Fakultät Informatik/Mathematik

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN
UNIVERSITY OF APPLIED SCIENCES

Lehrstuhl für Betriebliche Informations- und Kommunikationssysteme/Datenbanksysteme

Vorlesungsmitschrift

DATENBANKSYSTEME 2

Mitschrift von

Falk-Jonatan Strube

Vorlesung von

Prof. Dr.-Ing. Axel Toll

27. März 2018

INHALTSVERZEICHNIS

VORBEMERKUNGEN

PRÜFUNG

SP: 90 min
1 A4 einseitig handschriftlich beschrieben + Kurzreferenz (ohne Notizen)

PVL

Beleg (Abnahme im Praktikum)

- MS SQL
- Oracle

KLAUSURTHEMEN

- Beispiel mit 3 Tabellen mit Dateninhalt. Abfragen bzgl. Datenabfrage/Datenmanipulation, Correlation, Joins, Outerjoin, Leftjoin
- Wie läuft das Datenhandling mit ODBC ab, Befehle + Grundprinzip: Abruf, Datenbindung, Datenausgabe
- Datensicherheit, Transaktion: Was sind Transaktionen? Wie ist die aufgebaut? ACID-Prinzip
- Zugriffsschutz (Wissen, Besitz, Biometrie)
  Varianten ZGS: auf Tabelle mit (grant/revoke), mit views, stored procedures
  Authentifizierung
- Semantische Sicherung:
  DEFAULT
  Trigger (MSSQL) (auch für einfügen/ändern usw. mehrere eingaben (also keine zwischenspeicherung in variable), ohne Cursor)
- operationale Int.: nein
- physische Integrität: ja
- Logische/Physische Datenbank
  B*-Baum (Eigenschaften, Aufbau, Unterschied Binärbaum)
  Zugriffspfade außerhalb der Indexierung!
  keine Verkettung, keine Bitmap-Indizes, kein Objektorientiertes Datenmodell (außer in Bezug auf Oracle)
- Objectrelationale Featuers (keine Syntax, sondern Features)
- Projektierung: Schritte und Abhängigkeit (ohne ERM). Abfolge wichtig!

1 ERWEITERUNG VON DATENBANKSPRACHEN

1.1 ÜBERSICHT DER DATENBANK-SCHNITTSTELLEN

Bsp.: Abfrage aller Namen von Kunden des Ortes Dresden

	Programm (3. Gl) z.B. C	SQL

Pseudocode	`open (Kunden)`
	`while (not end of file) {`
	`  read (Kunde)`
	`  if (Kunde.Ort == 'Dresden')`
	`  print('Name ='+Kunde.Name)`
	`}`
	`close (Kunden)`
		`SELECT Name`
		`FROM Kunde`
		`WHERE Ort='Dresden'`

	prozedural (WIE?)	deskriptiv (WAS?)
Vorteile	• math. Operationen gut möglich • Eingabe/Ausgabe gut möglich • Steuerung des Programmablaufs (Algorithmus)	• Zugriff über Datenmengen • Fkt. des DBMS: – Zugriffsschutz – Integritätssicherung – ...
Nachteil	• Zugriff nur satzorientiert • Vorteile des DBMS kaum möglich	• Vorteile des Programms kaum möglich

1.1.1 PROBLEME VON DATENBANK-SCHNITTSTELLEN

Kap1.pdf
Folie 1

1.1.2 ZIELSTELLUNG DER ERWEITERUNGEN

- Verbindung der Möglichkeiten einer prozeduralen Sprache mit den Möglichkeiten von SQL
- satzorientierte Arbeit mit Ergebnismengen aus SQL („liefert“ Menge)
- Nutzung der Verarbeitungslogik

Lösung der Probleme...

- der Datenübertragung
- der Anpassung der unterschiedlichen Verarbeitungslogiken (Wie $\Leftrightarrow$ Was)

über:

1.1.3 AUFGABEN VON SCHNITTSTELLEN ZWISCHEN DATENBANK-SERVER UND CLIENT-ANWENDUNG

Kap1.pdf
Folie 2

Bsp.:

- Variablendeklaration in C
- ⋮
- SQLCHAR myName[21];
- ⋮
- // Abfrage: SELECT Name FROM Kunde
- ⋮
- // Binden: Spalte1 an Variable myName
- SQLBindCol(. . . ,1,SQL_ C_ CHAR,myName,. . .);

Die SQL-Verarbeitung geschieht über ODBC

1.1.4 AUSWAHL AN SCHNITTSTELLEN ZWISCHEN DATENBANK-SERVER UND CLIENT-ANWENDUNG

Kap1.pdf
Folie 3

1.1.5 ARTEN AN VERBINDUNGEN ZUR DATENBANK

- EIGENSTÄNDIG (Client des DBMS):
  MS SQL Management Studio, Oracle Developer, . . .
- eingebundene
  - eingebettet in Wirtssprache (HLI: host level interface)
    (Erweiterung des Sprachumfangs um SQL-Befehle + Precompiler):
    eSQL in C, eSQL in Java, . . .
  - CLI (call level interface)
    (Erweiterung der Sprache um Funktionen zum Zugriff auf DBMS):
    ODBC, JDBC, . . .

– objektorientiert
(Sprachumfang um Klassen erweitern, OR-Mapping):
ASP.NET, . . .

Bsp. Ablauf eSQL:

- Editor:
bsp.cpp (eSQL-Befehle)
- Precompiler:
bsp.c (Funktionsaufrufe an das DBMS)
- C-Compiler (lib/obj/. . .)
bsp.out
$\Rightarrow$ ausführbar

1.1.6 ARCHITEKTURVERFAHREN FÜR DATENBANK-SCHNITTSTELLEN

Kap1.pdf
Folie 4

1.2 ANWEISUNGEN VON SQL IN ANDEREN SPRACHEN (ODBC)

1.2.1 CALL-LEVEL INTERFACE (CLI)

Kap1.pdf
Folie 5

1.2.2 CALL-LEVEL INTERFACE DIALEKTE

Kap1.pdf
Folie 6

1.2.3 ARCHITEKTUR DER ODBC-SCHNITTSTELLE

Kap1.pdf
Folie 7

1.2.4 MERKMALE VON ODBC

Kap1.pdf
Folie 8

1.2.5 AUFBAU EINER DATENBANK-VERBINDUNG ÜBER ODBC

Kap1.pdf
Folie 9

Bsp. Aufbau:

```
// Headerdatei sqltypes.h
// ...
typedef void* SQLHANDLE;
typedef SQLHANDLE SQLHENN;
typedef SQLHANDLE SQLHDBC;
typedef SQLHANDLE SQLHSTMC;
// ...
```

⇒ Aufbau des C-Programms (Ablauf):

- Handle anlegen: (ODBC: SQLAllocHandle)
  - Umgebung
  - DB-Verbindung
  - Abfrage
- Umgebungsattribute setzen (ODBC: SQLSetEnvAttr)
- Verbindungsattribute setzen (ODBC: SQLSetCommentAttr)
- Vererbung zur DB aufbauen (ODBC: SQLDriverConnect / SQLConnect)
- SQL-Abfrage an DB senden (ODBC: SQLExecDirect)
- Spalten der Abfrage binden (ODBC: SQLBindCol)
  (je Spalte im Abfrageergebnis eine Bindung)
- Im Zyklus Datensatzweise lesen und ausgeben/verarbeiten (ODBC: SQLFetch)
- Verbindung zur DB beenden (ODBC: SQLDisconnect)
- Handle freigeben: (ODBC: SQLFreeHandle)
  - Umgebung
  - DB-Verbindung
  - Abfrage

1.2.6 BESCHREIBUNG DER WICHTIGSTEN CLI-FUNKTIONEN IN ODBC

Kap1B.pdf
Folie 1

Kap1B.pdf
Folie 2

Achtung: mit konstanten Bezeichnungen anstatt mit numerischen Werten arbeiten:

```
// Headerdatei sql.h
// ...
#define SQL_HANDLE_ENV 1
#define SQL_HANDLE_DBC 2
#define SQL_HANDLE_STMT 3
```

1.2.7 MINIMALBEISPIEL ODBC

Kap1B.pdf
Folie 3

Kap1B.pdf
Folie 4

Achtung: Variable muss ein Zeichen größer sein als SQL-Eintrag (→ Nullterminiert)

1.2.8 PROGRAMMAUSSCHNITT ODBC

Kap1.pdf
Folie 10

1.2.9 ERGEBNISTYPEN ODBC

Kap1.pdf
Folie 11

Beispiel Datentypen

Kap1C.pdf
Folie 1

$\Rightarrow$ FETCH liest immer einen Datensatz (Zeilenweise)
Achtung: Zeilen retcode braucht als 4. Argument ein @. . .

