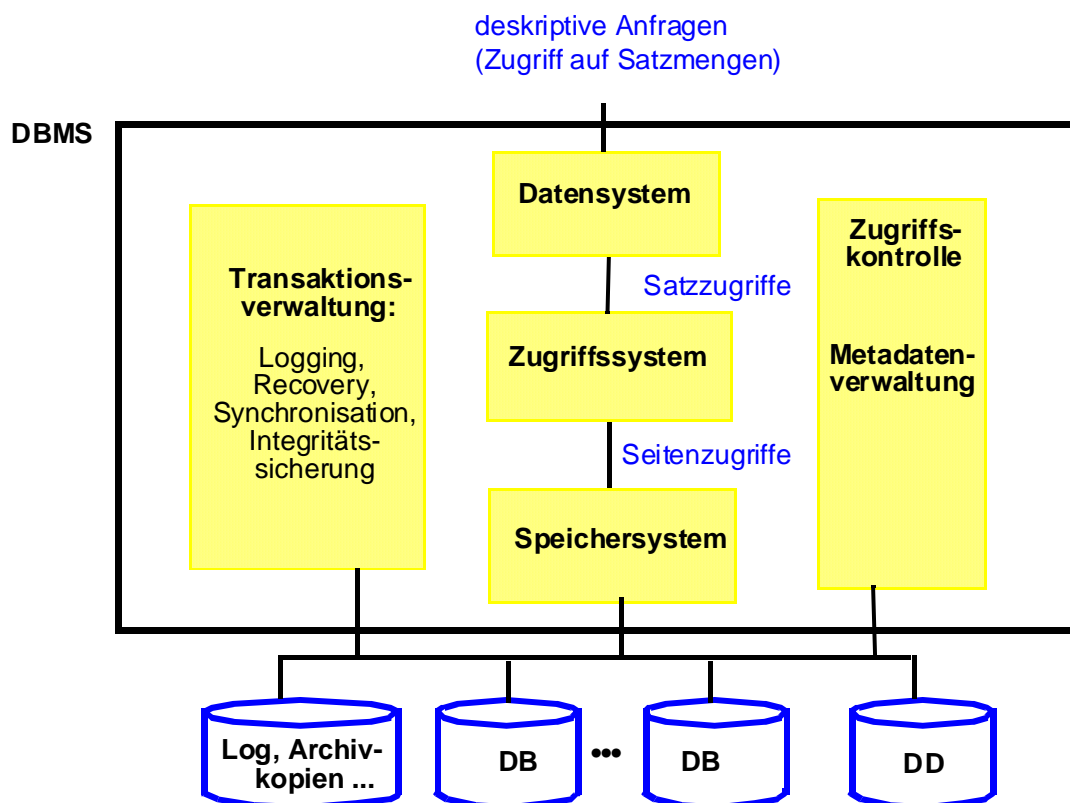


1. Transaktionsverwaltung

- Transaktionsparadigma (ACID)
- Synchronisationsproblem
- Recovery-Arten / Systemkomponenten
- Integritätskontrolle
 - Arten von Integritätsbedingungen
 - Integritätsregeln
 - Implementierung



Grobaufbau eines DBS



Das Transaktionsparadigma

Definition der Transaktion:

Eine Transaktion ist eine Folge von DB-Operationen (DML-Befehlen), welche die Datenbank von einem logisch konsistenten Zustand in einen neuen logisch konsistenten Zustand überführt. Das DBS gewährleistet für Transaktionen die sogenannten ACID-Eigenschaften.

ACID-Prinzip

- **A**tomicity: 'Alles oder Nichts'-Eigenschaft (Fehlerisolierung)
- **C**onsistency: eine erfolgreiche Transaktion erhält die DB-Konsistenz (Menge der definierten Integritätsbedingungen)
- **I**solation: alle Aktionen innerhalb einer Transaktion müssen vor parallel ablaufenden Transaktionen verborgen werden (logischer Einbenutzerbetrieb)
- **D**urability: Überleben von Änderungen erfolgreich beendeter Transaktionen trotz beliebiger (erwarteter) Fehler garantieren (*Persistenz*).



