18.4.18

(I)

PuD II - 3l.

- Allgemein: - Fragestellungen:
  - ▷ Kostenträgerinformationen (Production)
  - ▷ Mitarbeiterzahlen (Human Resources)
  - ▷ Kostentreiber (Financial Controlling)
  - ▷ Handelsbeziehungen & Warenströme (Logistics)

↳ DB im privaten Umfeld (Wikipedia, Amazon, Maps ...)
" geschäftlichen " (ERP-SAP R3, Schiff, Börse, Projekt DB)

⇒ Möglichkeiten der Datenhaltung (Kopf, Papierbasiert, Office-Dokumente) alle unzweckmäßig, daher DATENBANK

ARIS
- Datensicht
- DV-Entwicklung

DD-Rahmen
- Daten & Lebenszyklus
- Tools & Methoden

IDEE: Verschiedene Nutzer auf versch. Systemen weltweit auf gleichen Datenbestand
↳ Anforderungen:
- ▷ Performance
- ▷ Mehrbenutzerbetrieb & zugänglich
- ▷ Programmierbare Schnittstellen
- ▷ Einheitliche Logik
- ▷ Sicher bei Fehlern
- ▷ Einfach erweiter- & skalierbar

- Datenbank(systeme):

» gezielte Menge von logisch zusammengehörigen Daten (Datenbasis), die von einem Datenbankmanagementsystem (DBMS) elektronisch verwaltet werden. «

Ziele:
- Standardisierung d. Datenverwaltung
- Umgang mit großen Datenmengen
- Komplexitätsreduktion d. AWS & Portierbarkeit AWS

„Permanenz":
- ▷ [lt.: „stehen bleiben; verharren" // engl.: „andauerndes Gedächtnis"]
- Speicherung persistent, wenn unabhängig von Laufzeit des Programms/Rechners

„Konsistenz":
- ▷ „Stimmigkeit; Geschlossenheit"
- ▷ Redundanzfrei; Vermeidung Einfüge-/Änderungs-/Löschanomalien

Anforderungen:
- Kurze Zugriffszeiten
- Minimale Redundanz
- Trennung Datenorganisation & Anwendung
- Permanenz & Konsistenz d. Daten

Voraussetzungen:
- Techniken & Strukturdefinition zur Datenmanipulation
- Schutz vor unbefugten Änderungen
- Orga Mehrbenutzerbetrieb
- Realisierung Nutzerschnittstellen
- Anpassung d. Leistungsverhaltens

- Historie:

60's
- ▷ Basierend auf Hierarchie oder NW-Modell
- ▷ Trennung navigierende DML & Programmiersprache
- ▷ Zeigerstrukturen zwischen Daten
- ▷ Schwache Trennung interne/konzeptionelle Ebene

Relationales DB-Modell

70/80's
- ▷ IBM System R (erste relationale DB)
- ▷ Daten in Tabellenstrukturen
- ▷ 3-Ebenen-Konzept
- ▷ Deklarative DML getrennt von Programmiersprache

80/90's
- ▷ Mehrere Strukturen & größere Datenmengen
- ▷ Objektorientierte DB

TODAY
- ▷ Spezialisierung neue Datentypen (Video/Masse)
- ▷ Notwendig große, unstrukturierte Datenmengen
- ▷ Loslösung von festen Schemata (NoSql)

(II)

◦ Verteilte Systeme / Client Server Architektur:

▸ Verteilte Systeme sollen Ortstransparenz, konsistent, stimmig (Bsp. Uhrzeit)

↳ Realisierung mittels 2-/3-stufiger Client-Server-Architektur
(Benutzer Schnittstelle // Anwendungsfunktion / Datenverwaltung)
(Präsentationsschicht // Logikkomponente // Datenkomponente)

- Rechner stellen als SERVER Dienste zur Verfügung
- Andere nutzen jene als CLIENT

→ AWS (Client)

3-SCHICHTEN-MODELL
→ Appl.-Server
→ DB-System / Dateisystem } DB-Server

~> Architekturvarianten: „distributed logic": Präsentation / Verarbeitung // Verarbeitung / Verwaltung

---

◦ Ansi-Sparc // 3-Ebenen-Architektur:

▸ 1975 verabschiedete DB-Architektur Standard
▸ American National Standards Institute / Standards Planning And Requirements Committee

