Definitionen

Datei: Eine Datei ist eine benannte Ansammlung von Sätzen eines oder mehrerer Satztypen.

Dateisystem: Ein Dateisystem ist ein Softwarepaket, das den Zugriff auf einzelne Sätze in einer Datei besorgt, wenn das Anwendungsprogramm die entsprechenden Parameter liefert.

Datenbank: Eine Datenbank ist eine integrierte Ansammlung von Daten, die allen Benutzern eines Anwendungsbereiches als gemeinsame Basis aktueller Information dient. Die Daten sind entsprechend den natürlichen Zusammenhängen strukturiert, so dass für jede – auch ungeplante – Anwendung in der für sie benötigten Weise auf die Daten zugegriffen werden kann.

- **Datenbank** (DB): Die eigentlichen Daten des Anwendungsbereiches.
- **Datenbankmanagementsystems** (DBMS): Die (Datenbank-)Software, die den Zugriff auf die Daten der Datenbank steuert und überwacht.
- **Datenbanksystem** (DBS): Datenbank und Datenbankmanagementsystem zusammen bilden das Datenbanksystem.

Datenmodell: Ein Datenmodell legt fest, welche Daten(typen) wie strukturiert werden dürfen und welche Operationen auf den Daten und den Strukturen erlaubt sind.

Datenbankmodell: Ein Datenbankmodell legt die (atomaren) Datentypen fest, die benutzt werden dürfen, die erlaubten Typkonstruktoren, die Operationen auf Datentypen und Typkonstruktoren und die Regeln, die die genaue Verwendung von Datentypen und Typkonstruktoren festlegen.

Datenunabhängigkeit: Unabhängigkeit der Anwendungsprogramme von der Organisation, dem Aufbau und der Abspeicherung von Daten.

- **Physische Datenunabhängigkeit**: Isolation der Anwendungsprogramme von Änderungen der physischen Datenorganisation und des Datenzugriffs (z.B. Änderung der Dateiorganisation, Einfügen eines Zugriffspfades, …)
- **Logische Datenunabhängigkeit**: Isolation der Anwendungsprogramme von Änderungen der logischen Darstellung der Daten innerhalb der Datenbank (z.B. Hinzufügen von neuen Attributen, Umbenennung von Attributen, …)

ANSI/SPARC 3-Schema-Architektur

- **Internes Schema** (physische Realisierung, z.B. Files, Segmente, Indexe): Das interne oder physische Schema legt fest, in welcher Form die Daten der DB abgespeichert werden (Speicherungsstrukturen) und welche Zugriffspfade bzw. Indexstrukturen zur Gewährleistung eines schnelleren Zugriffs auf die Daten bereit gestellt werden. Diese Ebene ist für den normalen Benutzer eines DBS nicht relevant und daher auch nicht sichtbar.
- **Konzeptuelles Schema** (stabiles, globales Referenzschema, z.B. Relationen): Das konzeptuelle Schema stellt eine auf der Basis des Datenmodells entwickelte logische Beschreibung des Realweltausschnitts dar, der in der DB abgelegt werden soll. Die Beschreibung findet auf einer relativ abstrakten Ebene statt, bei der von Informationen über die interne Darstellung abstrahiert wird. Das konzeptuelle Schema stellt eine anwendungsneutrale Beschreibung des Realweltausschnitts dar. Es wird in der Regel aus dem beim konzeptuellen Datenbankentwurf entwickelten ER- oder objektorientierten Modell abgeleitet und stellt dessen DBMS-spezifische Repräsentation dar.

Diese Ebene ist für den Anwendungsprogrammierer relevant. Der normale Benutzer wird nur dann auf sie zurückgreifen, falls ihm kein eigenes Schema zur Verfügung steht.

- **Externes Schema** (Benutzersicht, z.B. Views, Subschemata): Das externe Schema, auch Sicht genannt, extrahiert aus dem konzeptuellen Schema einen anwendungsspezifischen Ausschnitt. Damit wird die Ebene modelliert, auf der üblicherweise die Anwendungen bzw. Benutzer arbeiten. Eine Sicht ist an die spezifischen Bedürfnisse der zu Grunde liegenden Anwendung angepasst. Das bedeutet insbesondere, dass die Anwendung nur den sie interessierenden Teil des konzeptuellen Schemas sieht und dies in einer auf ihre Bedürfnisse zugeschnittenen Form. Nicht relevante Daten werden ausgeblendet, andere Daten so dargestellt, wie es die Anwendung wünscht. Zusätzlich dient das externe Schema aber auch dem Datenschutz. Die Anwendung sieht genau das, was sie auch sehen soll und darf.
- Während es pro DB nur ein internes und ein konzeptuelles Schema gibt, ist die Anzahl der externen Schemata nicht begrenzt. Grundsätzlich kann jede auf der DB arbeitende Anwendung ihr eigenes externes Schema besitzen. In der Regel stehen auf der Ebene des externen Schemas die gleichen Modellierungsmechanismen zur Verfügung wie auf der Ebene des konzeptuellen Schemas.

Eigenschaften von DBMSen:

- **Persistenz**: dauerhafte Speicherung von Daten
- **Synchronisation, Transaktionsmanagement**: sicheres gleichzeitiges Arbeiten auf dem gemeinsamen Datenbestand
- **Recovery**: zentrale, automatische Fehlerbehandlung und Fehlerbehebung
- **Integrität**: zentrale Konsistenzüberwachung
- **Abfragesprache**: assoziativer (inhaltsbezogener) Zugriff auf Daten
- **Autorisierung**: zentrale Zugriffskontrolle

Eigenschaften von Relationen

- **Eindeutigkeit von Tupeln**: keine Duplikate, eindeutig identifizierbar
- **Attributwerte sind atomar**: d.h. aus Sicht der Anwendung nicht weiter zerlegbar
- **Keine Ordnung auf Tupeln oder Attributen**

Entity: identifizierbares Objekt der zu modellierenden Realwelt

Abbildungsprozess Realwelt ←→ relationales Datenbankmodell

- **Realwelt**
  - Vielschichtig, überwiegend Unikate
  - Umfangreiche Beziehungen
- **Semantisches Datenmodell**
  - Zusammenfassung von Entities zu Gruppen
  - Zwar schon stark abstrahierendes, aber vergleichsweise immer noch mächtiges Datenmodell
  - Beziehungen explizit modellierbar und komplexe Integritätsbedingungen
- **Relationales Datenmodell**
  - Abbildung auf Tupel und normalisierte Relationen, einfaches Datenmodell
  - Beziehungen nur implizit als Relation oder Attribut modellierbar

Typen von Relationen:

- **Benannte Relation**: mit einem Namen versehene Relation
- **Basisrelation**: benannte Relation, die Bestandteil des konzeptuellen Datenbankschemas ist; es existiert sowohl eine logische als auch eine physische Repräsentation in der DB
- **Abgeleitete Relation**: durch Anwendung von relationalen Operatoren aus einer oder mehreren Relationen abgeleitete Relation
- **Virtuelle Relation**: benannte abgeleitete Relation ohne physisches Abbild in der DB
- **Ergebnis-/Anfragerelation**: entsteht als Ergebnis der Ausführung einer Anfrage an die DB
- **Ausgangsrelation**: Basis für Anfragen, Anfragen von Mengenoperationen erwarten beispielsweise als Operanden ein oder zwei Ausgangsrelationen
- **Zwischenrelation**: unbenannte abgeleitete Relation, die als Zwischenschritt bei der Berechnung einer Ergebnisrelation erzeugt wird und als Ausgangspunkt für weitere Verarbeitungsschritte dient
- **Sicht**: benannte abgeleitete Relation, die eine speziell angepasste Sicht einer Benutzergruppe oder Anwendung auf einen Teil der DB widerspiegelt; im Normalfall virtuelle Relationen
- **Gespeicherte/Materialisierte Relation**: entspricht im Verhalten einer Basisrelation und wird auch physisch in der DB gehalten, ist jedoch i.d.R. nur für einen bestimmten Zeitraum gültig und existent
- **Schnappschussrelation**: materialisierte Relation, die genau einen Zustand einer Relation einfriert

Normalformen:

- **1. Normalform**: Wertebereiche aller Attribute der Relation müssen atomar sein.
- **2. Normalform**: Nicht-prime Attribute müssen vom ganzen Schlüssel funktional abhängen.
- **3. Normalform**: Nicht-prime Attribute dürfen nur direkt von einem Schlüssel abhängen.

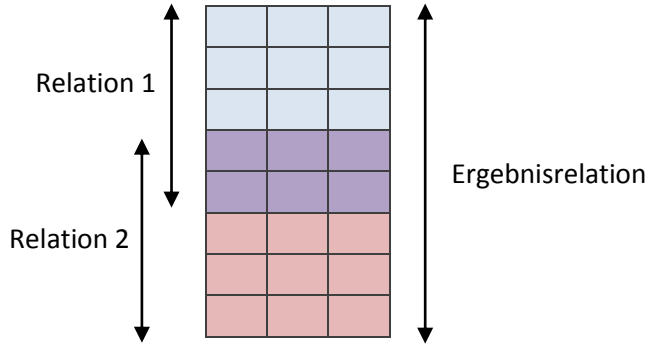
Anomalien:

- **Einfügeanomalie**: Bei einer Insert-Anomalie sind bei der Neueingabe eines Datensatzes Informationen so verknüpft, dass bestimmte Daten nicht eingegeben werden können, ohne dass andere Daten eingegeben werden müssen (Attribute haben die Eigenschaft *NOT NULL*).
- **Löschanomalie**: Eine Delete-Anomalie tritt auf, wenn die Datenbankstruktur nicht richtig entworfen wurde und Daten in einer Tabelle zusammengefasst wurden, die eigentlich nicht in einer einzelnen Tabelle hätten verwaltet werden sollen. Es müssen dann alle Daten eines Datensatzes (Tupels) gelöscht werden, obwohl an sich nur eine Teilmenge hätte gelöscht werden sollen.
- **Änderungsanomalie**: Eine Update-Anomalie tritt auf, wenn in der Datenbank redundant gespeicherte Daten nur teilweise aktualisiert werden. Eine Update-Anomalie resultiert meist aus einem falschen Datenbankentwurf, weil Daten in einer einzigen Tabelle verwaltet werden, die nicht in eine Tabelle gehören, sondern in mehrere aufgeteilt werden sollten (Stammdatentabellen).

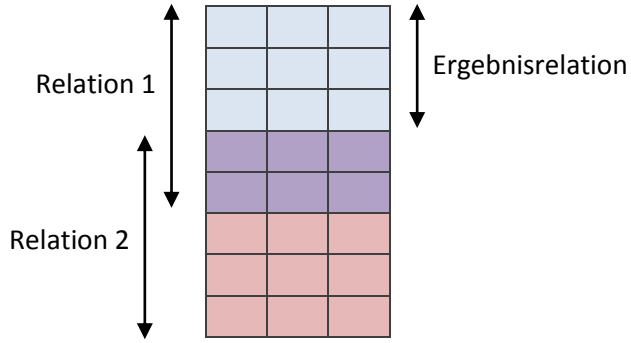
Operationen auf Relationen

Mengenoperationen: Sind zwei Relationen über ein identisches Schema definiert, so können die Mengenoperationen ∩ (Vereinigung), ∪ (Durchschnitt) und \ (Differenz) angewendet werden.

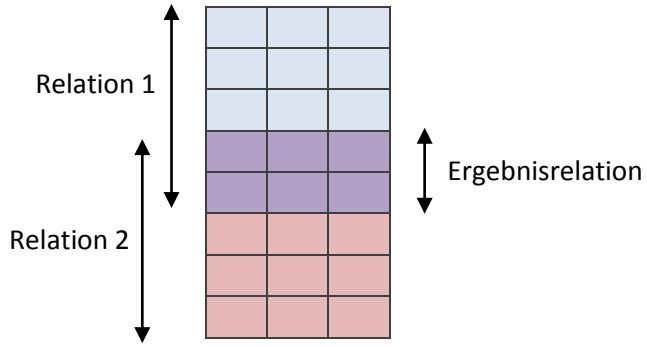
Vereinigung: $Rel_r \cup Rel_s := \{t | t \in r(R) \vee t \in s(R)\}$
Die Vereinigung zweier typkompatibler Relationen erweitert die eine Relation um alle noch nicht vorhandenen Tupel der anderen Relation.



Differenz: $Rel_r \setminus Rel_s := \{t | t \in r(R) \wedge t \notin s(R)\}$
Die Differenz zweier typkompatibler Relationen entfernt aus der ersten Relation all die Tupel, die auch in der zweiten Relation enthalten sind.



Durchschnitt: $Rel_r \cap Rel_s := \{t | t \in r(R) \wedge t \in s(R)\}$
Der Durchschnitt zweier typkompatibler Relationen ergibt die Tupel, die in beiden Relationen enthalten sind, ermittelt also die gemeinsame Teilmenge beider Mengen.



Kartesisches Produkt: $Rel_r \text{ x } Rel_s$
Das Kartesische Produkt zweier Relationen verknüpft jedes Tupel der einen Relation mit jedem Tupel der anderen Relation.

R

A	B	C
AA	D1	1K
BB	D4	6Z
CC	D7	8G

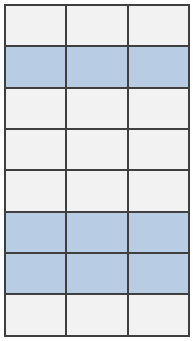
S

D	E
11	AA
22	BB

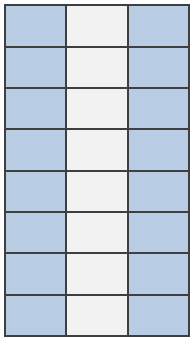
R x S

A	B	C	D	E
AA	D1	1K	11	AA
BB	D4	6Z	11	AA
CC	D7	8G	11	AA
AA	D1	1K	22	BB
BB	D4	6Z	22	BB
CC	D7	8G	22	BB

Selektion: $\sigma_{Bedingung}(Rel) := \{t | t \in Rel \wedge Bedingung(t) = TRUE\}$
Eine Selektion wählt die Tupel aus einer Relation aus, bei denen ein Attribut oder mehrere Attribute eine Selektionsbedingung oder mehrere Selektionsbedingungen erfüllen. Bei den Bedingungen kann man dabei zwischen einer Konstantenselektion und einer Attributselektion unterscheiden.



Projektion: $\pi_{attr_1,\ldots,attr_m}(Rel)$
Eine Projektion schneidet aus einer gegebenen Relation alle aus der Sicht der Anwendung irrelevanten Attribute aus. Gegebenenfalls entstehende Duplikate werden entfernt.



Division: $Rel_r \div Rel_s$
Die Division arbeitet auf zwei Relationen Rel_r und Rel_s, wobei gelten muss, dass alle Attribute aus Rel_s auch Bestandteil von Rel_t sind. Die Ergebnisrelation besteht nur aus den Attributen von $Rel_r^{\neq}$. Sie enthält genau dann den ≠-Anteil eines Tupels ($t^{\neq}$ genannt) aus Rel_r, wenn die Menge aller aus dem =-Anteil bestehenden Tupel von Rel_r, die im ≠-Anteil $t^{\neq}$ entsprechen, mindestens alle Tupel aus Rel_s

enthält.
Beispiel: Angenommen, die beiden Relationen Produkt und ProduktLagertIn sind auf die Attribute <produktNr> und <produktLagerNr, produktNr> reduziert worden. Sollen nun die Lager ermittelt werden, in denen alle vom Unternehmen vertriebenen Produkte gelagert werden, so kann dies wie folgt über die Division berechnet werden: ProduktLagertIn ÷ Produkt.

R

AA	11
BB	11
CC	11
AA	22
CC	22
AA	33
BB	33
CC	33

S

AA
BB
CC

R÷S

11
33

Thetaverbund: $Rel_r \bowtie_\Theta Rel_s := \sigma_{Thetaverbundbedingung}(Rel_r \text{ x } Rel_s)$

$\sigma_{Tab_L.C < Tab_R.C}\ (Tab_L \text{ x } Tab_R)$

Tab_L

A	B	C
a1	b1	1
c3	d2	2

Tab_R

C	D	E
1	d1	e1
3	d2	e2

Ergebnistabelle

A	B	C	C	D	E
a1	b1	1	3	d2	e2
c3	d2	2	3	d2	e2

Gleichverbund: $Rel_r \bowtie_= Rel_s := \sigma_{Gleichverbundbedingung}(Rel_r \text{ x } Rel_s)$

$\sigma_{Tab_L.C=Tab_R.C}\ (Tab_L \text{ x } Tab_R)$

Tab_L

A	B	C
a1	b1	1
c3	d2	2

Tab_R

C	D	E
1	d1	e1
3	d2	e2

Ergebnistabelle

A	B	C	C	D	E
a1	b1	1	1	d1	e1

Natürlicher Verbund: $Rel_r \bowtie Rel_s$
Ein allgemeiner Verbund, bei dem die Tupel zweier Relationen dann miteinander verbunden werden, wenn sie in allen namensgleichen Attributen den gleichen Wert aufweisen, in der Ergebnisrelation diese Attribute aber nur einmal vorkommen, nennt man einen natürlichen Verbund. Ein natürlicher Verbund ist also ein um die Projektion wird eine nur einfach Übernahme der namensgleichen Attribute bewirkt.