Transaktionsparadigma (2)

■ Programmierschnittstelle für Transaktionen

- begin of transaction (BOT)
- commit transaction („commit work“ in SQL)
- rollback transaction („rollback work“ in SQL)

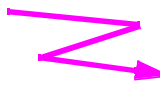
■ Mögliche Ausgänge einer Transaktion

BOT
DML1
DML2
•••
DMLn
COMMIT WORK

normales Ende

BOT
DML1
DML2
•••
DMLn
ROLLBACK WORK

abnormales Ende

BOT
DML1
DML2

erzwungenes
ROLLBACK
Systemausfall,
Programm-
fehler usw.

abnormales Ende

■ ACID vereinfacht DB-Anwendungsprogrammierung erheblich

- Fehlertransparenz (failure transparency)
- Transparenz der Nebenläufigkeit (concurrency transparency)
- erlaubt also fehlerfreie Sicht auf Datenbank im logischen Einbenutzerbetrieb



Transaktionsverwaltung

- Mechanismen zur Einhaltung der ACID-Eigenschaften
 - Synchronisation (Concurrency Control)
 - Logging, Recovery, Commit-Behandlung
 - Integritätskontrolle
- Enge Abhängigkeiten untereinander sowie zu anderen Systemfunktionen (Pufferverwaltung, etc.)
- ACID-Paradigma eignet sich vor allem für relativ kurze Transaktionen, die in den meisten Anwendungen vorherrschen



Transaktionsbeispiel: Debit/Credit

```
void main ( ) {
    EXEC SQL   BEGIN DECLARE SECTION
                int b /*balance*/, a /*accountid*/, amount;
    EXEC SQL   END DECLARE SECTION;
    /* read user input */
    scanf (, %d %d“, &a, &amount);
    /* read account balance */
    EXEC SQL   Select Balance into :b From Account
                Where Account_Id = :a;
    /* add amount (positive for debit, negative for credit) */
    b = b + amount;
    /* write account balance back into database */
    EXEC SQL   Update Account
                Set Balance = :b Where Account_Id = :a;
    EXEC SQL   Commit Work;
}
```



Beispiel paralleler Ausführung (Synchronisationsproblem)

P1

Werte (Variablen, DB)

P2

/* b1=0, a.Balance=100, b2=0 */

Select Balance Into :b1
From Account
Where Account_ID = :a

Select Balance Into :b2
From Account
Where Account_ID = :a

b1 = b1-50

/* b1=100, a.Balance=100, b2=100 */

/* b1=50, a.Balance=100, b2=100 */

b2 = b2 +100

Update Account
Set Balance = :b1
Where Account_ID = :a

/* b1=50, a.Balance=100, b2=200 */

/* b1=50, a.Balance=50, b2=200 */

Update Account
Set Balance = :b2
Where Account_ID = :a

/* b1=50, a.Balance=200, b2=200 */



Synchronisation


- DBS müssen Mehrbenutzerbetrieb unterstützen
- ohne Synchronisation kommt es zu so genannten Mehrbenutzer-Anomalien
 - Verlorengegangene Änderungen (lost updates)
 - Abhängigkeiten von nicht freigegeben Änderungen (dirty read, dirty overwrite)
 - inkonsistente Analyse (non-repeatable read)
 - Phantom-Probleme
- Anomalien sind nur durch Änderungen verursacht
- Synchronisation erfolgt automatisch durch das DBS
- zu klärende Fragen
 - Korrektheitskriterium ?
 - Realisierung ?
 - Leistungsfähigkeit ?



Atomaritätsproblem: Beispiel

■ Unterbrechung während einer Überweisung

```
void main () {  
    /* read user input */  
    scanf (",%d %d %d", &sourceid, &targetid, &amount);  
  
    /* subtract amount from source account */  
    EXEC SQL Update Account  
        Set Balance = Balance - :amount Where Account_Id = :sourceid;  
  
    /* add amount to target account */  
    EXEC SQL Update Account  
        Set Balance = Balance + :amount Where Account_Id = :targetid;  
    EXEC SQL Commit Work; }  
    
```



Recovery-Unterstützung

- automatische Behandlung aller erwarteten Fehler durch das DBVS
- Transaktionsparadigma verlangt:
 - Alles-oder-Nichts-Eigenschaft von Transaktionen
 - Dauerhaftigkeit erfolgreicher Änderungen
- Zielzustand nach Recovery: jüngster, transaktionskonsistenter Zustand vor Erkennen des Fehlers
- Voraussetzung: Sammeln redundanter Informationen während Normalbetrieb (Logging)

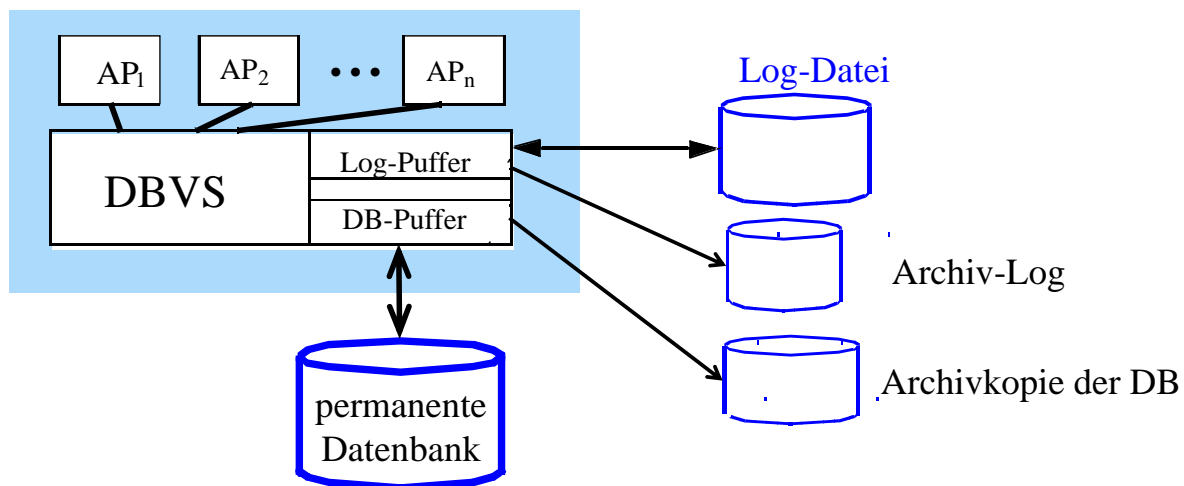


Fehlerarten

- **Transaktionsfehler:** vollständiges Zurücksetzen auf Transaktionsbeginn (Undo)
- **Systemfehler** (Rechnerausfall, DBVS-Absturz)
 - REDO für erfolgreiche Transaktionen (Wiederholung verlorengangener Änderungen)
 - UNDO aller durch Ausfall unterbrochenen Transaktionen (Entfernen derer Änderungen aus der permanenten DB)
- **Gerätefehler** (Plattenausfall):
 - vollständiges Wiederholen (REDO) aller Änderungen auf einer Archivkopie
 - oder: Spiegelplatten bzw. RAID-Disk-Arrays
- **Katastrophen** (Komplettausfall Rechenzentrum, etc.)
 - Verteilte Datensicherung auf geographisch separierten Systemen



Recovery: Systemkomponenten

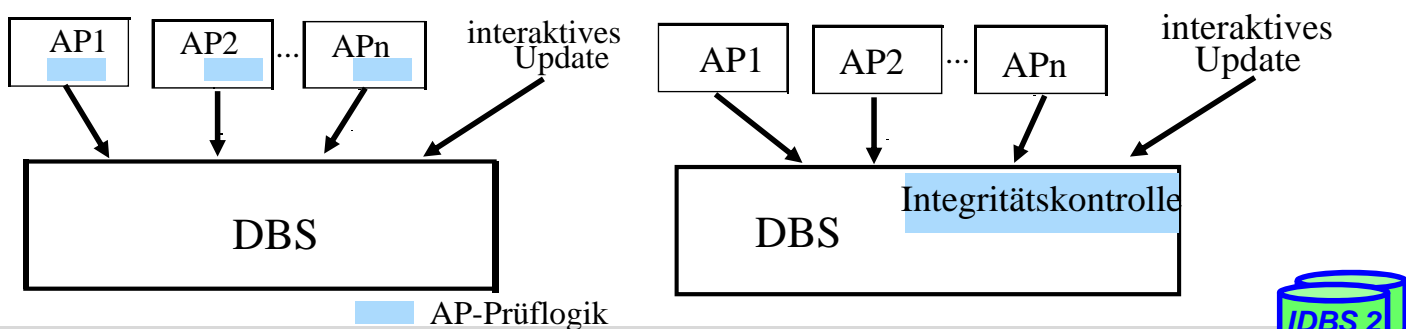


- Pufferung von Log-Daten im Hauptspeicher (**Log-Puffer**)
 - Ausschreiben spätestens am Transaktionsende ("Commit")
- **Temporäre Log-Datei** zur Behandlung von Transaktions- und Systemfehler
- Behandlung von Gerätefehlern: **Archivkopie + Archiv-Log**