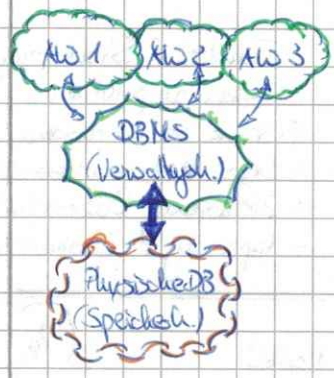
↳ Ziel: DB-Konzept: ◦ Unabhängigkeit von Programmiersprache, Datenschema, Hardware [Datenhaltung, Strukturierung, Nutzung]

PHYSISCH & LOGISCH DATENUNABHÄNGIGKEIT

(I) Externe Ebene:
▸ Benutzersicht auf Daten: UI & Schnittstellen
▸ individuell an Benutzer angepasst
▸ SQL, Views

(II) Konzeptionelle Ebene:
▸ Logische Gesamtsicht d. Daten & Datenstruktur
▸ Datenmodell spezifiziert Daten in DB
▸ Unterscheidung: - konzeptuelles Schema (ERM) unabhängig vom Implementierungsmodell
- logisches DB-Schema (relational) ← Modellierung ERM, SERM

(III) Interne Ebene:
▸ Physische Sicht auf Daten, Speicherung
▸ Aufbau Datenstrukturen auf phys. Speicher // Zugriffsmechanismen
▸ Indices Transaktionen



Bestandteile eines relat. DBMS

AWS: ◦ greifen über DBMS auf Daten zu

DBMS: ◦ relationale Sprache, arbeitet auf DB Basis
◦ Zentrale Funktion für Schutz, Sicherheit, Mehrbenutzerbetrieb, Geschwindigkeit & Reorganisation

DB-Basis: ◦ Daten- & Beziehungsinformationen in Tabellenform
◦ Beschreibungs-(Meta-)Daten & Systemtabellen

◦ Modellierung in der DB-Entwicklung:

◦ Idee → Konzeptuelles Schema → Logisches Schema
◦ Analog Abstraktions-Transformations-Graph

◦ 1.) Fachliche Analyse (Beurt. Datenfelder; Relationen) & Entwurf
2.) DV-technischer Entwurf (Überführung in relat. Datenmodell & Optimierung)
3.) Codierung (Datendefinition [Benutzersichten, Indexierungsoptionen], Datenmanipulation [Arbeiten mit DB])

21.4.18

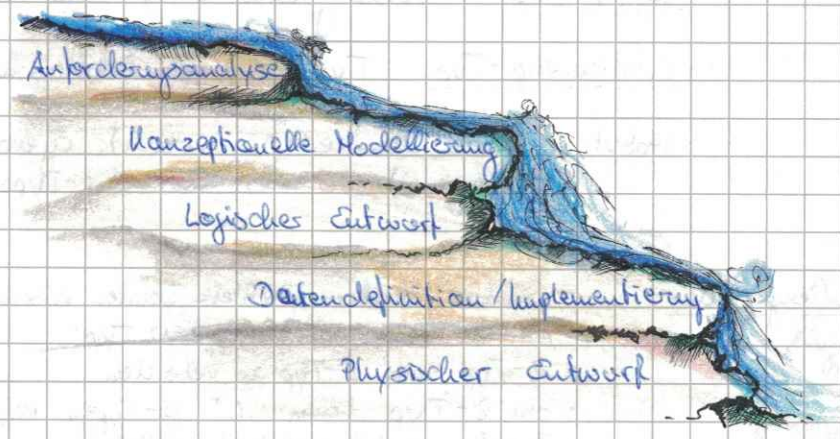
III

FuDB II 3sf.

- Datenbankentwurf: „DATENHALTUNG für MEHRERE AWS über MEHRERE Jahre."

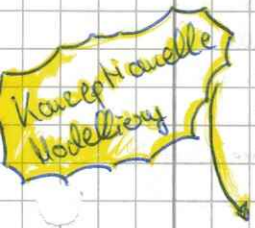
- Anforderungen:
  - Anwendungsdaten aus Daten der DB effizient ableitbar
  - Nur Speicherung benötigter Daten
  - Keine Redundanz
  - auf mit. Ebene (Aufbau plus Speicherstruktur & Zugriffsmechanismen)

PROZESS



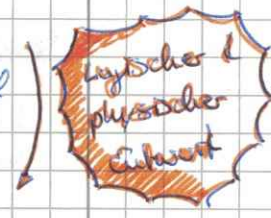
- Domäne: „Ein in sich schlüssiger, abgegrenzter Themenbereich."

↳ Ableitung aus: • Den Datentypen
⊕ Eigenschaften
⊕ Beziehungen



- Datenbankdesign mit ER-/SER-Skizze (Entity-Relationship-Diagramme)
- Daraus ER-Diagramm in relationales Modell überführen & Relationsschemata in SQL-Anweisungen

- DV-technischer Entwurf: Überführung in relat. Datenmodell & Optimierung tgt. Zielsystem
- Codierung (Datendefinition, & -manipulation)



- Datenmodellierung:

- Modellierungsansätze:
  - Funktionale Zerlegung
  - Datenflussansatz
  - Datenmodellierung
  - Objektorientierter Ansatz
  - Geschäftsprozessorientierter Ansatz

  } • Hierarchisierung v. Daten
  • Relative Unterteilung in Mengen

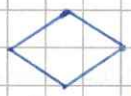
IV

• Entity-Relationship-Modell:

▸ Begriffe:

- Entity: Gegenständiges (konkretes) Objekt oder Vorstellungswelt
- Relationship: Beziehung/Verknüpfung zwischen zwei (gleichen) oder mehreren Entities
- Entity-Set: Menge gleichartiger (gleiche Attribute) Entities
- Relationship-Set: Menge gleichartiger Beziehungen zw. Entity(-Sets)
- Entity-Typ: Typisierung eines Entity-Sets (Abstraktion von Real/Vorstellungswe)
- Relationship-Typ: Typisierung von Relationship Sets
- Attribut: Besitzt Domäne (Wertebereich); Eigenschaft eines Entity-Typs (zwingend) oder eines Relationship-Typs (optional)


Entity-Typ


Relationship-Typ


Attribut

▸ Modellierungsregeln:

- Symbole durch ungerichtete Kanten verbinden
- Attribute haben GENAU EINEN Typ
- R-Typ mit E-Typ zu verbinden
- Ein R-Typ kann 2,* E-Typen verbinden
- Beziehung mit sich selbst ist möglich
- Komplexitäten analog Multiplizität

⊖ Unübersichtlich
⊖ keine Visual. von Ex-Abhängigkeit
⊖ Redundante Relationen
Gefahr der Unkonsistenz

▸ „schwacher E-Typ"

- nicht allein durch Attribute zu bestimmen
- besitzt Primärschlüsselattribute eines an den Beziehung bet. E-Typs
- 1:N nicht ausreichend R-Typ & Pfeil verwendet zu schlecht mit Doppellinien (□-◇-□)

▸ Schlüsselattribut:

- Spezielles Attribut zur Identifikation des Datensatzes
- Eindeutig & minimal, künstliche Schlüssel ohne Semantik
- Nicht-atomarer Schlüssel („Me-Kennzeichen, Soz-Vers-Nr)
- „ zusammengesetzter „ („Name-Vorname, ...)
- Künstlicher „ (Fortlaufende Nummer/Timestamps)

• Structured Entity Relationship Model:

- Motivation, Schwachstellen d. ERM zu eliminieren (Ferstl/Sinz 87/88)
- Analyse d. Datenstruktur; Beachtet Gegenstände & deren Abhängigkeiten

=> Strukturiert große Datenmengen (Unübersichtlichkeit ERM); Abhängigkeit +
=> „Quasi-Hierarchische" Graph („keine Querbeziehungen & eigene Abhängigkeiten §)

§ Inkonsistenzen (Zyklen/Quer-/Schrägbeziehungen)
§ Vermeiden unnötiger Relationstypen (ger. Schlüsselbereich schon auf konz. Ebene)

Elemente:

- E-Typ
- ex.abh. ER-Typ
- R-Typ
- Attribut

Kanten:

- Min-Wert=0
- Min-Wert=1
- Max-Wert=1
- Max-Wert=*

Kann so interpretiert

- Jede Kante ist gerichtet
- Immer Rechteck → Raute
- Immer Links nach Rechts
- Originäre Typen links Ex. Abh. rechts

26.4.'15

Zusammenf. PuDII

• SQL als relationale Sprache:

▷ Anforderungen:

1.) AD-HOC-FORMULIERUNGEN: - Anfrage ohne vollständiges Schreiben eines Programmes möglich

2.) DEKLARATIVITÄT: - Beschreibende Sprache
- „Was will ich haben?" (X Wie komme ich dahin)

3.) MENGENORIENTIERTHEIT: - Operationen auf in Tabellen organisierten Mengen
- vektorwertig (nur Ergebnismenge) „tuple-at-a-time"

4.) ABGESCHLOSSENHEIT: - Ergebnis ist Relation & kann für nächste Anfrage verw. werden

▷ Historie:
- Mitte '70er für System R (erste rel. DB) SEQUEL
- Standardisierung d. ANSI/ISO 1986 als Structured Query Language
- Erweiterungen (z.B. XML), allerdings immer noch Inkompatibilitäten
- (MySQL/ Oracle Database/ Infotmix/ ...)
- SQL = ad hoc / deklarativ / volles Datenbankensystem & Abstraktion

▷ Strukturen definieren:

Schema anlegen	CREATE SCHEMA <Schemaname>
Schema löschen	DROP SCHEMA <Schemaname>
Tabelle anlegen	CREATE TABLE <Tabellenname> (<Attribut/Name> <Datentyp> [Integr. Bed.]) [Integ. Bed.]
Tabelle löschen	DROP TABLE <Tabellenname>

Datentypen: INT / DEC / FLOAT / REAL / CHAR(n) / VARCHAR(n) / BOOLEAN / DATE / TIME / TIMESTAMP

Integritätsbed.:
- UNIQUE : Attribute ≠ endeutig
- NOT NULL : Attribut darf keine leere Zelle haben
- DEFAULT <Wert> : Standard-Wert, wenn nichts angegeben
- PRIMARY KEY : Attribut (Teil) Primärschlüssels
- FOREIGN KEY REFERENCES <Tabellenname> (<Attributname>)
- ON DELETE / UPDATE : Cascade // set Null // Set Default

▷ Primärschlüssel / Fremdschlüssel

▷ Datenabfrage: (SELECT ...)
- Projektion: Auswahl bestimmter Attribute einer Relation (Spalte)
- Selektion: " " Datensätze " " (Zeile)
- Verbund (Join): Verknüpfung zweier Relationen zu einer (gleiche Attribute)

WHERE XY = X²

▷ Struktur ergänzen:
- Attribut hinzufügen: ALTER TABLE <TN> ADD COLUMN <AN> <DT>
- " löschen: ALTER TABLE <TN> DROP COLUMN <AN> CASCADE

löscht auch Attribute, welche auf das zu löschende referenzieren.

(VI)

• Reihenfolge bei Datenabfragen:

SELECT (...) FROM (...) WHERE (...) GROUP BY (...) HAVING (...) ORDER BY

... oder ...

• Datenmanipulation:
- Pro Anweisung nur Datensätze einer Relation manipulierbar
- Ohne Angaben, alle Attribute angegeben (default...)
- Reihenfolge VALUES ≙ Reihenfolge Attribute

▷ Einfügen: INSERT INTO <TN> VALUES (<Wert1>;...)

▷ Ändern: UPDATE <TN> SET Attribut = Wert WHERE (...) (Eine Relation; mehrere Tupel)

▷ Löschen: DELETE FROM <TN> WHERE (...) (Eine Relation, Löschen von Datensätze, ohne Sel. alles)

Validity, Fremd-Schlüsselbeziehg

• Struktur SQL:
- (I) Data Definition Language - DDL
- (II) Data Manipulation Language - DML
- (III) Data Control Language - DCL
- (IV) Transaction Control Language - TCL

• Relationenmodell:
- ▷ Relation = Tabelle
- ▷ Attribut = Merkmal = Spalte
- ▷ Tupel = Datensatz = Zeile

} Attributwert = Datum = Zelle

Formale Definition:

