Zusammenfassung DB1, Kapitel 1-8

1 Inhaltsverzeichnis

2 Einführung, Kapitel 1

Zusammenhang von Information ↔ Daten:
- Information sind vom Menschen bearbeitete Daten
- Information wird aus Daten genommen

Redundanz:
- Mehrfachhaltung der gleichen Daten

2.1 Eigenschaften Dateiverarbeitungssystem

- datenflussorientiert
- Dateien dezentral
- Dateien sequentiell abarbeiten
- Redundanzen zwischen Dateien
- Programme und Daten sind eng gekoppelt
- Mehrbenutzerbetrieb nur beschränkt

2.2 Eigenschaften DBMS

- datenorientiert
- zentrale Datenbasis, verwaltet durch DBMS
- Anwendungen greifen nicht direkt auf Daten zu, via DBMS
- Mehrbenutzerbetrieb
- kontrollierte Redundanz
- Datenintegrität zentral von DBMS gesichert
- Datenpflege, -schutz, -sicherheit zentral sichergestellt

Vorteile:
- einheitliches Datenkonzept
- einmaliger Aufwand (Datendefinition, etc)
- einheitliche Manipulationssprache
- Bereitstellung zentraler Benutzerfunktionen

Nachteile: hohe Kosten, höhere Anforderungen, Programmieraufwand, Risiko, komplexer

2.2.1 Warum redundanzfreie Datenbanken
- keine Widersprüche (doppelte Haltung)

2.2.2 Was gehört zu Datenintegrität
- Datenkonsistenz (widerspruchsfrei)
    - Datenerfassung
- Datensicherheit (Backup)
    - Datenhaltung
- Datenschutz (unbefugte Zugriffe)
    - Verwendung

2.3 Benutzer eines DBMS

- End- Benutzer
- Angelernter Benutzer
- Applikationsprogrammierer
- DBA (Datenbankadministrator)

2.4 3- Ebenen Modell (3- Tier Model)

logische Ebene:

- definiert Datenelemente und Strukturen
- wirt mit DDL beschrieben

externe Sichten:

- Benutzerklassen

interne Ebene:

- Daten aus Sicht der Speicherstrukturen

Datenbank- Modelle:

- relationale DBMS → Daten als Tabellen
- Netzwerk DBMS → Datenobjekte als Records und Beziehungen zwischen Objekten mit Referenzen
- objektorientierte DBMS → Speicherung von Objekten

2.5 Allgemeines

DBMS und Datenbanken befinden sich heutzutage auf leistungsfähigen Servermaschinen. Die Applikationen laufen auf PC's oder Workstations und greifen übers Netzwerk auf die DB- Funktionen zu.

Datenbanksysteme haben die Aufgabe Daten konsistent zu verwalten. Ein Informationssystem ist benutzerorientiert.

2.6 Konzeptionelles Schema – Internes Schema – Externes Schema

Konzeptionelles Schema: Beschreibt einen Ausschnitt der realen Welt in Form von Objekten und deren Beziehungen noch losgelöst von systemtechnischen Aspekten und Einschränkungen. Heute sind hier Methoden der Objektorientierung üblich.

Internes Schema: zeigt wied Daten verwaltet werden; implemenntations-orientiert (Performanz, Verteilung, etc.) produktspezifisch (z.B. SQL); wurde über das logische Schema hergeleitet, hat typischerweise objekt-relationale Eigenschaften. Die Herleitung des internen Schemas aus dem konzeptionellen verlangt eine erste Transformation (OO zu relational).

Externes Schema: Sichten auf Originaldasten, was auch als Transformation aufgefasst werden kann (Views im Sinne von Funktionen => Transformation).

3 Datenmodellierung, Kapitel 2

- Datenmodell ist Metamodell
- Datenmodell legt Datenstrukturierungskonzepte des DBMS fest

3.1 Datenbankmodelle

- hierarchisches Modell (XML)
- Netzwerkmodell
- Relationenmodell
- postrelationale Datenbankmodelle (UML)

3.1.1 Physischer Datenbankentwurf

In Scriptsprache wie SQL

3.1.2 hierarchisches Modell

- Informationen in einer einzigen Hierarchie organisiert
- Beziehungen werden mit den Daten gespeichert
- sequentielle Verarbeitung

Vorteile: Schnelligkeit

Nachteile: zu wenig allgemein, keine Trennung zwischen Anwendung und Daten, Anpassungsarbeiten bei Änderungen

3.1.3 Netzwerkmodell

- Informationen in vernetzten Hierarchien organisiert
- Beziehungen mit Daten gespeichert, Mehrfachbeziehungen möglich
- gut für effiziente Verwaltung

3.1.4 Relationenmodell

- Information als Daten (1. Normalform) in Form von Tabellen gespeichert
- sehr flexibel
- klare und genaue Trennung zwischen Daten und Anwendung

Nachteil: Performance - Probleme

Entwurfsphasen:

1. Anforderungsanalyse
2. konzeptioneller DB- Entwurf → konzeptionelles Datenmodell (UML)
3. logischer DB- Entwurf → logisches Modell (Tabellen)
4. physischer DB- Entwurf → SQL- Scripts
5. interner DB- Entwurf

3.1.5 postrelationale Datenbankmodelle

objektrelational:

- Verschachtelung von Tabellen
- neben Daten auch Methoden gespeichert
- evolutionärer Ansatz: Erweiterung um OO- Konzepte

objektorientiert:

- Modelle basieren auf OO- Sprache
- Funktionalität für Transaktion, Persistenz, parallele Zugriffe
- revolutionärer Ansatz: Neuimplementierung

3.2 Grundbegriffe

Entität	Instanz einer Klasse, Zeile in einer Tabelle (entspricht Objekt)
Entitätsmenge	Menge von Datensätzen, die zusammengefasst werden können (entspr. Objektmenge)
Wertebereich	gültige Werte für ein Attribut
Attribut	Beschreibung eines Merkmals einer Entität
Entitätstyp	Zusammenfassung von Attributen
Schlüssel	Attribut oder Kombination von Attributen deren Wert eine Entität eindeutig identifiziert
Surrogat	künstlicher Schlüssel (wird zusätzlich eingefügt)
Relation	Beziehung zwischen Entitäten, beschreibt Entitätsmenge
Assoziation	verknüpft mehrere Entitätstypen
DDL	Data Definition Language, um Daten, -strukturen, Zusammenhänge etc. zu definieren
Primary Key	Darf nicht null sein, eindeutig identifiziert

3.3 Integritäts- und Konsistenzbedingung

Datenbestand heisst integer, wenn er einen korrekten Ausschnitt aus der Realität beschreibt. Datenbestand heisst konsistent, wenn er in sich widerspruchsfrei ist.

4 relationales Modell: DB- Entwurf, Kapitel 3

4.1 Begriffe

Tupel: repräsentiert eine Entität

PersNr	Name	Chef	Salär
...			
1001	Marxer	1000	10580

Zeile == Tupel

Spalte == Attribut
Feld == Attributwert

thema ©H. Huser

Primärschlüssel: der ausgewählte, identifizierende Schlüssel

- Eigenschaften
  - eindeutig
  - zuteilbar
  - kurz
  - invariant (konstant)

Entitäts- Integrität:
- NULL, undefinierter Attribut- Wert
- Entitäts- Integrität, kein Attribut des Primärschlüssels darf NULL sein
- Nichtschlüssel-Attribute müssen mind. Zugriffsklasse von Schlüssel besitzen

Einfüge-Anomalie:
- Inkonsistenz, welche bei einer Einfüge-Operation die Entitäts-Integrität verletzt (z.B. fehlende Attributwerte innerhalb oder von in Beziehung stehenden Tabellen).

Fremdschlüssel:
- gleichzeitig Primärschlüssel einer anderen Tabelle
- werden verwendet für die Modellierung von Beziehungen

4.1.1 Konzeptioneller Datenbankentwurf

Objekte & Beziehungen in UML

4.1.2 Logischer Datenbankentwurf

Tabellen TabName(Attrib1, Typ1, Attrib2, Typ2, ...)

4.1.3 Konzeptionelles und logisches Datenmodell

Konzeptionelles Datenmodell: Definiert Objekte und deren Eigenschaften; Beziehungen und Konsistenzbedingungen; ist lösungsneutral, Technologie- und Produktunabhängig
Logisches Datenmodell: Beschreibt Struktur der gespeicherten Daten; Technologie- und Produktabhängiges Datenmodell. Kann implementationsbedingt auch künstliche Klassen enthalten, die keinen direkten Bezug zur Realität haben.

4.2 logischer DB- Entwurf

Schritt 1: Abbildung konzeptionelles Modell (UML) in relationales Modell (Tabellen)
Schritt 2: überprüfen der Konsistenz (1. bis 3. Normalform)

Konzeptionelles Modell (UML)

Logisches DB-Modell

```
Student(StudId int ,
    Name string NOT NULL,
    Wohnort string NOT NULL)
```

Student

StudId	Name	Wohnort
1001	Müller	Uster
1002	Huber	Zürich
1003	Walder	Zug
1004	Huber	Zürich

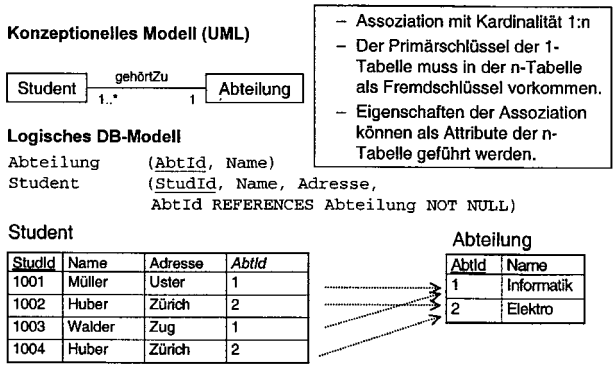
4.2.1 Regeln

4.2.1.1 Regel 1 - Abbildung von Klassen (siehe Bild rechts)

- jede Klasse wird auf eine Tabelle abgebildet

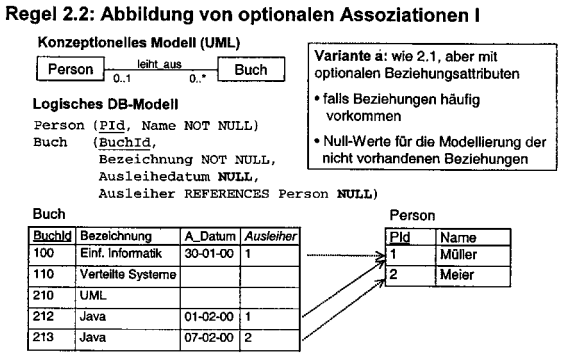
4.2.1.2 Regel 2.1 - Abbildung von 1:n Assoziationen

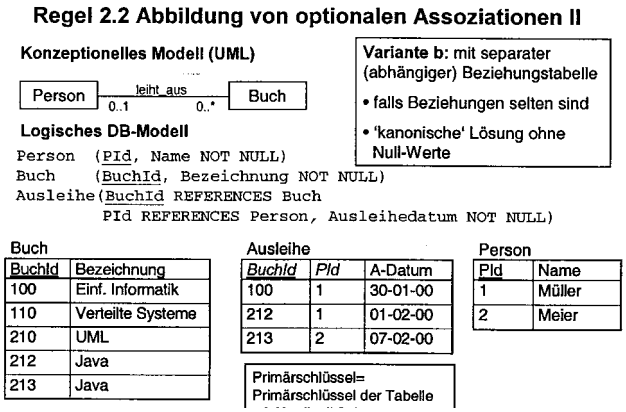
- Primärschlüssel der '1'- Tabelle muss in der 'n'-/'1..*'- Tabelle als Fremdschlüssel vorkommen



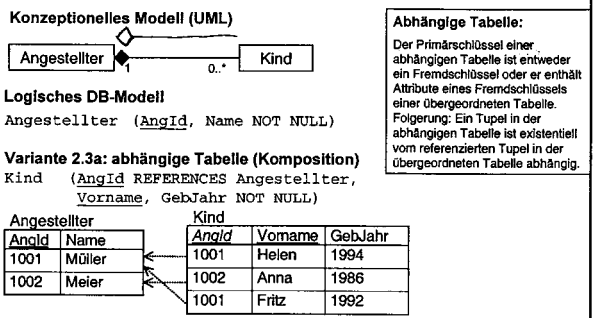
4.2.1.3 Regel 2.2 - Abbildung von optionalen Assoziationen

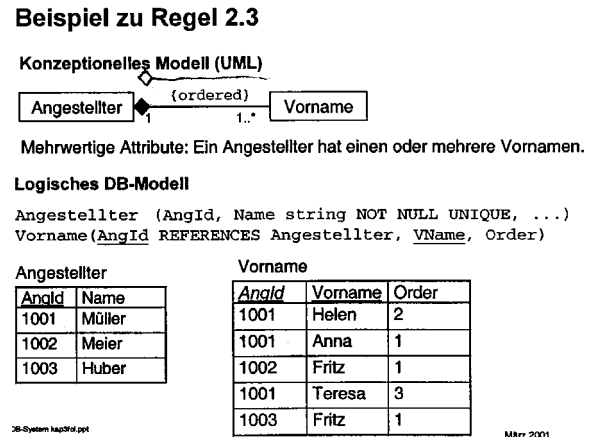
- mit separater Beziehungstabelle, in welcher die jeweiligen Primärschlüssel vorhanden sind





4.2.1.4 Regel 2.3 - Abbildung von Aggregationen

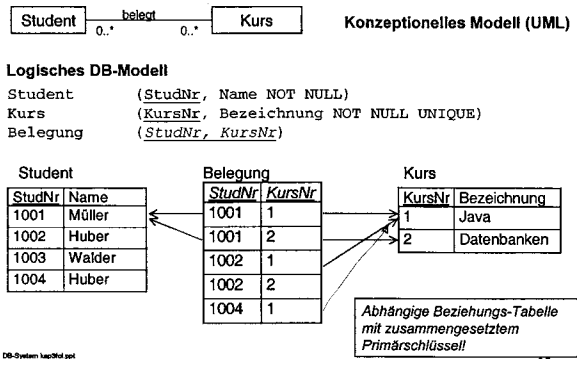




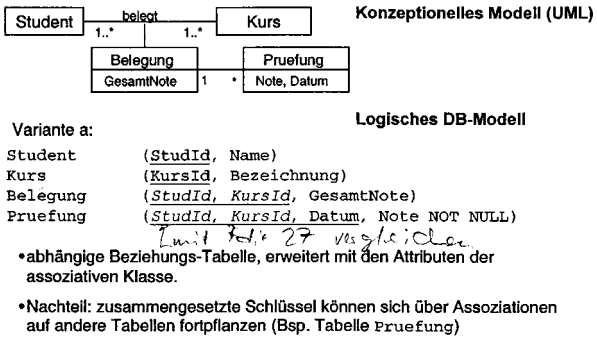
4.2.1.5 Regel 2.4 - Abbildung von n:m Assoziationen

- Beziehungstabelle mit zusammengesetztem Primärschlüssel

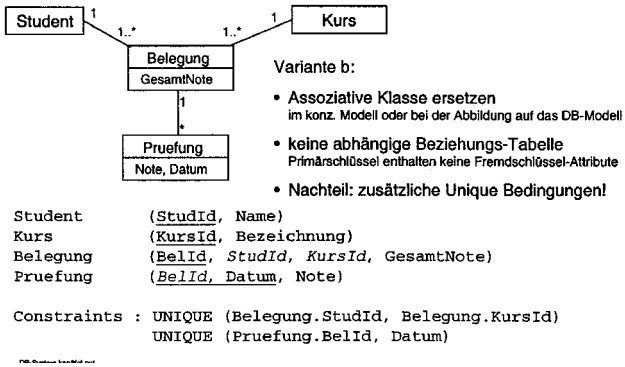
Regel 2.4: Abbildung von n:m Assoziationen



Regel 2.4a: Abbildung von assoziativen Klassen I



Regel 2.4b: Abbildung von assoziativen Klassen II



Regel 2.3 Abbildung von Aggregationen II

Variante 2.3b: unabhängige Tabelle (Aggregation)

```
Kind   (KindId,
        AngId REFERENCES Angestellter NOT NULL,
        Vorname NOT NULL, GebJahr)
Constraint: Unique (Kind.AngId, Kind.Vorname)
```

Angestellter

AngId	Name
1001	Müller
1002	Meier

Kind

KindId	AngId	Vorname	GebJahr
99	1001	Helen	1994
97	1002	Anna	1986
98	1001	Fritz	1992

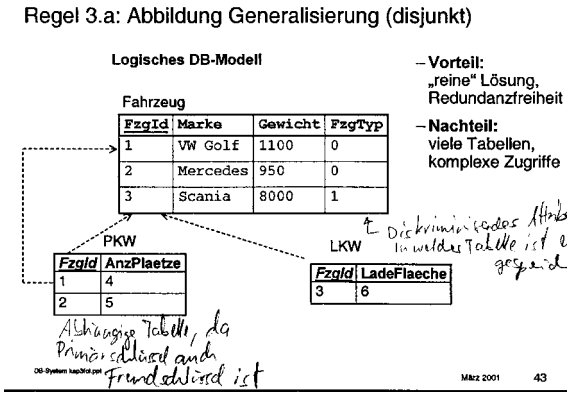
Ein Kind kann unabhängig von einem Angestellten existieren, falls `Kind.AngId NULL` sein darf (Kardinalität!)

4.2.1.6 Regel 3a - Abbildung Generalisierung

- Superklasse wird auf Tabelle abgebildet; enthält Attribut, welches Typ der Subklasse spezifiziert
- jede Subklasse wird auf Tabelle abgebildet
- Primärschlüssel der Subklasse = Primärschlüssel der Superklasse
- Primärschlüssel der Subklasse ist zugleich Fremdschlüssel

Vorteil: reine Lösung, "Redundanzfreiheit"
Nachteil: viele Tabellen, komplexe Zugriffe

Regel 3.a: Abbildung Generalisierung (disjunkt)



Regel 3.a: Abbildung Generalisierung (disjunkt)

- `Fahrzeug (FzgId int, Marke string, Gewicht float, FzgTyp int NOT NULL)`
- `PKW (FzgId REFERENCES Fahrzeug, AnzPlaetze int NOT NULL)`
- `LKW (FzgId REFERENCES Fahrzeug, LadeFlaeche float NOT NULL)`
- Das Attribut `FzgTyp` hat den Wertebereich `0 (=Pkw), 1 (=Lkw).`
- Jedes Fahrzeug wird in **zwei** Tabellen gespeichert.
- Falls die Basisklasse **nicht abstrakt** ist, kann FzgTyp **NULL** sein. Die Daten eines Fahrzeuges sind dann nur in der Tabelle `Fahrzeug` gespeichert.
- Falls die Generalisierung **nicht diskunkt** ist, können die Daten eines Objektes in **mehr als zwei** Tabellen gespeichert sein

4.2.1.7 Regel 3b - Abbildung Generalisierung

- keine Abbildung der Superklasse
- Subklassen werden auf Tabellen abgebildet; enthalten alle Attribute, gemeinsame und zusätzliche
- Schlüssel muss über alle Subklassen- Tabellen eindeutig sein

Vorteil: einfache Zugriffe auf die Elemente der Subklassen
Nachteil: Semantikverlust, komplexe Abfragen über Elemente der Superklasse, Eindeutigkeit des Schlüssels

Regel 3.b: Abbildung Generalisierung (disjunkt)

Logisches DB-Modell

- PKW (FzgId int, Marke string, Gewicht float, AnzPlaetze int NOT NULL)
- LKW (FzgId int, Marke string, Gewicht float, LadeFlaeche float NOT NULL)

PKW ~~Fahrzeug~~

FzgId	Marke	Gewicht	FzgTyp	AnzPlaetze
1	VW Golf	1100	0	4
2	Mercedes	950	0	5

LKW

FzgId	Marke	Gewicht	FzgTyp	LadeFlaeche
3	Scania	8000	1	6

Regel 3.c: Abbildung Generalisierung

Logisches DB-Modell

- Fahrzeug (FzgId int, Marke string, Gewicht float, FzgTyp int NOT NULL, AnzPlaetze int NULL, LadeFlaeche float NULL)
- Jedes Fahrzeug wird in **einer** Tabellen gespeichert. Funktioniert auch für überlappende Generalisierungen.

Fahrzeug

FzgId	Marke	Gewicht	FzgTyp	AnzPlaetze	LadeFläche
1	VW Golf	1100	0	4	
2	Mercedes	950	0	5	
3	Scania	8000	1		6

4.3 Normalisierung

4.3.1 1. Normalform

Relation ist in 1. Normalform, wenn alle Attribute Werte haben, die selber keine Mengen sind.

PC_Daten (Nr, Typ, Proz, RAM, HD, PlusRAM, SW, Vers, SWHersteller)

Nr	Typ	Proz	RAM	HD	Plus RAM	SW	Vers	SWHersteller
PC13	IBM AT02	80286	512	20	128	PC-DOS	3.3	IBM
PC13	IBM AT02	80286	512	20	128	dBASE IV	1.1	Ashton-Tate
PC25	COMPAQ 20	80386	4096	40	0	MS-DOS	3.31	COMPAQ
PC25	COMPAQ 20	80386	4096	40	0	WINDOWS	3.0	MICROSOFT
PC25	COMPAQ 20	80386	4096	40	0	dBASE IV	1.0	Ashton-Tate
PC37	IBM AT02	80286	512	20	1152	PC-DOS	3.3	IBM
PC37	IBM AT02	80286	512	20	1152	WINDOWS	3.0	MICROSOFT

- Die Attribute Nr und SW bilden den Primärschlüssel.
- Die Attribute Typ, Proz, RAM, HD und PlusRAM hängen von Attribut Nr alleine ab, nicht aber vom ganzen Primärschlüssel
- Das Attribut SWHersteller hängt vom Attribut SW alleine ab, nicht aber vom ganzen Primärschlüssel
- Nur das Attribut Vers hängt voll vom Primärschlüssel ab

4.3.2 2. Normalform

Relation ist in der 2. Normalform, wenn sie in der 1. Normalform ist und alle Attribute vom Primärschlüssel abhängig sind.

Relationen in 2. Normalform

Hardware (Nr, Typ, Proz, RAM, HD, PlusRAM)
Software (SW, SWHersteller)
Installiert (Nr, SW, Vers)

Nr	Typ	Proz	RAM	HD	Plus RAM
PC13	IBM AT02	80286	512	20	128
PC25	COMPAQ 20	80386	4096	40	0
PC37	IBM AT02	80286	512	20	1152

SW	SWHersteller
PC-DOS	IBM
dBASE IV	Ashton-Tate
MS-DOS	COMPAQ
WINDOWS	MICROSOFT

Nr	SW	Vers
PC13	PC-DOS	3.3
PC13	dBASE IV	1.1
PC25	MS-DOS	3.31
PC25	WINDOWS	3.0
PC25	dBASE IV	1.0
PC37	PC-DOS	3.3
PC37	WINDOWS	3.0

- Die Attribute Proz, RAM und HD hängen direkt vom Attribut Typ ab, und nur indirekt vom Schlüssel Nr

4.3.3 3. Normalform

Relation ist in der 3. Normalform, wenn sie in der 2. Normalform ist und keine transitiven* Abhängigkeiten hat.

Relation in 3. Normalform

Hardware (Nr, Typ, Proz, RAM, HD, PlusRAM)
Software (SW, SWHersteller)
Installiert (Nr, SW, Vers)
Typen (Typ, Proz, RAM, HD)

Nr	Typ	Plus RAM
PC13	IBM AT02	128
PC25	COMPAQ 20	0
PC37	IBM AT02	1152

SW	SWHersteller
PC-DOS	IBM
dBASE IV	Ashton-Tate
MS-DOS	COMPAQ
WINDOWS	MICROSOFT

Nr	SW	Vers
PC13	PC-DOS	3.3
PC13	dBASE IV	1.1
PC25	MS-DOS	3.31
PC25	WINDOWS	3.0
PC25	dBASE IV	1.0
PC37	PC-DOS	3.3
PC37	WINDOWS	3.0

Typ	Proz	RAM	HD
IBM AT02	80286	512	20
COMPAQ 20	80386	4096	40

* transitiv = ein direktes Objekt nach sich fordernd

5 SQL, Structured Query Language, Kapitel 4

- Abfragesprache von relationalen Datenbanken
- SQL als Data Definition Language (DDL)
- SQL als Data Manipulation Language (DML) für Abfrage und Modifikation
- SQL als Data Control Language für Verwaltung und Kontrolle

5.1 SQL als DDL

5.1.1 Datentypen

- VARCHAR, VARCHAR(20)
- NUMERIC(precision, scale): Dezimalzahlen, precision = Anz. Ziffern, scale = Anz. Ziffern nach Dezimalpunkt
- TINYINT, SMALLINT, INT, INTEGER
- DATE
- TIME
- DATETIME: DATE + TIME

5.1.2 Column-/Table- Constraints

Wenn der Primary Key aus mehr als einem Attribut zusammengesetzt ist, kann er nicht als Column-Constraint definiert werden.

Beispiel Column- Constraint:

```
CREATE TABLE Projekt (
        ProjNr          INTEGER                         PRIMARY KEY,
        …
        ProjLeiter      INTEGER         NULL            REFERENCES          Angestellter( PersNr)
);
```

Beispiel Table- Constraint:

```
CREATE TABLE Projekt (
        ProjNr          INTEGER         NOT NULL,
        …
        ProjLeiter      INTEGER         NULL,
        PRIMARY KEY (ProjNr),
        FOREIGN KEY (ProjLeiter) REFERENCES Angestellter
);
```

Tabelle abändern: Beispiel mit ALTER TABLE:

```
CREATE TABLE Projekt (
        ProjNr          INTEGER         NOT NULL,
        …
```

```
        ProjLeiter      INTEGER     NULL,
);

ALTER TABLE Projekt
        ADD CONSTRAINT pk_ProjNr
        PRIMARY KEY (ProjNr)
;

ALTER TABLE Projekt
        ADD CONSTRAINT fk_ProjAng
        FOREIGN KEY (ProjLeiter)
        REFERENCES Angestellter( PersNr )
;
```

TIP: Foreign- Keys mit Alter Table und die übrigen Constraints als Table- oder Column- Constraints

5.2 SQL als DML

5.2.1 Statements

- SELECT
- UPDATE
- INSERT
- DELETE

5.2.1.1 INSERT

Tupel in existierende Tabelle einfügen:
INSERT INTO *tabellenName* VALUES(*wert1*, *wert2*); oder
INSERT INTO *tabellenName(spaltenname(n))* VALUES(*wert1*, *wert2*);

Bemerkungen:

- wenn keine Spaltennamen angegeben, so muss die Reihenfolge nach VALUES in der entsprechenden Reihenfolge sein; die Anzahl der Werte muss mit der Anzahl Spalten übereinstimmen
- wenn Spaltennamen angegeben, werden nicht aufgeführte Spalten mit Defaults gefüllt bzw. mit NULL

5.2.1.2 SELECT

Auswahl von Spalten und Zeilen.

Beispiele aus AngProj- DB:

```
SELECT *
        FROM Abteilung
;
```
→ *alle Zeilen der Tabelle Abteilung*

```
SELECT Name
        FROM Abteilung
;
```
→ *Namen aller Abteilungen (Projektion)*

```
SELECT DISTINCT Wohnort
        FROM Angestellter
;
```
→ *Duplikate unterdrückt ('reine' Projektion)*

```
SELECT Name, Wohnort
        FROM Angestellter
        ORDER BY Name
;
```
→ *aufsteigend sortiert nach Name*

```
SELECT Name, Salaer
```

```
        FROM Angestellter
        ORDER BY Name DESC, Salaer ASC
;
```
→ *sortiert nach Name absteigend, Salär aufsteigend*

5.2.1.3 WHERE

- bestimmte Zeilen einer Tabelle auswählen (Selektion)
- die SELECT- Anweisung liefert Ergebnistabelle in der nur die Zeilen aufgenommen werden, welche WHERE- Klausel erfüllen
- AND bindet stärker als OR

```
SELECT Name, Wohnort
        FROM Angestellter
        WHERE Wohnort = 'Luzern'
        ORDER BY Name
;
```
→ *nur Angestellten mit Wohnort Luzern*

```
SELECT Name, Salaer, Wohnort
        FROM Angestellter
        WHERE AbtNr = 1 AND ( Salaer > 10000 OR Wohnort = 'Luzern' )
        ORDER BY Name
;
```
→ *Angestellten Abteilung 1 mit Salaer grösser 10000 oder Wohnort Luzern*

5.2.1.4 BETWEEN...AND und IN

- mit BETWEEN...AND Bereiche abfragen

```
SELECT Name, Salaer
        FROM Angestellter
        WHERE Salaer BETWEEN 3000 AND 5000
;

SELECT Name, Wohnort
        FROM Angestellter
        WHERE Wohnort IN ( 'Luzern', 'Zug', 'Horw' )
        ORDER BY Wohnort
;
```

5.2.1.5 LIKE

- Ausdrücke vom Typ CHAR oder VARCHAR mit Stringmasken vergleichen
- '_', beliebiges Zeichen
- %, beliebige Folge
- =, exakte Übereinstimmung

```
SELECT Name, Wohnort
        FROM Angestellter
        WHERE Wohnort LIKE 'Lu%'
;
```
→ *Wohnort beginnend mit Lu*

```
SELECT Name, Wohnort
        FROM Angestellter
        WHERE Wohnort LIKE '_ _ _ _ _ _ _'
;
```
→ *Wohnort mit genau 7 Zeichen*

```
SELECT Name, Wohnort
        FROM Angestellter
        WHERE Wohnort LIKE '_ _ _ _ _ _ _%'
;
```
→ *Wohnort mit mehr als 7 Zeichen*

5.2.1.6 Gruppenfunktionen

MAX, MIN; AVG, SUM, COUNT...

