 Sperrschritt zu Beginn & Ende jeder TA
      - Wachstumsphase: es werden Sperren angefordert, aber keine freigegeben
      - Schrumpfungsphase: es werden Sperren freigegeben, aber keine angefordert
      - schließt (kaskadierendes) Rücksetzen nicht aus, bei Abbruch einer TA
    - **Striktes 2PL:**
      - ➢ Freigabe aller Schreibsperren erst am Ende einer Transaktion
      - Verhindert kaskadierende Zurücksetzen sich gegenseitig beeinflussender TA

**Kompatibilitätsma**

| angeforderte Sperre \ gehaltene Sperre | NL | R | X |
|---|---|---|---|
| R | + | + | - |
| X | + | - | - |

NL = no lock (wird meist weggelassen)

    - **Deadlocks:**
      - **Erkennung:**
        DBMS erzeugt einen Wartegraphen, System prüft auf period. Zyklen.
        Anschließendes zurücksetzen einer zufälligen Transaktion
      - **Vermeidung:**
        Wenn eine Transaktion eine Sperre erwerben möchte die bereits von einer anderen Transaktion gehalten wird dann setze eine der beiden Transaktionen zurück
  - **Optimistische Synchronisation:**
    - ➢ 3-phasige Verarbeitung: Lesephase, Validierungsphase, Schreibphase
    - **Validierungsstrategien:**

- **Backward-Oriented:**
  Test gegenüber allen Änderungs-TA $T$, die seit BOT von$T$ erfolgr. validiert haben
- **Forward-Oriented:** Nur Änderungs-TA validieren gegenüber laufenden TA

# 6. Fehlerbehandlungen:

- **Recovery Klassen:**

| R1-partial undo | R2-partial redo | R3-global undo | R4-global redo |
|---|---|---|---|
| Transaktionsfehler | Systemfehler -> Abgeschlossene TAs müssen erhalten bleiben | Systemfehler -> Abgeschlossene TAs müssen zurückgesetzt werden | Fehler mit Hintergrundspeicherverlust (bspw. Feuer) |

- **Ersetzungsstrategie von Pufferseiten:**

| STEAL | NOSTEAL |
|---|---|
| ➢ Geänderte Seiten können jederzeit ersetzt und in die DB eingebracht werden<br>▪ **Vorteil:** große Flexibilität zur Seitenersetzung<br>▪ Nachteil: UNDO-Recovery vorzusehen | ➢ Seiten mit schmutzigen Änderungen dürfen nicht ersetzt werden<br>▪ Vorteil: Keine UNDO-Recovery auf DB vorzusehen<br>▪ Nachteil: Probleme bei langen Änderungstransaktionen |

- **Ausschreibstrategien:**

| FORCE | NOFORCE |
|---|---|
| ➢ Geänderten Seiten werden spätestens beim EOT in die DB eingebracht<br>▪ **Vorteil:** Keine REDO-Recovery nach Rechnerausfall<br>▪ **Nachteil:** Antwortzeitverlängerung für Änderungstransaktionen | ➢ Modifizierte Seiten werden nicht unbedingt nach EOT in die DB zurückkopiert<br>▪ Vorteil: nur REDO-Informationen in Log-Datei<br>▪ Nachteil: REDO-Recovery nach Rechnerausfall |

**Kombinationsmöglichkeiten**

| | FORCE | ¬FORCE |
|---|---|---|
| ¬STEAL | Kein Redo<br>Kein Undo | Redo<br>Kein Undo |
| STEAL | Kein Redo<br>Undo | Redo<br>Undo |

- **Protokollierung von Änderungen:**

Log (LSN,TA,PageID,Redo,Undo,PrevLSN)

  - **Protokollierungsarten:**
    - **Logisch:** Verwendung von Verweisen, wie LSN
    - **Physische:** Verwendung von After & Before Images
- **Wiederanlauf im Fehlerfall:**
  - **Grundregeln:**

- **WAL-Prinzip: Write Ahead Log für UNDO-Info**
  Vor Auslagerung einer modifizierte Seite, müssen alle Log-Einträge, die zu dieser Seite gehören, in die Log-Datei geschrieben werden
- **COMMIT-Regel (Force-Log-at-Commit)**
  REDO-Information muss vor dem COMMIT im Protokoll stehen

▪ **Phasen:**

1. **Analyse:**
   Ermittlung der Winners *(commit)* & der Losers *(ohne commit)* von TAs
2. **Redo:**
   Protokollierte Änder. werden in der Reihenf. ihrer Ausführ. in die Datenbasis eingebracht
3. **Undo:**
   Änderungsoperationen der Loser-Transaktionen werden in umgekehrter Reihenfolge ihrer ursprünglichen Ausführung rückgängig gemacht

▪ **Recovery bei mehrmaligen Absturz, mittels:**
  - Idempotenz Redo-Phase durch LSN
  - Idempotenz Undo-Phase durch CLR für jede Undo-Op.

**CLR (LSN, TA, PageID, Redo-Info, PrevLSN, UndoNxtLSN)**
*wobei Redo-Info = erfolg. Undo*

- **Sicherungspunkte *(SP)*:**
  - Ohne: alle Änderungen seit Start des DBMS müssten wiederholt werden

▪ **Arten:**

| Transaktionskonsistent | Aktionskonsistente | Unscharfe |
|---|---|---|
| DBS in Ruhezustand<br>-> aktive TA zu Ende führen<br>-> neue TA müssen warten | Vor Anlegung eines SP Abgeschlossenheit aller Operationen notwendig | Nur Kennung der modif. Seiten werden ausgeschrieben |

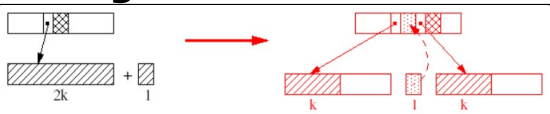
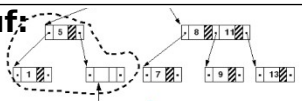
# 7. B/$B^{+}$ Baum:

- **B-Baum:**

▪ **Parameter k:** mind. Anzahl an Elementen pro Knoten/Blatt; max = 2k

▪ **Parameter h:** $\lceil \log_{2k+1}(n+1) \rceil \leq h \leq \lceil \log_{k+1}\left(\frac{n+1}{2}\right)+1 \rceil$

▪ **Operationen:**

| Einfügen | Löschen |
|---|---|
|  | - **Unterlauf:**  |
| Einfügen nach Reihenfolge der Werte: kl. -> gr. Zahl | **->** Verschmelzen mit Geschwisterknoten |

# 8. Anfrageverarbeitung: