Datensicherung und Zugriffsschutz

Datensicherung

Gewährleistung der Richtigkeit, Vollständigkeit und logischen Widerspruchsfreiheit der Daten

Gewährleistung der Integrität/Konsistenz der Daten

Zugriffsschutz

Schutz der Daten vor unberechtigter Benutzung

Folgen unzureichender Datensicherung (Schadenskategorien)

- **Zerstörung von Werten:** Beschädigung von Datenträgern und Maschinen
- **Ausfall des Datenverarbeitungsbetriebes:** Unterbrechung bis zur Instandsetzung
- **Finanzielle Verluste:** Geldwerte Manipulationen sowie Zusatzkosten für Wiederaufbau (Datenrekonstruktion, Notfallorganisation, Anlaufprobleme)
- **Personelle Probleme:** Stress, Frustration und Überarbeitung bei den Mitarbeitern

Klassifikation der Gefahrenbereiche

- **Schadensursache Personal :**
  - Fehlerhafte Bedienung / Versehentliches Löschen von Daten
  - Fehler bei der Dateneingabe
  - Unterlassung notwendiger Handlungen (Vergessen des Backup)
- **Schadensursache Hardware:**
  - Ausfall von Hardwarekomponenten (Festplatte)
  - Störung des Rechner-/ Kommunikationsnetzes
- **Schadensursache Software:**
  - Fehlerhafte Anwendungssoftware
  - Probleme bei Update/Upgrades
  - Computerviren
- **Schadensursache „Höhere Gewalt“:**
  - Feuer / Brand
  - Wasser / Hochwasser
  - Immission (Staub, elektromagnetische Wellen)
  - Blitzschlag
  - Stromausfall
  - (Erdbeben, Terrorakte, Kriegshandlungen etc.)

Sicherungskonzepte

- **Personelle Maßnahmen:**
  - Auswahl des Personals; Qualifikation
  - Sicherung des Zugangs (Ausweise, Closed-shop)
- **Bauliche Maßnahmen:**
  - Trennung des EDV-Bereiches von übrigen Arbeitsbereichen
  - Auswahl und Sicherung geeigneter Räumlichkeiten (Hochwasserschutz, Brandschutztüren, Code-Schlösser usw.)
- **Technische Maßnahmen:**
  - Auswahl zuverlässiger Hardware
  - Notstromaggregate, unterbrechungsfreie Stromversorgung (BBU)
  - Elektronische Verriegelung
- **Organisatorische / Softwareseitige Maßnahmen:**
  - Programmierte Kontrollen: - Vollständigkeitsprüfungen
    - Datenformatsprüfungen
    - Plausibilitätskontrollen
  - Regelung von Zugriffsberechtigung
  - Anlegen von Sicherheitskopien
  - File- und Record-Locking
  - Dateiattribute (RO), Datenträger-Sicherung (RO)
  - Prüfziffern
  - ...

Beispiel einer Schlüsselsicherung mit Prüfziffer

Aufbau der EAN (Europ. Artikelnummer)

40	27927	00029	2
Land	Ländereinheitliche Betriebsnummer	Interne Artikelnummer des Herstellers	Prüfziffer
BRD	BestMedia GmbH Holdorf	CD-R 700 MB	

EAN	4	0	2	7	9	2	7	0	0	0	2	9	2
Gewichtung	1	3	1	3	1	3	1	3	1	3	1	3	
Produkte	4 +	0 +	2 +	21 +	9 +	6 +	7 +	0 +	0 +	0 +	2 +	27	

Produktsumme: 78
Modul: 10
Quotient: 7
Rest: 8

Prüfziffer = Modul - Rest = 10 -8 = 2

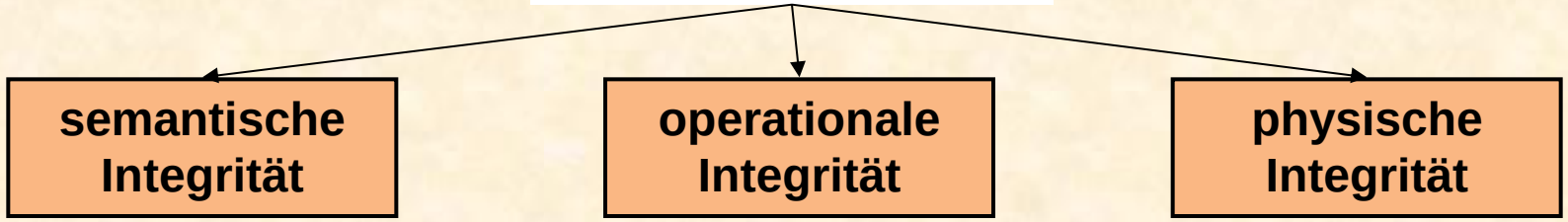
Überblick zur Integritätssicherung

Ziel der Integritätssicherung:
Unterstützung des Datenbanknutzer bei der Gewährleistung von Konsistenz und Integrität der gespeicherten Daten

Konsistenz bedeutet logische Übereinstimmung und innere Widerspruchsfreiheit der Daten.

Integrität bedeutet ganz allgemein inhaltliche Korrektheit/Richtigkeit und Vollständigkeit der Daten.

Unterscheidung in



semantische Integrität

- Ziel:
  Gewährleistung der Korrektheit der Daten bei jeglicher Nutzereingabe
- Maßnahmen:
  Erarbeitung von Integritätsbedingungen beim Datenbankentwurf

operationale Integrität

- Ziel:
  Sicherung der parallelen Nutzung der Datenbank durch verschiedene Nutzer
- Maßnahmen:
  Erstellung von Ablaufplänen und Sperren von Datenobjekten

physische Integrität

- Ziel:
  Wiederherstellung eines integeren Zustandes nach physischen Fehlern
- Maßnahmen:
  Protokollierung und Recovery

Was ist eine Transaktion

- Eine **Transaktion** ist eine Folge an SQL-Anweisungen, die wie eine einzelne Anweisung behandelt werden.

  Eine Transaktion garantiert, dass Änderungen entweder vollständig oder gar nicht ausgeführt werden.

- Das Transaktionsmanagement umfasst drei Komponenten:
  - **Transaktions-Kontrollanweisungen** werden vom Programmierer verwendet um sicherzustellen, dass eine Reihe von Anweisungen als Einheit behandelt wird.
  - Der Server **protokolliert alle Modifizierungen**, um die Möglichkeit der Rekonstruktion zu garantieren.
  - Der Server **sperrt Tabellenseiten** die während einer Transaktion verändert werden, so dass andere Anwender keinen Zugriff auf „vorübergehende“ Daten haben.

Syntax der Transaktions-Kontrollanweisungen:

Beginn der Transaktion:	`BEGIN TRANSACTION`
Bestätigen der Transaktion:	`COMMIT [TRANSACTION]`
Zurücksetzen der Transaktion:	`ROLLBACK [TRANSACTION]`

Eigenschaften einer Transaktion - ACID

- **Atomarität (atomicity)**

  „Unteilbarkeit der durch die Transaktionsdefinition (BEGIN bis COMMIT bzw. ROLLBACK) zusammengefassten Anweisungsfolgen

  Prinzip: „Alles oder Nichts“

  Der Benutzer weiß stets, in welchem Zustand die Datenbank ist.

- **Konsistenzhaltung (consistency)**

  Eine erfolgreich ausgeführte Transaktion erklärt ihre Änderungen für gültig und gewährleistet die Konsistenz der Datenbank.

  Auch umgekehrt: was immer eine vollständig ausgeführte Transaktion abliefert, akzeptiert das DBMS als konsistent.

- **Isolation (isolation)**

  Zwischenzustände der Daten, die eine Transaktion bearbeitet, sind möglicherweise inkonsistent und dürfen von parallel ablaufenden Transaktionen nicht benutzt werden.

  Umgekehrt darf eine Transaktion selbst auch keine Zwischenzustände anderer Transaktionen benutzen.

- **Dauerhaftigkeit (durability)**

  Die Ergebnisse vollständig ausgeführter Transaktionen dürfen nicht durch Fehler verloren gehen.

Probleme bei der Datenkonsistenz

Ursachen für mögliche Fehlerquellen:

– Flüchtigkeitsfehler / Eingabefehler
– Unkenntnis bei der Datenerfassung oder Datenänderung
– Datenimport aus externen Quellen
– Zusammenführung von Datenbeständen verschiedener DBMS

Beispielszenarien:

– widersprüchliche oder (teilweise) redundante Daten
→Nichtentscheidbarkeit von Fragen/Problemen
– Verletzung der Eindeutigkeit in einer Auftragstabelle
→ doppelte Rechnungen
– Buchstaben oder Sonderzeichen in Zeit-und Datumsangaben
→ falsche Ergebnisse einer Recherche

Daher Notwendigkeit von Schutzmechanismen zur Wahrung der Konsistenz
→ semantische Integritätssicherung

DBMS kann aber nur prüfen und gewährleisten, was an Bedingungen definiert wurde

Achtung! Optimierungsproblem –
Je mehr Integritätsbedingungen, desto langsamer das DBMS!!

Sicherung der semantischen Integrität

Sicherung der **semantischen Integrität**: die Gewährleistung der "Richtigkeit" und "Korrektheit" der Daten.

Bestandteile der Integritätsbedingungen:

- Angaben zur den Objekten (Attributwerte, Spalten, Tupel, Relationen) auf die sich die Integritätsbedingung bezieht,
- Auslöseregel, die angibt wann die Einhaltung der Bedingung überprüft werden soll (z.B. gleich nach Abschluß der Elementaroperation oder am Ende der Transaktion),
- Reaktionsregel, die angibt welche Aktionen bei der Verletzung der Bedingung auszulösen sind (z.B. Meldung an den Nutzer, Rücksetzen der Transaktion).

Anmerkung:

Die Integritätsbedingungen dürfen sich nicht nur auf die Zustände von Objekten beziehen, sondern müssen ebenso unerlaubte Zustandsübergänge zum Inhalt haben, z.B.:

- der Familienstand einer Person darf nicht von verheiratet in ledig geändert werden,
- der Primärschlüssel darf nicht geändert werden.

Möglichkeiten zur Sicherung der semantischen Integrität (1)

- **Formatkontrolle**

  Diese einfachste Form sichert, daß die definierten Datenformate eingehalten werden, z.B. daß keine Alphazeichen in Integerspalten eingetragen werden können oder daß Zeichenüberlauf verhindert bzw. signalisiert wird.

- **CHECK-Klauseln**

  Nehmen eine anwendungsbedingte Einschränkung der Wertebereiche der Attribute vor. Dienen der Plausibilitätskontrolle von Eingaben und Änderungen.

- **Domain**

  Festlegung der Wertebereiche der Attribute. Stellt eine Art Datentyp dar. Beschreibung einer Spalte durch Name, Datentyp, Länge, Standardwert und Integritätsbedingung.

- **Nutzerdefinierte Datentypen**

  Auf der Grundlage der vom DBMS angebotenen Datentypen und Integritätsbedingungen werden neue Datentypen entsprechend den Nutzererfordernissen kreiert. Analogien zu Domainen.

- **Starke Primärschlüssel und UNIQUE-Klausel**

  Beides dient der Zurückweisung von Duplikaten. Bei Primärschlüsseln müssen Duplikate ausgeschlossen werden, da sie die identifizierende Eigenschaft der Primärschlüssel beeinträchtigen.

Möglichkeiten zur Sicherung der semantischen Integrität (2)

- **Entity-Integrität**

  Primärschlüssel dürfen keine undefinierten Werte haben (s.o.). Angabe von NOT NULL.

- **Referentielle Integrität**

  Sicherung logischer Widersprüchsfreiheit inhaltlich in Beziehung stehender Spalten verschiedener Tabellen.

- **ASSERT-Anweisung**

  Prädikative, beschreibende Formulierung der semantischen Integrität. Integritätsbedingungen werden ähnlich wie Behauptungen formuliert.

- **Rules**

  Regeln zur Einschränkung der Wertebereiche der Attribute, die z.T. ähnliche Ausdrücke wie die WHERE-Klausel der SELECT-Anweisung enthalten können.

- **Trigger**

  Daten- und ereignisorientierte Prüfbedingungen und Sicherungsmaßnahmen, die die semantische Integrität auf prozeduralem Wege sichern.

- **Stored Procedure**

  Umfassende und flexible Prüf- und Sicherungsmöglichkeit, die alle Möglichkeiten des jeweiligen SQL-Dialektes nutzen kann und die in compilierter und optimierter Form auf dem SQL-Server bereitgestellt wird.

Spalten- und Tabellenbezogene Integritätsbedingungen

- **Constraint**

  Bedingung/Einschränkung, die an eine Spalte und/oder Tabelle gebunden ist

Schlüsselwort in SQL	Wirkungsweise	Wirkungskreis Spalte	Wirkungskreis Tabelle
NOT NULL	Missing Value werden nicht zugelassen	x	
UNIQUE	Dublikate werden abgelehnt	x	x
PRIMARY KEY	Duplikate werden abgelehnt	x	x
CHECK	Sichert Einhaltung d.Prüfbedingungen	x	x
REFERENCES	sichert referentielle Integrität	x	x
FOREIGN KEY	sichert referentielle Integrität		x

DEFAULTS und CHECK-Einschränkungen

Standardwerte / DEFAULTS

Definition an der Spalte einer Tabelle `DEFAULT <Wert>`

beim Anlegen einer neuern Tabelle oder beim Ändern einer vorhandenen Tabelle

Bsp.:
```
ALTER TABLE Kunde ADD Bundesland CHAR(20)
    CONSTRAINT DF_Kunde_Bundesland DEFAULT 'Sachsen'

ALTER TABLE Kunde ADD CONSTRAINT DF_Kunde_Ort
    DEFAULT 'Dresden' FOR Ort
```

Einschränkungen / CHECK

Definition an einer Tabelle: `CHECK (<logischer Ausdruck>)`

beim Anlegen einer neuern Tabelle oder beim Ändern einer vorhandenen Tabelle

Bsp.:
```
ALTER TABLE Kunde ADD CONSTRAINT CK_Kunde_Geschl
    CHECK (Geschl IN('M','W'))
```

Löschen über den Namen des CONSTRAINT

Bsp.:
```
ALTER TABLE Kunde DROP CONSTRAINT CK_Kunde_Geschl
```

DOMAINS / Nutzerdefinierte Datentypen und ASSERTIONS

DOMAINS / Nutzerdefinierte Datentypen

- Einschränkung der Wertebereiche der Attribute gegenüber den Standarddatentypen
- Einmal definiert, ermöglichen sie weiterhin die Vereinfachung und Vereinheitlichung weiterer Definitionen und wirken auch auf diese Weise integritätssichernd.

```
CREATE DOMAIN domainname [AS] datatype
                         [default-clause]
                         [domain-constraint]
                         [collate-clause]
domain constraint ::= [CONSTRAINT conname] CHECK (suchbedingung)
```

ASSERTION

Bei Assert-Anweisungen beschreibt man prädikativ welche Integritätsbedingungen einzuhalten sind. Man schreibt "Behauptungen", die dann vom DBMS zu überprüfen und einzuhalten sind.

```
CREATE ASSERTION constraintname CHECK (suchbedingung)
                        [DEFERRED | IMMEDIATE]
```

Referentielles Integritätsproblem

Das Problem der Sicherung des inhaltlichen "Zusammenpassens" zwischen Eigen- und Fremdschlüsselwert

Dabei ist folgendes zu gewährleisten:

- Ein Eigenschlüsselwert kann sich auf mehrere Fremdschlüsselzeichen beziehen, aber ein Fremdschlüsselwert darf sich nicht auf mehrere Eigenschlüsselwerte beziehen.
- Jeder Fremdschlüsselwert kann bei INSERT und UPDATE nur Werte vornehmen, die bereits als Eigenschlüssel vorhanden sind. Fremdschlüsselzeilen können problemlos gelöscht werden.
- Eigenschlüsselwerte, auf die Fremdschlüsselwerte Bezug nehmen, dürfen entweder nicht geändert oder gelöscht werden oder es müssen angemessene Kompensationsaktionen ausgelöst werden.

Das Einfügen von neuen Eigenschlüsselzeilen ist unproblematisch.

Foreign key Klausel

Syntax:

```
foreign key (spaltenname) references tabelle
[on delete {restrict | cascade | set null | set default}]
[on update {restrict | cascade | set null | set default}]
```

- **restrict**

  bedeutet, daß ein Versuch, eine Zeile von T1 zu löschen fehlgeschlagen wird, wenn irgendwelche referenzierte Zeilen in T2 existieren.

- **cascade**

  bedeutet, daß die referenzierte Zeile in T2 ebenfalls gelöscht wird

- **set null**

  bedeutet, daß Wert von T2.FK in den referenzierten Zeilen von T2 auf Null gesetzt werden (T2.FK darf in diesem Fall nicht als not null spezifiziert sein).

- **set default**

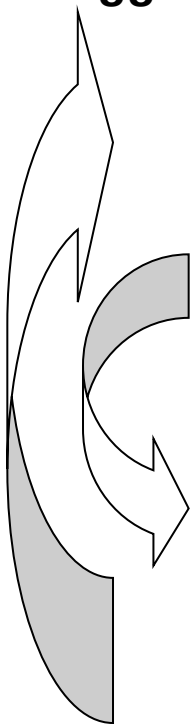
  bedeutet, daß Werte von T2.FK in den referenzierten Zeilen von T2 auf den anwendbaren Default-Wert gesetzt werden.

  (T2.FK muß in diesem Fall einen Default-Wert haben, weiterhin muß eine Zeile von T1 schon existieren die diesem Default-Wert als Wert von T1.PK hat)

Beachtung zyklischer Abhängigkeiten

Problem: Chicken/Egg

Zyklische Abhängigkeit



```
create table Abteilung(
Abtnr char(3) not null primary key,
Abtname char(15) not null,
…
Abtleiter char(5)  );
```

REFERENCES Mitarbeiter

```
create table Mitarbeiter (
Mitnr char(5) primary key,
Name char(10),
…
Abtnr char(3) REFERENCES Abteilung);
```

```
ALTER TABLE Abteilung
ADD CONSTRAINT co_forkey
FOREIGN KEY(Abtleiter) REFERENCES Mitarbeiter(Mitnr);
```

Benutzerdefinierte Funktionen

	Skalarwertfunktion	Tabellenwertfunktion
Rückgabewert:	ein einzelner Wert	eine Tabelle
Einsatz in einer Abfrage:	wie ein Spaltenname (nach dem *select*)	wie ein Tabellenname (nach dem *from*)

```
CREATE FUNCTION <funktionsname>
([@parameter_name1 <datentyp1> [,...]])
RETURNS [<datentyp> | @tabellenname TABLE <tabellendefinition>]
AS
  BEGIN
       <SQL_statements>
       RETURN [rueckgabewert]
  END

      <SQL_statements >::= Befehle von Transact-SQL zum Ermitteln
                           des Rückgabewertes
```

Beispiel für eine Skalarwertfunktion

Skalarwertfunktion zur Ermittlung des Gebührensatzes der Kreditbearbeitung in Prozent in Abhängigkeit der als Parameter übergebenen Kredithöhe:

```
CREATE FUNCTION Gebuehrensatz(@Kredit int)
RETURNS DECIMAL(28,10)
AS
  BEGIN
    DECLARE @Gebuehrsatz decimal(28,10)

    IF @Kredit > 1000
      SELECT @Gebuehrsatz = 0.5
    ELSE
      IF @Kredit > 500
         SELECT @Gebuehrsatz = 0.8
      ELSE
         SELECT @Gebuehrsatz = 1.2
    RETURN @Gebuehrsatz
  END
```

Beispiel für eine Tabellenwertfunktion

Tabellenwertfunktion zur Anzeige von Kunr, Name, Vorname der Kunden für einen Ort aus der Tabelle Kunde :

```
CREATE FUNCTION ErmittleKundenImOrt (@ort
char(20))
RETURNS @table TABLE
( Kunr          int,
  Name          char(20),
  Vorname       char(10)
)
AS
BEGIN
   INSERT INTO @table
       SELECT Kunr, Name, Vorname
       FROM    Kunde
       WHERE Ort = @ort
RETURN
END
```

Trigger

Trigger lösen beim Eintreten bestimmter Ereignisse für bestimmte Datenobjekte eine Folge von Aktionen aus.

Man spricht in diesem Zusammenhang auch von ECA-Regeln (event - condition - **action rules**; Ereignis - Bedingungen - Aktionsregeln).

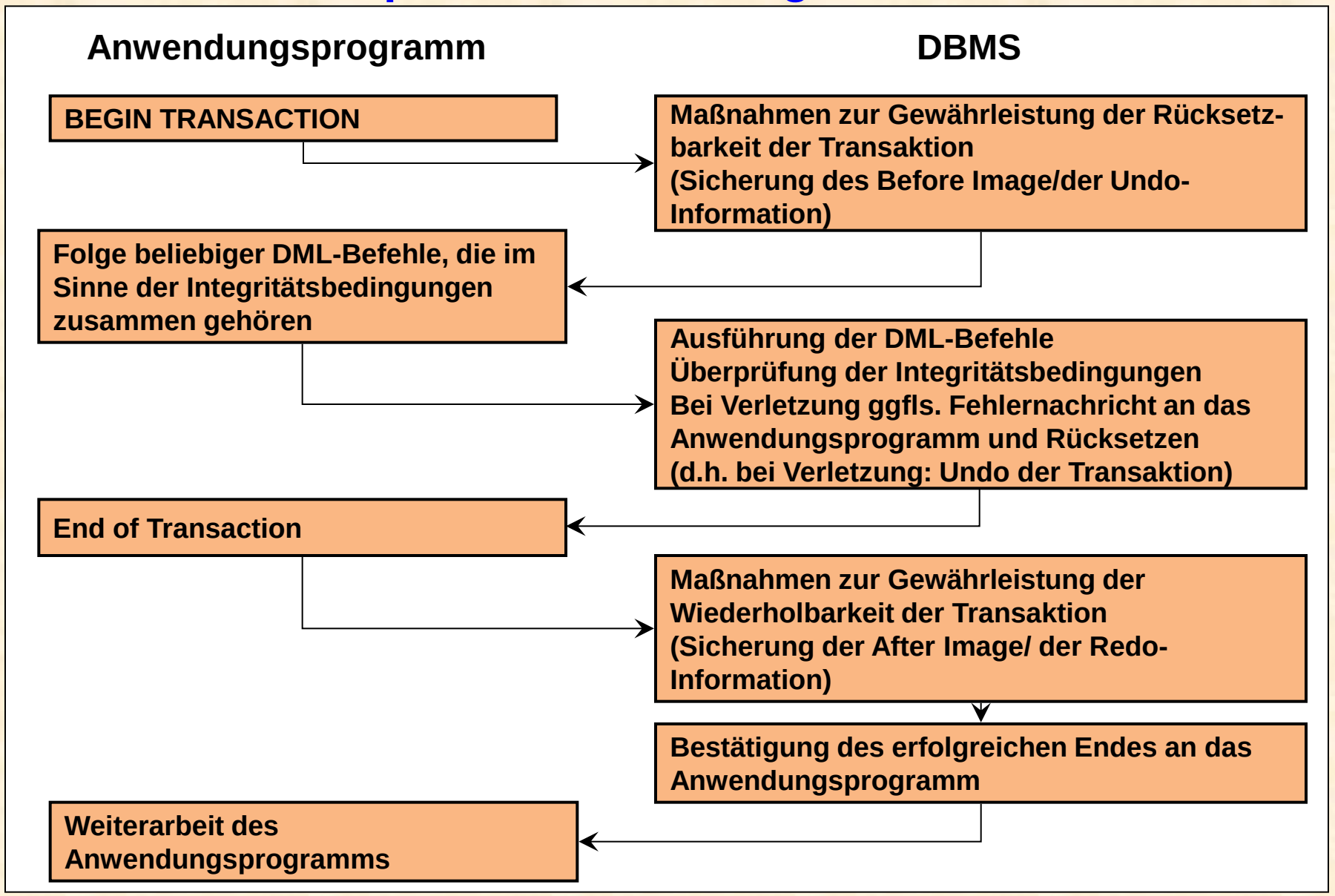
Anmerkungen:

- Besonders durch Trigger können Datenbanken zu „aktiven Datenbanken“ werden.
- Bei der Benutzung „reagieren die Datenbanken nach dem Eintreten bestimmter Bedingungen von allein“, sie sind aktiv. Das sind die Datenbanken natürlich nur, wenn der Entwickler vorher entsprechende Trigger programmiert hat.
- Mit Triggern kann man auch die referentielle Integrität sichern (in prozeduraler Art und Weise).

Syntax:

```
CREATE TRIGGER <trigger_name> ON <table_name>
        {FOR {INSERT, UPDATE, DELETE} AS SQL_statements|
         FOR  {INSERT, UPDATE} AS IF UPDATE (column_name)
                        [{ AND|OR} UPDATE (column_name)]
                                SQL_statements}
<SQL-statements >::= Befehle von Transact-SQL zur Formulierung
                     der Integritäts-bedingungen und Aktionen
```

Zusammenspiel zwischen Programm und DBMS



Beispiele der Arbeit mit Triggern (1)

1. Protokollierung von Änderungen (UPDATE) der Mittel in der Tabelle Projekt

Tabelle Project

Projnr	Proj_bezeich	Mittel	....	...
1234	DV-Projekt CAD-Hausbau	~~150.000,00~~ 180.000,00	...	
1235	Statistik-Projekt	125.000,00	...	

Tabelle Mittel_protokoll

```
CREATE TABLE Mittel_protokoll
        (Projnr char(4) null,
        Benutzer char(16) null,
        Zeit datetime null,
        Mittel_alt float null,
        Mittel_neu float null)
```

Projnr	Benutzer	Zeit	Mittel_alt	Mittel_neu
1234	MeierU	19.12.09 12:05:20	150000,00	180000,00

Beispiele der Arbeit mit Triggern (2)

2. Änderungen in den Mitteln der Tabelle Project sind nur zulässig, wenn die Erhöhung der Gesamtmittel kleiner als 50% ist

Tabelle Project

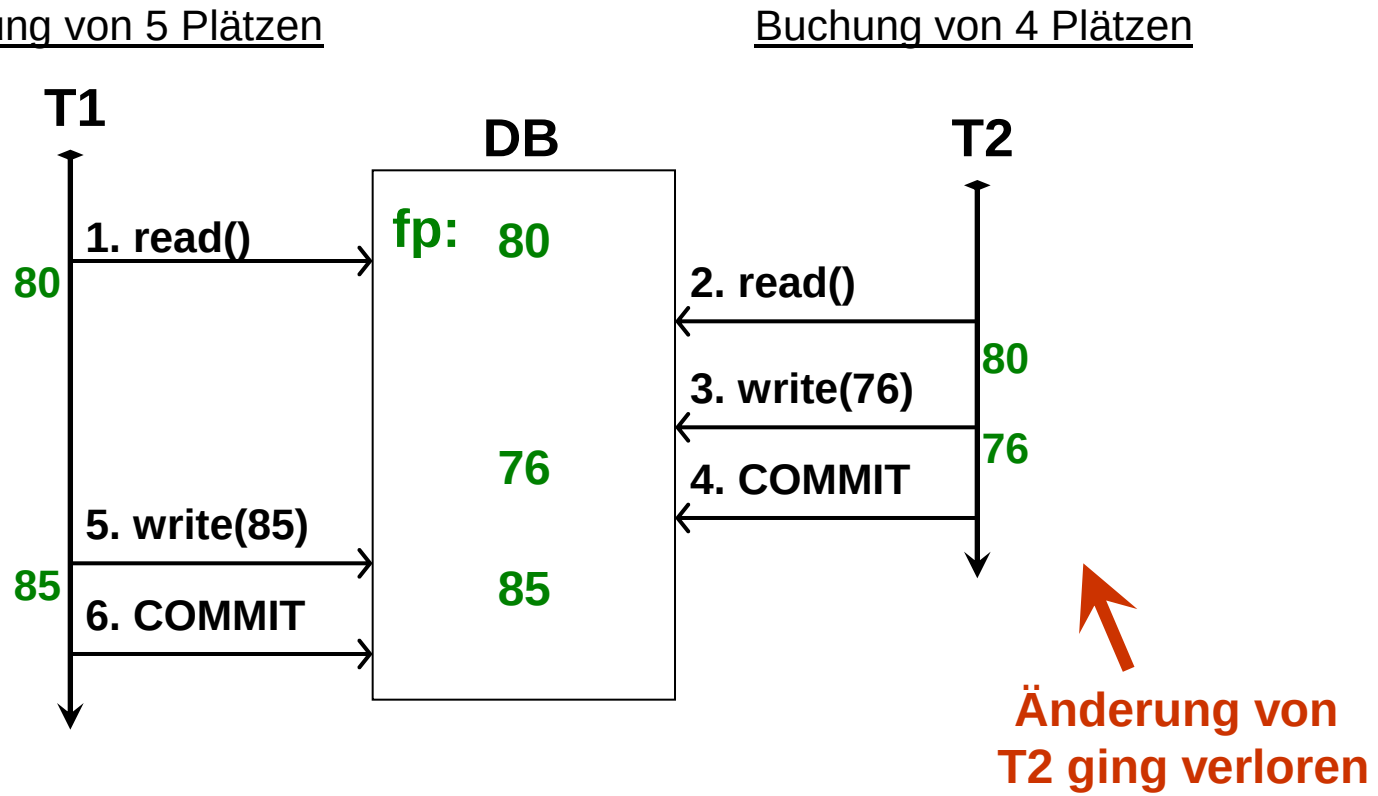
Projnr	Proj_bezeich	Mittel	....	...
1234	DV-Projekt CAD-Hausbau	150.000,00	...	
1235	Statistik-Projekt	125.000,00	...	

```
CREATE TRIGGER Mittel_pruef ON Project
    FOR UPDATE AS
    IF UPDATE(Mittel)
    BEGIN
        DECLARE @alte_summe float
        DECLARE @neue_summe float
        SELECT @alte_summe = (SELECT SUM(Mittel) FROM DELETED)
        SELECT @neue_summe = (SELECT SUM(Mittel) FROM INSERTED)
        IF @neue_summe  > @alte_summe * 1.5
              BEGIN
              ROLLBACK TRANSACTION
              PRINT "Die Änderung der Projektmittel nicht ausgeführt!"
              END
        ELSE
              PRINT "Die Änderung der Projektmittel wurde ausgeführt"
    END
```

Lost Update

read() … Lesen der Anzahl freier Plätze in der Buchungstabelle
write(fp) … Schreiben der Anzahl freier Plätze in der Buchungstabelle

Bsp. Buchungssystem: Ablauf der beiden Transaktionen



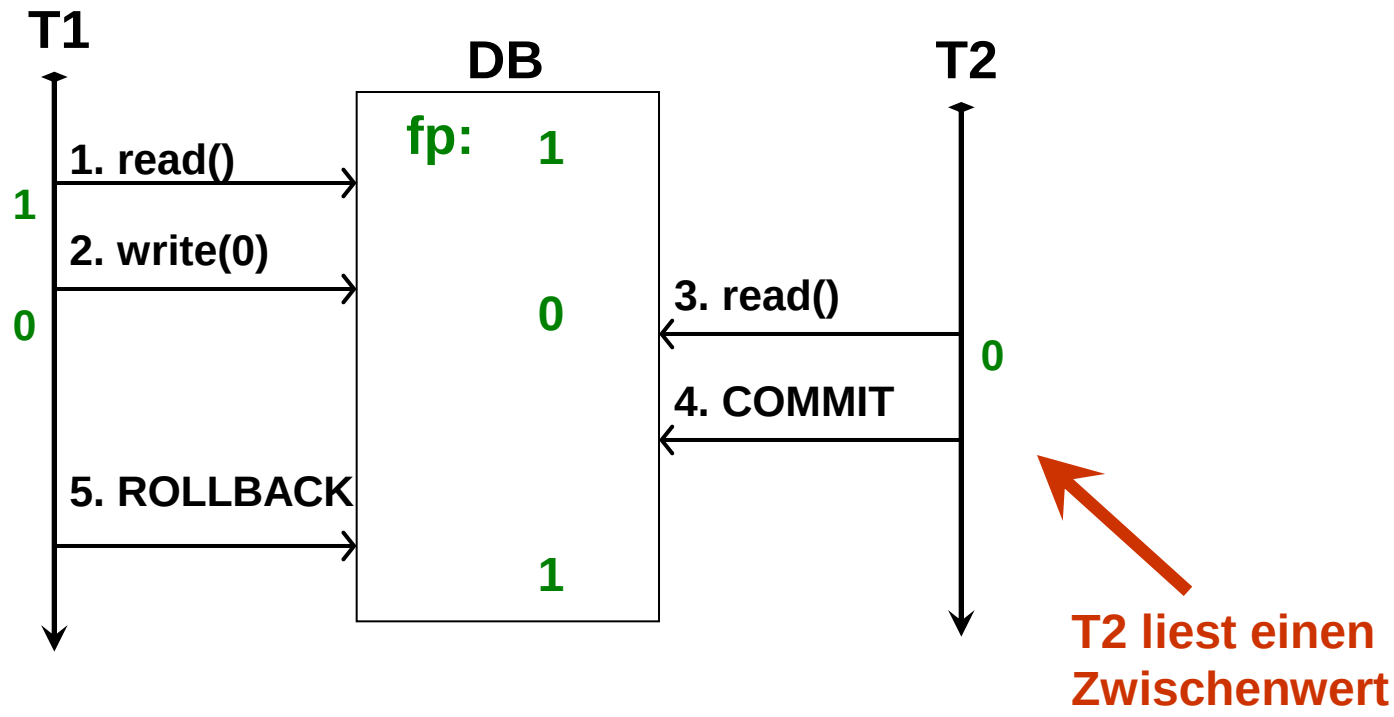
Dirty Read

read() … Lesen der Anzahl freier Plätze in der Buchungstabelle
write(fp) … Schreiben der Anzahl freier Plätze in der Buchungstabelle

Bsp. Buchungssystem: Ablauf der beiden Transaktionen

Buchung von 1 Platz mit Rücksetzen

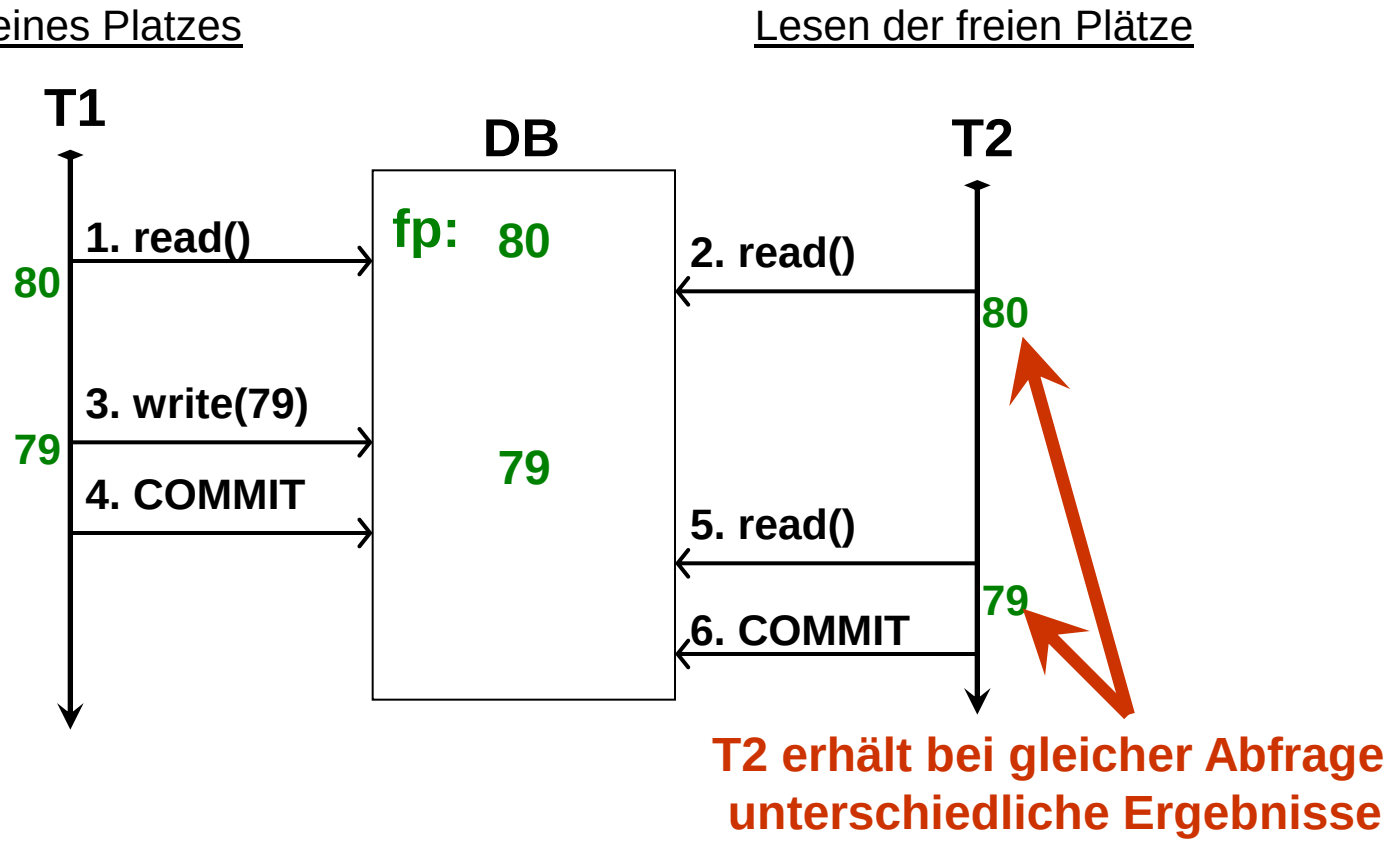
Versuch einer Buchung



Non repeatable Read

read() … Lesen der Anzahl freier Plätze in der Buchungstabelle
write(fp) … Schreiben der Anzahl freier Plätze in der Buchungstabelle

Bsp. Buchungssystem: Ablauf der beiden Transaktionen



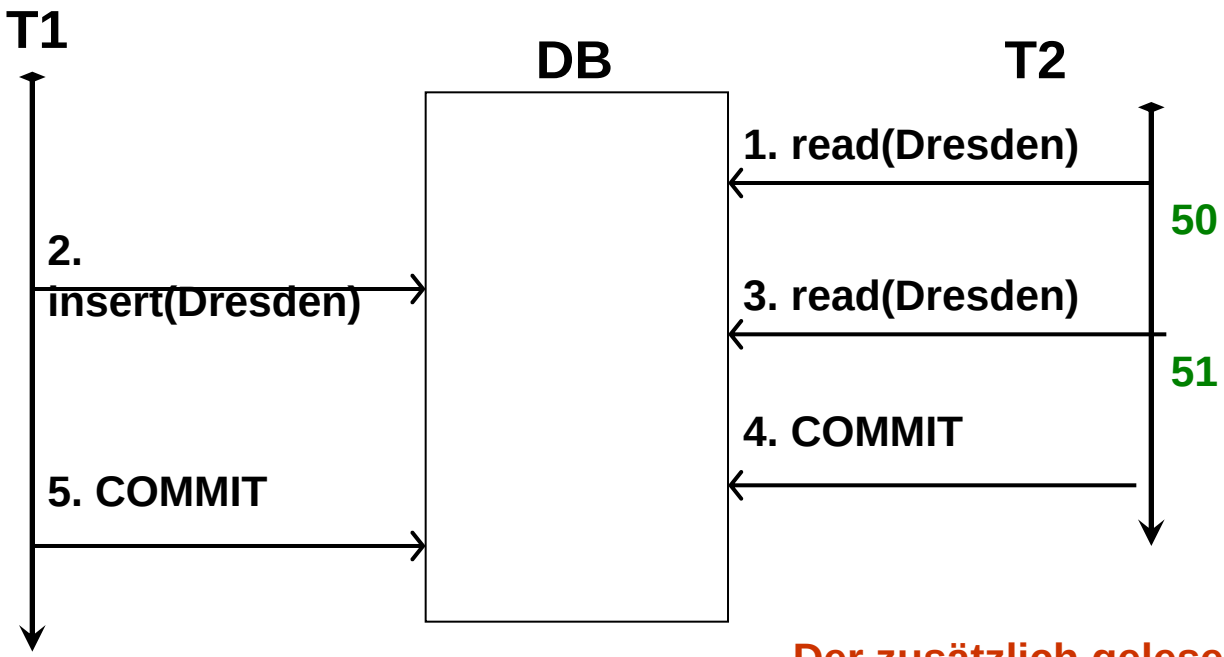
Phantomproblem

insert(Dresden) … insert into Katalog (…,Ziel,…) values (… , ‘Dresden‘, …)
read(Dresden) … select count(*) from Katalog where Ziel = ‘Dresden‘

Bsp. Buchungssystem: Ablauf der beiden Transaktionen

Einfügen eines weiteren Ziels in Dresden

Lesen der Anzahl an Zielen in Dresden



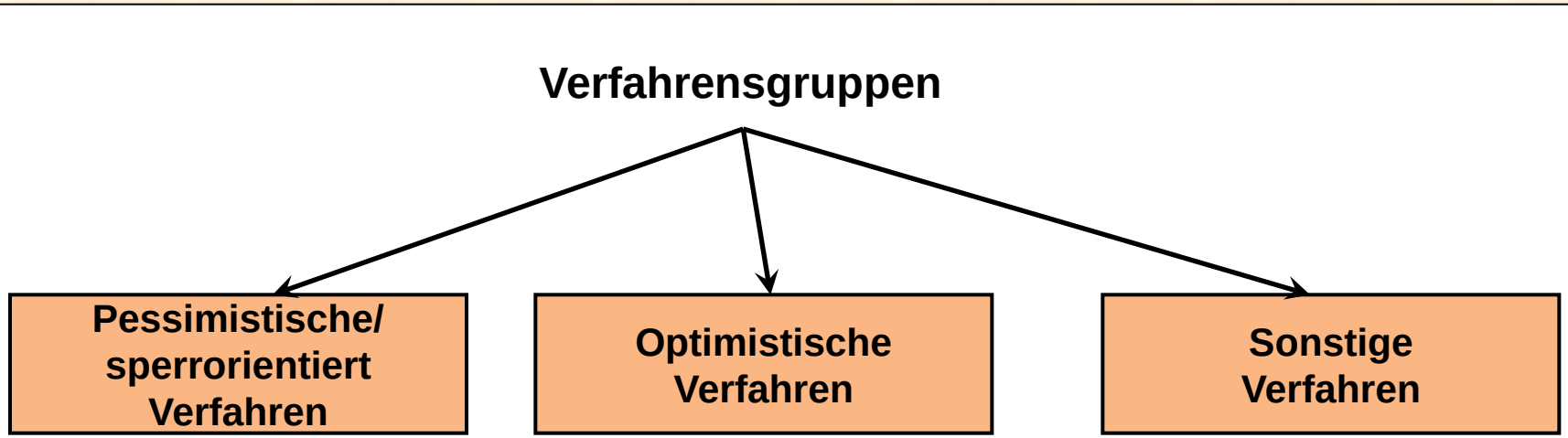
Der zusätzlich gelesene Datensatz wird als Phantom bezeichnet.

Sicherung der operationalen Integrität

Sicherung der **operationalen Integrität** heißt, Vermeidung von Nebenwirkungen des Mehrnutzerbetriebes durch die Realisierung eines fiktiven Einzelnutzerbetriebes und die Gewährleistung der Eigenschaft der Isolation einer Transaktion.

Das wird auch als Synchronisation oder Serialisierung (Concurrency Control-Nebenläufigkeitsteuerung) bezeichnet.

Transaktionssystem



Sperrverfahren

Bei den **Sperrverfahren** werden Schreib -bzw. Lesesperren als Mechanismus zur Gewährleistung der Serialisierungbarkeit benutzt.

So werden 5 Bedingungen dafür genannt, dass bei "nach außen" gleichzeitiger Ausführung von n Transaktionen das gleiche Ergebnis erhalten wird wie bei einer seriellen Abfolge:

1. Jedes Datenobjekt auf das innerhalb einer Transaktion zugegriffen werden soll, muss vorher gesperrt werden.
2. Eine Transaktion muss Sperren, die von anderen Transaktionen gesetzt werden beachten.
3. Eine Transaktion fordert Sperren, die sie bereits erworben hat, nicht noch einmal an.
4. Jede Transaktion durchläuft 2 Phasen:
   - eine Wachstumsphase, in der sie Sperren anfordert, aber keine freigibt und
   - eine Schrumpfungsphase in der sie bisher erworbene Sperren freigibt, aber keine neuen Sperren erwirbt (2-Phasen-Sperrprotokoll).

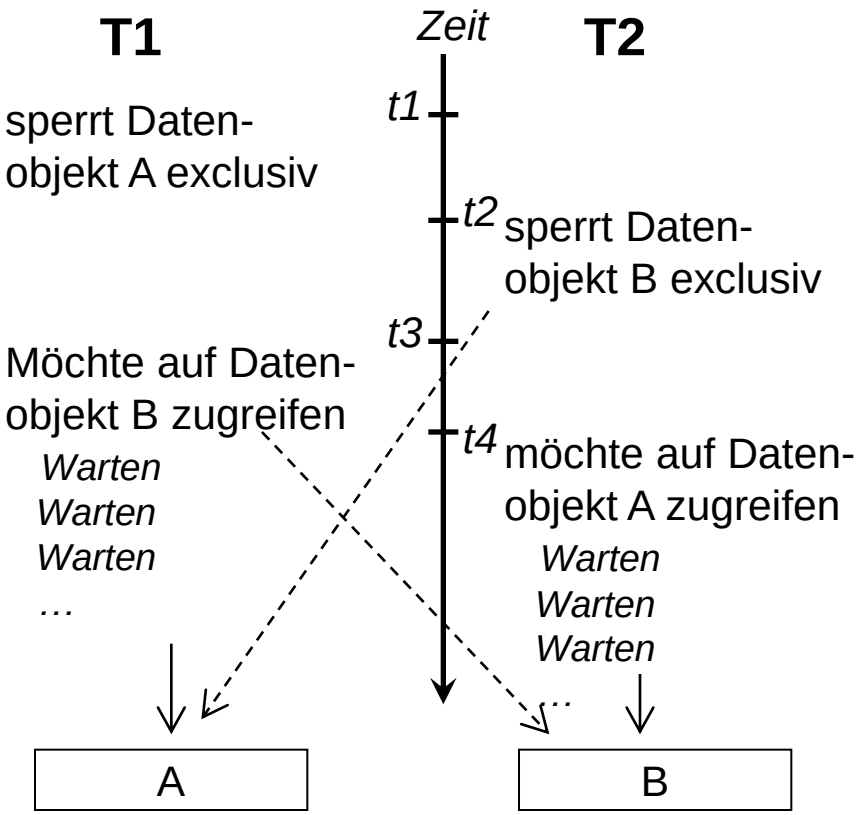
   Bei Einhaltung des 2-Phasen-Sperrprotokolls ist die Serialisierbarkeit der Transaktionen gegeben.
5. Spätestens am Ende der Transaktion sind alle Sperren wieder freizugeben.

Statische- und dynamische Sperren

Das Sperren kann statisch oder dynamisch erfolgen:

- Beim **statischen Sperren** werden zu Beginn der Transaktion alle Sperren angefordert (preclaiming). Man sperrt alles, was man "gebrauchen könnte". Also manchmal auch etwas mehr.
- Beim **dynamischen Sperren** werden während der Transaktion (aber nicht in der Schrumpfungsphase) Sperren nach Bedarf angefordert. Dadurch kann es zu Verklemmungen (**Deadlocks**) kommen.

Deadlocksituation bei dynamischen Sperren



Viele DBMS nutzen das strikte 2-Phasen-Sperrprotokoll, d.h. sämtliche Sperren werden erst am Ende der Transaktion freigegeben

Anforderungen und Sperrmodi

- Zur Vermeidung der vorn aufgeführten Anomalien (außer Phantomanomalie) ist ein striktes 2-Phasensperrprotokoll mit Preclaiming sinnvoll.
- Vom Standpunkt der Effizienz des DBMS, eines möglichst großen Durchsatzes von Transaktionen, sollen
  - die Sperren möglichst kurz sein,
  - nur berechtigte Sperren gesetzt werden,
  - so wenig wie möglich Datenobjekte gesperrt werden.

Sperrmodi

S-Sperre (shared) - Lesesperre

Dabei können Datenobjekte, die mit einer S-Sperre belegt sind, auch von anderen Transaktionen mit einer S-Sperre versehen und von diesen gelesen werden. Schreiben und das Setzen von Schreibsperren wird abgewiesen.

X-Sperre (exclusive) - Schreibsperre

Dabei kann auf Datenobjekte, die mit einer X-Sperre versehen sind, von keiner anderen Transaktion (weder lesend noch schreibend) zugegriffen werden.

Stufen der Isolation paralleler Transaktionen

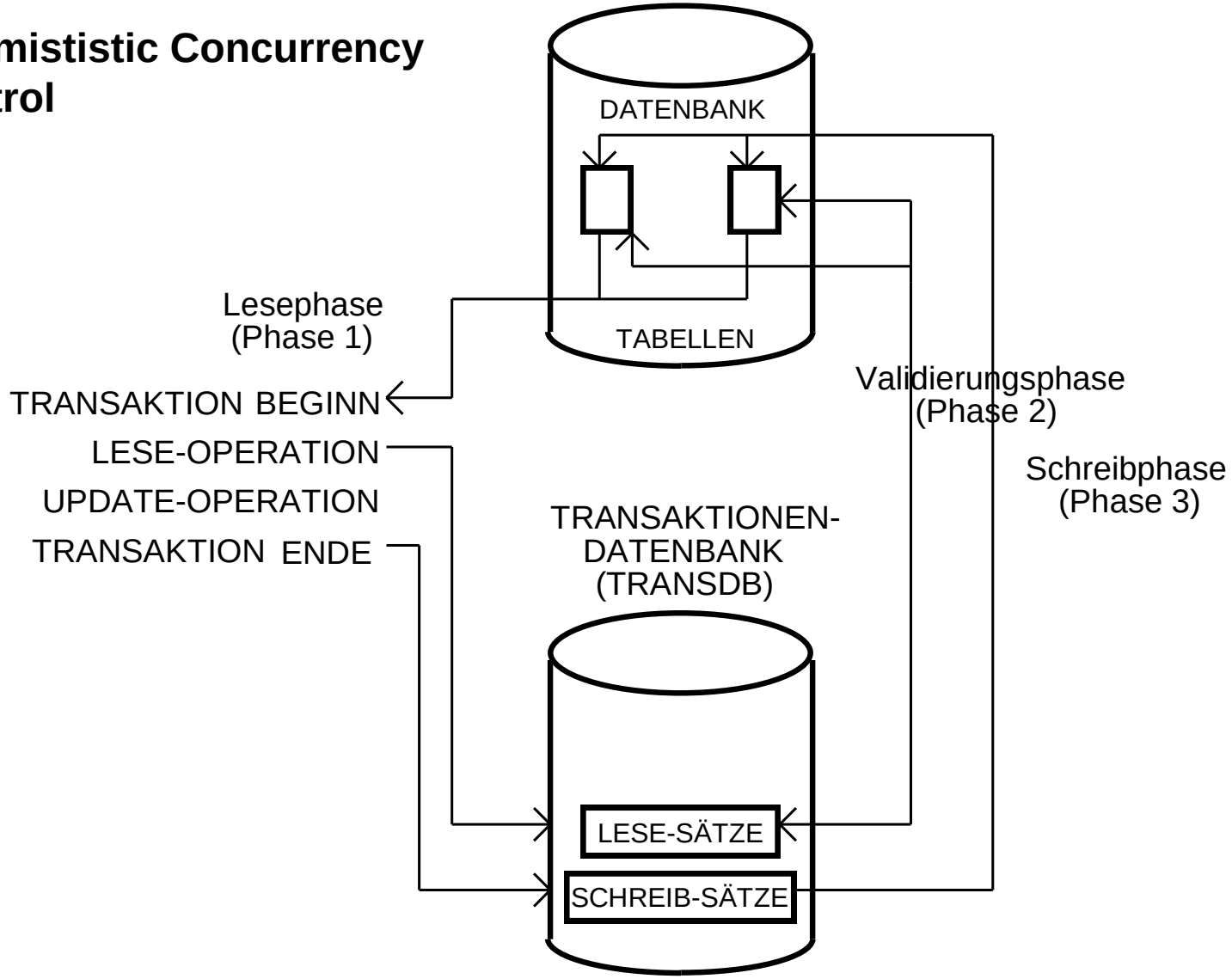
Es wird zwischen kurzen und langen Sperren unterschieden:

- **kurze Sperren** wurden sofort nach Gebrauch des Datenobjektes, also vor EOT freigegeben,
- **lange Sperren** werden bis EOT gehalten.

Aus sinnvollen Abstufungen ergeben sich 4 Konstistenz- oder Isolationsebenen.

Ebene	Bezeichnung	Inhalt
0	isolation level read uncommitted	Transaktion hält kurze X-Sperren auf den von ihr geänderten Datenobjekten
1	isolation level read commited	Transaktion hält lange X-Sperren auf den von ihr geänderten Datenobjekten
2	isolation level repeatable read	Transaktion hält lange X-Sperren auf den von ihr geänderten und kurze S-Sperren auf den von ihr gelesenen Datenobjekten
3	isolation level serializable	Transaktion hält lange X- und lange S-Sperren auf den von ihr geänderten bzw. gelesenen Datenobjekten

Dreiphasen-Ablauf bei Optimistischer Synchronisation



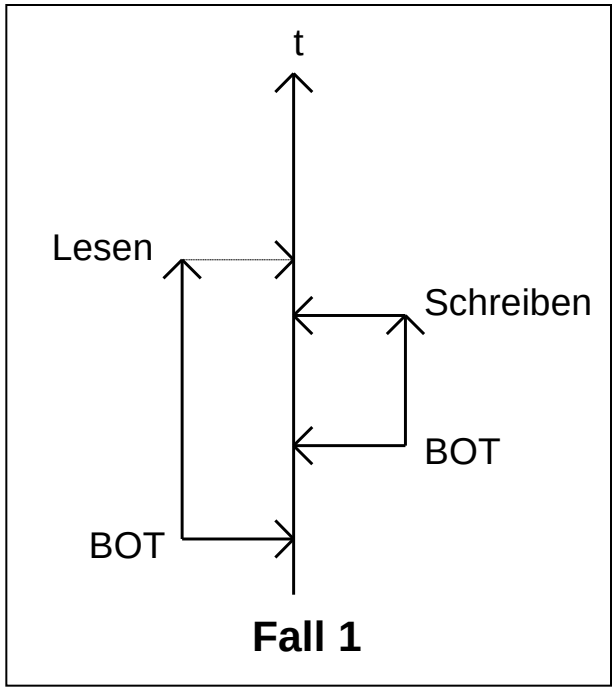
Zeitstempelverfahren

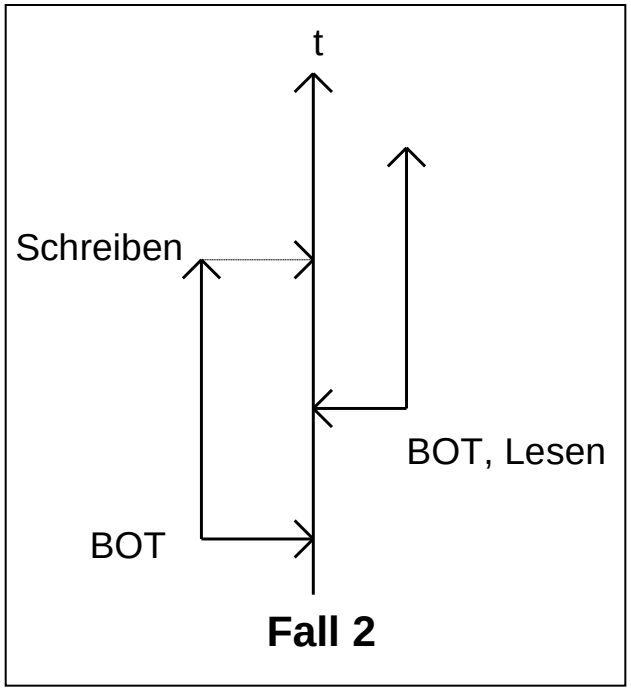
Basis: Jeder Transaktion erhält zu ihrem Beginn einen Zeitstempel.

- Jedes gelesene Datum erhält dies Marke als Lesezeitmarke.
- Jedes geschriebene Datum erhält diese Marke als Schreibzeitmarke.

Regeln:

- Keine Transaktion darf ein Datum lesen, dessen Schreibzeitmarke jünger (oder höher) ist als die der Transaktion (Fall 1).
- Keine Transaktion darf ein Datum ändert, dessen Lesezeitmarke jünger (oder höher) ist als die der Transaktion (Fall 2).





Sicherung der physischen Integrität

Sicherung der Vollständigkeit der Daten zur Vermeidung von Datenverlusten

Ziel:

- Nach einem Fehler automatisch, ohne Eingriffe des Datenbank-administrators, einen Zustand in der Datenbank wieder herstellen, in dem die Ergebnisse sämtlicher erfolgreich abgeschlossener Transaktionen und keine Ergebnisse gescheiterter Transaktionen enthalten sind.

Gewährleistung der Transaktionseigenschaften:

- Atomarität
- Dauerhaftigkeit

Strategie:

- bestimmte Daten/ Datenbestände duplizieren und auf einen anderen, 2. Datenträger speichern
  (Benutzt werden dafür spezielle Dateien, die ***Protokoll-, Log- oder Journaldateien*** genannt werden.)

Eintragungen in der Protokolldatei

- **Undo-Information** oder **Before Image**
  - Zustand der beteiligten Datenobjekte wie er zu Beginn der Transaktion war
  - Undo-Informationen ("ungetan") weist auf das Rücksetzen hin.
  - Before-Image weist auf den Zustand vor der Änderung hin.
  - Forderung: "write ahead log" - WAL-Prinzip
- **Redo-Information** oder **After-Image**
  - Zustand der beteiligten Datenobjekte nach dem Commit, d.h. noch bevor das Ende der Transaktion dem Programm/Nutzer mitgeteilt wird,
  - Damit die Ergebnisse abgeschlossener Transaktionen nicht verloren gehen können, wird dieser Zustand auf der Sicherungsplatte protokolliert.
  - Redo-Information ("noch einmal tun") weist auf das Wiederherstellen hin.
  - After-Image auf den Zustand nach der Änderung hin.
  - Forderung "write ahead log" - WAL-Prinzip gilt auch
  - LRU - last recently used bedeutet, dass bei Platzmangel im Datenpuffer des Hauptspeichers die am längsten nicht benutzte Seite auf die Platte geschrieben wird.

Einbringstrategien

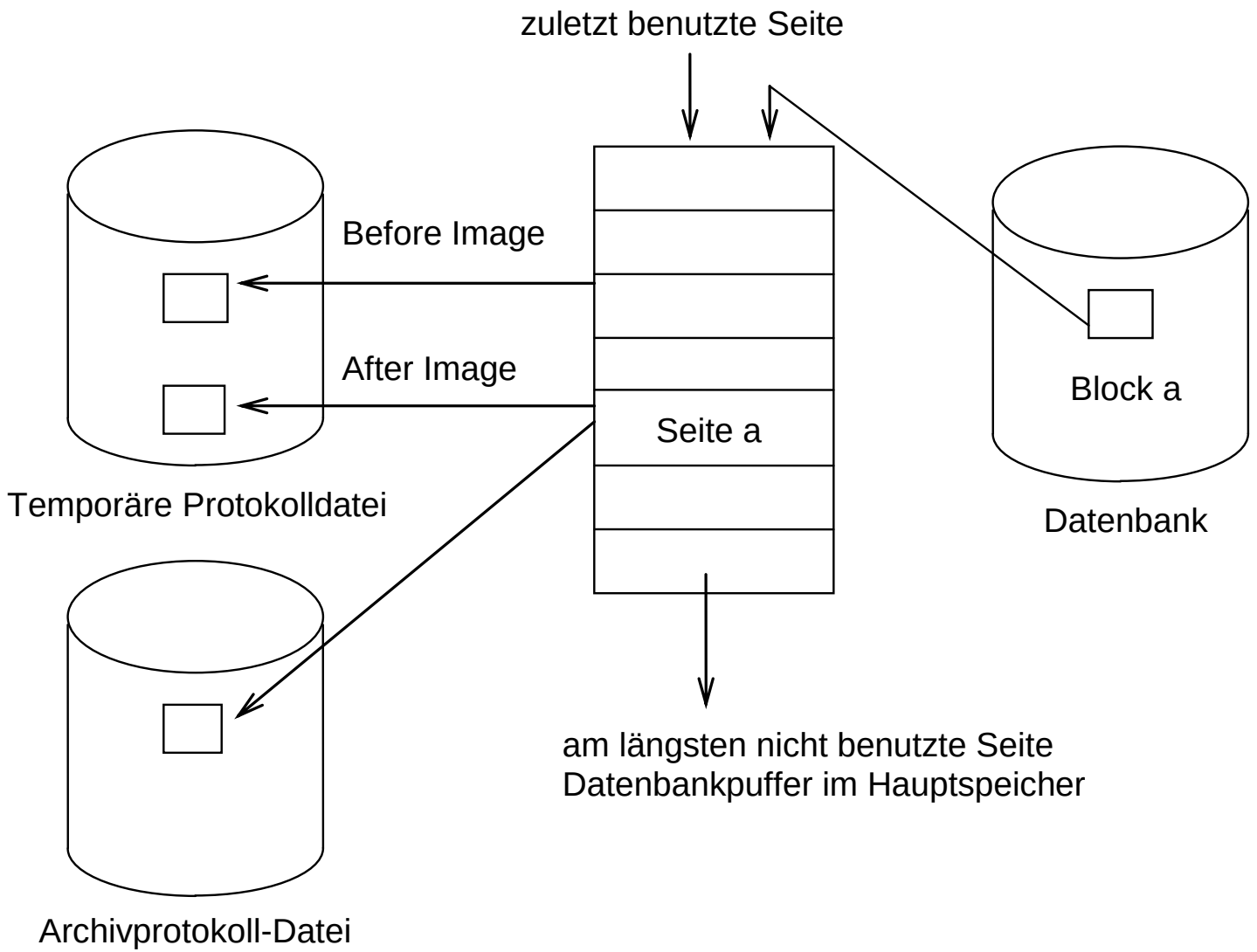
direkt
Schreiben der geänderten Seiten über den alten Inhalt desselben Blockes aus dem sie gelesen wurden

indirekt
Schreiben der Seiten in einen anderen noch freien Block geschrieben

Durch das indirekte Einbringen nachdem Schattenspeicherkonzept haben die ursprünglichen Seiten (Schattenseiten) noch eine Zeit bestand und können als Before Image verwendet werden

- Bei **logischer Protokollierung** werden die ausgeführten Operationen bzw. ihre universen Operationen oder die Datenobjekte auf einer logischen Ebene (meistens die Tupel) protokolliert.
- Bei **physischer Protokollierung** werden Datenobjekte der physischen Ebene, in der Regel die Datenseiten, protokolliert.
- Letztlich unterscheiden sich die Protokollverfahren noch dadurch, ob sie **Zustände** (Tupel/Sätze oder Seiten) oder **Zustandsübergänge** (Operationen/Befehle) sichern.

Prinzip der physischen Protokllierung (1)



Prinzip der physischen Protokllierung (2)

Dem Bild auf der vorherigen Folie liegt folgendes Beispiel zugrunde:

1. In einer Seite S wird ein Datenobjekt a eingefügt.

   Dadurch geht der Zustand der Seite von S (1) nach S (2) über.

2. In der gleichen Seite S wird ein Objekt b von b alt in b neu geändert.

   Dadurch geht der Zustand der Seite von S (2) in S (3) über.

	logisch	physisch
Zustände	Tupel a Tupel b alt Tupel b neu	Protokollierung der Before- und After Images 1. S (1) und S (2) 2. S (2) und S (3)
Übergänge	Protokollierung der Operationen mit Parametern: 1. INSERT 2. UPDATE (b alt, b neu)	Differenzenlogging (kaum Bedeutung) 1. S (1) **+** S(2) 2. S (2) **+** S(3)

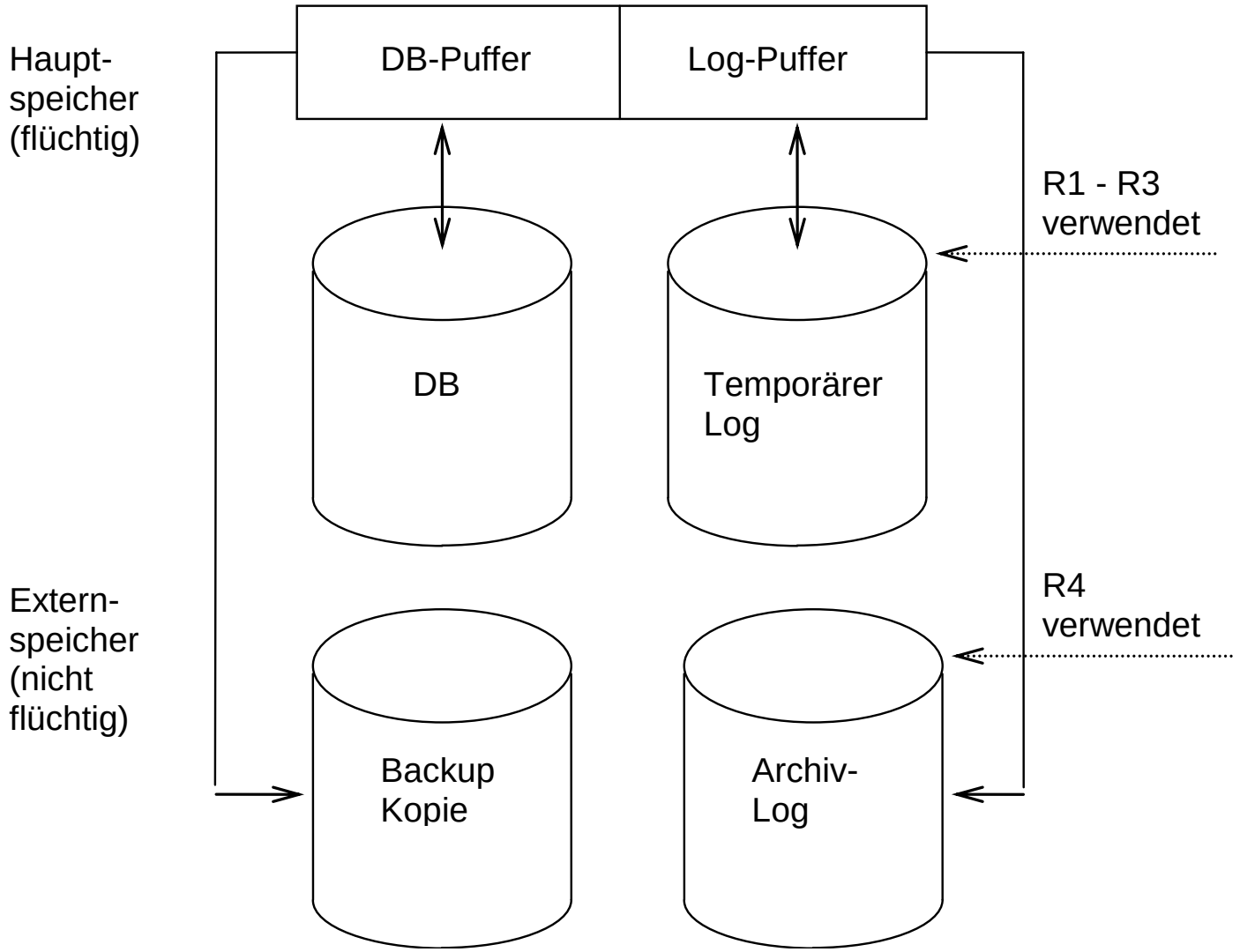
Wiederherstellungsverfahren

Transaktionsfehler (z.B. Verletzung der semantischen Integrität	Transaction UNDO (R1) partielles Zurücksetzen	Zurücksetzten gescheiterter Transaktionen; beeinflusst andere TA nicht
Systemfehler (Verlust des Haupt-speicherinhalts	Partial REDO (R2) partielles Wiederholen	Wiederholung derjenigen abge-schlossenen Transaktionen, deren Ergebnisse verloren gegangen sind
	Global UNDO (R3) vollständiges Zurücksetzen	Zurücksetzen aller gescheiterten Transaktionen
Medienfehler (Plattenfehler)	Global REDO (R4) vollständiges Wiederholen	Wiederholen aller abgeschlossenen Transaktionen, die nach der letzten Plattensicherung ausgeführt wurden

Zusammenhang zwischen Hauptfehlerquellen und Recoveryverfahren:

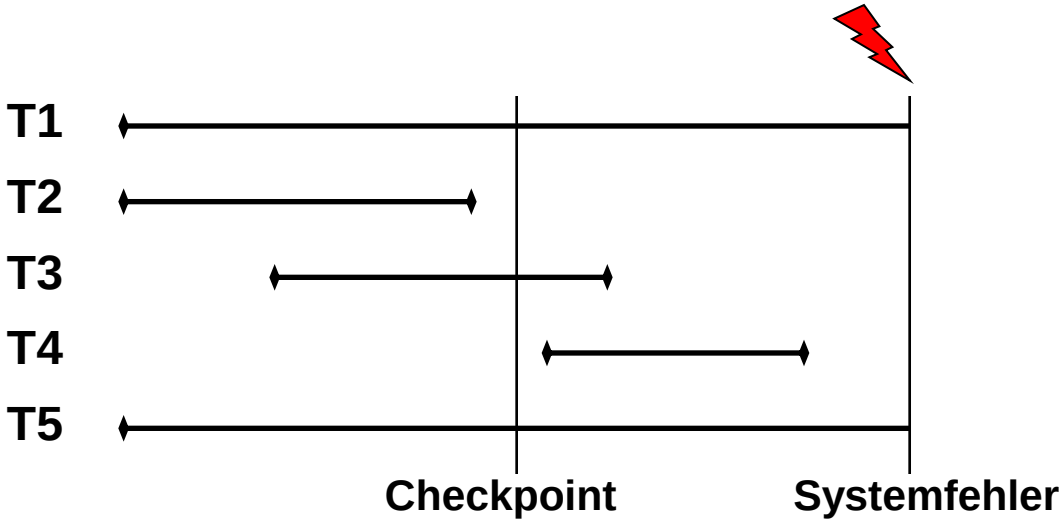
- **R1-Recovery** oder Transaction UNDO oder partiell Rollback oder intransaction backout oder partielles Zurücksetzen
- **R2-Recovery** oder partial REDO oder partielles Wiederholen
- **R3-Recovery** oder global UNDO oder globales Rücksetzen
- **R4-Recovery** oder global REDO oder vollständiges Wiederholen oder Restore

Zusammenhang zwischen Protokoll- und Recoveryverfahren



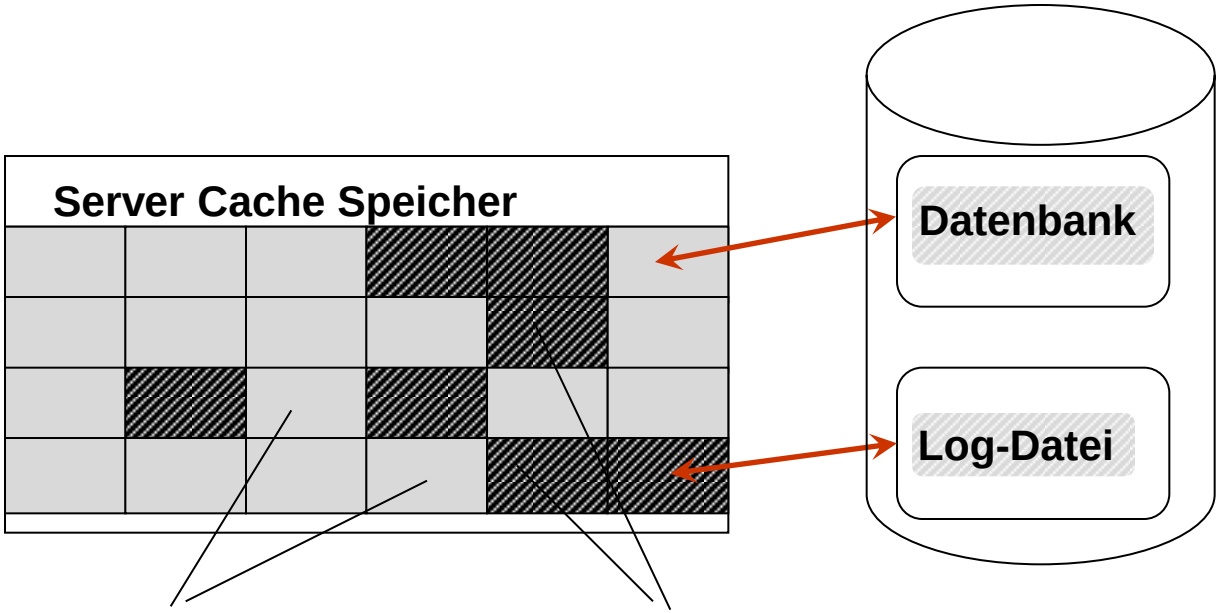
Weitere Recoveryverfahren

- Transaktionsinternes Zurücksetzen mittels sogenannter **savepoints** (oder auch Binnensicherungspunkte genannt).
- Kompensatives Recovery
- Notmaßnahmen für den Fall, dass die temporären Protokolldateien auch defekt sind.
- Konzept der "externen" Sicherungspunkte (Checkpoints)



T1 - nicht beendet, UNDO notwendig (bis Checkpoint)
T2 - beendet und Ergebnisse durch Checkpoint in die Datenbank eingebracht
T3 - beendet, REDO ab Checkpoint
T4 - vollständiges REDO notwendig
T5 - nicht beendet, UNDO notwendig

Übertragung Datenbankpuffer-Platte



Datenseiten

werden ausgelagert bei
- Platzbedarf
- CHECKPOINT

Log Seiten

werden ausgelagert bei
- COMMIT
- Platzbedarf
- CHECKPOINT

Hochverfügbarkeit (1)

Hochverfügbarkeit: Ein System gilt als hochverfügbar, wenn eine Anwendung auch im Fehlerfall weiterhin verfügbar ist und ohne unmittelbaren menschlichen Eingriff weiter genutzt werden kann.

→ *Hochverfügbarkeit (abgekürzt auch HA, abgeleitet von engl. High Availibilty) bezeichnet also die Fähigkeit eines Systems, bei Ausfall einer seiner Komponenten einen uneingeschränkten Betrieb zu gewährleisten.*

Verfügbarkeitsklassen:

3	99,9%	43,80 min/Monat oder 8,76 h/Jahr Downtime
4	99,99%	4,38 min/Monat oder 52,6 min/Jahr
5	99,999%	0,44 min/Monat oder 5,26 min/Jahr

Hochverfügbarkeit verursacht bei steigenden Anforderungen überproportionale Kosten!

Kosten ≤ Schaden * Eintrittswahrscheinlichkeit

Kosten für Verfügbarkeit:

- Hardware
- Software (Lizenzen)
- erhöhter Aufwand für Wartung und Monitoring

Hochverfügbarkeit (2)

Technologien zur Sicherung der Hochverfügbarkeit:

- Festplattenspiegelung
- Clusterdienst
- Datenbank-Spiegelung mit Replikation

Datenbank-Spiegelung mit Replikation:

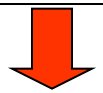
- Kopie der Datenbank auf zweiter Instanz eines SQL Servers
- Spiegeldatenbank erhält Änderungen über dedizierte LAN-Verbindung
- Automatischer Failover benötigt zusätzlichen Zeugenserver (Witness)

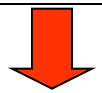
Beachtung des Konflikts Performance – Transaktionssicherheit

→ **"Hot or Cold Standby"**

Datenschutz und Zugriffsschutz

Datenschutz im engeren Sinne	Datenschutz im weiteren Sinne
Schutz personenbezogener Daten	Schutz aller betrieblichen Daten
vor bewusster und unbewusster Manipulation / Spionage / Zerstörung und vor Datenverlust	
BundesDatenSchutzGesetz (BDSG)	Zugriffsschutz





Öffentliches Recht

Access & Identity Management

Richtlinien und Vorschriften

- nationale und internationale Richtlinien
- BDSG und Datenschutzgesetze der Länder
  - regelt den Umgang mit personenbezogenen Daten
  - Erhebung, Nutzung und Verarbeitung von Daten nur bei Zustimmung der Person oder durch Rechtsgrundlage erlaubt
- IT Richtlinien
  - Namensrichtlinie
  - Passwortrichtlinie
  - Sicherheitsrichtlinie

Datenschutz

Probleme beim Datenschutz

Schutz der Persönlichkeitssphäre ⟺ **Erhalt der Funktionsfähigkeit und Effizienz in Staat und Wirtschaft**

Maßnahmen des Datenschutzes (nach BDSG)

1. Zutrittskontrolle
2. Zugangskontrolle
3. Zugriffskontrolle
4. Weitergabekontrolle
5. Eingabekontrolle
6. Auftragskontrolle
7. Verfügbarkeitskontrolle
8. Gebot der Datentrennung

Rechte der Bürger nach BDSG

- Recht auf Auskunft über die gespeicherten Daten
- Recht auf Berichtigung fehlerhafter Daten
- Recht auf Sperrung von Daten (best. Daten vor Weitergabe bewahren)
- Recht auf Löschung von unzulässigen Daten (Daten die für ein Amt irrelevant sind)

Access & Identity Management (IAM)

Identity Management

- regelt die sichere und zielgerichtete Arbeit mit Identitäten von Personen und Objekten → Zutritts- und Zugangskontrolle

→ **Authentifizierung**

Access Management

- steuert zentral die Zugriffsberechtigungen
- verwaltet Privilegien an Objekten und Ressourcen
- kontrolliert und protokolliert alle Aktivitäten → Zugriffskontrolle

→ **Autorisierung**

Ziele des IAM:

- Sicherung des Zugriffsschutzes
- Vielzahl der Kennungen und personenbezogene Informationen reduzieren
- Schaffung einer einzigen digitalen Identität

Authentifizierung

Methoden der Authentifizierung

- Wissen
- Besitz
- Biometrie

Methoden	Erklärung	Beispiele
Wissen	Der Benutzer **weiß** etwas. Er wurde in Augenschein genommen und registriert. Im gleichen Zug wurde ihm ein Passwort (Wissen) mitgeteilt, mit dem er seine Identität nachweisen soll.	PIN, Passwort, Antwort auf eine bestimmte Frage
Besitz	Der Benutzer **besitzt** etwas. Er wurde in Augenschein genommen und registriert. Er hat etwas erhalten (Besitz), was er verwenden soll, um seine Identität nachzuweisen.	SmartCard, digitales Zertifikat, RFID-Karte,
Biometrie	Der Nutzer **ist** ein unverwechselbares Individuum. Er wurde in Augenschein genommen und registriert. Dabei wurde ein persönliches Merkmal vermessen und im System hinterlegt. Dieses muss dann genutzt werden, um die Identität nachzuweisen	Fingerabdruck, Gesichtserkennung, Iriserkennung, Stimmenerkennung

[Quelle: Richter 2007]

Lebenszyklus der digitalen Identität

- Anlegen von Benutzern und Gruppen
- Ändern/Benutzen/Zuweisen von Gruppen
- Löschen

Verwaltung von Nutzern und Gruppen in SQL (Standard 2003)

Befehle zur Nutzerverwaltung:

CREATE USER *name* WITH PASSWORD '*password*' [*parameter*]
ALTER USER name WITH [*parameter*]
DROP USER name

Befehle zur Gruppenverwaltung:

CREATE GROUP *gruppenname* [[WITH] USER *benutzername*, ...]

ALTER GROUP *gruppenname* ADD USER *benutzername* [, ...]
ALTER GROUP *gruppenname* DROP USER *benutzername* [, ...]

DROP GROUP *gruppenname* [[WITH] USER *benutzername*, ...]

Beispiele:

CREATE USER *meier* WITH PASSWORD '12345' CREATEUSER;
CREATE USER schmidt WITH PASSWORD '12346' VALID UNTIL '2010-05-31;
CREATE GROUP *fibu* WITH USER *meier, schmidt*;
ALTER GROUP *fibu* DROP USER *schmidt*;
DROP USER *schmidt*;

Aufgabenkomplexe Autorisierung

Rechte- und Zugriffsverwaltung / -überwachung:

- **Kategorien von Rechten**
  - Objektbezogene Rechte´(z.B. SELECT, UPDATE …)
  - Systemrechte / Privilegien an Ressourcen (z.B. Datenbank erstellen)
  - Profile / Rollen
- **Überwachung von Zugriffen / Auswertungen**
  - Systemtabellen
  - Auditing / Protokolle
- **weitere Maßnahmen zur Erhöhung der Sicherheit**
  - zusätzliche Produkte der Hersteller
  - Verschlüsselung und Maskierung von Daten
  - Zugriffsschutzkontrolle / Firewall
  - Indirekte Zugriffe (Arbeit mit Views/Rules/Systemprozeduren)

Methoden für die Zuweisung von Rechten:

- direkter Zugriff über Benutzerkonten
- Zuweisung über Gruppen
- Zuweisung über Rollen

Aufgaben der Zugriffskontrolle

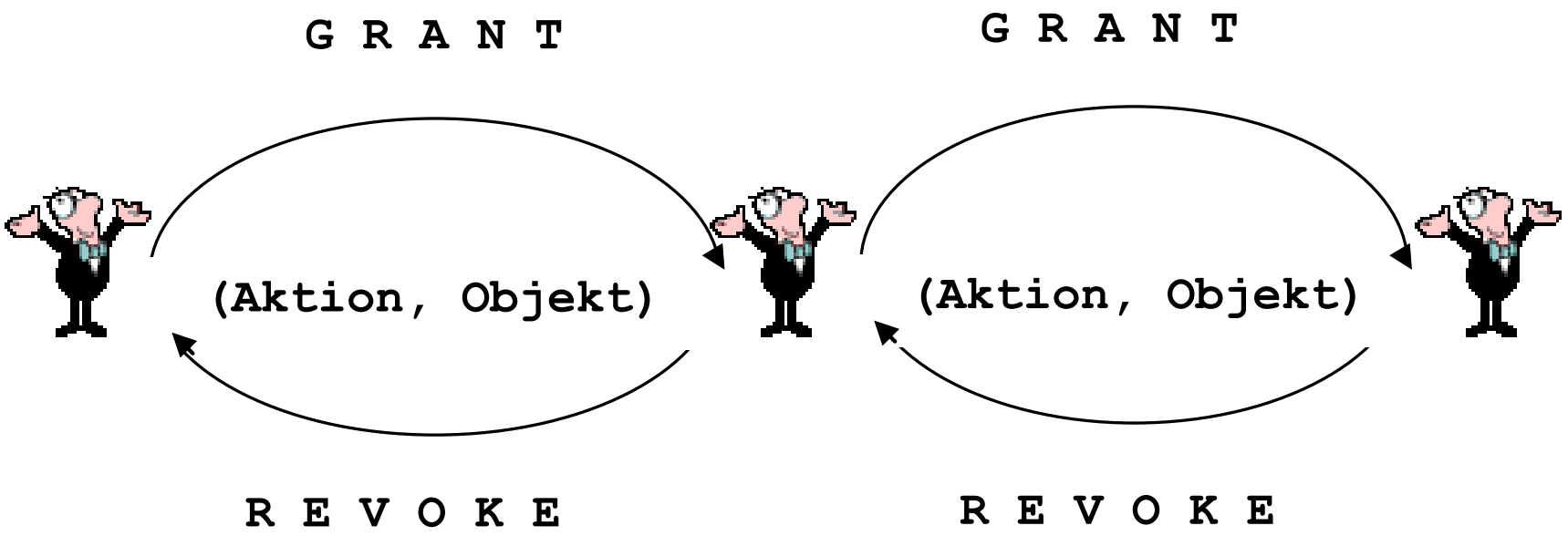
Die Zugriffskontrolle hat zwei wesentliche Aufgaben:

- Kontrolle von ZUGRIFFEN, d.h. Feststellung ob ein bestimmter Zugriff erlaubt ist oder nicht und die Verhinderung unerlaubter Zugriffe
- Administration von RECHTEN, d.h. Verwaltung der Daten, die für die Zugriffskontrolle nötig sind

Ein Zugriff umfasst drei Komponenten:

1. **Subjekt**
   Der Initiator eines Zugriffs ist aus Systemsicht ein Nutzer, eine Gruppe, ein Programm, oder etwas anderes, in jedem Fall aber der aktive Teil.
2. **Aktion**
   Durch die Aktion wird die Art und Weise des Zugriffs bestimmt.
3. **Objekt**
   Dieser passive Teil erfährt den Zugriff und bestimmt hiermit das Ziel desselben.

Komponenten der Autorisierung



- **Grantor** (Subjekt, das ein Recht vergibt)
- **Capability** (ein Paar aus Aktion und Objekt)
- **Grantee** (Subjekt, das ein Recht empfängt)

Zugriff auf den Server (Authentifizierung MS SQL Server)

- Um auf den Server und seine Datenbanken zugreifen zu können, muss der Anwender angelegt werden.
- Nur der Systemadministrator kann den Zugriff auf den Server bewilligen.
- Syntax:

```
CREATE LOGIN loginname WITH PASSWORD = password,
      DEFAULT_DATABASE = databasename
CREATE LOGIN domainname\username FROM Windows
```

Beispiel:

```
CREATE LOGIN user1 WITH PASSWORD 'mypassword',
      DEFAULT DATABASE = pubs2
```

- Hinweise:

Der Systemadministrator muss sich in der *Master*-Datenbank befinden, um `CREATE LOGIN` zu benutzen.

Das Standardpasswort ist null.

Wenn der Anwender einloggt, ist er mit der Standard-Datenbank verbunden.

Fehlt die Standard-Datenbank wird die Master-Datenbank zur Standard-Datenbank. Spezifizieren Sie dann eine andere!

Zugriff auf Datenbanken (MS SQL Server)

- Der Zugriff auf den Server ermöglicht nicht den Zugriff auf jede Datenbank.
- Nur der Datenbank-Besitzer oder der Systemadministrator kann den Zugriff auf eine Datenbank bewilligen.
- Hinzufügen eines Anwenders in eine Datenbank:

```
CREATE USER username FOR LOGIN loginname
```

Beispiel:

```
use pubs2
CREATE user1 FOR LOGIN user1
```

- Der Datenbank-Besitzer ist ein spezieller Benutzer:
  - Wenn Sie eine Datenbank besitzen, ist Ihr Anwender-Name immer dbo .
  - Der Datenbank-Besitzer besitzt die Systemtabelle in dieser Datenbank, und besitzt typischerweise alle Tabellen in der Datenbank.
- Löschen eines Anwenders aus einer Datenbank mit:

```
DROP USER username
```

Beachte: Man kann nie den Besitzer eines Objektes löschen.

Zugriff auf Befehle kontrollieren (Systemprivilegien)

```
                 create database
                 create default
                 create procedure
                 create rule
grant            create table                 to
revoke           create view                  from    user[,user]…
                 dump database
                 dump transaction
                 all
```

- Beispiele:

```
grant all to fred, john
revoke create database, create procedure from fred
grant create procedure to public
```

- Jeder Anwender kann temporäre Tabellen erstellen.
- Nur der Systemadministrator kann die Erlaubnis Datenbanken zu erzeugen vergeben oder zurückziehen, und der Anwender in der Frage muss ein Anwender in der Datenbank master sein.

Zugriff auf Objekte kontrollieren (Objektprivilegien)

```
grant  { select, insert, delete, update, execute, all }
revoke
        on object [(column)] { to | from } user [,user]…
```

- Beispiele:

```
grant insert on titles[title_id] to fred, mary
revoke select on publishers from robert
```

- Die Chronologie bestimmt die resultierende Berechtigung.
- Um nur einige Benutzer vom Zugriff auszuschließen, werden die Rechte zunächst auf Public übertragen und dann die entsprechenden Benutzer ausgeschlossen.

```
grant select on sales to public
revoke select on sales from nancy
```

- Um nur einige Spalten vom Zugriff auszuschließen, wird zunächst die ganze Tabelle ein- und danach die entsprechenden Spalten ausgeschlossen.

```
grant select on payroll to public
revoke select on payroll (salary) from public
```