- Gruppenfunktionen liefern als Resultat nur eine Zeile
- Nullwerte werden durch die Gruppenfunktionen übersprungen
- COUNT: NULL- Werte werden nicht gezählt

SELECT MAX(Salaer) FROM Angestellter; → *höchstes Salär*
SELECT MIN(Salaer) FROM Angestellter; → *tiefstes Salär*
SELECT AVG(Salaer) FROM Angestellter; → *durchschnittliches Salär*
SELECT SUM(Salaer) AS "Salaersumme" FROM Angestellter; → *Salärsumme*
SELECT COUNT(*) AS "Anz Mitarbeiter" FROM Angestellter; → *Einträge zählen*

5.2.1.7 GROUP BY und HAVING

GROUP BY:

- teilt Resultattabelle in Gruppen auf, die in der GROUP BY- Spalte gleiche Werte aufweisen
- NULL- Werte als separate Gruppe

```
SELECT Wohnort, COUNT( * )
        FROM Angestellter
        GROUP BY Wohnort
;
```
→ *Tabelle mit verschiedenen Wohnorten und Anzahl Angestellten des jeweiligen Wohnorts*

HAVING:

- nur nach GROUP BY- Klausel
- erlaubt Auswahl von Zeilen
- Bedingung der HAVING- Klausel muss mit Funktion beginnen, welche in SELECT- Klausel vorkommen muss

```
SELECT AbtNr, COUNT( * )
        FROM Angestellter
        GROUP BY AbtNr
        HAVING COUNT( * ) >= 5
;
```
→ *Tabelle mit den Abteilungen und deren Anzahl Mitarbeiter, Abteilung muss mehr als 4 Mitarbeiter haben*

5.2.1.8 JOINS

- Join- Operationen verbinden Tabellen über Spalten mit dem gleichen Datentyp
- Beziehungen zwischen Tabellen, über Fremdschlüssel realisiert
- Join- Operation mit SELECT Statement ausgeführt
- karthesisches Produkt (*tab1* x *tab2*) über Tabellen werden gebildet, d.h. Resultat ist Tabelle mit allen Zeilen von beiden Tabellen

```
SELECT Abteilung.Name, Angestellter.Name
        FROM Abteilung CROSS JOIN Angestellter
;
```

5.2.1.8.1 Inner Joins

Tupel mit Eigenschaft, dass Wert von *AbtLeitung.AbtChef* gleich Wert von *Angestellter.PersNr* ist...

```
SELECT AbtLeitung.AbtNr, Name
        FROM AbtLeitung INNER JOIN Angestellter
        ON AbtChef = PersNr
;
```

Mit zusätzlichem Join erhalten wir den Namen der Abteilung...

```
SELECT AbtLeitung.AbtNr, Abteilung.Name, Angestellter.Name
        FROM Angestellter INNER JOIN
                (Abteilung INNER JOIN AbtLeitung
```

```
                ON Abteilung.AbtNr = AbtLeitung.AbtNr)
        ON Angestellter.PersNr = AbtLeitung.AbtChef
;
```

Die Angestellten der Abteilung Verkauf...

```
SELECT Angestellter.Name AS " MA von 'Verkauf' "
        FROM Angestellter INNER JOIN Abteilung
        ON Angestellter.AbtNr = Abteilung.AbtNr
        WHERE Abteilung = 'Verkauf'
;
```

5.2.1.8.2 Outer Joins

LEFT OUTER JOIN: die Tupel der linken Argumentrelation bleiben auf jeden Fall erhalten
RIGHT OUTER JOIN: die Tupel der rechten Argumentrelation bleiben auf jeden Fall erhalten

Beispiele:

```
SELECT Angestellter.Name, Projekt.Bezeichnung
        FROM Projekt LEFT OUTER JOIN Angestellter
        ON Projekt.ProjektLeiter = Angestellter.PersNr
        ORDER BY Projekt.ProjNr
```

; → *Alle Projekte werden ausgegeben mit zugehörigenm Projektleiter; auch Projekte ohne Mitarbeiter werden ausgegeben...*

```
SELECT Angestellter.Name, Projekt.Bezeichnung
        FROM Projekt RIGHT OUTER JOIN Angestellter
        ON Projekt.ProjektLeiter = Angestellter.PersNr
        ORDER BY Projekt.ProjNr
```

; → *auch Mitarbeiter ohne Chef...*

5.2.1.8.3 Bemerkungen

- SQL muss Spaltennamen immer eindeutig identifizieren können
- für Tabellennamen kann Alias verwendet werden
- bei Equi- Joins sind Alias erforderlich

5.2.1.9 SELECT- Unterabfragen

Beispiele:

```
SELECT Angestellter.Name, Salaer
        FROM Angestellter INNER JOIN Abteilung
        ON Abteilung.AbtNr = Angestellter.AbtNr
        WHERE Abteilung.Name = 'Entwicklung'
        AND Salaer =
                (SELECT MIN( Salaer )
                        FROM Angestellter INNER JOIN Abteilung
                        ON Abteilung.AbtNr = Angestellter.AbtNr
                        WHERE Abteilung.Name = 'Entwicklung'
                )
```

; → *innere Abfrage: kleinstes Salär; äussere Abfrage: welcher Angestellte*

```
SELECT Name
        FROM Angestellter
        WHERE PersNr NOT IN
                (SELECT PersNr
                        FROM Projektzuteilung
                )
```

; → *Mitarbeiter die in keinem Projekt mitarbeiten*

```
SELECT Name
        FROM Angestellter a
```

```
        WHERE EXISTS
                (SELECT *
                        FROM Projektzuteilung
                        WHERE PersNr = a.PersNr
                )
;
```
→ *Mitarbeiter die in einem Projekt mitarbeiten*

```
SELECT Ang.Name, Ang.Salaer
        FROM Angestellter Ang INNER JOIN Abteilung Abt
        ON Ang.AbtNr = Abt.AbtNr
        WHERE Abt.Name = 'Marketing'
        AND Ang.Salaer < ANY
                (SELECT Salaer
                        FROM Angestellter Ang1 INNER JOIN Abteilung Abt1
                        ON Ang1.AbtNr = Abt1.AbtNr
                        WHERE Abt1.Name = 'Entwicklung'
                )
;
```
→ *MA der Abteilung 'Marketing', die weniger verdienen als alle von der 'Entwicklung'*

```
SELECT Ang.Name, Ang.Salaer
        FROM Angestellter Ang INNER JOIN Abteilung Abt
        ON Ang.AbtNr = Abt.AbtNr
        WHERE Abt.Name = 'Marketing'
        AND Ang.Salaer > ALL
                (SELECT Salaer
                        FROM Angestellter Ang1 INNER JOIN Abteilung Abt1
                        ON Ang1.AbtNr = Abt1.AbtNr
                        WHERE Abt1.Name = 'Entwicklung'
                )
;
```
→ *MA der Abteilung 'Marketing', die mehr verdienen als alle von der 'Entwicklung'*

Bemerkungen:
- Unterabfragen haben keine ORDER BY
- Unterabfrage hat nur einen Spaltennamen

5.2.2 Views

- virtuelle Tabellen; wird von einer oder mehreren Tabellen oder Views abgebildet
- Daten werden zur Ausführungszeit aus darunter liegenden Tabellendaten hergeleitet
- können nur unter bestimmten theoretischen Bedingungen verändert werden
- CREATE VIEW

Warum Views?
- Daten unterschiedlich strukturiert
- Zugriffsschutz

Einschränkungen:
Es gibt aus logischen Gründen oder aus praktischen Gründen Regeln, die Update-Operationen (inkl. Insert und Delete) auf Views einschränken:
- Falls mehrere Tabellen verknüpft werden (da SQL-Update nur auf Tupels erlaubt)
- Primary Key: kein Insert falls NOT NULL fehlt
- Weitere: Joins, Distinct, Gruppen-Funktionen und Set-Ops

View anlegen:
```
CREATE VIEW AngPublic (PersNr, Name, Tel, Wohnort) AS
SELECT PersNr, Name, Tel, Wohnort
FROM Angestellter
```

View löschen:
```
DROP VIEW AngPublic
```

View AbtChef als virtuelle Tabelle mit Attributen aus mehreren Tabellen:
CREATE VIEW AbtChef (Name, Salaer, AbtName) AS
SELECT Ang.Name, Salaer, Abt.Name
FROM Angestellter Ang INNER JOIN
(Abteilung ABT INNER JOIN AbtLeitung AL
ON AL.AbtNr=Abt.AbtNr)
ON AL.AbtChef=Ang.PersNr

5.3 SQL als DML

5.3.1 Update

- bestehende Tupel modifizieren

Beispiel:
UPDATE Angestellter
SET Wohnort = 'Luzern'
WHERE Name = 'Marxer, Markus'
; → *'Marxer, Markus' wohnt neu in 'Luzern'*

5.3.2 Delete

- Tupel einer Tabelle löschen

Beispiel:
DELETE Angestellter
WHERE Name = 'X, Y'
; → *'X, Y' wird gelöscht*

5.4 SQL als Kontrollsprache

DB- Benutzer benötigt in der Regel nur Zugriff auf eine kleine Teilmenge der gespeicherten Daten. Die übrigen Daten sollten gar nicht sichtbar sein.

Die Berechtigung für welche Zugriffe von welchem Benutzer erlaubt sind regelt das DBMS. Es ist ebenfalls für die Authentifizierung und Autorisierung verantwortlich.

Erzeugen eines Benutzers:
CREATE USER *user* IDENTIFIED BY *password;*

Ändern des Passwortes:
ALTER USER *user* IDENTIFIED BY *password;*

Löschen eines Benutzers:
DROP USER *user;*
→ löscht den Benutzer nur, wenn keine Objekte existieren die von diesem Benutzer angelegt wurden

Löschen eines Benutzers:
DROP USER *user CASCADE;*
→ löscht den Benutzer sowie seine Objekte

5.4.1 Privilegien

Privilegien werden mit dem Befehl GRANT vergeben und mit REVOKE wieder entzogen. Typische Privilegien sind:

- create session, create table, create view, create user, grant any privilege, create profile, create role, grant any role, audit system, audit any

Syntax:
GRANT *privilege* (können auch mehrere sein; durch ',' abgetrennt)

TO *user* (können auch mehrere sein; durch ',' abgetrennt);

REVOKE *privilege* (können auch mehrere sein; durch ',' abgetrennt)
TO *user* (können auch mehrere sein; durch ',' abgetrennt);

Privilegien können auch auf Objekte vergeben werden. Typische Privilegien sind:

- DELETE, INSERT, REFERENCES, SELECT, UPDATE, EXECUTE, ALTER

Syntax:
GRANT SELECT ON *Objekt* TO *user*;
z.B. GRANT SELECTION ON Angestellter TO public

GRANT ALL ON Abteilung TO Mueller WITH GRANT OPTION

5.5 Constraints

5.5.1 Column-Constraints oder: Was darf ein Attribut?

CREATE TABLE Abteilung(
AbtNr INTEGER **PRIMARY KEY**,
Name VARCHAR2(20) **NOT NULL UNIQUE**,
Salaer DECIMAL(7,2) **NOT NULL CHECK (Salaer between 1000 AND 20000)**
ProjNr INTEGER **NOT NULL REFERENCES Project**);

5.5.2 Table Constraints

PRIMARY KEY(AbtNr),
FOREIGN KEY(ProjNr) REFERENCES Project,
CHECK (Salaer between 1000 AND 20000));

Mit ALTER TABLE:
ALTER TABLE Angestellter
ADD CONSTRAINT TelCnstr
CHECK (Wohnort IN ('Zuerich', 'Basel', 'Rapperswil'))

5.5.3 Referentielle Integrität

Löschen eines Tupel in einer referenzierten Tabelle:

Strategie	**Operation Löschen von Tuples der Tabelle A**
ON DELETE CASCADE Fortpflanzung auf abh. Tabelle B: Alle Tupels in B, die eines der zu löschenden Tupels in A referenzieren, werden auch gelöscht	CREATE TABLE B (bKey INTEGER PRIMARY KEY, aRef INTEGER REFERENCES A(aKey) **ON DELETE CASCADE**)
ON DELETE RESTRICT Falls ein Tupel in B eines der zu löschenden Tupels aus A referenziert, wird die Löschoperation nicht ausgeführt Falls nichts angegeben wird, ist RESTRICT default	CREATE TABLE B (bKey INTEGER PRIMARY KEY, aRef INTEGER REFERENCES A(aKey) **ON DELETE RESTRICT**)
ON DELETE SET NULL Das Fremdschlüsselattribut aRef eines Tupels aus B wird auf NULL gesetzt, falls das referenzierte Tupel aus A gelöscht wird	CREATE TABLE B (bKey INTEGER PRIMARY KEY, aRef INTEGER REFERENCES A(aKey) **ON DELETE SET NULL**)
ON DELETE SET DEFAULT Das Fremdschlüsselattribut aRef eines Tupels aus B wird auf Defaultwert gesetzt, falls das referenzierte Tupel aus A gelöscht wird	CREATE TABLE B (bKey INTEGER PRIMARY KEY, aRef INTEGER REFERENCES A(aKey) **ON DELETE SET DEFAULT**)

ALTER TABLE AbtLeitung
ADD CONSTRAINT fk_AbtLAbt

FOREIGN KEY (AbtNr) REFERENCES Abteilung
ON DELETE CASCADE

Ändern des Primärschlüssels eines Tupel

Strategie	**Operation Löschen von Tuples der Tabelle A**
ON DELETE CASCADE Fortpflanzung auf abh. Tabelle B: Der Wert von aRef wird aktualisiert	CREATE TABLE B (bKey INTEGER PRIMARY KEY, aRef INTEGER REFERENCES A(aKey) **ON UPDATE CASCADE**)
ON DELETE RESTRICT Falls ein Tupel in B eines der zu ändernden Tupels aus A referenziert, wird die Änderung nicht ausgeführt Falls nichts angegeben wird, ist RESTRICT default	CREATE TABLE B (bKey INTEGER PRIMARY KEY, aRef INTEGER REFERENCES A(aKey) **ON UPDATE RESTRICT**)
ON DELETE SET NULL Das Fremdschlüsselattribut aRef eines Tupels aus B wird auf NULL gesetzt, falls der referenzierte Primärschlüssel aus A geändert wird	CREATE TABLE B (bKey INTEGER PRIMARY KEY, aRef INTEGER REFERENCES A(aKey) **ON UPDATE SET NULL**)
ON DELETE SET DEFAULT Das Fremdschlüsselattribut aRef eines Tupels aus B wird auf einen Defaultwert gesetzt, falls der referenzierte Primärschlüssel aus A geändert wird	CREATE TABLE B (bKey INTEGER PRIMARY KEY, aRef INTEGER REFERENCES A(aKey) **ON UPDATE SET DEFAULT**)

6 Recovery (Datensicherung), Kapitel 6

6.1 Fehlerklassifikation

lokale Fehler einer Transaktion:

- Fehler in der Applikations- SW
- expliziter Abbruch einer Transaktion durch Benutzer
- System führt zum Scheitern der Transaktion
- Lösung: Änderungen müssen rückgängig gemacht werden

globales Undo und globales Redo:

- alle nicht mit 'Commit' abgeschlossenen Transaktionen zurücksetzen
- allfällige Änderungen, die bereits auf die Disk geschrieben wurden, rückgängig machen
- Änderungen von abgeschlossenen Transaktionen, die noch nicht auf die Disk geschrieben wurden, nachführen

Fehler mit Hintergrundspeicherverlust:

- bereits auf der Disk gespeicherte Daten gehen verloren (z.B. durch ein Feuer, head crash...)
- Rekonstruktion mit archivierten Backup's und Log- Files

6.2 Strategien für das Ein- und Auslagern von Puffer- Seiten

6.2.1 steal/ no steal

no steal: Das Ersetzen einer Seite, die von einer aktiven Transaktion modifiziert wurde und noch nicht mit unifix freigegeben wurde, ist ausgeschlossen.

steal: Jede nicht fixierte Seite ist Kandidat für die Ersetzung.

6.2.2 force/ no force

Nach 'Commit' müssen alle Änderungen auf die Disk geschrieben werden.

force: macht dies sofort bei Ausführung des 'Commits'

no force: erzwingt unmittelbares Rückschreiben nicht

6.2.3 Bemerkungen

Kombination 'no steal/ force' ist vorteilhaft, da sie weniger Probleme mit dem Recovery bereitet. In Bezug auf die Performance ist jedoch die Kombination 'steal/ no force' vorzuziehen.

6.3 Logfile- Techniken

Update- Transaktion:

- Positionieren des Datensatzes auf Disk
- Einlesen des Datensatzes in Systempuffer
- aktualisieren der Werte
- Rückschreiben des aktualisierten Datensatzes auf Disk
- nach System- Crash, Mechanismen:
    - UNDO
    - REDO

Technik für die Lösung dieser Probleme ist das Führen eines Log- Files, in dem sämtliche Informationen über Änderungsoperationen auf den Daten gespeichert werden

6.3.1 incremental log with deffered updates

Log- File enthält folgende Informationen:

- Transaction Record
- Modifikationen auf Datenwerte als Schreiboperationen beschrieben

Notation:

<T, starts>	Start einer Transaktion
<T, commits>	Ende Transaktion mit Commit
<T, aborts>	Ende Transaktion mit Rollback
<T, X, c>	Schreibbefehl, der das Feld X auf den Wert c setzt

6.3.1.1 Ablauf

- Markiere Start (<T, starts>)
- für jede Modifikations- Operation, <T, X, c>
- Commit- Befehl zuerst auf Log- File (<T, commit>), dann Log- File sichern
- DBMS interpretiert Transaktionsbeschreibung auf Log- File
- DBMS führt Modifikationen auf Datenbank aus

6.3.1.2 Verhalten nach Failure

Bei Datenverlust wird Log- File analysiert. Alle nicht abgeschlossenen Transaktionen haben keine Auswirkungen auf der gespeicherten Datenbank hinterlassen und müssen daher nicht weiter behandelt werden. Alle abgeschlossenen Transaktionen müssen aber nochmals durchgeführt werden (REDO).
Periodisch Checkpoint durchgeführt:

- modifizierte Datenblöcke von Systempuffern im Hauptspeicher auf die Disk schreiben
- Log- Records vom Hauptspeicher auf Disk
- Checkpoint- Record auf das Log- File schreiben

6.3.2 incremental log with immediate updates

6.3.2.1 Ablauf

- Markiere Start der Transaktion (<T, starts>)
- für jede Modifikations- Operation, <T, X, alter Wert, neuer Wert>
- DBMS schreibt jetzt die Änderung in die DB- Puffer im Haupspeicher
- falls Commit, zuerst auf Log- File vermerkt, dann Log- File auf Disk speichern

6.3.2.2 Verhalten nach System Failure

DBMS kann mit Hilfe des Log- Files eine Transaktion zurücksetzten, indem das Log- File in umgekehrter Richtung interpretiert wird. Zuerst werden alle Transaktionen zurückgesetzt, die noch nicht mit Commit abgeschlossen wurden. Anschliessend werden Transaktionen seit dem letzten Checkpoint nochmals ausgeführt.

6.4 Recovery nach Disk- Crash

Im schlimmsten Fall sind die gespeicherten Daten verloren. Dann muss der letzte Datenbackup geladen werden. Mit Hilfe des Log- Files werden dann sämtliche Transaktionen rekonstruiert.

6.4.1 Full Backup, Incremental Backup

Jeden Abend wird ein Backup der Datenbank und des Log- Files erstellt (Full Backup), und nach jeder Stunde wird ein Backup des Log- Files erstellt (Incremental Backup).

7 Transaktionen, Synchronisation von parallelen Transaktionen

- Transaktionsmanager koordiniert die von den Applikationen gestarteten Transaktionen, kommuniziert mit dem Scheduler
- Recovery- Manager muss Daten wieder in einen konsistenten Zustand zurückführen mit Hilfe des Log- Files
- Puffer- Manager die internen DB- Puffer

Probleme bei fehlender oder fehlerhafter Synchronisation:

- the lost update problem
- the uncommited update problem
- the problem of inconsistent analysis
- phantoms

7.1.1 'the lost update' problem

Zwei Transaktionen holen sich den Selben Wert aus der Datenbank, verändern ihn und schreiben ihn wieder zurück. Der geänderte Wert der ersten Transaktion wird überschrieben wenn die zweite Transaktion ihren Wert zurück schreibt, da sie immer noch mit den alten Wert gearbeitet hat.

Zeit	Transaktion 1	Transaktion 2	A
0	begin();		100
1	a = read(A);	begin();	100
2	a = a +10;	a = read(A);	100
3	write(a,A);	a = a+20;	100
4	commit();	write(a,A);	110
5		commit();	120

Das zweite `write()` überschreibt den geänderten Wert mit einem alten Ergebnis!

7.1.2 dirty read

Zwei Transaktionen verändern das Selbe Attribut einer Tabelle. Die erste Transaktion schreibt ihre Änderung in die DB zurück und die zweite liest den geänderten Wert aus. Die erste nimmt dann jedoch ihre Änderungen mit einem `rollback` zurück während die zweite Transaktion mit dem geänderten Wert weiter rechnet.

Zeit	Transaktion 1	Transaktion 2	A
0	begin();		100
1	a = read(A);		100
2	a = a +10;		100
3	write(a,A);	begin();	110
4		a = read(A);	110
5	rollback();	a = a+10;	100
6		write(a,A);	120
7		commit();	120

Das zweite `read()` liest einen Wert aus, der durch das `rollback()` ungültig wird und rechnet damit!

7.1.3 non-repeatable read

Eine Transaktion liest von mehreren DB-Objekten Werte, welche gleichzeitig von einer anderen Transaktion verändert werden.

7.1.4 phantom rows

Eine Transaktion liest eine Menge von Tupels aus einer Tabelle, welche das Resultat einer SELECT-Abfrage sind. Danach verändert eine andere Transaktion die Tabelle. Wenn nun die Selbe Abfrage wiederholt würde, so würde das Resultat andere Tupels zurückliefern. Die zusätzlichen Tupels erscheinen dann als „Phantom Rows“ im Resultat.

7.2 Serialisierbarkeit

Ablaufplan heisst seriell, wenn eine Transaktion nach der anderen vollständig aufgeführt wird.

Wenn alle Transaktionen strikte seriell durchgeführt werden, führt dies zu einem schlechten Durchsatz mit langen Reaktionszeiten.

Ablaufplan heisst konsistent, wenn er dieselben Resultate liefert, wie ein ausgezeichneter serieller Ablaufplan. Ein solcher nicht-serieller Ablaufplan heisst serialisierbar.

Für die Serialisierbarkeit ist Reihenfolge der Read- und Write- Operationen relevant. Regeln:

- falls Transaktionen nur lesend zugreifen, Reihenfolge beliebig
- falls Transaktionen auf disjunkte (getrennte) Datenbereiche zugreifen, Reihenfolge beliebig
- falls Transaktion Datenbereich modifiziert und andere Transaktion gleichen Datenbereich liest oder schreibt, Reihenfolge wichtig

7.3 Synchronisationsverfahren

- Sperrprotokolle: gemeinsam genutzte Daten mit Sperren (Locks) schützen
  - ist optimal, wenn Wahrscheinlichkeit für Konflikte hoch
- optimistische Verfahren gehen davon aus, dass im Normalfall keine Konflikte auftreten
  - ist optimal, wenn Wahrscheinlichkeit für Konflikte klein

7.4 Konsistenzprotokolle

Transaktion T sieht konsistenten Zustand falls:

- T keine temporären Inkonsistenzen einer anderen Transaktion liest oder schreibt
- T keine durch T geänderten Daten vorzeitig freigibt
- keine andere Transaktion Daten von T vorzeitig liest oder überschreibt

7.5 Sperrprotokolle

Sie legen fest, wie die gemeinsam genutzten Daten durch Sperren geschützt werden. Folgende Sperren stehen zur Verfügung:

- Exklusive Sperren (x- locks), reserviert Datenbereiche für sich
- Teilsperren (s- locks), Datenbereich für Lesezugriff reservieren; andere Transaktion darf lesend zugreifen

Verträglichkeit von Sperren:

		vorhandene Sperre	
		s-lock	x-lock
neu verlangte Sperre	s-lock	*Verträglich*	*Konflikt*
	x-lock	*Konflikt*	*Konflikt*

7.6 Zwei- Phasen- Sperrprotokoll (Two- Phase- Locking)

Sie verlangt, dass eine Transaktion in einer ersten Phase für alle Objekte, die sie lesen bzw. schreiben möchte, Teilsperren bzw. exklusive Sperren verlangt. Nach dem Zugriff auf die Objekte gibt die Transaktion in einer zweiten Phase die Sperren wieder frei. Zwei- Pasen- Sperrprotokoll garantiert die Serialisierbarkeit.
Regeln:

- Datenelement vor ersten Benutzung mit Sperre belegen
- Nach Freigabe keine neue Sperre
- Am Ende alle Sperren freigegeben

Nachteil: *Änderungen werden sichtbar bevor Transaktion beendet ist*

7.7 Deadlocks

Deadlocks können auftreten, wenn Transaktionen die selben Ressourcen reservieren und sich so gegenseitig blockieren.
Um Deadlocks zu erkennen kann ein *Wait-For-Graph* aufgezeichnet werden. Zyklen im Graph bedeuten einen Deadlock.

Bedingungen:

- gegenseitiger Ausschluss
- hold and wait
- non preemption
- circular wait

Beispiel:

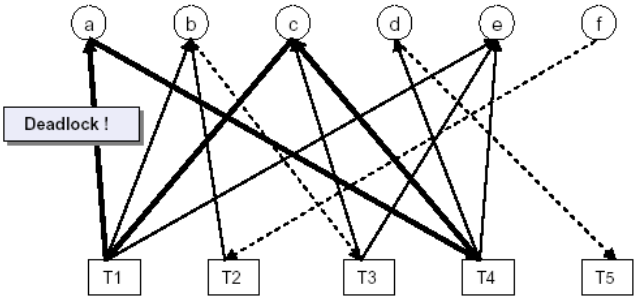
Ressourcen
a,b,c,d,e,f

Sperren
X = Exclusive
S = Shared (nur-lesen)

Aktuelle Sperren:

T1(X,a)	T1(S,b)	T1(S,e)
T2(S,b)		
T3(S,c)	T3(S,e)	
T4(S,c)	T4(X,d)	T4(S,e)

Zusätzliche Sperren:
T1(X,c)
T2(S,f)
T3(X,b)
T4(S,a)
T5(X,d)



Wait-For-Graph

7.7.1 Deadlock- Erkennung

Konstruktion des Wait- For- Graph:

- für jede Transaktion Ti Knoten Ti
- gerichtete Kante Ti > Tj, falls Ti auf Gewährung einer Sperre wartet, die von Tj gehalten wird

Wenn der Graph eine Schlaufe aufweist liegt eine 'Circular Wait' Bedingung und damit ein Deadlock vor.

7.8 Beispiel:

T1 = RA WA WB
T2 = RA RB WA
T1: Liest Wert von A, erhöht diesen um 100, schreibt Wert nach A zurückT1 schreibt alter Wert von A nach B.
T2: erhöht Wert von A um 200 und liest Wert von B.
Anfangswert von A = 10 und B = 5

S1	R1A	W1A	W1B	R2A	R2B	W2A
A	10	110	110	110	100	310
B	5	5	10	10	10	10

S2	R2A	R2B	W2A	R1A	W1A	W1B
A	10	10	210	210	310	310
B	5	5	5	5	5	210

S3	R1A	R2A	R2B	W1A	W1B	W2A
A	10	10	10	110	110	210
B	5	5	5	5	10	10

Lost Update!

S4	R1A	W1A	R2A	R2B	W1B	W2A
A	10	110	110	110	110	310
B	5	5	5	5	10	10

Dirty Read!

Rollback möglich

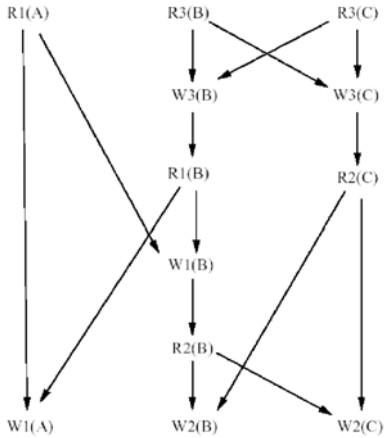
Endwerte: **S5:** A = 210, B = 10
S6: A = 210, B = 10

S1 und S2 sind seriell.
S3, S5 und S6 sind nicht serialisierbar wegen abweichender Endwerte. S4 ist nicht serialisierbar wegen falsch gelesenem Wert.

7.8.1 Dependency-Graph (D-Graph)

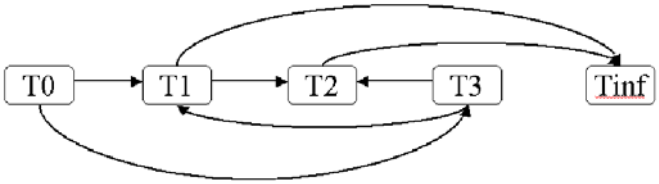
D-Graph bestimmen für folgenden Schedule:
S = R1A R3C R3B W3C W3B R1B R2C W1B R2B W2B W2C W1A

immer R zu W mit gleicher Zahl gegen unten



7.8.2 Conflict-Graph (C-Graph)

S = R1A R3C R3B W3C W3B R1B R2C W1B R2B W2B W2C W1A
mit Hilfe des C-Graphen überprüfen, ob S serialisierbar ist:

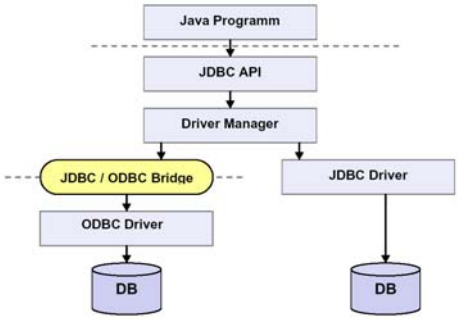


Der C-Graph enthält keinen Zyklus, damit ist der Schedule serialisierbar und äquivalent zu T3, T1, T2, also: S' = R3C R3B W3C W3B R1A R1B W1B W1A R2C R2B W2B W2C

8 JDBC

8.1 Überblick

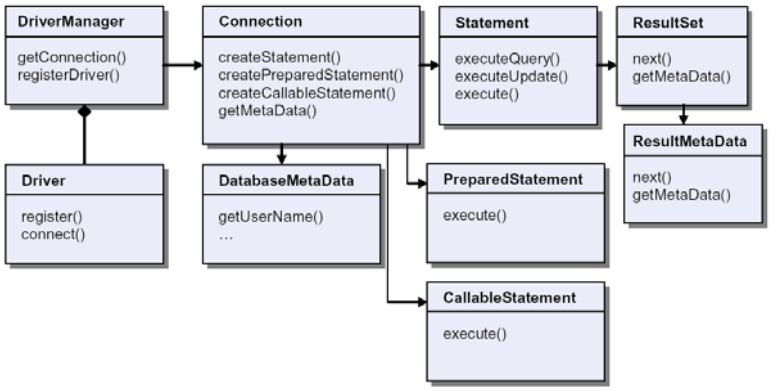
JDBC steht für **J**ava **D**ata **B**ase **C**onnection und ist eine standardisierte API, die es ermöglicht aus Java-Programmen auf eine beliebige Datenbank zuzugreifen, welche einen geeigneten Treiber zur Verfügung stellt.



8.2 Schnittstellen bei 2- / 3-Tier(Schichten)-Architekturen

2-tier: Client und Database (DB-Schnittstelle auf dem Client)
3-tier: Client, Middle Tier Server, Database (DB-Schnittstelle auf dem Middle Tier Server)

8.3 Die wichtigsten JDBC - Klassen:



DriverManager:	Laden / Registrieren
Driver:	Umwandlung in SQL
Connection:	Verbindungsaufbau
Statement:	Führt Query-Operationen durch
ResultSet:	Verwaltet das Resultat

8.4 Treiber laden und registrieren

Der Treiber wird entweder während der Laufzeit dynamisch durch den ClassLoader geladen und registriert…

```
try {
        Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
} // keine überprüfung, ob schon vorhanden
catch( ClassNotFoundException e) {
        System.out.println("Treiber nicht gefunden: "+e);
}
```

…oder mit ins Projekt eingebunden und manuell beim Treibermanager registriert.

```
try {
        DriverManager.registerDriver( new sun.jdbc.odbc.JdbcOdbcDriver() );
        } //zur Kompilierungszeit wird geprüft, ob schon vorhanden
catch( SQLException e) {
        System.out.println(e);
}
```

8.5 Datenbankverbindung aufbauen

```
String url = "jdbc:odbc:mydatabase";
```

```
try {
        con = DriverManager.getConnection( url, "username", "password" );
} //analog zu sql-Befehl: CONNECT username/password
catch( SQLException e) {
        System.out.println("Fehler beim Verbinden "+e);
}
```

Beispiele für Connection Strings:

```
String url = "jdbc:oracle:oci8:@mydatabase";
String url = "jdbc:orcale:thin:@localhost:1521:mydatabase";
```

8.6 Standard SQL-Statements

Normale Statements werden dazu benötigt um SQL-Queries oder Befehle auf der Datenbank auszuführen. Statements können vom Connection-Objekt erzeugt werden.

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

Das Statement Objekt besitzt drei Methoden um Befehle auszuführen:

ExecuteQuery für Abfragen, welche ein Resultset zurückliefern

```
ResultSet result;
result = stmt.executeQuery("SELECT * FROM Daten");
```

Execute Update für `CREATE`, `DELETE`, `INSERT` und `UPDATE` Aufrufe:

```
int changedTupels
= stmt.executeUpdate("INSERT INTO Daten VALUES(0,1,2)");
```

Execute für den Aufruf von Stored Procedures

```
boolean success = stmt.execute("SELECT * FROM Daten");//true falls ok
```

8.7 Lesen von NULL-Werten

```
ResultSet rs = stmt.executeQuery(„SELECT * FROM Table");
while( rs.next() )
{
        int x = rs.getInt("Spalte1"); //getString(1), getDate(), getTime(), ...
        if(rs.wasNull)
        {
                x = 2; // aus Defaultwert setzen
        }
}
```

8.8 Result Set

- ResultSet hat einen Cursor, der auf das aktuelle Tupel zeigt
- Zugriff auf Daten via Cursor mit der getXXX() – Methode
- Cursor kann mit next() – Methode bewegt werden
- ResultSet sollte nach Gebrauch mit der Close – Methode geschlossen werden

Spalten

Zeilen

Daten	Daten
Daten	Daten

Result Set

8.9 Beispiel für eine Datenabfrage

```
Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
Connection con = DriverManager.getConnection("jdbc:odbc:angproj", "user", "pass");
Statement stmt = con.createStatement();
String query = "SELECT * FROM Daten";
ResultSet rs = stmt.executeQuery(query);
while( rs.next() ) {
        String vorname = rs.getString("Vorname");
        String name = rs.getString(2); // 2. Spalte
        Float kontostand = rs.getFloat("Kontostand");
        System.out.println(vorname+" "+name+": "+kontostand);
} stmt.close(); rs.close(); con.close();
```

8.10 Prepared Statements

Prepared Statements werden mit Platzhaltern initialisiert und auf dem Datenbankserver vorkompiliert. Anschliessend können für jeden Aufruf nur noch die Parameter für die Platzhalter übergeben werden, was den Aufruf schnell macht! Prepared Statements werden dort eingesetzt wo eine grosse Menge von gleichartigen

`UPDATES` durchgeführt werden müssen.

```
String str = "UPDATE Table SET Name = ?
WHERE userID = ?";
PreparedStatement preStmt = con.preparedStatement(str);
for(int i=0; i<10; i++)
{
String name = "Name "+i;
// Werte zuweisen
preStmt.setString(1, name);
preStmt.setInt(2, i);
preStmt.setNULL(3, java.SQL.Types.DECIMAL );
preStmt.executeUpdate();
}
preStmt.close();
```

8.11 Callable Statements

Callable Statements werden für den Aufruf von Stored Procedures mit IN oder OUTParametern verwendet.

Beispiel: Aufruf einer Stored Procedure mit einem IN und einem OUT-Parameter

```
PROCEDURE example( y OUT VARCHAR2, x IN VARCHAR2 )
```

Escape-Sequenz: Driver muss standart SQL auf propriätäres Orcale-SQL abbilden.

Java-Code:

```
CallableStatement callStmt =
conn.prepareCall( " { call example(?,?) } " ); // Typ für OUT-Parameter festlegen
callStmt.registerOutParameter(1, Types.VARCHAR); // IN-Parameter definieren
callStmt.setString(2, "Test");
callStmt.execute();
String result = callStmt.getSring(1);
```

8.12 Transaktionen

Standardmässig ist des Auto-Commit Mode eingeschaltet, was heisst, dass jede SQLAnweisung als eine Transaktion angesehen wird und automatisch commited wird.

Auto-Commit Mode ausschalten:

```
con.setAutoCommit( false );
```

Manuell Transaktionen abschliessen

```
con.commit(); con.rollback();
```

8.13 ResultSet Meta-Daten

Jedes ResultSet enthält eine Referenz auf eine Meta-Daten-Klasse, welche folgende Zusatzinformationen enthält:

Anzahl der Kolonnen des ResultSets

für jede Spalte/Kolonne: Name, Typ, NULL oder NOT NULL, schreibbar, Darstellungsinfo, Tabellenname

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM Daten");
ResultSetMetaData rsmd = rs.getMetaData();
int anzahlSpalten = rsmd.getColumnCount();
String spalteAusTabelle= rsmd.getTableName(spaltenNr);
String spaltenTyp = rsmd.getColumnTypeName(spaltenNr);
```

8.14 Database Meta-Daten

Über die DatabaseMetaDaten können globale Informationen über die Datenbank ausgelesen werden wie: SQL Sprachumfang, DB Produktname, Driver Eigenschaften, unterstützte Typen, Eigenschaften der Transaktionen

```
DataBaseMetaData dbmd = con.getDataBaseMetaData();
String dbProdName = dbmd.getDatabaseProductName();
String version = dbmd.getDatabaseProductVersion();
String driverName = dbmd.getDriverName();
String driverVers = dbmd.getDriverVersion();
String userName = dbmd.getUserName();
```

8.15 Isolation Level lesen / ändern

Der Isolation-Level kann ausgelesen und geändert werden.

```
int isolationLevel = con.getTransactionIsolation(); //Lesen des aktuellen Levels
if(DataBaseMetaData.supportsTransacionIsolationLevel
(isolationlevel))
{ con.setTransactionIsolation( isolationLevel ); //Setzten des Levels }
TRANSACTION_NONE, TRANSACTION_READ_UNCOMMITED, TRANSACTION_READ_COMMITED,
TRANSACTION_REPEATABLE_READ, TRANSACTION_SERIALIZABLE
```

8.16 Prüfungsaufgaben zu JDBC

Aufgabe1:
Das folgende JDBC-Programm baut eine Verbindung zur Datenbank auf und führt einige 'prepared'-Abfragen aus. Ergänzen Sie die fehlenden Codestücke. Die Aufgabestellung zu den Teilaufgaben finden Sie direkt in den betreffenden Code-Abschnitten. Lösen Sie diese in der Reihenfolge 4.1, 4.2, 4.3, 4.4.

```
import java.sql.*;
import java.io.*;
public class prf200012 {
Connection con;
PreparedStatement queryByProjektName = null;
public prf200012 (Connection c) {
con = c;
}
```

Aufgabe 2:
Die Prozedur prepareQueryByProjektName soll die folgende SQL-Anweisung an den DBMS-Server für eine prepared-Ausführung senden. Diese vorkompilierte Anweisung wird dann über die Member-Variable queryByProjektName referenziert. Bei der Ausführung der Anweisung wird die Projektbezeichnung als Argument übergeben. (2 P)

```
public void prepareQueryByProjektName () throws SQLException {
if ( queryByProjektName == null )
{
queryByProjektName = con.prepareStatement (
"SELECT P.Bezeichnung AS ProjektName, A.Name AS PLeiter, A.Salaer AS PL-Salaer " +
"FROM Projekt P, Angestellter A " +
"WHERE P.Bezeichnung = ? AND P.ProjLeiter = A.PersNr");
}
}
}
```

Aufgabe 3
PrintResultSet ist eine generische Methode, welche einen beliebigen Result-Set ausgibt. Ergänzen Sie die markierten Stellen. (5P)

```
void printResultSet (ResultSet rs) throws SQLException {
/*generische Routine: gibt beliebigen Resultset aus in der Form
SpalteName1 SpaltenName2 ... SpaltenNamei
wert11 wert21 werti1
wert12 wert22 werti2
*/
```

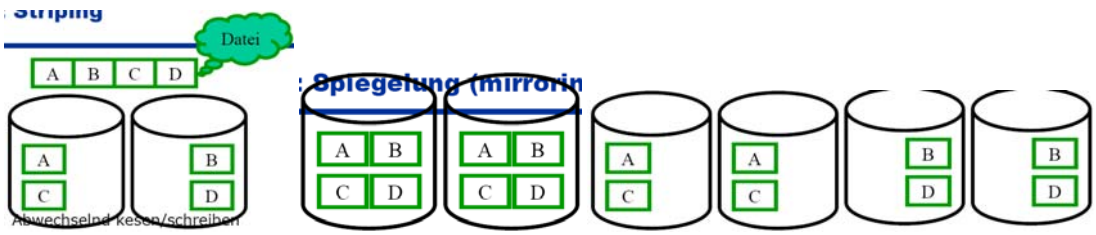
Aufgabe 4
Ergänzen Sie die for-Anweisung, sodass diese über alle Kolonnen des Resultsets iteriert. Geben Sie jeweils die Kolonnen-Ueberschrift aus. Deklarieren Sie allfällige notwendigen Variablen.