2 DATENSICHERHEIT/ZUGRIFFSSCHUTZ

2.1 ÜBERSICHT DATENSICHERHEIT/TRANSAKTIONEN

Kap2.pdf
Folie 1

2.1.1 DATENSICHERHEIT IN DATENBANKSYSTEMEN

	höhere Gewalt	Ausfall technischer Mittel
Gefahrenbereich	• Brand, Wasser • Stromausfall • Blitzschlag • Staub	• HDD defekt • SW gelöscht
Maßnahmen	• Backup / Spiegelung • Brandschutzmaßnahmen • Maßnahmen nach Bundes Datenschutzgesetz (BDSG)	• Backup / Spiegelung • sichere HW/SW

unbewusste menschliche Fehler	parallele Datennutzung	Computerkriminalität
• vergessene Handlung (Eintrag nicht komplett ausgefüllt) • versehentliches Löschen	• gleichzeitiges schreiben von Daten	• Spam, Hacker • Viren, Trojaner
• ReadOnly-Felder • Pflichfelder • Abhängigkeiten	• interne Serialisierung (Warteschlange)	• Maßnahmen Datenschutz • Kryptographie • sichere Passwörter • Zertifikate

Diese lassen sich in Kategorien einteilen:

- physische Integrität
  - höhere Gewalt

  - Ausfall technischer Mittel
- semantische Integrität
  - unbewusste menschliche Fehler
- operationale Integrität
  - parallele Datennutzung
- Zugriffsschutz
  - Computerkriminalität

2.1.1.1 KLASSIFIKATION DER GEFAHRENBEREICHE

Kap2.pdf
Folie 2

Schadensursachen

Bedienfehler	25%
Blitzschlag	17%
Kurzschluss	16%
Diebstahl	10%
Wasser	8%
Feuer	6%
Sturm	1%
Sonstige	17%

2.1.1.2 SICHERHEITSKONZEPTE

Kap2.pdf
Folie 3

BEISPIEL EINER SCHLÜSSELSICHERUNG MIT PRÜFZIFFER

Kap2.pdf
Folie 4

2.1.1.3 ÜBERBLICK ZUR INTEGRITÄTSSICHERUNG

Kap2.pdf
Folie 5

2.1.2 ÜBERSICHT TRANSAKTIONSVERARBEITUNG

WAS IST EINE TRANSAKTION?

Kap2.pdf
Folie 6

2.1.2.1 EIGENSCHAFTEN EINER TRANSAKTION - ACID

(potentielle Prüfungsfrage!!!)

Kap2.pdf
Folie 7

PRINZIPIELLER ABLAUF

PROGRAMM		DBMS
BEGIN TRANSACTION	$\rightarrow$	Maßnahmen für pot. Rücksetzen (Before Image / Undo Information)
Folge an DML-Befehlen	$\swarrow$	
	$\searrow$	Ausführen der Befehle (Integritätsprüfung)
Ende der Transaktion (COMMIT/ROLLBACK)	$\swarrow$	
	COMMIT $\searrow$	Maßnahmen zur Gewährleistung der Wiederholung der Transaktion (After Image / Redo Information)
(alternativ)	ROLLBACK $\searrow$	Lade Before Image
Weiterer Programmablauf	$\swarrow$	

Before/After Image: gespeichert als Log, nicht direkt in der Datenbank

Beispiel 2 Kontennr. in Tabelle Kunde

Ktnr	Inhaber	Wert
1001	Meyer	$22000,$-
1211	Lehmann	$1100,$-

Umbuchung: Meyer $\rightarrow$ Lehmann $(1000,$-$) \Rightarrow$ 2 Updates:

```
BEGIN TRANSACTION
  UPDATE Kunde SET Wert = Wert - 1000
    WHERE Ktnr = 1001
  -- ohne Transaktion würde hier eine Inkonsistenz bei Störung
     auftreten
  UPDATE Kunde SET Wert = Wert + 1000
    WHERE Ktnr = 1211
COMMIT
```

Beispiel Bestellwert Kontrolle: Maximale Summe des Bestellwertes je Kunde = $10000,-$

```
BEGIN TRANSACTION
  UPDATE Kauf SET Menge = Menge * 2
    WHERE Kunr = 123
  IF (SELECT SUM(Menge * vPreis) FROM Kauf WHERE Kunr = 123) >
     10000
    BEGIN
      PRINT 'Bestellwert > 10000 unzulässig'
      ROLLBACK
    END
  ELSE
    COMMIT
```

Hinweise

- in Transaktion NICHT:
  - DB, Index anlegen
  - DB, Tabelle ändern
  - Objekt löschen
  - SELECT INTO
  - Zugriffsrechte vergeben
- Aufrufe von stored proc können NICHT rückgängig gemacht werden.

2.2 SEMANTISCHE INTEGRITÄT

Kap2.pdf
Folie 8

Kap2.pdf
Folie 9

2.2.1 MÖGLICHKEITEN ZUR SICHERUNG DER SEMANTISCHEN INTEGRITÄT

Kap2.pdf
Folie 10

Trigger:
z.B. für Insert/Update/Delete (sodass dieser Trigger immer dann ausgeführt wird, wenn eine der Aktionen ausgeführt wird)

Kap2.pdf
Folie 11

2.2.2 SPALTEN- UND TABELLEN-BEZOGENE INTEGRITÄT

Kap2.pdf
Folie 12

Integritätsbedingungen (=constraints)

- Spalten-bezogen (column-constraint)
- Tabellen-bezogen (table-constraint)

```
CREATE TABLE Kunde(
  Kunr INT,
  Name CHAR(10) NOT NULL,
  ...,
  CONSTRAINT PK_Kunde PRIMARY KEY (Kunr) -- wobei "CONSTRAINT
     PK_Kunde" optional ist
  CONSTRAINT DF_Kunde_Ort DEFAULT 'Dresden' FOR Ort -- Default
  CONSTRAINT CH_Kunde_GebDat CHECK (GebDat < GETDATE()) -- Check
)

CREATE TABLE Artikel ( ... )

CREATE TABLE Kauf (
  Kunr INT,
  Artnr SMALLINT,
  Menge INT NOT NULL,
  vPreis DECIMAL(8,2) NOT NULL,
  CONSTRAINT PK_Kauf PRIMARY KEY (Kunr, Artnr), -- table constraint
  CONSTRAINT FK_Kauf_Kunde FOREIGN KEY (Kunr) REFERENCES Kunde(Kunr
     ),
  CONSTRAINT FK_Kauf_Artikel FOREIGN KEY (Artnr) REFERENCES Artikel
      (Artnr)
)
```

2.2.3 DEFAULTS UND CHECK-EINSCHRÄNKUNGEN

Kap2.pdf
Folie 13

Bsp.: siehe oberhalb

2.2.4 DOMAINS / NUTZERDEFINIERTE DATENTYPEN UND ASSERTIONS

Kap2.pdf
Folie 14

Bsp. 1:

```sql
CREATE DOMAIN Note INT
  CHECK (Note BETWEEN 1 AND 5)

CREATE TABLE Zeugnis (
  ...
  Fach CHAR(20),
  Zensur Note,  -- definierter Datentyp kann bspw. so wieder
     verwendet werden.
  ...
)

CREATE TABLE Pruefung (
  ...
  Matnr CHAR(5),
  Modul CHAR(10),
  Zensur Note,
  ...
)
```

Bsp. 2:

```sql
CREATE DOMAIN PLZ AS CHAR(5)
  CONSTRAINT DO_PLZ CHECK (CONVERT(INT,PLZ) > 1000 AND CONVERT (INT
     ,PLZ)>10000)
  DEFAULT '01069'
```

Löschen:

```sql
DROP DOMAIN <domainname> {RESTRICT|CASCADE} -- bspw. DROP DOMAIN
   PLZ
  -- RESTRICT: kann nicht weg gelöscht werden, wenn der Constraint
     noch in einer Tabelle in Verwendung ist
  -- CASCADE: ersetzt an der Stelle, in der Constraint in
     Verwendung ist, durch Datentyp um Constraint löschen zu können
```

Bsp. Bibliothek-Datenbank:
Tabellen: Vorgemerkte_Buecher, Ausleihbare_Buecher
Bedingung: nicht mehr als 10% der ausleihbaren Bücher vormerken.

```sql
CREATE ASSERTION AS_Vormerkungen
  CHECK ((SELECT COUNT(*) FROM Vorgemerkte_Buecher) / (SELECT COUNT
     (*) FROM Ausleihbare_Buecher) < 0.1) DEFERRED
  -- DEFERRED: am Ende der Transaktion
```

Anmerkung: Assertion nicht allzu relevant.

2.2.5 REFERENTIELLES INTEGRITÄTSPROBLEM

Kap2.pdf
Folie 15

2.2.6 FOREIGN KEY KLAUSEL

Kap2.pdf
Folie 16

2.2.7 BEACHTUNG ZYKLISCHER ABHÄNGIGKEIT

Kap2.pdf
Folie 17

Bsp.:

Abteilung	Mitarbeiter
Abtnr	Abtltr
1001	13
1002	15

Mitnr	Abtnr
12	1001
13	1002
14	1002
15	1001

Problem: Tabelle 1 referenziert die 2. (Abtltr) und die 1. die 2. (Abtnr)
Vorgehen: Erste Referenz im erstellen der Tabelle normal definieren, die zweite Referenz erst im Nachhinein mit ALTER TABLE einfügen

2.2.8 BENUTZERDEFINIERTE FUNKTIONEN

Kap2.pdf
Folie 18

```
SELECT <Spaltenliste> -- Skalarwertfunktion
FROM <Tabellenliste>  -- Tabellenwertfunktion
WHERE ...
```

2.2.8.1 SKALARWERTFUNKTION

Kap2.pdf
Folie 19

Anwendung für diese Funktion:

```
SELECT Kunr, Name, Kredit,
  dbo.Gebuehrensatz(Kredit) AS Gebuehrensatz,
  Kredit*dbo.Gebuehrensatz(Kredit)/100 AS Kreditgebuehr
FROM Kunde
```

DETAILS ZUM BEISPIEL

Kap2B.pdf
Folie 1

2.2.8.2 TABELLENWERTFUNKTION

Kap2.pdf
Folie 20

wichtiger Unterschied zwischen „View“ und „Tabellenwertfunktion“: View können keine Parameter übergeben werden.

DETAILS ZUM BEISPIEL

Kap2B.pdf
Folie 2

WEITERES BEISPIEL

```
declare @ort char(20)
set @ort = 'Dresden'

SELECT a.*, b.Artnr, b.Menge, b.VPreis
FROM ErmittleKundenImOrt(@ort) a
  JOIN Kauf b ON a.Kunr = b.Kunr
```

2.2.9 TRIGGER

Kap2.pdf
Folie 21

Trigger ist mit Tabelle verknüpft (für Aktionen wie Insert, Update, Delete, . . .).
Wenn nun ein Insert, Update, Delete, . . . auf die Tabelle ausgeführt wird, wird der entsprechende Trigger ausgeführt.
Trigger können nur vom Besitzer der Tabelle erstellt werden.
Beispiel für einfachen Trigger:

```
CREATE TRIGGER delKunde
 ON Kunde
 FOR DELETE
 AS
  print 'Sie haben Kunden gelöscht'
RETURN
```

Zugriff auf das Transaktionlog:
t-sql: virtuelle Tabellen (deleted, inserted)

ABLAUF DER DATENMANIPULATION

Tabelle liegt auf der Festplatte. Tabelle wird zum Bearbeiten in den Hauptspeicher des Rechners kopiert. Transaktion ist auf separater Festplatte mit Transaktionlog (Protokolldatei) abgelegt.

Beispiel DELETE:

	DB		Speicher		Log
1.	DELETE Kunde	↘			
2.			Kunden-Seite anlegen	↘	
3.				↙	deleted = before image = Undo Information
4.		↙	Kunden-Seite löschen, dann bekannt geben nach außen, dass gelöscht		
5.	Löschung schreiben (verzögert)				

Beispiel INSERT:

	DB		Speicher		Log
1.	INSERT INTO Kauf	↘			
2.			Kauf-Seite anlegen	↘	
3.		↙		←	inserted=after image= Redo Information, dann bekannt geben nach außen, dass angelegt
4.	Löschung schreiben (verzögert)				

Fall UPDATE: ist bloß Kombination aus DELETE+INSERT
Mit dem Trigger erhält man Zugriff auf das letzte Stück des Transaktionlog (inserted/deleted), welches die Transaktion betrifft, die gerade bearbeitet wird.

2.2.9.1 ZUSAMMENHANG ZWISCHEN PROGRAMM UND DBMS

Kap2.pdf
Folie 22

TRIGGER ZUR ABBILDUNG DER REF. INTEGRITÄT

Löschweitergabe: kaskadierendes Löschen
Kunde löschen in Tabelle Kunde ⇒ Löschen seiner Einkäufe in Tabelle Kauf

```
CREATE TRIGGER Kund_loesch
  ON Kunde
  FOR DELETE
  AS
    DELETE Kauf
    WHERE Kunr IN (SELECT Kunr FROM deleted)
RETURN
```

Achtung: Trigger sollten für mehrere Datensätze funktionieren, nicht nur für einen (Deswegen Kunr IN ... und nicht Kunr = ... – im Fall von mehreren und nicht nur einem Datensatz). Trigger sollte möglichst performant sein!

ZEITPUNKT DES TRIGGERAUFRUFS

Bsp.:

```
DELETE FROM Kunde
WHERE Ort = 'Dresden'
```

Server arbeitet mit AUTOCOMMIT:

ohne Trigger	mit Trigger
Beginn der Transaktion	Beginn der Transaktion
deleted schreiben (Tabelle)	deleted schreiben (Tabelle)
	DELETE-Trigger aufrufen
Transaktion beendet	Transaktion beendet ODER Trigger setzt zurück ↑ (Rollback)

BEISPIEL

Löschen in Kunde unterbinden, wenn in Kauf noch Datensätze für den Kunden vorhanden sind (RESTRICT):

```
CREATE TRIGGER delKund1
  ON Kunde FOR DELETE
  AS
    IF (SELECT COUNT(*) FROM Kauf k JOIN delete d ON k.Kunr = d.
       Kunr) > 0
    BEGIN
      print 'Abbruch, da noch Daten in Kauf'
      SELECT ... -- Datensätze, die referenziellen Integrität
         widersprechen auch mit print ausgeben
      ROLLBACK TRANSACTION -- Trigger steckt mitten in Transaktion,
          deswegen kann hier ein Rollback der Transaktion
         veranlasst werden, obwohl sie hier nicht explizit begonnen
          wurde
    END
RETURN
```

BEISPIEL

Umsatzsumme je Kunde in Kauf max $10.000,-$€.

```
CREATE TRIGGER itr_Kauf ON Kauf
FOR insert, update
  AS
    IF EXISTS (
      SELECT SUM(Menge*vPreis)
      FROM Kauf
      GROUP BY Kunr
      HAVING SUM(Menge*vPreis) > 10000)
    BEGIN
      print 'Kundenumsatz größer 10.000 unzulässig.'
      ROLLBACK TRANSACTION
      -- wenn hier RETURN steht, dann können nach dem END weitere
         Bedingungen (IF...) angehängt werden.
```


```
    END
RETURN
```

WEITER BEISPIELE

Kap2.pdf
Folie 23

```
CREATE TRIGGER MittelAenderung
ON Projekt
FOR UPDATE AS IF UPDATE (Mittel) -- aus Performancegründen nur
   triggern, wenn Mittel geändert
BEGIN
  INSERT INTO Mittel_protokoll
    SELECT i.Pronr, user_name(), GETDATE(), d.Mittel, i.Mittel
    FROM inserted i JOIN deleted d ON i.Kunr = d.Kunr AND i.Artnr =
        d.Artnr
END
```

Kap2.pdf
Folie 24

2.3 OPERATIONALE INTEGRITÄT

„Gefahrenbereich“ Mehrbenutzerbetrieb.

2.3.1 ANOMALIEN IM MEHRBENUTZERBETRIEB

2.3.1.1 LOST UPDATE

Kap2.pdf
Folie 25

2.3.1.2 DIRTY READ

Kap2.pdf
Folie 26

2.3.1.3 NON REPEATABLE READ

Kap2.pdf
Folie 27

2.3.1.4 PHANTOMPROBLEM

Kap2.pdf
Folie 28

2.3.2 SICHERUNG DER OPERATIONALEN INTEGRITÄT

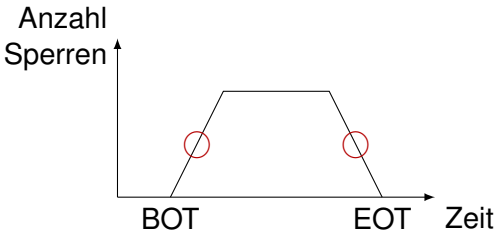
Kap2.pdf
Folie 29

2.3.2.1 SPERRVERFAHREN

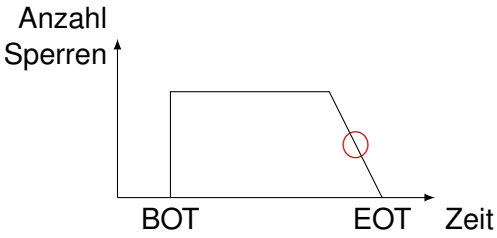
Kap2.pdf
Folie 30

BOT. . . begin of transaction
EOT. . . end of transaction
◯: gefährdete Bereiche

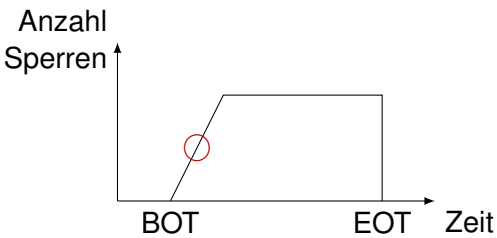
EINFACHES ZWEIPHASEN-SPERRPROTOKOLL



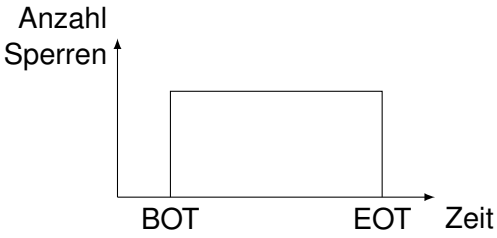
ZWEIPHASEN-SPERRPROTOKOLL MIT PRECLAIMING



STRIKTES ZWEIPHASEN-PROTOKOLL



STRIKTES ZWEIPHASEN-PROTOKOLL MIT PRECLAIMING



2.3.2.2 STATISCHE UND DYNAMISCHE SPERRVERFAHREN

Kap2.pdf
Folie 31

2.3.2.3 ANFORDERUNGEN AN SPERRMODI

Kap2.pdf
Folie 32

Sperrgranulate:

- logische Objekte:
  Tupel, Tabelle
- physische Objekte:
  Seite (Block auf Festplatte), Segmente

Sperrverwaltung über Lockmanager.
$+$ ... erlauben, $-$ ... ablehnen (Warteschlange)

Sperrwunsch	aktuell keine Sperre	aktuell S-Sperre	aktuell X-Sperre
S	$+$	$+$	$-$
X	$+$	$-$	$-$

2.3.2.4 STUFEN DER ISOLATION PARALLELER TRANSAKTIONEN

Kap2.pdf
Folie 33

SQL:

```
SET TRANSACTION ISOLATION LEVEL ...
```

Mögliche Anomalien nach Ebene:

Ebene	Lost Update	Dirty Read	Non repeat. Read	Phantom
0	X	X	X	X
1			X	X
2				X
3				

Also: Nur mit Ebene-3 können alle Anomalien abgedeckt sein. Aber: tiefere Ebenen kosten mehr Leistung!

2.3.2.5 OPTIMISTISCHE VERFAHREN

Kap2.pdf
Folie 34

Grundgedanke:

- Transaktionen nutzen oft verschiedene Daten
- Konflikte selten
- kein Overhead für Sperrverwaltung

2.3.2.6 ZEITSTEMPELVERFAHREN

Kap2.pdf
Folie 35

Letzten Endes werden Kombinationen verschiedener Verfahren verwendet:
Jeder Nutzer bekommt einen Zeitstempel, im Backend läuft allerdings ein pessimistisches Verfahren.

2.4 PHYSISCHE INTEGRITÄT

Kap2.pdf
Folie 36

- abgeschlossene Transaktion $\Rightarrow$ Ergebnis herstellen
  nicht abgeschlossene Transaktion $\Rightarrow$ Rücksetzen
- Verfahren des Sicherns $=$ Logging, Backup
  Verfahren des Wiederherstellens $=$ Recovering

2.4.1 PROTOKOLLIERUNG

- Abspeicherzeitpunkt (Beginn/Ende der Transaktion)
- Protokollgranulat (Tupel/Befehle)
- Niveau (logisch: Tupel/Befehl oder physisch: Seite)

EINTRAGUNGEN IN DER PROTOKOLLDATEI

Kap2.pdf
Folie 37

2.4.2 EINBRINGSTRATEGIEN

Kap2.pdf
Folie 38

2.4.3 PRINZIP DER PHYSISCHEN PROTOKOLLIERUNG

Kap2.pdf
Folie 39

Kap2.pdf
Folie 40

2.4.4 DATENSICHERUNG UND WIEDERHERSTELLUNG AM BEISPIEL MS SQL-SERVER

Log (*.ldf Datei)
↑
Daten- und Log-Seiten im Hauptspeicher
↕
Datenbank (*.mdf Datei)

ARCHIVSICHERUNG

- Datenbank: Vertrieb.bak

```
BACKUP DATABASE Vetrieb TO DISK = 'E:\Backups\Vertrieb.bak'
```

- Log: V_log1.bak

```
BACKUP LOG Vertrieb TO DISK = 'E:\Backups\V_log1.bak'
```

Sobald Log gesichert wird, wird *.ldf Datei geleert!

2.4.5 WIEDERHERSTELLUNGSMODI

Kap2.pdf
Folie 41

- vollständig (Standard)

```
ALTER DATABASE SET RECOVERY FULL
```

Nachteil:
− Log-File erst mit Log-Sicherung abgeschnitten
Vorteil:
+ Wiederherstellung bis letzte Log-Sicherung möglich (bei Festplatten-Fehler)

```
RESTOR DATABASE Vertrieb FROM DISK = 'E:\Backups\Vertrieb.bak'
```

Zum Wiederherstellen bei einem Ausfall (bspw. um 9:30) wird die gesicherte Datenbank hergestellt (die bspw. um 6:00 erstellt wurde) und alle danach vorhandenen Logs (die bspw. um 7:00, 8:00 und 9:00 erstellt wurden) angewandt, um wieder zum verlorenen Zustand zurück zu kommen. Achtung: Logs sind inkrementell! Ist ein Log kaputt, können folgende Logs nicht mehr zur Wiederherstellung genutzt werden.
Anhand der Logs kann auch bei falscher Manipulation bis zu einem bestimmten Zeitpunkt wiederhergestellt werden (mit dem Backup von 9:00 Zustand von genau 8:20 herstellbar).

- einfach

```
1 ALTER DATABASE SET RECOVERY SIMPLE
```

Nachteil:
− Wiederherstellung nur bis letztem DB-Backup Vorteil:
+ Log-Datei „wächst nicht“ (Wird nur für Transaktionen benutzt)
Zum Wiederherstellen bei einem Ausfall (bspw. um 9:30) kann nur die gesicherte Datenbank (von bspw. 6:00) wieder hergestellt werden, es gibt keine Logs dazwischen, die einen neueren Zustand wiederherstellen können!

2.4.5.1 ZUSAMMENHANG ZWISCHEN PROTOKOLL- UND RECOVERYVERFAHREN

Kap2.pdf
Folie 42

2.4.5.2 WEITERE RECOVERYVERFAHREN

Kap2.pdf
Folie 43

2.4.5.3 ÜBERTRAGUNG DATENBANKPUFFER-PLATTE

Kap2.pdf
Folie 44

2.4.6 HOCHVERFÜGBARKEIT

Kap2.pdf
Folie 45

Kap2.pdf
Folie 46

2.4.6.1 SPIEGELUNG

1. Eintragung in Datenbank 1
2. REDO ins Log der DB 1
3. Spiegelung der Eintragung in Datenbank 2
4. REDO der Spiegelung ins Log der DB 2
5. Bestätigung an DB 1

2.4.7 DB-SERVERSTRUKTUREN

am Beispiel MS SQL / Sybase
Datenbanken (DB):

- Benutzerdatenbanken (siehe Praktikum)
  - Benutzertabellen (z.B. Mitarbeiter, Projekt)
  - Systemtabellen sys.. . . (Speicherung der Metadaten) + Systemsicht auf Systemtabellen
- Systemdatenbanken
  - master
    - Verwaltung der Benutzer-DB
    - Berechtigungen auf dem Server

      ⋮
  - model
    - Prototyp einer Benutzer-DB
  - tempdb
    - Ablage nicht dauerhafter Objekte

WICHTIG: Metadaten liegen in Systemtabellen sys.***!

2.5 ZUGRIFFSSCHUTZ/DATENSCHUTZ

Kap2.pdf
Folie 47

2.5.1 PROBLEME BEIM DATENSCHUTZ

Kap2.pdf
Folie 48

	Datenschutz	Zugriffsschutz
Bereich	Bundesweit	Informationssystem
Gegenstand	personenbezogene Daten	Daten aller Fachgebiete
Mittel	juristisch/organisatorisch	organisatorisch, SW/HW-Schutzfunktion

2.5.2 ACCESS- UND IDENTITY MANAGEMENT (IAM)

Kap2.pdf
Folie 49

2.5.3 AUTHENTIFIZIERUNG

Kap2.pdf
Folie 50

Beispiel: Privilegien vergeben (Tabelle Mitarbeiter)

	MitNr	Name	Qualif.	Gehalt
Meier (Produktplaner)	lesen	lesen	lesen	X
Schmidt (Geschäftsführung)	lesen	lesen	lesen	lesen
Lehmann (HR/Personal)	schreiben	schreiben	schreiben	schreiben

2.5.4 VERWALTUNG VON NUTZER UND GRUPPEN IN SQL

Kap2.pdf
Folie 51

2.5.5 AUFGABENKOMPLEXE AUTORISIERUNG

Kap2.pdf
Folie 52

2.5.6 AUFGABEN DER ZUGRIFFSKONTROLLE

Kap2.pdf
Folie 53

2.5.7 KOMPONENTEN DER AUTORISIERUNG

Kap2.pdf
Folie 54

2.5.8 ZUGRIFF AUF DEN SERVER (MS SQL SERVER)

Kap2.pdf
Folie 55

ANWEISUNGEN ZUM ZUGRIFFSSCHUTZ

Zugriffsschutz im MS SQL Server
sa . . . Systemadministrator
dbo . . . DataBaseOwner

```
1 -- sa:
2 CREATE LOGIN MeierU WITH PASSWORD '123'
3 CREATE LOGIN Schmidt WITH PASSWORD '234'
4 CREATE LOGIN Richter WITH PASSWORD '345'
```

```
5
6  CREATE USER MeierU FOR LOGIN MeierU
7  GRANT CREATE DATABASE TO MeierU
8  -- Logout sa
9  -- MeierU:
10 CREATE DATABASE Test
11 USE Test  -- in Datenbank Test springen
12 CREATE USER Schmidt FOR LOGIN Schmidt
13 GRANT CREATE TABLE TO Schmidt
14 -- Logout MeierU
15 -- Schmidt:
16 USE Test
17 CREATE TABLE Adresse (Ort, ...
```

Login: Zugriff auf Datenbankserver (ohne Berechtigungen an lesen/schreiben/...)
User: Nutzer der Datenbank (Zugriffsberechtigung nach Vorgabe)

2.5.9 ZUGRIFF AUF DATENBANK (MS SQL SERVER)

Kap2.pdf
Folie 56

2.5.9.1 ZUGRIFF AUF BEFEHLE KONTROLLIEREN (SYSTEMPRIVILEGIEN)

Kap2.pdf
Folie 57

2.5.9.2 ZUGRIFF AUF OBJEKTE KONTROLLIEREN (OBJEKTPRIVILEGIEN)

Kap2.pdf
Folie 58

2.5.9.3 VARIANTEN DER UMSETZUNG VON ZUGRIFFSSCHUTZ

Beispiel: Tabelle Mitarbeiter
public: Mitnr, Name, Vorname, Anschrift
geschützt: Gehalt

1. Direkter Schutz

```
1 GRANT SELECT ON Mitarbeiter(Mitr, Name, Vorname, Anschrift) TO
    MeierU
```

2. Indirekter Schutz (über VIEW)

```
1 CREATE VIEW MitAdresse AS
2   SELECT Mitnr, Name, Vorname, Adresse
3   FROM Mitarbeiter
4 GRANT SELECT ON MitAdresse TO MeierU
```


3. Indirekter Schutz (über stored procedure)

```
CREATE PROC MitProc AS
  SELECT Mitnr, Name, Vorname, Adresse
  FROM Mitarbeiter
GRANT EXEC MitProc TO MeierU
```

2.5.9.4 SQL-INJECTION ABWEHREN

Beispiel: e-Business-Application

ID	Name	PW
1	Meier	abc
2	Schulze	xyz

```
IF (SELECT COUNT (*))
  FROM User
  WHERE Name='Meier'
  AND PW='abc') > 1
```

⇒ Wenn ein Ergebnis zurückkommt ist das Passwort richtig geraten.

```
string sql= "SELEC COUNT(*) FROM user WHERE Name='"
  +strName
  +"' AND PW='"
  +strPW+"'";
```

Mittels einer SQL Injection kann der SQL Code so erweitert werden, dass eine Anmeldung immer erfolgreich ist.

```
strName = A'OR'x'='x
strPW = A'OR'x'='x

sql = SELECT COUNT(*)
  FROM user
  WHERE Name='A'OR'x'='x'
  AND PW = 'A'OR'x'='x'
```

Lösung: stored procedure einsetzen:

```
CREATE PROC logintest (@Name char(10), @Pw char(10))
BEGIN
  IF EXISTS (
    SELECT COUNT(*)
    FROM user
    WHERE Name=@Name
    AND PW=@pw
  )
    RETURN 1
  ELSE
    RETURN 0
END
```

Aufruf der stored procedure:

```
string sql= 'exec dlogintest('=
  + strName
  + "','"
  + strPw
  + "')";
```

Aufruf der fehlerhaften stored procedure vie SQL-Injektion

```
exec logintest('A'OR'x'='x','A'OR'x'='x')
```


3 LOGISCHE DATENMODELLE

Datenmodellierung ist ein zweistufiger Abstraktionsprozess:

Kap3.pdf
Folie 15

3.1 LOGISCHE UND PHYSISCHE DATENORGANISATION / ÜBERBLICK ÜBER DATENMODELLE

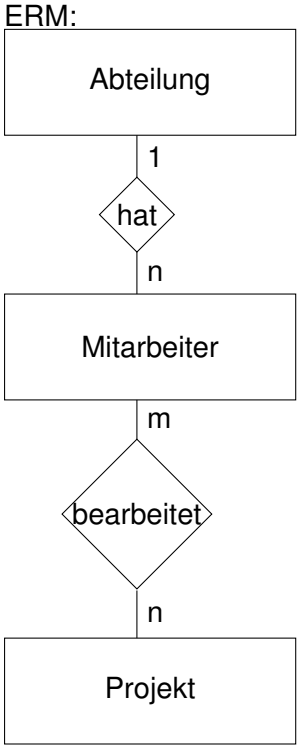
3.1.1 ZUSAMMENHANG ZWISCHEN SEMANTISCHEN UND LOGISCHEN DATENMODELLEN

Kap3.pdf
Folie 16

3.1.2 SEMANTISCHE DATENMODELLE

Kap3.pdf
Folie 17

ERM:



3.1.3 LOGISCHE DATENMODELLE

Kap3.pdf
Folie 18

Kap3.pdf
Folie 19

3.2 KLASSISCHE LOGISCHE DATENMODELLE

3.2.1 HIERARCHISCHES MODELL

Kap3.pdf
Folie 20

Speicherung der Beziehung in den Datensätzen
Vorteil: hohe Abfrageperformance
Nachteil: eingeschränkte Datenunabhängigkeit („unflexibel“)

Kap3.pdf
Folie 21

3.2.2 NETWERKDATENMODELL

Kap3.pdf
Folie 22

Speicherung der Beziehung in den Datensätzen
Vorteil: hohe Abfrageperformance
Nachteil: eingeschränkte Datenunabhängigkeit („unflexibel“)

Kap3.pdf
Folie 23

3.2.3 RELATIONALES DATENMODELL

3.2.3.1 RELATIONENBEGRIFF

Kap3.pdf
Folie 24

Beispiel:
Entity-Typ: Zeugnis
Attribute:
A_1 Fach $W_1\{Ma, Ph\}$
A_2 Note $W_2\{1, 2, 3, 4, 5\}$
$PM = W_1 * W_2$

3.2.3.2 CHARAKTERISTIKA

Kap3.pdf
Folie 25

Abfrage der Beziehungen zur Laufzeit
Vorteil: Datenunabhängig, flexibel (Erweiterung, Abfrage)
Nachteil: Performance schlechter als hierarchisch oder Netzwerk-DM

3.2.3.3 GRENZEN DES RELATIONALEN DATENMODELLS: MODELLIERUNG POLYEDER

Kap3.pdf
Folie 26

Abfrage aller Punkte mit `F.fid<3`

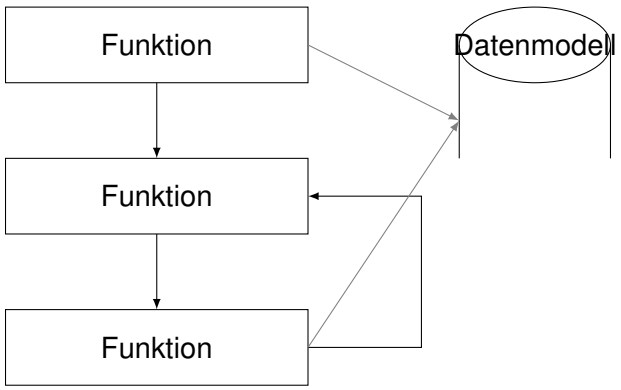
```
SELECT F.fid, P.x, P.y, P.z
FROM Punkt P, KP_Rel, KP, FK_Rel FK, Flaeche F
WHERE F.fid<3
  AND P.pid = KP.pid
  AND KP.kid = K.kid
  AND K.kid = FK.kid
  AND FK.fid = F.fid
```

$\Rightarrow$ NF^2-Modell wäre schneller (keine 1. Normalform)

Kap3.pdf
Folie 27

3.3 OBJEKTORIENTIERTES DATENMODELL

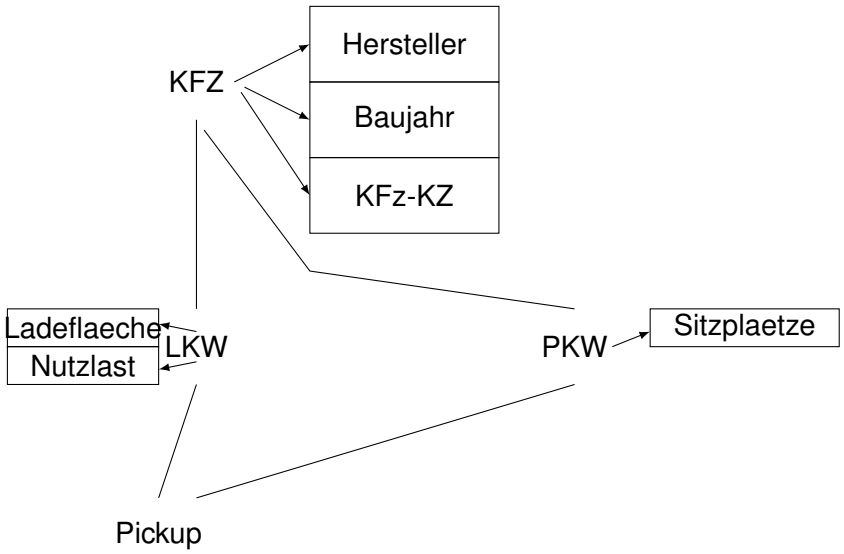
Konventionelles Programm



Objektorientiertes Programm

- Objekt:
  - Daten
  - Funktion
  - Funktion
  - Funktion
- Objekt:
  - Daten
  - Funktion
- Objekt:
  - Daten
  - Funktion
  - Funktion

Kap3.pdf
Folie 28



3.3.1 CHARAKTERISIERUNG DER OBJEKTORIENTIERUNG

Kap3.pdf
Folie 29

Kap3.pdf
Folie 30

3.3.2 BEDEUTUNG DES OBJEKTORIENTIERTEN DATENMODELLS

Kap3.pdf
Folie 31

3.3.3 OBJEKTORIENTIERTEN UND OBJEKTRELATIONALES DATENMODELL

Kap3.pdf
Folie 32

3.3.4 OBJEKTRELATIONALES DATENMODELL

Kap3.pdf
Folie 33

3.3.4.1 NF^2-AUSPRÄGUNG (1 TUPEL)

Kap3.pdf
Folie 34

3.4 DBMS-SPEZIFISCHE ERWEITERUNGEN VOM STANDARD-SQL (ORACLE)

3.4.1 DATENDEFINITION IN ORACLE

Kap3.pdf
Folie 1

3.4.1.1 ORACLE SQL DATENTYPEN

Kap3.pdf
Folie 2

3.4.2 ARBEIT MIT SYSTEMTABELLEN DES DATA DICTONARY

Kap3.pdf
Folie 3

3.4.3 SEQUENZ

Kap3.pdf
Folie 4

Kein SELECT ohne FROM.
SELECT in Oracle:

```
SELECT sysdate FROM dual;
--Output: 16.12.2016 09:41

SELECT user FROM dual;
--Output: sxxxxx

SELECT Name || ',' || Vorname
       || ' (' || SUBSTR(Beruf,1,3)
       || ')' AS Kundenname
FROM Kunde

-- Output:
-- Meier, Uwe (Ing)
-- Adler, Sabine (Inf)
```

3.4.4 SPEZIFIKA BEI ABFRAGEN

Kap3.pdf
Folie 5

3.4.4.1 HIERARCHISCHE ABFRAGEN

Kap3.pdf
Folie 6

3.4.5 AUSGEWÄHLTE SQL-FUNKTIONEN IN ORACLE

Kap3.pdf
Folie 7

3.4.6 PL/SQL-STATEMENTS IN PROCEDUREN UND TRIGGERN

Kap3.pdf
Folie 8

Variablendeklaration:

```
DECLARE v_name VARCHAR2(10) := 'Otto';
        v_ort VARCHAR2(20);
        v_alt NUMBER(3);
```

Wertzuweisung:

```
v_alt := 27;
v_ort := 'Dresden';
```

3.4.7 STORED PROCEDURES

Kap3.pdf
Folie 9

3.4.8 CURSOR IN PL/SQL

Kap3.pdf
Folie 10

```
CREATE OR REPLACE PROCEDURE p1(K_wert IN INTEGER)
IS
    CURSOR C1 IS
        SELECT Kunr, Name
        FROM Kunde
        WHERE Kredit > K_wert;
    FOR satz IN C1 LOOP
        DBMS_OUTPUT.PUTLINE('Kunr:' || satz.Kunr);
        DBMS_OUTPUT.PUTLINE('Name:' || satz.Name);
    END LOOP;
    END;
/
```


2. Variante CURSOR-Nutzung:

```
...
CURSOR C1 IS
  SELECT Kunr, Name, Vorname
  FROM Kunde
  WHERE Kredit > K_wert;
FOR satz IN C1 LOOP
  DBMS_OUTPUT.PUTLINE('Kunr: '|| satz.Kunr);
  DBMS_OUTPUT.PUTLINE('Name: '|| satz.Name);
  DBMS_OUTPUT.PUTLINE('Vorname: '|| satz.Vorname);
END LOOP;
...
```

3.4.9 TRIGGER IN PL/SQL

Kap3.pdf
Folie 11

3.4.9.1 TRIGGER-BEISPIELE

Kap3.pdf
Folie 12

TRIGGER ZUR SICHERUNG REFERENTIELLER INTEGRITÄT

(DS in Kauf einfügen oder ändern ⇒ prüfen, ob Kunde vorhanden ist)

```
CREATE OR REPLACE TRIGGER CheckKunde
  AFTER INSERT OR UPDATE ON Kauf
  FOR EACH ROW
    DECLARE i int;
            nicht_moeglich EXCEPTION;
    BEGIN
      SELECT COUNT(*) into i
      FROM Kunde k
      WHERE k.Kunr = :NEW.Kunr;
      IF i=0 THEN
        RAISE nicht_moeglich;
      END IF
      -- ggf. weitere Prüfungen

      EXCEPTION
        WHEN nicht_moeglich
          raise_application_error(-2100,'Kunr nicht vorhanden');
        -- ggf. weitere Exceptions

      END;
    /
```

Kap3.pdf
Folie 13

Zeilen :NEW :OLD nur im Row-Trigger

Operation	:OLD	:NEW
INSERT	NULL	einzufügende Tupel
UPDATE	Tupel vor Update	Tupel nach Update
DELETE	zu löschender Tupel	NULL

Trigger:

- Beim Einfügen in Artikel
  ⇒ ArtNr als nächsten Wert einer Sequenz setzen
- Beim Ändern
  ⇒ keine Änderung der ArtNr (primary key) zulassen

1. Sequenz anlegen:

```
create sequence sq_artikel
  increment by 1
  start with 10000;
```

2. Trigger anlegen:

```
create or replace trigger tr_artikel
  befor insert or update on Artikel
  for each row
  begin
    if inserting then    -- nur für Insert
      select sq_artikel.nextval
        into :NEW.Artnr from dual
    end if;
    if updating then     -- nur für Update
      :NEW.Artnr := :OLD.Artnr
    end if;
  exception
    when others then
      raise_application_error(-2100, sqlerrm);
  end;
```

Kap3.pdf
Folie 14

3.4.10 UNTERSCHIEDLICHE STRATEGIEN BEIM UMGANG MIT DEM TRANSAKTION-LOG IN TRIGGERN

	MS SQL-Server/Sybase	Oracle
Ansatz	Tabellen-orientiert	Datensatz-orientiert
Umsetzung	„magic“ TABLES: deleted, inserted	TUPEL (einzelne Datensätze!): :OLD, :NEW
Verwendung	in jedem Trigger	nur im ROW-Trigger
Transaktion zurücksetzen	ROLLBACK TRAN	RAISE_APPLICATION_ERROR(...)

Beispiel:

```
1 UPDATE Kauf SET vPreis = vPreis * 1.1 WHERE Kunr=123
```

MS-SQL
table deleted:

Kunr	Artnr	Menge	vPreis
123	1234	10	360
123	5555	10	320
123	2222	10	3600

table inserted:

Kunr	Artnr	Menge	vPreis
123	1234	10	396
123	5555	10	352
123	2222	10	3960

ORACLE
1. row:

	Kunr	Artnr	Menge	vPreis
:OLD	123	1234	10	360
:NEW	123	1234	10	396

2. row:

	Kunr	Artnr	Menge	vPreis
:OLD	123	5555	10	320
:NEW	123	5555	10	352

3. row:

	Kunr	Artnr	Menge	vPreis
:OLD	123	2222	10	3600
:NEW	123	2222	10	3960

3.5 OBJEKTRELATIONALE ERWEITERUNGEN IM RDM (ORACLE)

3.5.1 OBJEKTRELATIONALE FEATURES IN ORACLE

Kap3.pdf
Folie 35

3.5.2 KOMPLEXBEISPIEL FÜR RELATIONALE FEATURES

Kap3.pdf
Folie 36

3.5.3 ABSTRAKTE DATENTYPEN - OBJEKT

Kap3.pdf
Folie 37

Kap3.pdf
Folie 38

3.5.4 OBJEKTVIEWS

Kap3.pdf
Folie 39

Überschreibt den eigentlichen INSERT-Befehl.

Kap3.pdf
Folie 40

3.5.5 METHODEN

Kap3.pdf
Folie 41

Kap3.pdf
Folie 42

3.5.6 VERSCHACHTELTE TABELLEN (NESTED TABLES)

Kap3.pdf
Folie 43

Kap3.pdf
Folie 44

Kap3.pdf
Folie 45

3.5.7 VARIABLE ARRAYS (VARRAYS)

Kap3.pdf
Folie 46

3.5.8 LARGE OBJECTS (LOBS)

Kap3.pdf
Folie 47

Kap3.pdf
Folie 48

3.5.9 EVOLUTION VON DATENMODELLEN

Kap3.pdf
Folie 49

3.5.9.1 AUFGABENVERTEILUNG IM WANDEL DER ZEIT

Kap3.pdf
Folie 50

4 PHYSISCHE DATENORGANISATION

4.1 ÜBERSICHT/ABGRENZUNG

2 Aufgabenstellungen:

- Bildung und Strukturierung internen Sätze
  - variable oder feste Satzlänge
  - feste Formate oder Kennzeichenformate
  - Datentypen
  - Blockung, Segmentierung
    z.B.: 1 Black = 2 kByte = 2048 Byte
- Festlegung der Zugriffspfade
  - Pfade der Daten (Kapitel 4.2)
  - Pfade innerhalb der Daten (Kapitel 4.3)

Kap4.pdf
Folie 1

Datensatz (Mitnr, Name, Vorname, Alter)

- feste Länge ohne Kennzeichen:
  _103 | Meier____ | Paul __ | _37
- feste Länge mit Kennzeichen
  I | _103 | N | Meier____ | A | _37
- variable Länge ohne Kennzeichen
  L | 103 | L | Meier | L | Paul | L | 37
- variable Länge mit Kennzeichen
  I | L | 103 | N | L | Meier | A | L | 37

4.2 ZUGRIFFSPFADE AUSSERHALB DER DATENSÄTZE/INDEXIERUNG

4.2.1 GRUNDPRINZIP DER INDEXIERUNG

Kap4.pdf
Folie 2

4.2.1.1 GEORDNETER BINÄRBAUM

Kap4.pdf
Folie 3

4.2.1.2 GEORDNETER, BALANCIERTER BINÄRBAUM

Kap4.pdf
Folie 4

Binärbaum:

- binäre Suche möglich
- einfaches Einfügen/Ändern
- alle Knoten enthalten Daten
- gut für Datenhaltung im Hauptspeicher

aber DBS $\Rightarrow$ Datenhaltung auf Festplatte $\rightarrow$ Datenhaltung an Blöcken (Seiten) orientieren

4.2.2 B*-BÄUME

Kap4.pdf
Folie 5

BEISPIELSTRUKTUR EINES B*-BAUMS

Kap4.pdf
Folie 6

BSP.: DATENMENGE IM B*-BAUM

DS-Länge:	$80\,\mathrm{Byte}$
Schlüssel + Zeiger:	$10\,\mathrm{Byte}$
Seitengröße:	$2048\,\mathrm{Byte}$
Anzahl DS/Seite:	$\frac{2048}{80} = 25$
durchschnittliche Anzahl DS/Seite:	$\frac{5}{6} \cdot 25 = \boxed{20}$
Anzahl Schlüssel/Seite:	$\frac{2048}{10} = 204$
durschnittliche Anzahl Schlüssel/Seite	$\frac{5}{6} \cdot 204 = \boxed{170}$

Ebenen-Anzahl	DS-Anzahl
1	20
2	$170 \cdot 20 = 3\,400$
3	$170 \cdot 170 \cdot 20 = 578\,000$
4	$170 \cdot 170 \cdot 170 \cdot 20 = 98\,260\,000$

4.2.2.1 B*-BAUM-TRANSFORMATION (SPLITTING)

Kap4.pdf
Folie 7

Struktur der DB aus physischer Sicht:

- Verzeichnis der Seiten
  - Freie Seiten
- je Index
  - Wurzelseite
  - Index-Schlüssel-Seiten
  - Datenseiten

Zugriff von „oben“ nach „unten“: Wurzelseite → Schlüssel-Seiten → Datenseite

B*-Baum:

- HDD-Speicherung
- optimiert für Einzel-Tupel-Zugriffe

4.2.2.2 ERMITTLUNG DER BENÖTIGTEN KAPAZITÄT (ANZAHL DER SEITEN)

Kap4.pdf
Folie 8

Beispiel:

Tupelanzahl	$10\,000$
Spaltenanzahl	10
DS-Länge	85

$$\begin{aligned}\text{Seitenanzahl:} &= \frac{10\,000 \cdot (5+10+85)}{(2\,048-90)\cdot\left(1-\frac{10}{10\,000}\right)} \\ &= \frac{1\,000\,000}{1\,958 \cdot 0,9} = \frac{1\,000\,000}{1\,762,2} \\ &= 567,5 = 568\,\text{Seiten}\end{aligned}$$

BEGRIFFSWEGE

- getrennt von Datensätzen ⇒ INDEX
- im Datensatz ⇒ VERKETTUNG

4.3 ZUGRIFFSPFADE AUSSERHALB DER DATENSÄTZE/VERKETTUNG

4.3.1 ADRESSVERKETTUNG

Kap4.pdf
Folie 9

4.3.1.1 OWNER-MEMBER

Kap4.pdf
Folie 10

4.3.1.2 OWNER1-MEMBER-OWNER2

Kap4.pdf
Folie 11

4.3.2 ANFRAGEBEISPIEL AUF RELATIONALES DBMS

Kap4.pdf
Folie 12

4.4 SPEICHERUNG AUF BASIS VON BITMAP-INDIZES

Problemstellung: Index für Abfrage nutzen?
Strategie klassischer DBMS:
Index: B*-Baum
1. Abfrage (ja, Index nutzen, da Einzel-DS lesen)

```
SELECT Name, Vorname
FROM Kunde
WHERE Kunr=345
```

2. Abfrage (nein, Index nicht nutzen, da alle DS lesen)

```
SELECT AVG(Kredit), Ort
FROM Kunde
GROUP BY (Ort)
```

4.4.1 VERTIKALE POSITIONIERUNG UND BITMAP INDEX

Kap4.pdf
Folie 13

4.4.2 BEISPIEL FÜR I/O-REDUZIERUNG

Kap4.pdf
Folie 14

4.4.3 BEISPIEL INDEX FÜR RELATIONALES DBMS

Kap4.pdf
Folie 15

4.4.4 FAST PROJECTION

Kap4.pdf
Folie 16

4.4.4.1 FFP UND FFFP

Kap4.pdf
Folie 17

4.4.5 LOW FAST

Kap4.pdf
Folie 18

4.4.6 HIGH GROUP

Kap4.pdf
Folie 19

4.4.7 HIGH NON GROUP

Kap4.pdf
Folie 20

4.4.8 BEISPIEL FÜR DEN EINSATZ DER INDEXARTEN

Kap4.pdf
Folie 21

5 PROJEKTIERUNG VON RELATIONALEN DATENBANKSYSTEMEN (BETRIEBLICHE DATENMODELLIERUNG UND DATENBANKANWENDUNGEN)

Datenbank-Anwendungssystem:

- DBS
  - DBMS
    - Datenbasis
- Nutzer/Anwender-Programme

5.1 LEBENSZYKLUS VON DATENBANKANWENDUNGEN

1. Systemdefinition
2. Entwurf
3. Implementierung
4. Dateneingabe und -analyse
5. Test und Validierung
6. Nutzung
7. Verbesserung und Wartung / Anpassung

5.2 ÜBERBLICK ÜBER DEN ENTWURFSPROZESS VON DBS

- Anforderungsanalyse
- Konzeptioneller Entwurf
- Logischer Entwurf
- Implementierungsentwurf
- Physischer Entwurf
- Wartung

5.2.1 DISKUSSION AUSGEWÄHLTER PROBLEMFELDER

	Analyse	Entwurf	Implem.	Test	Installation	Nutzung
Fehlerkosten	0,2	0,5	1	2	5	10-20
Fehlerentdeckung	bei Nutzung	im Test	in Impl.			
Aufwand (soll)	hoch	hoch	hoch	sehr hoch	hoch	gering
Aufwand (oft)	sehr gering	gering	mittel	hoch	sehr hoch	extrem hoch

5.2.2 MITARBEITERANZAHL UND BEARBEITUNGSDAUER

Eigentlich: mit steigender Anzahl der Mitarbeiter sollte die Bearbeitungsdauer linear sinken. Aber: Kommunikationswege heben die Dauer wieder. Zusammen mit den Kommunikationswegen schaffen zu viele Mitarbeiter wieder mehr Aufwand als dass sie die Dauer senken → Badewannenkurve. Die optimale Anzahl an Mitarbeitern ist wichtig.

5.2.3 SOFTWARE-ENTWICKLUNG

- Coding-Guides
  - Namens-Konventionen
  - Kommentierung
- Software-Tests
  - automatisiert (Unit-Tests)
- formale Methoden
  - Versionsverwaltung
  - Ticketsysteme
  - Netzplantechnik
  - Ressourcenplanung
  - Terminplanung
- soziale Kompetenz

5.3 PHASEN DES ENTWURFSPROZESSES

Kap5_1.pdf
Folie 1

1. Anforderungsanalyse/-spezifikation
2. Konzeptioneller Entwurf (ERM)
3. Logischer Entwurf (RM)
4. Implementierungsentwurf
5. Physischer Entwurf
6. Wartung + Anpassung

bis 2.: unabhängig vom logischen Modell
bis 3.: unabhängig vom konkreten DBMS