Integritätskontrolle

- Wahrung der logischen DB-Konsistenz
- Überwachung von semantischen Integritätsbedingungen durch Anwendungen oder durch DBS
- DBS-basierte Integritätskontrolle
 - größere Sicherheit
 - vereinfachte Anwendungserstellung
 - unterstützt interaktive und programmierte DB-Änderungen
 - leichtere Änderbarkeit von Integritätsbedingungen



Arten von Integritätsbedingungen

- Modellinhärente Integritätsbedingungen (vs. Anwendungsspezifische IB)
 - Primär- und Fremdschlüsseleigenschaften (referentielle Integrität)
 - Definitionsbereiche (Domains) für Attribute
- Reichweite der Bedingung
 - *Attributwert-Bedingungen* (z. B. Geburtsjahr > 1900)
 - *Satzbedingungen* (z. B. Geburtsdatum < Einstellungsdatum)
 - *Satztyp-Bedingungen* (z. B. Eindeutigkeit von Attributwerten)
 - *satztypübergreifende Bedingungen* (z. B. referentielle Integrität zwischen verschiedenen Tabellen, Assertions / Check-Klauseln zwischen Tabellen)
- Statische vs. dynamische Bedingungen
 - *Statische Bedingungen* (Zustandsbedingungen) beschränken zulässige DB-Zustände (z.B. Gehalt < 500000)
 - *dynamische Integritätsbedingungen* (Übergangsbedingungen): zulässige Zustandsübergänge (z. B. Gehalt darf nicht kleiner werden)
- Zeitpunkt der Überprüfbarkeit
 - *unverzögerte vs. verzögerte Integritätsbedingungen*

Integritätsregeln

- Standardreaktion auf verletzte Integritätsbedingung: ROLLBACK
- Integritätsregeln erlauben Spezifikation von Folgeaktionen, z.B. um Einhaltung von IB zu erreichen
 - SQL92: deklarative Festlegung referentieller Folgeaktionen (CASCADE, SET NULL, ...)
 - Trigger bzw. Verallgemeinerung durch ECA-Regeln
- Probleme von Triggern
 - i.a. beschränkt auf Änderungsoperationen einer Tabelle (UPDATE, INSERT, DELETE)
 - i.a. keine verzögerte Auswertung von Triggern
 - Gefahr zyklischer, nicht-terminierender Aktivierungen
 - Korrektheit (Regelabhängigkeiten, parallele Regelausführung, ...)



Integritätsregeln (2)

- Trigger / ECA-Regeln sind jedoch sehr flexibel und mächtig
 - Realisierungsmöglichkeit für nahezu alle Integritätsbedingungen, u.a. dynamische IB
 - viele Einsatzformen über Integritätskontrolle hinaus

```
CREATE TRIGGER GEHALTSTEST
AFTER UPDATE OF GEHALT ON PERS
REFERENCING OLD AS AltesGehalt,
NEW AS NeuesGehalt
WHEN (NeuesGehalt < AltesGehalt)
ROLLBACK;
```



Implementierungsaspekte der Integritätskontrolle

- IB-Überprüfung verlangt vom DBS Entscheidungen
 - für welche DB-Operationen welche Überprüfungen zusätzlich vorzunehmen sind
 - wann Überprüfungen durchzuführen sind (direkt, verzögert)
 - wie Überprüfungen vorzunehmen sind (Ausführungsplan)
- in einfachen Fällen können IB über **Anfragemodifikation** (query modification) behandelt werden
 - Transformation von Änderungsoperation durch Hinzunahme einzuhaltender IB-Prädikate
 - verhindert Ausführung integritätsverletzender Änderungen

```
UPDATE PERS
SET GEHALT = GEHALT * 1.05
WHERE PNR = 4711
```

Integritätsbedingung $GEHALT < 500000$



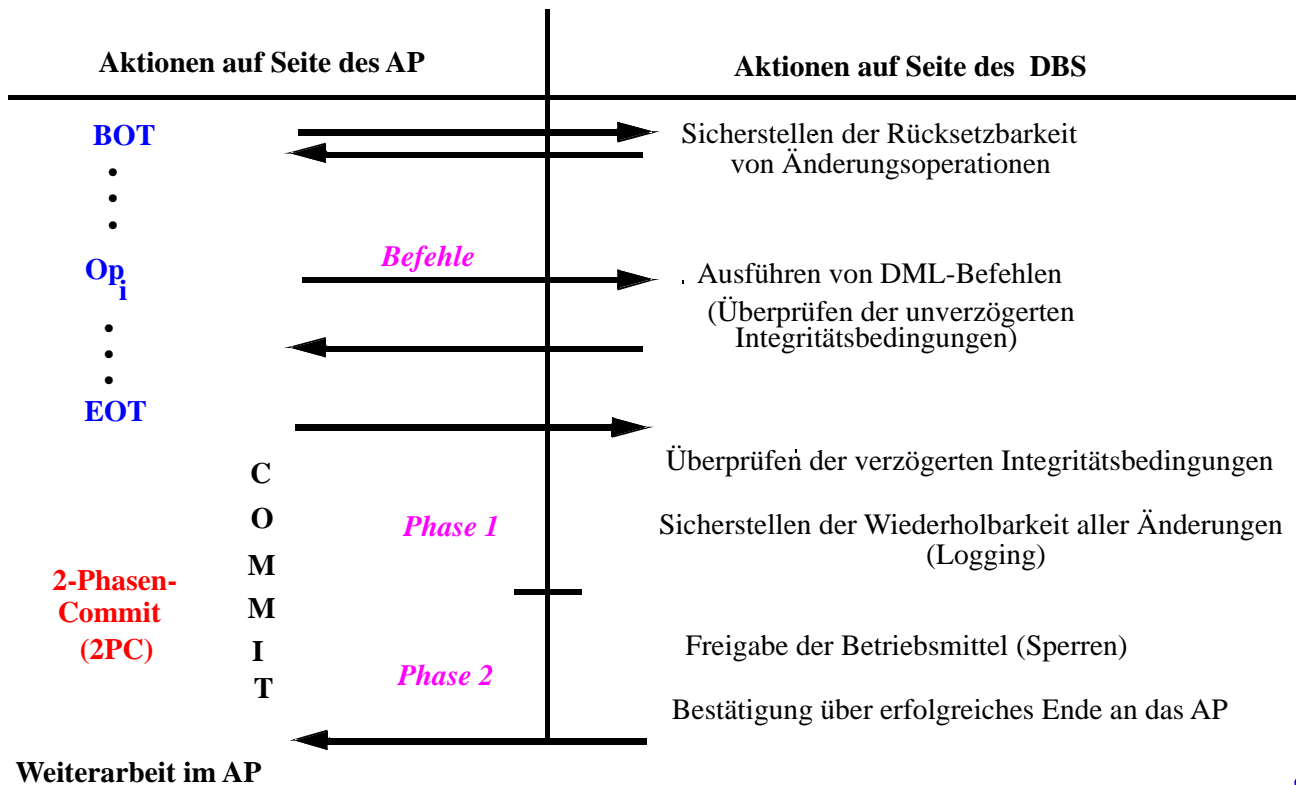
Integritätskontrolle über Regelsystem

- DBS-interne Verwendung von ECA-Regeln zur Überwachung von IB
- IBM-Prototyp Starburst
 - mengenorientierte Regelauswertung am Ende von Transaktionen
 - pro Transaktion werden für jede geänderte Tabelle vier temporäre Relationen (transition tables) mit den von Änderungen betroffenen Sätzen geführt:
deleted, inserted, old-updated, new-updated
 - Begrenzung der Regelauswertung auf minimale Menge relevanter Daten
 - Nutzung der Tabellen innerhalb von Regeln zur Integrationskontrolle (i.a. automatisch vom Compiler erzeugt)
- Beispiel: Wahrung der Integritätsbedingung $GEHALT < 500000$

```
CREATE RULE GehaltsCheck ON PERS
WHEN INSERTED, UPDATED (GEHALT) // Event: Disjunktion
IF EXISTS // Bedingung (Condition)
  (SELECT *
   FROM inserted UNION new-updated
   WHERE GEHALT >= 500000)
THEN ROLLBACK // Aktion
```



Die Transaktion als Schnittstelle zwischen Anwendungsprogramm und DBS



Zusammenfassung

- Transaktionskonzept vereinfacht DB-Programmierung und – Nutzung
 - Transparenz gegenüber Fehlern und Mehrbenutzerbetrieb
- Transaktionsverwaltung sichert ACID-Eigenschaften
 - Synchronisation
 - Logging / Recovery
 - Integritätskontrolle
- Integritätskontrolle kann regelbasiert erfolgen
 - Anfragemodifikation für einfache Fälle
- **Zwei-Phasen-Commit** zum erfolgreichen Transaktionsabschluss
 - Änderungen werden erst gültig nach Sicherstellung aller Integritätsbedingungen sowie der Wiederholbarkeit (Redo-Fähigkeit) der Änderungen
- ACID v.a. für kurze Transaktionen geeignet
-> Bedarf für erweiterte Transaktionskonzepte