Tab_L

A	B	C
a1	b1	1
c3	d2	2

Tab_R

C	D	E
1	d1	e1
3	d2	e2

Ergebnistabelle

A	B	C	D	E
a1	b1	1	d1	e1

Linker äußerer Verbund: $Rel_r \supset\!\bowtie Rel_s$
Beim linken äußeren Verbund werden alle Tupel der Relation, die links vom Verbundoperator steht, vollständig übernommen. Analog verfährt man beim rechten und vollständigen äußeren Verbund.

Tab_L

A	B	C
a1	b1	1
c3	d2	2

Tab_R

C	D	E
1	d1	e1
3	d2	e2

Ergebnistabelle

A	B	C	D	E
a1	b1	1	d1	e1
c3	d2	2	⊗	⊗

Semiverbund: $Rel_r \ltimes Rel_s \coloneqq \pi_{\{attr_1,\dots,attr_r\}} Rel_r \bowtie Rel_s$
Beim Semiverbund wird zunächst eine der üblichen Verbundoperationen durchgeführt. Dann werden aus der Ergebnisrelation die Attribute einer der beteiligten Relationen wieder herausgenommen.

Tab_L

A	B	C
a1	b1	1
c3	d2	2

Tab_R

C	D	E
1	d1	e1
3	d2	e2

Ergebnistabelle

A	B	C
a1	b1	1

Umbenennung: Die Umbenennung weist

1. einem Attribut a einer Relation Rel einen neuen Namen a‘ zu
   $\rho_{attr_{a'} \leftarrow attr_a} Rel$
2. einer Relation Rel_a einen neuen Namen Rel_n zu
   $\rho_{Rel_n} Rel_a$

Datentypen:

- **Numerische Datentypen**: `SMALLINT, INTEGER, DECIMAL, NUMERIC, REAL, FLOAT, DOUBLE PRECISION`
- **Chronologische Datentypen**: `DATE, TIME, TIMESTAMP, INTERVAL`
- **Zeichenbasierte Datentypen**: `CHARACTER, CHARACTER VARYING`
- **Bitbasierte Datentypen**: `BIT, BIT VARYING`

SQL (Structured Query Language)

DDL (Data Definition Language, Datendefinitionssprache)

Prädikate:

Prädikat	Datentypen	Bedeutung
BETWEEN	alle Datentypen	Enthaltensein des Wertes im spezifizierten Wertebereichsausschnitt
IN	alle Datentypen	Enthaltensein eines Wertes in einer Menge
LIKE	alphanumerische Zeichenketten	(partielle) Übereinstimmung zweier Werte (% beliebig viele Zeichen, _ein Zeichen)
NULL	alle Datentypen	Belegung eines Wertes mit der Nullmarke
OVERLAPS	Zeitintervalle	Überlappung zweier Zeitintervalle

Anlegen eines **Wertebereiches** mit **Default**:

```
CREATE DOMAIN ShortString AS CHAR(5)
        DEFAULT '99999';
```

Anlegen einer **Tabelle**:

```
CREATE TABLE Mitarbeiter (
        plz:        ShortString,
        ort:        VARCHAR(20),
        strasse:    VARCHAR(20),
        hausnr:     SMALLINT);
```

Zeitpunktprüfung für **Wertebereicheinschränkungen**:

- NOT DEFERRABLE: Test wird direkt ausgeführt
- DEFERRABLE: Test kann grundsätzlich verschoben werden
- INITIALLY: legt den Default bei Beginn einer Transaktion fest
- IMMEDIATE: sofortige Überprüfung vor Ende der Transaktion

```
CREATE DOMAIN FünfstelligeZahl AS INTEGER
        CONSTRAINT Fünfstellig CHECK (VALUE BETWEEN 10000 AND 99999)
        DEFERRABLE INITIALLY IMMEDIATE;
```

Ist die Überprüfung einer Wertebereichseinschränkung grundsätzlich verschiebbar, kann über die SET CONSTRAINTS-Klausel innerhalb einer Transaktion die Ausführung einer (verschobenen) Überprüfung sofort (IMMEDIATE) angeordnet oder auf einen späteren Zeitpunkt (DEFERRED) verschoben werden.

1) SET CONSTRAINTS Fünfstellig IMMEDIATE;
2) SET CONSTRAINTS Fünfstellig DEFERRED;

Dynamische Wertebereichsfestlegung:

```
CREATE DOMAIN BustypDomain AS CHAR(1)
        DEFAULT 'S'
        CONSTRAINT Typeinschränkung CHECK (VALUE IN ('D','G','M','R','S'));
```

Hinweis: Soll Nullmarke nicht zum Wertebereich gehören: AND VALUE IST NOT NULL

```
CONSTRAINT Typeinschränkung
        CHECK (VALUE IN (SELECT DISTINCT Bustyp FROM Beschreibung));
```

Spalten- und tabellenbezogene Integritätsbedingungen:

```
produktNr       INTEGER         CHECK (VALUE IS NOT NULL)
produktNr       INTEGER         NOT NULL
CREATE TABLE ProduktLagertIn (
        produktNr       INTEGER         NOT NULL CHECK (VALUE BETWEEN 1000 AND 99999),
        lagerNr         SMALLINT        NOT NULL CHECK (lagerNr BETWEEN 100 AND 999),
        istBestand      SMALLINT        (CHECK VALUE ≥ 0) DEFAULT 0,
        qualität        CHAR(8)         CHECK (VALUE IN ('sehr gut','gut','mittel','schlecht')));
CREATE TABLE Abteilung (
        abteilungsNr            DreistelligeZahl        PRIMARY KEY,
        abteilungsName          VARCHAR(25)             UNIQUE NOT NULL);
```

Check-Klausel im Anschluss an die Deklaration der Spalten:

```
CREATE TABLE Mitarbeiter (
        mitarbeiterNr   INTEGER         NOT NULL,
        plz             CHAR(5),
        …,
        CHECK (mitarbeiterNr BETWEEN 1000 AND 99999 AND
                plz LIKE 'D-_____' OR plz LIKE 'NL-_____'));
```

Tabellenbezogene Integritätsbedingungen:

```
CONSTRAINT VerlustVermeidung CHECK (nettoPreis > (stueckKosten * 1,5));
CONSTRAINT NrKontrolle CHECK (teilNr NOT IN (SELECT produktNr FROM SindBestandVon));
```

Primärschlüssel und **Schlüsselkandidaten**:

```
CREATE TABLE Produkt (
        produktNr       FünfstelligeZahl        PRIMARY KEY,
        produktBez      VARCHAR(25)             UNIQUE);
```

Fremdschlüssel:

```
CREATE TABLE Liefert (
        zuliefererNr    AchstelligeZahl,
        teilNr          FünfstelligeZahl,
        produktNr       FünfstelligeZahl,
        CONSTRAINT LiefertPS PRIMARY KEY (zuliefererNr, teilNr),
        CONSTRAINT LiefertFS1 FOREIGN KEY (zuliefererNr) REFERENCES Zulieferer
                ON DELETE CASCADE ON UPDATE CASCADE,
        CONSTRAINT LiefertFS2 FOREIGN KEY (teilNr) REFERENCES Teil
                ON DELETE CASCADE ON UPDATE CASCADE,
        CONSTRAINT LiefertFS3 FOREIGN KEY  (produktNr) REFERENCES Produkt
                ON DELETE SET NULL ON UPDATE CASCADE);
```

Um trotz zyklischer Verweise zwischen zwei Tabellen Daten einfügen zu können, kann vor der Einfügeoperation die Überprüfung der Fremdschlüsselbedingung außer Kraft gesetzt werden (SET CONSTRAINTS VerweisUnterTabelle DEFERRED; zu Beginn und am Ende SET CONSTRAINTS VerweisUnterTabelle IMMEDIATE;).

Behandlungen von **Integritätsverletzungen**:

- **RESTRICT**: Es ist verboten, Zeilen zu ändern/löschen, auf die noch von anderen Tabellen referenziert wird.
- **CASCADE**: Wird eine Zeile in der Haupttabelle gelöscht, so werden automatisch rekursiv auch alle Zeilen in den betroffenden Untertabellen gelöscht. Änderungen am Primärschlüssel werden auch in den betroffenen Untertabellen aktualisiert.
- **SET NULL/DEFAULT**: Alle betroffenen Fremdschlüsselspalten werden entweder mit der Nullmarke oder dem angegeben Default belegt, sofern dies zugelassen bzw. spezifiziert wurde.
- **NO ACTION**: keine automatische Reaktion des DBMS, ist die Default-Einstellung

Tabellen ändern:

```
ALTER TABLE ProduktLagertIn
        DROP COLUMN produktLagerBez RESTRICT;
```

Die Spalte wird somit nur gelöscht, falls produktLagerBez in keiner anderen Tabelle Bestandteil eines Fremdschlüssels ist.

```
ALTER TABLE ProduktLagertIn
        ALTER COLUM qualität CHAR(8)
        CHECK (VALUE IN (‘sehr gut‘,‘gut‘,‘mittel‘,‘schlecht‘));
ALTER TABLE ProduktLagertIn
        ADD CONSTRAINT QualitätsEinschränkung
        CHECK (qualität IN (‘sehr gut‘,‘gut‘,‘mittel‘,‘schlecht‘));
ALTER TABLE ProduktLagertIn
        ADD COLUMN qualität CHAR(8) SET DEFAULT ‘mittel‘;
```

Wertebereichänderungen:

```
ALTER DOMAIN BustypDomain
        DROP CONSTRAINT Typeinschränkung;
ALTER DOMAIN BustypDomain
        ADD CONSTRAINT Neue Typeinschränkung
        CHECK (VALUE IN (SELECT DISTINCT Bustyp FROM Busbeschreibung));
ALTER DOMAIN BustypDomain
        DROP DEFAULT;
ALTER DOMAIN BustypDomain
        SET DEFAULT ‘D’;
```

Datenbankobjekte löschen:

```
DROP DOMAIN FünfstelligeZahl RESTRICT;
DROP TABLE Mitarbeiter;
```

Drop-Befehl löscht Datenbankobjekt vollständig. Delete hingegen löscht nur die aktuellen Ausprägungen des Datenbankobjekts, die Strukturbeschreibung bleibt erhalten.

Operationen auf **Zeit-** und **Datumsintervallen**:

INTERVAL '3 6' DAY TO HOUR/3 = INTERVAL '1 2' DAY TO HOUR

DATE '2003-04-01' – DATE '2000-01-01' = INTERVAL '3-3' YEAR TO MONTH

DRL (Data Retrieval Language, Datenbankanfragesprache)

Einfache **SQL-Anfrage**:

```
SELECT mitarbeiterNr, mitarbeiterName
        FROM Mitarbeiter
        WHERE einstellung < DATE '1965-01-01';
```

Ergebnistabelle mit und ohne **Duplikate**:

```
SELECT ALL produktTyp
        FROM Produkt;
SELECT DISTINCT produktTyp
        FROM Produkt;
```

Aggregatfunktionen:

Name	Bezug	erwarteter Argumenttyp	Ergebnistyp	Semantik
COUNT()	Tabelle	keiner (*)	numerisch	Anzahl von Zeilen der Ergenistabelle
COUNT()	Spalte	beliebige Spalte	numerisch	Anzahl der Zeilen, die bei der als Argumenttyp angegebenen Spalte einen Wert ungleich der Nullmarke aufweisen
SUM()	Spalte	numerisch	numerisch	Summe aller aktuellen Werte der Spalte
AVG()	Spalte	Numerisch	numerisch	Durchschnitt aller aktuellen Werte der Spalte
MAX()	Spalte	Zeichen / numerisch / chronologisch	wie Argumenttyp	höchster der Werte einer Spalte
MIN()	Spalte	Zeichen / numerisch / chronologisch	Wie Argumenttyp	niedrigster der Werte einer Spalte

Das **Kartesische Produkt**:

```
SELECT auftragsNr, mitarbeiterName, mitarbeiterVorname
        FROM Auftrag, Mitarbeiter
        WHERE bearbeiterNr = mitarbeiterNr;
SELECT auftragsNr, mitarbeiterName, mitarbeiterVorname
        FROM Auftrag CROSS JOIN Mitarbeiter
        WHERE bearbeiterNr = mitarbeiterNr;
```

Der **natürliche Verbund**:

```
SELECT DISTINCT produktBez, abteilungsName
        FROM Produkt NATURAL JOIN ArbeitetAn NATURAL JOIN Abteilung
        WHERE Produkt.produktTyp = 'Waschmaschine';
```

Hinweis: Beim natürlichen Verbund fallen übereinstimmende Spalten weg.

Der **Spaltenverbund**:

```
SELECT Zulieferer.*, Teil.*
        FROM Zulieferer INNER JOIN Liefert USING (zuliefererNr)
        INNER JOIN Teil USING (teilNr);
```

Hinweis: Die Spalten, die im USING-Teil angegeben werden, werden nicht doppelt ausgegeben.

Der **Bedingungsverbund**:

```
SELECT Mitarbeiter.*
        FROM Mitarbeiter INNER JOIN KUNDEN ON (mitarbeiter.Nr = kundenNr);
```

Hinweis: redundante Spalten werden nicht ausgeblendet.

Der **äußere Verbund**:

```
SELECT *
        FROM Produkt NATURAL LEFT OUTER JOIN ArbeitetAn
        NATURAL LEFT OUTER JOIN Abteilung;
SELECT *
        FROM Produkt LEFT OUTER JOIN ArbeiteAn USING (produktNr)
        LEFT OUTER JOIN Abteilung USING (abteilungsNr);
```

Der **Vereinigungsverbund**:

```
SELECT *
        FROM Produkt UNION JOIN ArbeitetAn;
```

Boolsche Operatoren:

NOT	Schließt all diejenigen Zeilen aus, die die dem NOT folgende Bedingung erfüllen.
AND	Sowohl die vor als auch die nach dem AND stehende Bedingung müssen erfüllt sein.
OR	Entweder die vor oder nach dem OR stehende Bedingung müssen erfüllt sein (oder beide).

Null-Prädikat:

```
SELECT *
        FROM Mitarbeiter
        WHERE abteilungsNr IS NULL;
```

Group-By- und **Having-Klausel**:

```
SELECT produktLagerBez AS Lager, COUNT(produktNr) AS #produkte
        FROM ProduktLagertIn
        GROUP BY produktLagerBez
        Having produktLagerBet = 'München' OR produktLagerBez = 'Berlin';
```

Geschachtelte Anfragen:

```
SELECT produktTyp, produktTyp, AVG(ØnettoPreis) AS ØnettoPreisÜberAlleLager
        FROM (SELECT produktTyp, produktLagerNr, AVG(nettoPreis) AS ØnettoPreis
                FROM ProduktLagertIn NATURAL JOIN Produkt
                WHERE istBestand > 0
                GROUP BY produktTyp, produktLagerNr)
        GROUP BY produktTyp;
SELECT *
        FROM Mitarbeiter
        WHERE ort = (SELECT ort
                FROM Mitarbeiter
                WHERE gehalt = (SELECT MIN(gehalt)
                        FROM Mitarbeiter));
```

Mengenorientierte Prädikate:

	Semantik
IN	Ist der Wert links vom Prädikat in der Menge rechts vom Prädikat enthalten?
EXISTS	Ergibt die Auswertung der zum Prädikat gehörenden Unteranfrage mindestens einen Wert?
Θ ANY	Gilt, dass der Θ-Vergleich des Wertes links vom Prädikat mit den Werten der Menge rechts vom Prädikat für mindestens einen Vergleich wahr ist?
Θ SOME	Ist eine andere Bezeichnung für ANY.
Θ ALL	Gilt, dass der Θ-Vergleich des Wertes links vom Prädikat mit den Werten der Menge rechts vom Prädikat für alle Vergleiche wahr ist?

Vereinigung:

```
SELECT kundenName AS name, plz, ort, strasse, hausNr
        FROM Kunden
UNION
SELECT zuliefererName AS Name, plz, ort, strasse, hausNr
        FROM Zulieferer;
```

Durchschnitt:

```
SELECT kundenName AS name, plz, ort, strasse, hausNr
        FROM Kunden
INTERSECT
SELECT zuliefererName AS Name, plz, ort, strasse, hausNr
        FROM Zulieferer;
```

Differenz:

```
SELECT kundenName AS name, plz, ort, strasse, hausNr
        FROM Kunden
EXCEPT
SELECT zuliefererName AS Name, plz, ort, strasse, hausNr
        FROM Zulieferer;
```

Order-By-Klausel:

```
SELECT mitarbeiterName, mitarbeiterVorname
        FROM Mitarbeiter
        ORDER BY mitarberName ASC, mitarbeiterVorname DESC;
```

Abarbeitung von SQL-Anfragen:

FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY

DML (Date Manipulation Language, Datenmanipulationssprache)

Neue Zeile einfügen:

```
INSERT INTO Mitarbeiter (mitarbeiterNr, mitarbeiterName)
        VALUES (123456,'Mueller'');
```

Mehrere neue Zeilen einfügen:

```
INSERT INTO MitarbeiterAlsKunden
        (SELECT mitarbeiterNr, mitarbeiterName, mitarbeiterVorname
        FROM Mitarbeiter INNER JOIN Kunden ON (mitarbeiterNr = kundenNr));
```

Ändern von Zeilen:

```
UPDATE TeilLagertIn
        SET istBestand = istBestand + 50000
        WHERE teilNr = 55555 AND teileLagerNr = 12006;
```

Zeile löschen:

```
DELETE FROM Teil
        WHERE teilBez = 'Hammer';
```

Löschen aller Zeilen einer Tabelle:

```
Delete FROM Mitarbeiter;
```

DCL (Data Control Language, Datenkontrollsprache)

Vergeben aller Rechte an einer Tabelle an eine Benutzergruppe:

```
GRANT ALL PRIVILIGES
        ON Mitarbeiter
        TO PersonalAngelegenheiten;
```

Vergeben von **einzelnen Rechten**:

```
GRANT
        SELECT (mitarbeiterNr, mitarbeiterName),
        DELETE (mitarbeiterNr, mitarbeiterName)
        On Mitarbeiter
        TO PersonalAngelegenheiten;
```

Widerruf von Rechten:

```
REVOKE ALL PRIVILEGES
        ON Mitarbeiter
        TO PersonalAngelegenheiten;
```

Rechtevergabe für Operationen in SQL-92:

Operation	Bedeutung
SELECT	Ermöglicht das Lesen aller Spalten einer Tabelle.
DELETE	Ermöglicht das Löschen von Zeilen aus einer Tabelle oder nur das Löschen von Spaltenwerten von Zeilen, falls eine Liste von Spalten angegeben wurde.
INSERT	Erlaubt das Einfügen von Zeilen, wobei festgelegt werden kann, dass nur eine vorgegebene Menge von Spalten mit Werten belegt werden darf. Für die anderen Spalten kommt der (hoffentlich) spezifizierte Default zum Tragen. Aus offensichtlichen Gründen muss in der angegebenen Spaltenliste der Primärschlüssel und jede Spalte enthalten sein, für die die NOT NULL-Option ohne Angabe eines Default spezifziert wurde.
UPDATE	Ermöglicht das Ändern von Zeilen in einer Tabelle oder nur das Ändern von Spaltenwerten von Zeilen, falls eine Liste von Spalten angegeben wurde.
REFERENCES	Erlaubt die Nutzung von Spalten aus der Spaltenliste als Fremdschlüssel bei der Definition einer anderen Tabelle.

SSL (Storage Structure Language) bzw. DSDL (Date Storage Definition Language, Speicherungsstrukturdefinitionssprache)

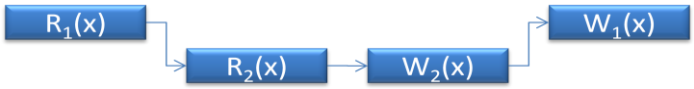
Erstellen eines **Sekundärindexes**:

```
CREATE INDEX mitarbeiterSort
        ON Mitarbeiter (mitarbeiterName DESC, mitarbeiterVorname DESC);
```

Transaktionsmanagement

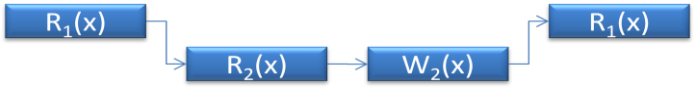
Probleme bei der Parallelarbeit auf der DB

- **Verloren gegangene Änderungen (lost update)**



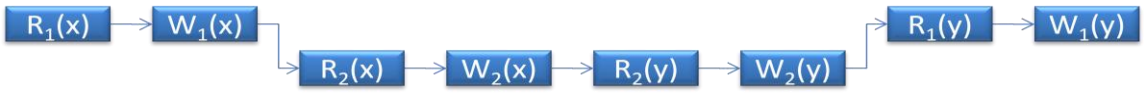
  Die Änderungen der Transaktion 2 werden von Transaktion 1 überschrieben und sind somit wirkungslos.

- **Nicht wiederholbares Lesen (unrepeatable read)**



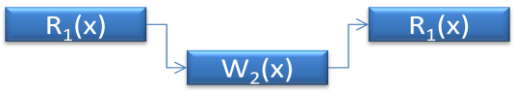
  Der Wert des ersten Lesevorgangs von Transaktion 1 hat sich geändert, ohne dass Transaktion 1 etwas davon mitbekommt. Beim Lesen desselben Objektes erhält die Transaktion 1 unterschiedliche Werte.

- **Inkonsistente DB**



  Möchte Transaktion 1 zum Beispiel bei allen Objekten den Preis um 10 % erhöhen, und Transaktion 2 schlägt auf alle Preise 50 € auf, so entstehen im obigen Beispiel unterschiedliche Preise für Objekte, die eigentlich den gleichen Preis besitzen sollten. Die Datenbank ist somit inkonsistent.

- **Phantomproblem**



  Transaktion 2 fügt nach dem Lesen von Transaktion 1 ein neues Objekt ein. Bei einem erneuten Lesen durch Transaktion 1 wird nun ein Objekt mehr ausgegeben.

Definition Transaktion: Eine Transaktion stellt eine Folge von logisch zusammenhängenden Operationen auf der DB dar. Die von einer Anwendung durchgeführt werden. Diese Folge repräsentiert eine konsistenzwahrende Einheit, was insbesondere heißt, dass die DB durch die Transaktion von einem konsistenten Zustand in einen nicht zwangsläufig unterschiedlichen konsistenten Zustand überführt wird.

Wirkungslose Transaktionen bestehen ausschließlich aus

- **Leseoperationen** auf einem Objekt, das in der selben Transaktion **nicht mehr geschrieben** werden, und/oder
- **Schreiboperationen** auf einem Objekt, das in einer anderen Transaktion **überschrieben** wird, ohne dass es zwischenzeitlich gelesen wurde.

ACID-Kriterium für Transaktionen:

- **Atomicity/Unteilbarkeit**: Da eine Transaktion eine Folge von Operationen beschreibt, die nur in ihrer Gesamtheit ausgeführt die Konsistenz der DB garantieren, müssen sie entweder vollständig oder gar nicht ausgeführt werden. Tritt also während der Transaktionsausführung ein nicht korrigierbarer Fehler auf, so muss jede von der Transaktion durchgeführte Änderung (automatisch) rückgängig gemacht werden.
- **Consistency/Konsistenz**: Eine erfolgreich durchgeführte Transaktion überführt die DB von einem konsistenten Zustand in einen nicht zwangsläufig unterschiedlich konsistenten Zustand. Sollten während des Ablaufes einer Transaktion inkonsistente Zwischenzustände vorkommen, müssen diese bis zum Transaktionsende beseitigt sein.
- **Isolation**: Eine Transaktion muss immer so ablaufen, als wäre sie die einzige im System. Parallele Transaktionen dürfen einander nicht beeinflussen.
- **Durability/Dauerhaftigkeit**: Änderungen, die von einer erfolgreich abgeschlossenen Transaktion durchgeführt werden, überleben jeden nachfolgenden Fehlerfall.

Definition Serialisierbarkeit: Ein paralleles System von Transaktionen ist dann korrekt synchronisiert, wenn es serialisierbar ist, d.h. wenn es mindestens eine (gedachte) serielle Ausführung derselben Transaktionen gibt, die

- denselben Datenbankzustand erzeugt und bei der
- alle Transaktionen dieselben Ausgabedaten liefern.

Eine strikte Serialisierbarkeit liegt vor, wenn eine Vertauschung von nacheinander ausgeführten Transaktionen ausgeschlossen ist.

Definition Schedule: $T^* = (T_1, \ldots, T_n)$ sei eine Menge von fest vorgegebenen Transaktionen. Eine Schedule S (auch Historie/history, log oder computation genannt) ist eine beliebige Permutation der Operationen der Transaktionen aus T^* mit der Einschränkung, dass die relative Reihenfolge der Operationen jeder Transaktion erhalten bleiben muss. Eine Schedule ist also eine beliebige Aneinanderreihung der Operationen aller Transaktionen, wobei aber die Operationen einer Transaktion in ihrer Reihenfolge nicht vertauscht werden dürfen.

Definition Serielle und serialisierbar Schedule: Werden Transaktionen sequenziell nacheinander ausgeführt, entsteht eine Schedule, in der immer erst alle Operationen einer Transaktion stehen, bevor die Operationen der nächsten Transaktion kommen. Eine solche Transaktion heißt seriell. Entsprechen die Auswirkungen einer Schedule denen einer seriellen Schedule, so spricht man von einer serialisierbar Schedule. Solche Schedule erzeugen denselben Datenbankendzustand und dieselben Ausgaben wie mindestens eine serielle Schedule.

Definition Konflikt: Zwei Operationen o_{im} und o_{jn} verschiedener Transaktionen stehen in Konflikt zueinander, wenn sie auf dasselbe Objekt zugreifen und mindestens eine der Operationen eine Schreiboperation (WRITE) ist.

Synchronisationsverfahren:

Synchronisationsverfahren haben die Aufgabe, eine beliebige Folge von eingehenden Operationsaufforderungen so anzuordnen, dass die Serialisierbarkeit gewährleistet ist.

- **Optimistische (verifizierende) Vorgehensweise**: Es wird nur eingegriffen, wenn die Serialisierbarkeit vermutlich bereits verletzt wurde. Die problemverursachende Transaktion wird dann zurückgesetzt. Bei dieser Vorgehensweise wird davon ausgegangen, dass ein Konflikt nur selten auftritt, weshalb das Rücksetzen als Mittel zur Synchronisation wegen seines seltenen Auftretens akzeptabel ist.
- **Pessimistische (präventive) Vorgehensweise**: Es wird immer eingegriffen, wenn eine Konsistenzverletzung vorliegt oder eintreten könnte. Hier werden Sperrverfahren eingesetzt, welche auch überwiegend in zentralen DBMS verwendet werden.

Definition Lese-/Schreibsperre:

- Die **Lesesperre** oder gemeinsame Sperre (shared lock, S-Sperre) erlaubt dem "Sperrbesitzer" das Lesen des gesperrten Objektes. Andere Transaktionen dürfen ebenfalls Lesesperren setzen, wohingegen das Setzen einer Schreibsperre verboten ist.
- Die **Schreibsperre** oder exklusive Sperre (exclusive lock, X-Sperre) erlaubt dem "Sperrbesitzer" sowohl lesenden als auch schreibenden Zugriff auf das Objekt. Anderen Transaktionen ist jeglicher Zugriff auf das Objekt verwehrt.

Sperrhierarchie:

- DB
- Area (oder Segment)
- Tabelle (oder Datei)
- Zeile (oder Satz)

Ein negativer Nebeneffekt der Sperrhierarchien: Je feiner die Sperreinheiten sind, desto mehr Parallelität ist zwischen den Transaktionen möglich, desto größer ist aber auch der Verwaltungsaufwand für die Sperren.

Definition Warnsperren: Warnsperren sind Sperren, die auf allen dem aktuellen Sperrgranulat übergeordneten Ebenen gesetzt werden. Dort drücken sie die Absicht aus, auf tieferer Ebene ein Objekt zu sperren. Um Konflikten beim Setzen von Sperren aus dem Weg zu gehen, müssen Sperren immer von oben nach unten (von der DB zum Datensatz) angefordert werden.

Sperrmodi von Warnsperren:

- **IS**: Erklärt die Absicht der Transaktion, auf einer tiefer liegenden Ebene mindestens ein Objekt zu lesen. Daher wird das Recht erworben, dort IS- und S-Sperren zu setzen.
- **IX**: Erklärt die Absicht der Transaktion, auf einer tiefer liegenden Ebene mindestens ein Objekt zu verändern bzw. zu schreiben. Daher wird das Recht erworben, dort IS-, S-, IX-, X- und SIX-Sperren zu setzen.
- **SIX**: Wirkt auf der gegeben Ebene wie eine S-Sperre, indem alle Objekte auf dieser Ebene lesegesperrt werden. Zusätzlich erklärt die Transaktion die Absicht, auf einer tiefer liegenden Ebene mindestens ein Objekt zu schreiben. Für tiefer liegende Ebenen wird das Recht erworben, IX-, X- und SIX-Sperren zu setzen. Da bereits diese Ebene komplett S-gesperrt ist, sind alle tieferen Ebenen implizit mit gesperrt, weshalb das Setzen von IS- und S-Sperren auf tieferen Ebenen nicht

mehr notwendig ist. Die SIX-Sperre ist beispielsweise dann sinnvoll, wenn sich Ute zunächst eine Vielzahl von Hotels ansehen (lesen) möchte, bevor sie dann in einem Hotel ein Zimmer reserviert.

Kompatibilitätsmatrix der Warnsperrmodi:

	IS	IX	S	SIX	X
IS	👍	👍	👍	👍	👎
IX	👍	👍	👎	👎	👎
S	👍	👎	👍	👎	👎
SIX	👍	👎	👎	👎	👎
X	👎	👎	👎	👎	👎

Definition Fortgepflanztes Rollback: Von einem fortgepflanzten Rollback spricht man immer dann, wenn der Abbruch einer Transaktion von anderen, von dieser Transaktion abhängigen, möglicherweise beendeten Transaktion zur Folge hat. Ein fortgepflanztes Rollback kann immer dann auftreten, falls eine Transaktion T_a von ihr geänderte Daten bereits vor ihrem Ende freigibt. Werden diese Daten jetzt von einer anderen Transaktion T_b genutzt, so muss auch diese zurückgesetzt werden, falls T_a, aus welchen Gründen auch immer, scheitern sollte. Diese Abhängigkeit wird dadurch hervorgerufen, dass T_b ein von Ta geändertes Datum gelesen hat. Da die Änderung von T_a jedoch zurückgesetzt wurde, hat T_b ein ungültiges Datum gelesen. Diese Situation kann sich nun fortpflanzen, indem eine Transaktion T_c zurückgesetzt werden muss, weil sie Änderungen der Transaktion T_b gelesen hat usw.

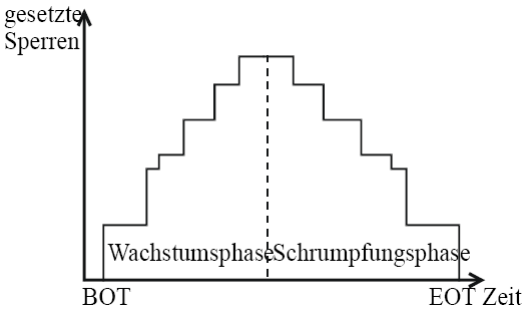
Das Problem mit fortgepflanztem Rollback ist neben dem Verlust von bereits durchgeführter Arbeit (Rücksetzen von Transaktionen) insbesondere die Tatsache, dass möglicherweise bereits abgeschlossene Transaktionen zurückgesetzt werden müssen. Das steht im Widerspruch zum D-Kriterium der ACID-Eigenschaft, welches ja garantiert, dass einmal abgeschlossene Transaktionen jeden nachfolgenden Fehlerfall überleben. Um dieses Problem aus dem Weg zu gehen, wurde das strikte Zweiphasen-Sperrprotokoll entwickelt.

Definition Zweiphasen-Sperrprotokoll: Das Zweiphasen-Sperrprotokoll wird beachtet, wenn eine Transaktion wohlgeformt und zweiphasig ist. Eine Transaktion ist wohlgeformt (well formed), falls sie für jedes Objekt, das sie bearbeiten möchte, vorher eine entsprechende Sperre anfordert. Sie ist zweiphasig (two phase), falls sie kein Objekt mehr sperrt, nachdem sie zum ersten Mal eine Sperre auf einem Objekt freigegeben hat. Eine zweiphasige Transaktion besteht demnach aus einer so genannten Wachstumsphase, während der sie alle Sperren anfordert, und einer daran anschließenden Schrumpfungsphase, während der sie alle Sperren wieder freigibt, aber keine Sperren mehr Anfordern darf. Wenn eine Transaktion endet, werden alle noch gehaltenen Sperren automatisch freigegeben.

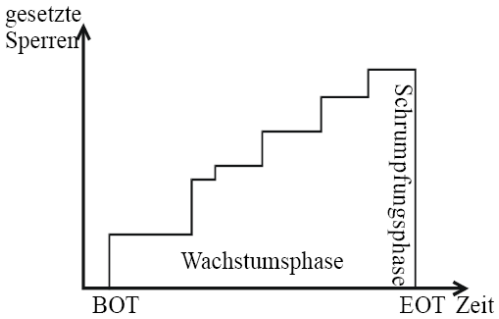
Definition Preclaiming: Preclaiming bedeutet, dass alle Objekte, die eine Transaktion möglicherweise verwenden wird, zu Beginn der Transaktion auf einen Schlag gesperrt werden. Können nicht alle Objekte im gewünschten Modus gesperrt werden, wird die anfordernde Transaktion in einen Wartezustand versetzt. Nach Ablauf eines Zeitintervalls versucht sie erneut, die benötigten Sperren zu setzen.

Die vier verschiedenen Varianten der Zweiphasen-Sperrprotokolle:

- **Einfaches Zweiphasen-Sperrprotokoll**:

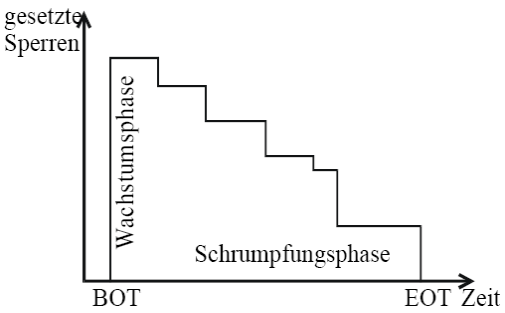


- **Striktes Zweiphasen-Sperrprotokoll**:



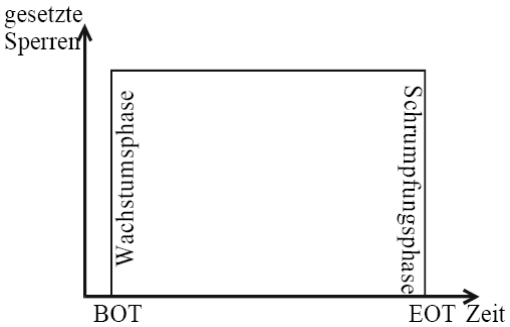
Alle von der Transaktion gesperrten Objekte werden auf einmal freigegeben und zwar mit Hilfe einer aus der Sicht des DBMS atomaren Operation am Ende der Transaktion. Erfüllt die höchste Sicherheitstufe (Konsistenzstufe 3), nur Phantomproblem muss bei physischen Sperren auf anderem Weg gelöst werden.

- **Zweiphasen-Sperrprotokoll mit Preclaiming**:



Das Preclaiming verhindert Deadlocks. Aufgrund der Probleme, dass eine sehr grobe Ebene und sehr früh gesperrt werden muss und des dadurch entstehenden Parallelitätsverlust wird es jedoch kaum eingesetzt.

- **Striktes Zweiphasen-Sperrprotokoll mit Preclaiming**:



Vermeidet zwar alle Varianten im Zusammenhang mit Sperrprotokollen bekannten Probleme, macht aber eine Parallelarbeit an einer Datenbank unmöglich.

Definition Deadlock: Ein Deadlock ist eine Situation, bei der Prozesse wechselseitig darauf warten, dass der jeweils andere Prozess von ihnen benötigte Betriebsmittel freigibt.

Deadlocks können in allen System auftreten, in denen allgemein zugängliche, exklusiv zu nutzende Betriebsmittel an parallel laufende, konkurrierende Prozesse vergeben werden und bei denen die folgenden vier Bedingungen alle erfüllt sind:

- **Bedingung des gegenseitigen Ausschlusses**: Prozesse fordern und erhalten exklusive Kontrolle über von ihnen benötigte Betriebsmittel.
- **Bedingung der Nichtentziehbarkeit**: Ein einmal zugeteiltes Betriebsmittel muss explizit durch den das Betriebsmittel haltenden Prozess freigegeben werden. Es kann also nicht zwangsweise entzogen werden.
- **Warte-und-Halte-Fest-Bedingung**: Wenn ein beantragtes Betriebsmittel blockiert ist, also nicht zugewiesen werden kann, wird der beantragende Prozess in den Zustand "blockiert" überführt, wobei bereits zugewiesene Betriebsmittel weiterhin festgehalten werden.
- **Bedingung der Mehrfachanforderung**: Prozesse, die bereits ein Betriebsmittel besitzen, können die Zuteilung weiterer Betriebsmittel beantragen.

Verfahren zur **Bewältigung** der **Deadlock-Problematik**:

- Verhinderung (prevention), z.B. Preclaiming
- Vermeidung (avoidance) (theoretisch: nur dann Zuteilung von Betriebsmitteln an einen Prozess erlauben, wenn sichergestellt ist, dass noch mindestens ein Weg vorhanden ist, der allen im System befindlichen Prozessen ein normales Ende ermöglicht → komplexer Analyseprozess)
- Auflösung durch Zeitbeschränkung (time-out), besitzt viele Vorteile wie den geringen Verwaltungsoverhead, besitzt aber auch Schwächen bei wechselnder Systemauslastung, da i.d.R. immer die teuerste (am längsten laufende) Transaktion zurückgesetzt wird
- Erkennung und Auflösung (detection and resolution), z.B. Wartegraphen, bedeutet hoher Aufwand

Deadlockerkennung und vor allem Time-out sind gängige Verfahren in DBMS.

Die Instanz der DBMS, die für die Synchronisation der Transaktionen verantwortlich ist, wird üblicherweise **Sperrmanager** genannt. Die **Aufgaben** des Sperrmanagers:

- Setzen und Freigabe von Sperren unter Beachtung des Zweiphasen-Sperrprotokolls
- Verhinderung von dauerhafter Blockade durch "faire" Vergabe von Sperren an Transaktionen
- Verhinderung oder Entdeckung und Auflösung von Deadlocks

Definition Prädikatsperren: Prädikatsperren sind Sperren, die auf logischer Ebene dadurch gesetzt werden, dass ein Prädikat angegeben wird, welches exakt und eindeutig die Menge aller zu sperrenden Objekte beschreibt.

Mit Prädikatsperren kann das Phantomproblem gelöst werden, jedoch entstehen andere erhebliche Probleme. Bisher werden logische Sperren noch in keinem bekannten, kommerziellen DBMS eingesetzt, insbesondere auch, weil noch keine effizienten Algorithmen für einen hinreichend allgemeine Klasse von Prädikaten existieren.

Isolationsstufen in SQL-92:

- **Level 0, read uncommited**: Bedeutet, dass die entsprechende Transaktion Daten, die von einer anderen noch im Gang befindlichen Transaktion geschrieben wurden, lesen kann ("dirty read"). Die Transaktion hält keine Sperren. Diese Isolationsstufe ist nur für Read-Only-Transaktionen gestattet.
- **Level 1, read commited**: Bedeutet, dass entsprechende Transaktionen nur "definitive" Daten lesen kann. Die Transaktion hält keine Sperren für Tupel, die von ihre gelesen wurden, wohl aber für solche, die von ihr modifiziert wurden. Da keine Sperren für gelesen Tupel gehalten werden, ist es möglich, dass bei wiederholtem Lesen eines bestimmten Tupels dieses zwischenzeitlich verändert wurde. Auch das Phänomen des inkonsistenten Lesens sowie das Phantom-Problem kann auftreten.
- **Level 2, repeatable read**: Bedeutet ebenfalls, dass die entsprechende Transaktion nur "definitive" Daten lesen kann. Zudem hält die Transaktion Sperren für alle Tupel, die von ihre gelesen und/oder geschrieben wurden. Sie hält jedoch keine Prädikatsperren. Damit kann nur noch das Phantomproblem auftreten.
- **Level 3, serializable**: Garantiert, dass die Transaktion in jedem Fall so auf der DB arbeiten kann, als sei sie die einzige Aktive. Einflüsse von anderen Transaktionen, gleich welcher Art, sind ausgeschlossen.

	Schmutziges Lesen	Nicht wiederholbares Lesen	Phantomproblem
READ UNCOMMITTED	kann auftreten	kann auftreten	kann auftreten
READ COMMITTED	nicht möglich	kann auftreten	kann auftreten
REPEATABLE READ	nicht möglich	nicht möglich	kann auftreten
SERIALIZABLE	nicht möglich	nicht möglich	nicht möglich

Definition Recovery: Unter Recovery versteht man alle Aktionen eines DBMS, die im Zusammenhang mit der Sicherung der Konsistenz der DB in einem Fehlerfall stehen. Im Idealfall wird durch die Recoveryaktion der Fehler kaschiert, bleibt also aus der Sicht der Anwendung transparent. Recovery deckt eine Vielzahl unterschiedlicher Fehler ab, die vom Zurücksetzen einer einzigen Transaktion bis hin zum Erzeugen des letzten konsistenten Datenbankzustandes aus einer Archivkopie nach Auftritt eines besonders schweren Fehlers reichen.

Da das Betriebssystem alleinig für die Hauptspeicherverwaltung zuständig ist, kann die Situation auftreten,

1. dass Seiten mit modifizierten Datensätzen von zum Fehlerzeitpunkt noch aktiven Transaktionen bereits aus dem Hauptspeicher verdrängt worden waren oder
2. dass Seiten mit modifizierten Datensätzen von zum Fehlerzeitpunkt bereits abgeschlossenen Transaktionen sich noch im Hauptspeicher befinden, ohne zusätzlich auch auf dem Sekundärspeicher zurückgeschrieben worden zu sein.

Im Fall 1 müssen die Daten aus dem Sekundärspeicher mit einem UNDO entfernt werden. Im Fall 2 müssen mit Hilfe eines REDO die gültigen Daten in den Sekundärspeicher gespeichert werden.

Definition UNDO: Unter einem UNDO versteht man das Zurücksetzen eines aktuellen Zustandes eines Datums auf einen früheren, konsistenten Zustand. Dabei kann zurückgesetzt werden, indem entweder der (irgendwo festgehaltene) ehemalige Zustand direkt in die DB zurückgeschrieben wird oder indem die zwischenzeitlich auf dem Datum ausgeführten Operationen durch in der richtigen (also umgekehrten) Reihenfolge ausgeführten inversen Operationen rückgängig gemacht bzw. kompensiert werden.

Definition REDO: Unter einem REDO versteht man das Wiederherstellen des verloren gegangenen aktuellen und konsistenten Zustandes eines Datums. Dies kann erreicht werden, indem entweder der (noch woanders gesicherte) letzte Zustand direkt in die DB zurückgeschrieben wird oder indem ausgehend von einem alten konsistenten Zustand die danach auf dem Datum ausgeführten Operationen erneut ausgeführt werden.

Zwei Fälle von UNDO/REDO:

- **steal**: Von einer Transaktion modifizierte Seiten können grundsätzlich jederzeit, insbesondere auch vor dem Ende einer Transaktion auf dem Sekundärspeicher zurückgeschrieben werden. Es ist Vorsorge für ein UNDO zu treffen.
- **¬steal**: Von einer Transaktion modifizierte Seiten können grundsätzlich nicht vor dem Ende einer Transaktion auf dem Sekundärspeicher zurückgeschrieben werden. Eine Vorsorge für ein UNDO ist nicht notwendig.

	force	**¬force**
steal	kein REDO UNDO	REDO UNDO
¬steal	kein REDO kein UNDO	REDO kein UNDO

- **force**: Von einer Transaktion modifizierte Seiten müssen grundsätzlich spätestens bis zum Ende der Transaktion auf dem Sekundärspeicher zurückgeschrieben werden. Eine Vorsorge für ein REDO ist nicht notwendig.
- **¬force**: Von einer Transaktion modifizierte Seiten können grundsätzlich zu einem beliebigen Zeitpunkt, also insbesondere auch erst nach dem Ende einer Transaktion auf den Sekundärspeicher zurückgeschrieben werden. Es ist Vorsorge für ein REDO zu treffen.

Definition Logdatei: Eine Logdatei ist vereinfacht dargestellt eine sequentielle Datei, in der jeder Transaktionsbeginn, jedes Transaktionsende und jede (atomare) Änderungsoperation auf der DB in der richtigen, d.h. in der durch die Realität vorgegebenen Reihenfolge protokolliert wird.

Definition logische Protokollierung: Von einer logischen Protokollierung spricht man immer dann, wenn der Logdateieintrag dokumentiert, welche Änderungsoperation mit welchem Operanden auf welcher Spalte durchführt wurde. Für einen vollständigen Logeintrag muss daher mindestens die verantwortliche Transaktion, die betroffene Zeile/Spalte, die ausgeführte Operation und, falls es sich um eine binäre Operation handelt, der zweite Operand festgehalten werden. Um ein UNDO auszuführen, muss zudem die zur durchgeführten Operation inverse Operation angegeben werden.

Definition physische Protokollierung: Bei einer physischen Protokollierung wird in der Logdatei notiert, welches der Wert einer Spalte vor seiner Änderung war und welches der Wert nach seiner Änderung ist. Ersterer wird Before-Image, Letzerer After-Image genannt. Für einen vollständigen Logdateieintrag muss daher mindestens die verantwortliche Transaktion, die betroffene Zeile/Spalte, das Before- und das After-Image abgelegt werden.

Definition Rollback: Von einem Rollback spricht man, wenn das DBMS im laufenden Betrieb aufgrund eines Transaktionsrecoveryereignisses eine bestimmte Transaktion T zurücksetzen muss. Damit müssen von T in der DB bewirkten Änderungen so rückgängig gemacht werden, als ob T nie existent gewesen wäre. Da vom Zurückrollen i.d.R. keine parallelen Transaktionen betroffen sind, kann das DBMS den Rücksetzprozess parallel zur Abarbeitung anderer Transaktionen vornehmen.

Wird eine ¬steal-Politik gefahren und auf der Ebene von Seiten synchronisiert, so kann ein Rollback einer Transaktion T durch ein einfaches Freigeben der betroffenen Hauptspeicherseiten vorgenommen werden. Schwieriger ist der steal-Fall. Da bereits Daten auf dem Sekundärspeicher zurückgeschrieben sein können, müssen diese Änderungen wieder rückgängig gemacht werden. Dazu wird die Log-Datei solange von hinten nach vorne gelesen, bis die Beginnmarke von T gefunden wurde. Auf dem Weg dorthin wird für jedes Before-Image mit der Transaktionsidentifikation von T der alte Objektwert in die DB zurückgeschrieben.

Bei einem Systemzusammenbruch, der meist den Verlust des gesamten Hauptspeicherinhaltes einschließlich aller Puffer und Verwaltungsinformationen bedingt, kann im Prinzip zunächst wie im Fall des Rollback vorgehen, indem die Änderungen der zum Absturzzeitpunkt noch nicht beendeten Transaktionen rückgängig gemacht werden. Um die Transaktionen ermitteln zu können, die zurückgesetzt werden müssen, wird die Logdatei rückwärts gelesen. Für jede BOT-Marke wird überprüft, ob für diese Transaktion auch eine entsprechende EOT-Marke in der Logdatei vorhanden ist. Ist dem nicht so, war die Transaktion offiziell noch nicht beendet und es muss ein UNDO gemacht werden. Im anderen Fall war die Transaktion bereits beendet, es muss aber evtl. ein REDO durchgeführt werden.

Definition Checkpoints: Checkpoints sind dauerhaft gemachte Verwaltungsinformationen, die den aktuellen Zustand des Transaktionssystems beschreiben. Sie werden in regelmäßigen Abständen in die Logdatei geschrieben. Im Fall eines Systemzusammenbruchs dienen sie als Ausgangspunkt für die Recoveryaktion.

Bei Neustarts muss das DBMS die Logdatei bis zum jüngsten Checkpoint rückwärts lesen. Ab jetzt ist bekannt, welche Transaktionen zurückgesetzt werden müssen: alle die, die im Checkpoint aufgeführt oder erst danach gestartet wurden und für die bis zum Ende der Logdatei keine EOT-Marke gefunden wurde.

Arten von Checkpoints:

- **Transaktionsorientiert**: Zum Zeitpunkt des Schreibens des Checkpoints darf keine Transaktion aktiv sein. Somit ist ein Rollback über den Checkpoint hinaus nie nötig.
- **Aktionskonsistent**: Checkpoint wird nach fest vorgegebenen Zeitintervallen geschrieben, es wird nur unterstellt, dass jede Operation abgeschlossen ist und die in der Logdatei notierten Operationen atomar sind. Ein REDO ist nie nötig, dies gilt aber nicht für UNDO-Befehle.
- **Unscharf**: Datensätze der Logdatei werden zusätzlich über einen monoton aufsteigenden eindeutigen Identifizierer durchnummeriert. In der Logdatei wird nur noch festgehalten, welche Seiten noch nicht auf den Sekundärspeicher zurückgeschrieben wurden, aber Daten von bereits beendeten Transaktionen enthalten.

Definition Dump: Unter einem Dump versteht man eine Sicherungskopie der kompletten DB auf einem nichtflüchtigen Medium.

JDBC

Bevor ein **JDBC-Treiber** verwendet werden kann, muss er im JDBC **DriverManager** registriert werden:

```
try {
      Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
}
catch (ClassNotFoundException e) {
      // Handle Exception
}
```

Das java.sql.**Connection**-Objekt kapselt genau eine **Verbindung** zu einer bestimmten Datenbank. Die Methode `getConnection()` des DriverManager-Objekts erzeugt eine Verbindung:

```
Connection con = DriverManager.getConnection
("jdbc:db2:studivwt","administrator","adminPW");
```

Nachdem die Verbindung aufgebaut wurde, können **SQL-Abfragen** mit Hilfe von **Statements** ausgeführt werden:

```
Statement stmt = con.createStatement();
```

Das **ResultSet** enthält das **Abfrageergebnis** und verwendet einen internen Zeiger:

```
ResultSet rs = stmt.executeQuery("SELECT * FROM Studenten");
```

Das **Auslesen der Daten** aus einer Spalte kann entweder über die Angabe der **Spaltennummer** oder über den **Spaltennamen** erfolgen:

```
while(rs.next()){
      System.out.println("Matrikelnummer=" +
      rs.getString("Matrikelnr"));
      System.out.println("Studienfach=" +
      rs.getString("Studienfach"));
}
```

Schließen aller offenen Datenbank-Objekte:

```
rs.close();
stmt.close();
con.close();
```

Erzeugen von **vorkompilierten Anfragen**:

```
PreparedStatement pstmt = null;
pstmt = con.prepareStatement("SELECT * FROM Student WHERE
      matrikelnummer = ? AND name = ?");
pstmt.setInt(1,12345);
pstmt.setString(2,"Petersen");
rs = pstmt.executeQuery();
```

Beispiel einer einfachen **Transaktion**:

```
try {
      //BOT
      con.setAutoCommit(false);
      stmt.executeUpdate("UPDATE Lager1 SET Bestand=10");
      stmt.executeUpdate("INSERT INTO Versand (AnzArtikel)
            VALUES (5)");
      con.commit();
      //EOT
}
catch(SQLException e) {
      con.rollback();
}
```

Methode zum Behandeln von **Nullmarken** ist wasNull():

```
if (rs.wasNull())
      System.out.println("Nullmarke vorhanden.");
```