$R(A_1, A_2, \dots, A_n)$	...	Relationstyp
A	...	Attribut
X, Y	...	$\subseteq A$
$X \rightarrow Y$	...	Y funktional abhängig von X
$X \twoheadrightarrow Y$	...	Y voll " " " "

wenn keine Untermenge von X existiert, von der Y abhängig ist

identifizierte Attribut = Schlüssel: unterstreichen

• Schlüsselattribute:
- ▷ Für eindeutige Identifikation von Datensätzen ↝ best. Attributwerte
- ▷ Oft einzelnes Attribut nicht ausreichend ↝ nicht eindeutig

↳ ⊕ künstliches, eindeutiges Attribut ID
ODER Kombination von Elementen

Definition: „Teilmenge S des Relationsschemas R heißt Schlüssel wenn gilt":

1 Eindeutigkeit: Keine Ausprägung von R kann zwei versch. Tupel enthalten, die sich in allen Attributen von S gleichen

0 | 1 Minimalität: Keine echte Teilmenge von S erfüllt bereits die Bedingung der Eindeutigkeit

keine überflüssigen Attribute

↝ Es kann verschiedene Schlüsselkandidaten geben!

VII

Zusammenfassung – RuD II

• ER-Modell zum Relationenmodell:

Entity-Typ
- Zu jedem ET eine Relation
- Attribute des ET wird Attribut d. Relation
- Primärschlüssel des ET wird PK d. Relation

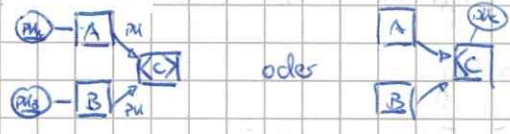
} weitere Attribute können hinzugefügt werden

Relationship-Typ
- 1:1 & 1:N → zusätzliche Fremdschlüssel-Attribute in best. Relationen
- M:N → Erzeugung zusätzlicher Relation (R-Typ wird eig. Relation)
- immer möglich zusätzliche Relationen einzuführen

} abhängig von Kardinalität

Vererbung
- Vererbung entlang SEEM-Leserichtung
- PK wird entweder PK & FK in modif. DOT (PK-Vererbung) oder nur FK
- Vererbung kontextabhängig (bei M:N; FK & PK gleichzeitig)

Ziel = R-Typ → PK
= ER-Typ → FK



optional

C(PK_A, PK_B, ...) C(PK_A, PK_B : {PK_C}) // mit FK: C(PK_C : FK_A : FK_B)

• Normalisierung:

↯ ER-Modell nicht immer redundanzfrei (Anomalie-Gefahr)
↳ Insert-/Delete-/Update-Anomalien

⇒ Vermeidung Redundanzen/Anomalien ⇒ komplex; ungeformt ⊖

↪ LÖSUNG: Schrittweises, normalisiertes Zerlegen in Äquivalent

„NORMALISIEREN"

~ „Funktionale Abhängigkeit – FD": • [A → B] B funktional abhängig, wenn ∀ ∈ R gilt zu jedem Wert A genau 1 Wert in B
- Wenn ich von A, dann auch von jeder Obermenge
- A' ⊆ A ∈ | Abhängigkeit ⇒ voll // A' ⊆ A ∉ partiell abhängig

1.) Normalform: • Alle Attribute elementar/atomar

2.) Normalform: • Vermeidung von Attributen, die NICHT voll funkt. vom Schlüssel abhängen
⇒ Informationen redundant

• Alle Nichtschlüsselattribute voll funktional abhängig

⇒ (1.) Attribute die voll abhängig sind verbleiben in Relation
(2.) Löschen von partiell abhängigen ⇒ eigene Tabelle mit dem Teil von A

3.) Normalform: • Keine transitiven Abhängigkeiten/Abhängigkeiten von Nichtschlüsselattributen

⇒ Analog 2.) NF, Hoß das transitiver Vorfahre Schlüssel der neuen Relation wird.

Durchdachtes ER liefert normalisierte Tabellen; Allerd. auch Universal Relation Assumption
(1 Tabelle → Normalisierung)

VIII

• Allgemeines DB-Struktur:

- Mensch möchte lieber mit graphischer Oberfläche kommunizieren als mit DB
  ↳ Oberfläche muss mit DB kommunizieren

⇒ Programmierung in Java per 1.) Verbindungserstellung
2.) Schreiben von Daten
3.) Lesen von Daten
} mittels Java DataBase Conn. & Hibernate als Objekt-Relationales-Mapping

---

• Java & SQL:

- Verwenden d. Paketes java.sql.*
  ↳ Objekte: Verbindung → Connection (Interface!)
  SQL-Statement → Statement "
  Abfrageergebnis → ResultSet "

▸ Herstellt Verbindung zw. gew. Programm & DB (auch mehrere DB)

▸ Verbindung über Treiber-Verwaltung:

jdbc : [Treiber-Name] : [Datenbank-Ort]

• Kommunikation Java & DB:

- Verbindung existiert: Kommunikation: `Statement statement = connection.createStatement();`

▸ execute ≙ SQL-Statement (Insert, Update, Delete)
execute Query ≙ SQL-Abfragen (Select)

▸ Abfrage mittels Methoden die Attributwerte auslesen & Zeiger bewegen

[next; previous; first, last] → alle boolean

[getString; getInt; getDouble...]

`ABFRAGE: ResultSet ergebnisMenge = statement.executeQuery ("Select-Statem.");`

Alle Java-SQL-Objekte [Connection, Statement, ResultSet] sollten geschlossen werden! [xyz.close();]

• Exceptions:

- Verbindungsabbruch; DB ∉; SQL-Statement nicht interpretierbar; etc.
- Lösung: Methode meldet das etwas Unvorhergesehenes existiert

⇒ Erzeugt Exception-Object

throw/throws

Auffangen durch try catch-Block

try analysiert Fehlertyp; dann unter catch Fehlertypen; finally ... garantierte Durchführung

21. PuD II

• Generelles Ablauf:

(1) JDBC Treiber laden
(2) URL der DB-Verbindung def.
(3) Verbindung zur DB herstellen
(4) Anweisung als Statement Objekt erzeugen
(5) SQL-Statement ausführen
(6) Abfrageergebnisse verwalten
(7) Schließen der DB-Verbindung

Connection = Verbindung zu DB
Statement = Abfrage als SQL-String
ResultSet = Zeiger auf Zeile der Ergebnismenge
Exception = Ausnahmebehandlung

• Objekt-Relationales-Mapping:

- Selbst Effizienz-, Wartbarkeits- & Wiederverwendbarkeits-Probleme
  ↳ SQL & DB-Anbindung IN JAVA // SQL-DB-Anbindung in decl. KLASSE
  ⊕ Bekannte Probleme
  ⊕ Konsistente Speicherung d. Laufzeitdaten
  ⊕ Vererben im O.O.P.

» Abbildung von Laufzeitdaten eines in objektorientierter Sprache geschriebenen Programms auf zugehörige Daten in einer relationalen DB & umgekehrt «

Klasse = Tabelle
Attribut = Spalte
Objekt = Datensatz
Assoziation = Schlüsselattribute

→ Speichern von Objekten als relationale Datensätze
→ Bearbeiten, Abfragen etc. möglich ohne manuellen SQL-Code
→ ORM generiert SQL-Code & sichert Konsistenz

- Hibernate als Open Source Programm (Framework) für Java
- Mittels XML-Konfiguration // & Java Annotations

@ XY (...) (Sprachelemente die das Einbinden von Metadaten (METADATEN) im Code erlauben)

Discriminator Column = Unterscheidung
Join Column = Fremdschlüssel

» Klassenhierarchie Vererbung:

A) One Table per Class Hierarchy (Single Table)
- Alles in einer Tabelle; Attribut ∉ in Klasse = NULL; Spalte mit Klassenname
B) One Table per Subclass (Joined)
- Jede Klasse ein. Tabelle mit Ref-Attributen
C) One Table per Concrete Class (Table-per-Class)
- Ein. Tabelle für je konkr. Klasse; Attribute der abstrakten eingebettet
- Abstrakte x Tabelle

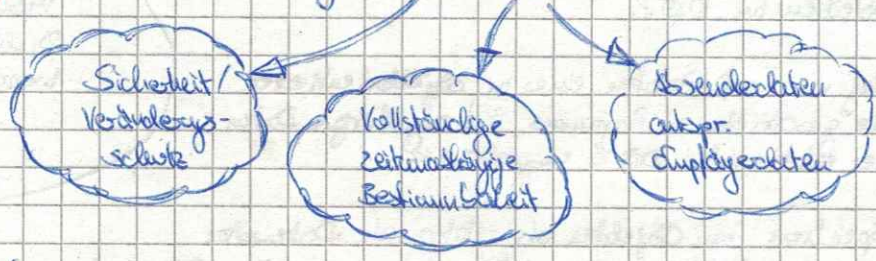
» Assoziationen:
1.) @OneToOne — ∈ aus A genau eins aus B zugeordnet
2.) @OneToMany — Parent-Seitig
@ManyToOne — Child-Seitig
3.) @ManyToMany

Probleme:
↯ Umwandlung von Datentypen & nullwertige Attribute
↯ Verbindungsaufbau & Vererbungsbeziehungen
↯ Methoden persistieren will

- Normalformen: 1. NF: nur atomare Attribute
  2. NF: keine teil-funktionalen Abhängigkeiten von PK
  3. NF: keine transitiven Abhängigkeiten von NK

- Transaktionskonzept & konkurrierender Zugriff:
  - Einordnen in die 3. (internen) Ebene des Ansi-Sparc-Modells, als Teilaufgabenbereich des DBMS
  - Transaktion:
    - „Folge von OPERATIONEN die einen KONSISTENTEN Zustand einer Datenbank in einen zweiten ebenfalls konsistenten, ggf. veränderten Zustand überführen. → ACID"
    - Folge von Lese- & Schreiboperationen
    - Der Speicherung vollständiger Transaktionen
    - Auch Logical Unit of Work (LUW) // Unit of Recovery (UOR)

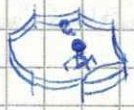
→ Ziel: Gewährleistung der INTEGRITÄT von Daten im MEHRBENUTZERBETRIEB



→ ACID-Prinzip: „garantiert konsistente Datenbasis"

Atomicity:
- Alles oder Nichts - Prinzip
- Keine Zwischenergebnisse // Sperren unvollst. Transaktionen
- Bei Fehler Rollback zu letztem Haltezustand

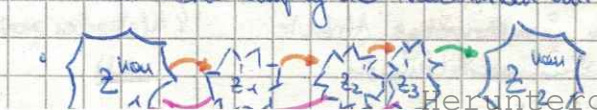
Consistency:
- Transaktion als konsistenzerhaltende Einheit von $Z^{konsistent}_1 \rightarrow Z^{konsistent}_2$
- Innerhalb Transaktion temporäre Inkonsistenzen möglich
- Während Abarbeitung Zustände für andere Nutzer gesperrt



Isolation:
- Parallele Aktionen führen zum gleichen Ergebnis wie Serielle
- Sperren sichern, dass nicht parallel die gleichen Daten geändert werden
- Transaktion = Einheit d. Serialisierbarkeit

Durability:
- Erfolgreiche Transaktionen sind persistent gespeichert & lösen Sperren
- Zustände gültig bis zu neuer Aktivität
- Fehler können abgeschlossene T. durch unvollst. T. nicht beschädigen

- Begin-of-Transaction (BOT) & End-of-Transaction (EOT) aufgegen Transaktion
- Teilen DBMS Ablauf der Transaktion mit



29.6.'18

(XI)

Zsf. PuD II

· Konkurrierender Zugriff:

- Serieller Ablauf:
- Alle Operationen der 1. T. müssen beendet sein, bevor die 2. beginnen kann.
- Gewährleistung der Konsistenz
- ⊖ Performance
- Bei Mehrbenutzerbetrieb ⇒ Serialisierung → Ausschluss konk. Zugriffe; → Konsolidierungssicherung

- Paralleler Ablauf:
- Ausführung versch. Aktionen, während 1. T. noch nicht abgeschlossen ist.

Dirty Read	Non-Repeatable-Read	Phantom Read	Lost Update
T2 liest von T1 ein Zwischenergebnis bevor T1 abgeschlossen ist.	T erhält für die gleiche Abfrage untersch. Daten, z.B. bei paralleler Änderung	T erhält für die gleiche Operation unterschiedlich große Ergebnismengen	Operationen von ≥2 T überschneiden sich & es ergeben sich falsche Werte

Σ
- Read-Operationen nur für AwS kritisch
- Write-Operationen für AwS & DB kritisch

↳ S: Share-Lock: Niemand Schreiben / Jeder Lesen
X: Exclusive-Lock: Nur Einer Lesen & Schreiben; Sonst kein Zugriff; nur ohne S-Lock

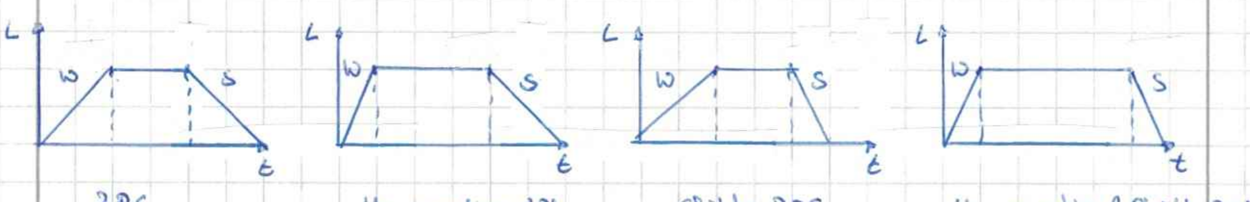
ISOLATIONSSCHICHTEN

	Dirty Read	Non-Repeatable-Read	Phantom Read	Lost Update
Read Uncommitted: Alle Daten lesbar, hohe Geschwindigkeit; keine Sperren, keine T-Isolation	✓	✓	✓	✓
Read Committed: Lesen nur nach commit(); X-Locks	✗	✓	✓	✓
Repeatable Read: Mehrfache Zugriffe ≙ gleichen Werten; T nur Daten vor Ausführung; S&X-Locks	✗	✗	✗	✓
Serializable: Sperren für ges. DB; Anfrage wie seriell, sicher, langsam	✗	✗	✗	✗

PARALLELISIERUNG VS. INTEGRITÄT

- Sperrverfahren:
- ~ Sperren sichern Objekt exklusive Rechte
- ~ Deadlocks verhindern!

⇒ Zwei-Phasen-Sperrprotokoll:
- LOCK-Aufbauphase ⇒ UNLOCK-Abbauphase
- Sperren nicht wechselseitig auf/abbauen
- Sperren zu Anfang ⊖ Parallelisierungsgrad



2PS — Konservative 2PS — Strikte 2PS — Konservativ & Strikt 2PS

ZWECK DER DATENMODELLIERUNG

- A) Komplexitätsreduktion
- B) Abstraktion d. Realwelt als Implementierungsbasis
- C) Schaffung eines Kommunikations- & Architekturmodells zur Verständigung verschiedener Gruppen

Logical Unit of Work...

Unit of Recovery...

da eine Transaktion eine wiederherstellbare Sequenz von ausgeführten Aktionen darstellt.

{ANSI SPARC}

A) EXTERNE

Für Benutzer & Anwendungen individuelle Sichten

B) KONZEPTIONELLE

Beschreibung des in der Datenbank gespeicherten Daten, sowie Beziehungen + Normalisierung

C) INTERNE

Physische Sicht darauf, wo die Daten gespeichert werden. Ziel ist die effiziente Zugriff

⊕ Physische Datenunabhängigkeit: Trennung der Ebenen & Entkopplung bei Änderungen

⊕ Logische Datenunabhängigkeit: Änderungen des Speicherstruktur ≠ Änderung extern (wenn)

Modellierung:

1.) Anforderungsanalyse: (Eigenschaften / Bez. / Dateit)

2.) DB-Design-Skizze: ERM / SERM

↳ Übführung in RELATIONALE MODELL (Relationen / Attribute / Integritätsbed.)

↳ Aus Relationsschema SQL-Ausdrücke

3.) DV-Entwurf: Übführung in das RELATIONALE DATENMODELL

⊕ Optimierung

4.) Codierung:
- Datendefinition
- Datenmanipulation

Bestandteile des SQL

(I) Datendefinition: Anlegen, Ändern, Löschen von STRUKTUREN

(II) Datenmanipulation: Anleg., Änd., Löschen von DATEN

(III) Datenabfrage: Abfragen von Daten

Nachteile anderer DB-Alternativen:

- Kopf: ⊖ Speicher / ⊖ Verfügbarkeit
- Papier: ⊖ Struktur / ⊖ Verfügbarkeit
- Karteikarte: ⊖ Mehrbenutzer / ⊖ Zugriffsrechte
- Office: ⊖ Verteilung / ⊖ Verknüpfung

↳ DATENBANK:

Anforderungen?

DEADLOCKS SQL