```
ResultSetMetaData mrs = rs.getMetaData(); --1P
int nCols = mrs.getColumnCount();
for (int i=1; i<=nCols; i++--1P ) {
      System.out.print (/*Label der Kolonne i*/ mrs.getColumnLabel(i)—1P + "\t\t");
}
System.out.println();
```

9 Interne Ebene

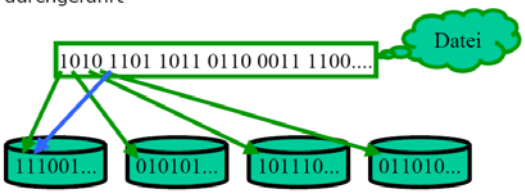
9.1 RAID

RAID 0 RAID 1 RAID 0+1



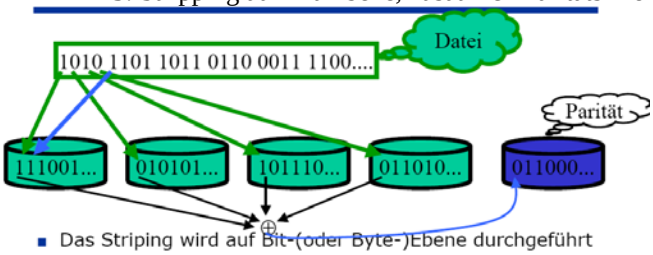
RAID 2: Stripping auf Bitebene:

- Anstatt ganzer Blöcke, wie bei RAID 0 und RAID 0+1, wird das Striping auf Bit- (oder Byte-) Ebene durchgeführt



- Es werden zusätzlich auf einer Platte noch Fehlererkennungs- und Korrekturcodes gespeichert
- In der Praxis **nicht** eingesetzt, da Platten sowieso schon Fehlererkennungscodes verwalten

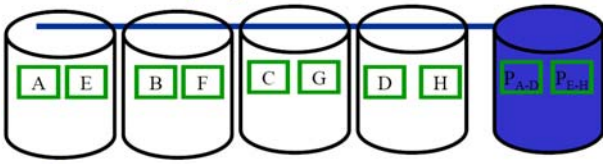
RAID 3: Stripping auf Bit-Ebene, zusätzlich Paritätsinfo



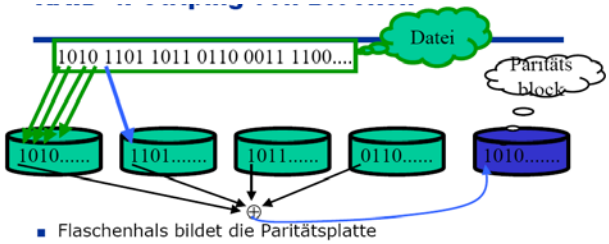
- Das Striping wird auf Bit-(oder Byte-)Ebene durchgeführt
- Es wird auf einer Platte noch die Parität der anderen Platten gespeichert. Parität = bit-weise xor ⊕
- Dadurch ist der Ausfall einer Platte zu kompensieren
- Das Lesen eines Blocks erfordert Zugriff auf alle Platten
  - Verschwendung von Schreib/Leseköpfen
  - Alle marschieren synchron

H.Huser + S.F.Keller, 23.05.1998 Organisatorisches Datenbanksysteme 1 - 17

RAID 4: Striping von Blöcken

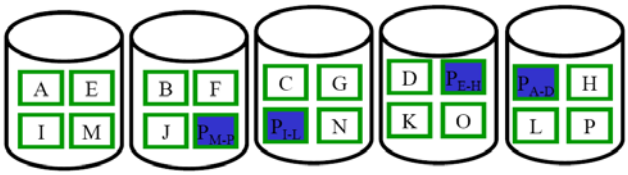


- Bessere Lastbalancierung als bei RAID 3
- Flaschenhals bildet die Paritätsplatte
- Bei jedem Schreiben muss darauf zugegriffen werden
  - [Bei Modifikation von Block A zu A' wird die Parität P_{A-D} wie folgt neu berechnet: $P'_{A-D} := P_{A-D} \oplus A \oplus A'$]
- D.h. bei einer Änderung von Block A muss der alte Zustand von A und der alte Paritätsblock gelesen werden und der neue Paritätsblock und der neue Block A' geschrieben werden



- Flaschenhals bildet die Paritätsplatte
- Bei jedem Schreiben muss darauf zugegriffen werden
  - [Bei Modifikation von Block A zu A' wird die Parität P_{A-D} wie folgt neu berechnet: $P'_{A-D} := P_{A-D} \oplus A \oplus A'$]
- D.h. bei einer Änderung von Block A muss der alte Zustand von A und der alte Paritätsblock gelesen werden und der neue Paritätsblock und der neue Block A' geschrieben werden

RAID 5: Striping von Blöcken, Verteilung der Paritätsblöcke



- Bessere Lastbalancierung als bei RAID 4
- die Paritätsplatte bildet jetzt keinen Flaschenhals mehr
- Wird in der Praxis häufig eingesetzt
- Guter Ausgleich zwischen Platzbedarf und Leistungsfähigkeit

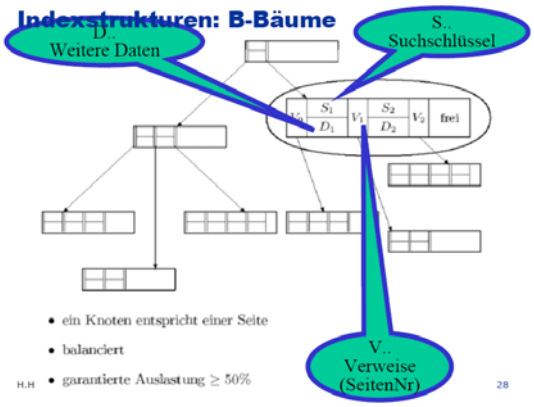
9.2 B-Bäume

Begriff:
- Balancierte Mehrwege-Suchbäume
- Nicht Binär-Bäume (deshalb gut für Hauptspeicher)

Eigenschaften:
- Geeignet für Hintergrundspeicher

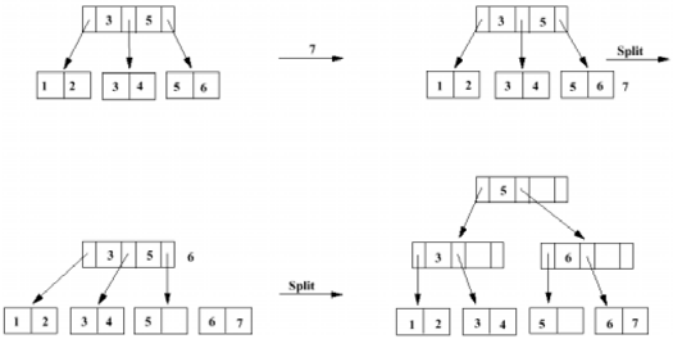
Idee:
- Abbilden von Baum-Knoten auf Seiten im Hintergrundspeicher

9.2.1 Indexstrukturen



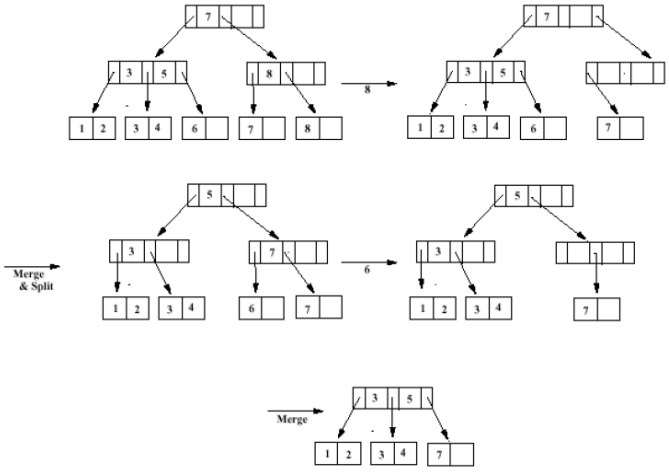
Ein B-Baum der Ordnung (m,l) m>3, l+1 ist ein Vielwegden vom Grad m (oder k) mit folgenden Eigenschaften:
(1) Die Wurzel ist entweder ein Blatt oder hat mindestens zwei Söhne.
(2) Jeder innere Knoten - ausser der Wurzel - hat mindestens [m/2] und höchstens m Söhne. Die garantierte Mindestspeicherplatzauslastung beträgt somit 50%.
(3) Alle Blätter befinden sich auf dem gleichen Level. Sei N die Anzahl Blätter. Für die Höhe h des Baumes gilt dann höchstens $h=1+[\log_{(m/2)}(N/2)]$ und mind. $h= [\log_m N]$
(4) Einfügen und Löschen kann in Zeit proportional zur Höhe des Baumes durchgeführt werden.

9.2.2 Einfügen eines neuen Objekt



1. Führe eine Suche nach dem Schlüssel durch; diese endet (scheitert) an der Einfügestelle.
2. Füge den Schlüssel dort ein.
3. Ist der Knoten überfüllt, teile ihn
   - Lege einen neuen Knoten an und belege ihn mit den Schlüsseln, die rechts vom mittleren Eintrag des überfüllten Knotens liegen.
   - Füge den mittleren Eintrag im Vaterknoten des überfüllten Knotens ein.
   - Verbinde den Verweis rechts des neuen Eintrags im Vaterknoten mit dem neuen Knoten

9.2.3 Löschen eines Objekts



Beispiel: