DB Summary Fortlaufend

1. Oktober, 2022; rev. 7. Januar 2023

Linda Riesen (rieselin)

Inhaltsverzeichnis

Introduction

0.1 Was sind Daten?

1. Zeichen kommmen aus Zeichenvorrat (Alphabete, Zahlen, Satzzeichen, Piktogramme, etc) der beliebig angeordnet sein kann.
2. Daten sind Strukturierte Zeichen, die sich auf einem Datenträger befinden (bsp. Papier)
3. Daten im Kontext = Information
4. Intormation und verknüpfung mit Intellektueller Einbettung = Wissen
5. Nachher gits na Weisheit aaaber... eher unmöglich da... für normalsterbliche

Datenmenge verdoppelt sich alle 2-3 Jahre.

Strukturierte Daten	Unstrukturierte Daten	Semi-Strukturierte Daten
Fest vorgegebene Struktur	keine *explizite* Struktur	irregulläre/unvollständige Struktur
Mehrere gleichartige Datensätze	Keine bestimmte Anordung	
gut Tabellarisch darstellbar		
Aktienmarkt/ Rationale DB	Bilder/ Filme/ Texte/ etc.	Email/ Suchresultate/ JSON

1 Relationales Datenmodell: Vorlesung 1

1.1 Dateisysteme vs Datenbanken

Persistenz vs Volatilität der Daten:
Daten sollten länger vorhanden sein. ⇒ Nutzung von Nichtflüchtigen Speichermedien

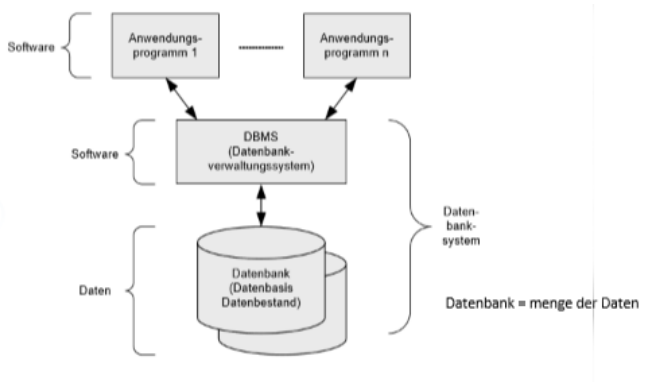

Abbildung 1: A Database System

Relationale Datensysteme:
So entkoppelt wie möglich auf 3 Ebenen um Unabhängigkeit bei anfallenden nötigen Änderungen zu bringen. (Externe, Konzeptionelle und Physische Ebene)

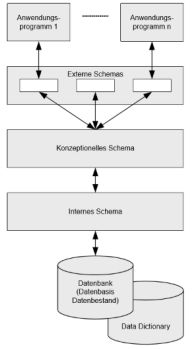

Abbildung 2: 3 Ebenen System der Rationalen MDBS (Database Management System)

Dateisystem	Relationale Datenbanksysteme
Vorteile	
• Einfach Angepasst , effizient • nicht auf andere Rücksicht nehmen • eigene Formate möglich • weit verbreitet	• Hohe Datenunabhängigkeit • Hohe Leistung und Skalierbarkeit • Datenmodelle und Abfragesprachen ⇒ leichte Handhabbarkeit • Transaktionskonzept[1], Datenkontrolle • Ständige Betriebsbereitschaft • Mehrere Anwendungen gleichzeitig • Automatisierung von Integritätskontolle, Redundanzverwaltung, Zugriffskontrolle, Datensicherung / Wiederherstellung

[1] https://dbs.uni-leipzig.de/buecher/mrdbs/mrdbs-12.html = ACID Chunnt wohl später na

Dateisystem	Relationale Datenbanksysteme
Nachteile	
• Mehrfachanwendung schwierig • Datenstrukturänderung = Programmänderung • Simultanarbeiten schwierig • Zugriffsrechte schwierig • mehrfache Speicherung der Daten • Integration und Austausch komplex	• Aufwändiger Aufbau und Betrieb • führt zu Flaschenhals
Logische und Physische Daten*ab*hängigkeit	Logische und Physiche Daten*un*abhängigkeit

1.2 Relationales Modell

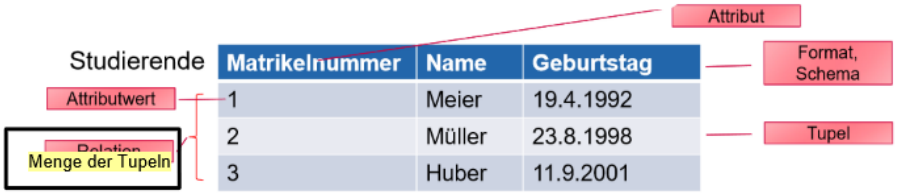

Abbildung 3: Ansicht einer Relation (das ganze = Relation)

Definitionen:

- Domänen = endliche Menge von Werten desselben Types dh. es macht keienen Sinn eine Domäne Hobbies zu haben die unterschiedlich viele Werte (je nach Tupel) enthalten kann ⇒ unauswertbarkeit/unkompatibel
- Attributwerte entstammen genau 1 Domäne
- Attribute
  - Bezeichnung/ Name (bsp. Anrede)
  - Domäne/ Wertebereich (bsp. {"Herr", Frau"})
- n-Tupel = Extension
  - Menge zusammengehöriger Attributwerte
  - Feste Anzahl von Komponenten (=Attributen)
  - Attributwerte verändern sich, Attribute bleiben konstant
  - Menge ⇒ Keine Doppelten Elemente, Ungeordnet
- Format (= Schema / Heading)
  - Menge zusammengehöriger Attribute (feste Anzahl, beliebige Anordung
- Notation
  - Tabellendarstellung
  - $\{< Meier, 19, 2002 >, < Miller, 20, 4000 >, < Abc, 15, 300 >\}$ Zum Format Studierende(Name, Alter, Kontostand)
  - Nur als Format: Studierende(Name, Alter, Kontostand)

1.3 Schlüsselkandidat

Eine Teilmenge der Attribute gilt als Schlüsselkandidat wenn:

- Es gibt nicht 2 Tupel mit denselben Schlüsselattributwerten (zu keinem Zeitpunkt).
- Man kann keines der Schlüsselattribute weglasse ohne diese Eigenschaft zu verlieren

Falls mehrere möglich sind, wird eine Auswahl getroffen. Es gibt immer eine Möglichkeit in einer Relation da jedes Tupel verschieden sein muss (Menge).

1.3.1 Primärschlüssel (primary key, PK)

Ein Ausgewählter Schlüsselkandidat der Explitzit als Primärschlüssel bezeichnet wird.

- Attributwerte sollten nie ändern
- Eindeutigkeit der Werte soll über Zeit gelten
- Attribut(e) sollten möglichst wenig Speicherplatz benötigen (= kurze Schlüssel)
- Falls kein Schlüssel passend ⇒ *Surrogatsschlüssel* (künstlicher Schlüssel)

1.3.2 Fremdschlüssel (foreign key, FK)

Ein Fremdschlüssel ist eine Menge von Attributen einer Relation die den Primärschlüssel einer anderen darstellen (sie können, müssen aber nicht auch den Primärschlüssel von beiden Relationen sein). Über diese Attribute kann auf die anderen Tupel referenziert werden.

- Der Name des PK und FK muss nicht übereinstimmen
- Wenn 2 Attribute den gleichen Namen haben müssen sie noch nicht direkt PK/FK sein, dies muss ausdrücklich deklariert werden
- Es ist sinnvoll gleiche Namen für PK und FK zu verwenden

2 Relationale Algebra (T. 1): Vorlesung 2

2.1 Äquivalenz

Zwei Relationen sind äquivalent wenn sich die eine durch umordnen der Attribute der anderen herstellen lässt (Domänen stimmen überein, Attribute bedeuten dasselbe)

$$R1 \sim R2 \tag{1}$$

Equation 1: Äquivalent aber nicht Gleich

2.2 Relationale Algebra

- Operatoren und Regeln für das Rechnen mit Relationen
- Rechenvorschriften = Abrfragesprache für beliebige Abfragen an die DB
- Anfragen (Ausdrücke der Relationalen Algebra) = Queries

2.2.1 Selektion σ

Unärer Operator[1], erzeugt neue Relation mit gleichem Format aber Teilmenge der Tupel, nur Tupel die der Selektionsbedingung (Kombination von logischen Ausdrücken und/oder Konstanten (Selektionsprädikat), die für jedes Tupel geprüft wird) entsprechen werden ins Resultat genommen sonst {} (Leere Relation) Die Selektion entspricht nicht Select von SQL (mit Select lässt sich mehr machen).

[1]Nur ein (1) Operand

$$R' = \sigma_{Selektionsbedingung}(R) \tag{2}$$

Equation 2: Selektion

$$\sigma_{\phi 1}(\sigma_{\phi 2}(r) = \sigma_{\phi 2}(\sigma_{\phi 1}(r) \tag{3}$$

Equation 3: Selektions Regel

2.2.2 Projektion π

Unärer Operator[2], erzeugt eine neue Relation mit einer Teilmenge der Ursprünglichen Attribute. Dabei können Dublikate entstehen, die entfernt werden müssen. Die Projektion wählt eine Menge von Spalten aus mit Hilfe der attributliste (welche auch Berechnungen, Konstanten, Funktionsaufrufe enthalten kann und auswertbar sein muss). Die Entfernung von Dublikaten wird in DBMS oft nicht automatisch getan (grosser Rechenaufwand).

$$R' = \pi_{Attributliste}(R) \tag{4}$$

Equation 4: Projektion

2.2.3 Umbenennung ρ

Wenn verschiedene Relationen dieselben Attributnamen haben können sie mit R.A identifiziert werden. Dies geschieht durch den Umbennenungsoperator

[2]Siehe Fussnote 1

$$\pi_A(\pi_B(R) \neq \pi_B(\pi_A(r) \tag{5}$$

Equation 5: Projektions Regel

$$R' = \rho_{neuerName}(R) \tag{6}$$

Equation 6: Umbenennung (Rename)

2.2.4 Kombinierend: (kartesisches) Produkt x

Binär, Kreuzprodukt von 2 Relationen = Menge aller Tupel die man erhält wenn man jedes Tupel mit jedem kombiniert. (Matrixmultiplixation kinda), bei Namensgleichheit wird umbenannt.

$$R' = RxS \tag{7}$$

Equation 7: (kartesisches) Produkt

2.2.5 Verbund, Natural Join $\bowtie$

Kreuzprodukt, aber nur mit Tupeln, die irgendwie zusammenpassen: Übereinstimmung der Attributwerte in allen gemeinsamen Attribute (und dann alle Möglichkeiten wo das so ist), evtl. Umbennenung, falls nötig, Format = Vereinigung der Attributmengen. Nachteile: Gemeinsame Attribute (gl. Bezeichnung, Domäne, Bedeutung), Übereinstimmung in allen Attributwerten der gemeinsamen Attribute $\Rightarrow$ nur ein Test auf Gleichheit möglich.

$$RxS \neq SxR \tag{8}$$

Equation 8: (kartesisches) Produkt Regel $\Rightarrow$ kann mit π erzeugt werden.

$$R' = R \bowtie S \tag{9}$$

Equation 9: Natural Join

3 Relationale Algebra (T.2): Vorlesung 3

3.1 Relationale Algebra (Continued)

3.1.1 Theta-Join $\bowtie_P$ (= JOIN in SQL)

Viel Flexibler als natural Join, darum der Normalfall, Kreuzprodukt bilden und mittels Join-Prädikat selektieren (beliebiger logischer Ausdruck, kann auch Vergleiche enthalten). Evtl ist Umbennenung nötig für das neue Format.

Andere Join Varianten:
Siehe SQL Kapitel

3.1.2 Vereinigung (Mengenoperator), $\cup$

Binäre Operation von zwei Mengen (beide müssen das gleiche Format haben, in SQL mind gleiche Domänen = Vereinigungskompatibel) mit dem Resultat von allen Tupel die entweder in einem oder anderem oder beiden vorkommen (Duplikate werden entfernt), kommutativ.

3.1.3 Durchschnitt (Mengenoperator), $\cap$

Binäre Operation von zwei Mengen, (beide müssen das gleiche Format haben, in SQL mind gleiche Domänen = Vereinigungskompatibel), Resultat von allen Tupeln die in beiden Relationen vorkommen, kommutativ

$$R \bowtie S = \pi_{A1,...,AM,R.Bk,C1,....Cn}(\sigma_{R.B1=S.B1und...undR.Bk=S.Bk}(RxS)) \tag{10}$$

Equation 10: Natural Join ist eigentlich zusammengesetzt aus Selektion und Projektion

$$u = u \bowtie u \tag{11}$$

Equation 11: Natural Join Regel

3.1.4 Differenz (Mengenoperator), $\backslash oder -$

Binäre Operation von zwei Mengen, (beide müssen das gleiche Format haben, in SQL mind gleiche Domänen = Vereinigungskompatibel), Resultat sind die Tupel die in R aber nicht in S vorkommen, nicht kommutativ.

3.2 Multimengen = Bag-Algebra (Bsp SQL)

In Multimengen lässt sich nicht unterscheiden zw. identischen Elementen, es ist aber unterscheidbar wie oft ein Bag ein Element enthält (= Multiplizität). RA = Relationale Algebra

- σ Selektion: Gleich wie RA, Dupliakte ändern sich nur wenn Selektionsbedingung so, wie normale Tupel
- π Projektion: Gleich wie RA, aaber keine Entfernung der Duplikate
- $\pi_{E \to X}$ Projektionsprädikat mit Ausdrücken (Normal + Konstanten, Operationen, Funktionen, ect.)
- x Kreuzprodukt: Gleich wie RA
- $\bowtie$ Joins: Gleich wie RA
- $\cup$ Vereinigung (bag union): Gleich wie RA, bei Duplikaten zählen auf beiden Seiten und die grössere Anzahl nehmen.

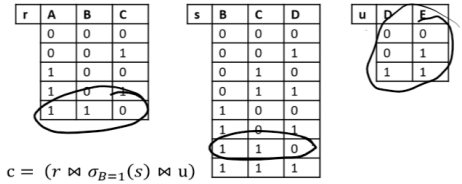

Abbildung 4: An Example of Select and Natural Join $\{< 1,1,0,1,1 >, < 1,1,0,0,1 >, < 1,1,0,0,0 >\}$ zum Format $c(A,B,C,D,E)$

$$R \bowtie_P = \sigma_P(RxS) \tag{12}$$

Equation 12: Theta-Join

- $\sqcup$ Vereinigung (bag concatenation): Gleich wie RA, bei Duplikaten die Summe von beiden Seiten nehmen.
- $\cap$ Durchschnitt: Gleich wie RA, bei Duplikaten zählen auf beiden Seiten und die kleinere Anzahl nehmen.
- $\backslash$ oder $-$ Differenz: Gleich wie RA, bei Duplikaten zählen auf beiden Seiten und Differenz ausrechnen, falls weniger als 0, ist sie einfach 0. (nicht Kommutativ)
- δ Duplikatelimation: Entfernt Duplikate, stellt aus Bag wieder Relation her, spärlich verwenden

3.2.1 Outer Joins: ($\rtimes, \ltimes, \rtimes\ltimes$)

Bei outer Joins werden jeweils alle Tupel der linken/rechten/beiden Operatoren ins Resultat übernommen. Wenn es einen passenden Partner für Join gibt, so wird normal gejoint, sonst mit NULL (kein Wert, Indikation fehlender Werte) aufgefüllt.

$$R' = R \cup S \tag{13}$$

Equation 13: Vereinigung

$$R' = R \cap S \tag{14}$$

Equation 14: Durchschnitt

3.2.2 Aggregat-Funktionen, Aggregieren

Verknüpfung von mehreren Tupeln (bsp Summe aller Verkäufe pro X).

3.2.3 Gruppierung

Gruppierung der Gleichen Tupeln aus der Multimenge, => Keine Duplikate mehr nur noch 1 Relation.

$\sigma_\Phi(\sigma_\psi(r)) = \sigma_\psi(\sigma_\Phi(r))$ **Kommutativität**
$\pi_A(\sigma_\Phi(r)) = \sigma_\Phi(\pi_A(r))$ falls Φ nur Attribute aus der Menge A referenziert
$r \bowtie s = s \bowtie r$ (Achtung: Relationenformat ist verschieden!)

$r \bowtie (s \bowtie t) = (r \bowtie s) \bowtie t$ **Assoziativität**

$\pi_A(\pi_C(r)) = \pi_A(r)$ falls $A \subseteq C$
$\sigma_\Phi(\sigma_\psi(r)) = \sigma_{\Phi \wedge \psi}(r)$ **Idempotenz**

$\pi_A(r \cup s) = \pi_A(r) \cup \pi_A(s)$
$\sigma_\Phi(r \cup s) = \sigma_\Phi(r) \cup \sigma_\Phi(s)$ **Distributivität**
$\sigma_\Phi(r \bowtie s) = \sigma_\Phi(r) \bowtie s$ falls Φ nur Attribute von r referenziert
$\pi_{A,B}(r \bowtie s) = \pi_A(r) \bowtie \pi_B(s)$ falls für die Joinattribute J gilt: $J \subseteq A \cap B$
$r \bowtie (s \cup t) = (r \bowtie s) \cup (r \bowtie t)$

Abbildung 5: Äquivalenzen der Relationalen Algebra

$$R' = R \setminus S \tag{15}$$

Equation 15: Differenz

3.3 Anwendung Relationale Algebra

Biersorten, die nur von einer einzigen Person bevorzugt werden (Tipp: wie finden Sie Biersorten, die von mindestens zwei Besuchern bevorzugt werden?)
$\pi_{BS}(v) \setminus \pi_{BS}(\sigma_{x.BES<y.BES \wedge x.BS=.BS}(vASx \times vASy))$
Alle Biersorten die von allen Gästen des Restaurants Schwanen bevorzugt werden = Alle Biersorten für die es keinen Gast gibt der die Biersorten nicht mag
$\pi_{BES}(\sigma_{RES='Schwanen'}(S)) \quad \setminus \quad \pi_{BES}((\pi_{BES}(\sigma_{RES='Schwanen'}(g) \quad \times \pi_{BES}(v)) \setminus v$

4 Design einer DB mit Entity Relationship: Vorlesung 4

ER: Diagrammsprache für Design von Datenstrukturen

Entitätstyp: (= Rechteck) undefined notion (Punkt in Geometrie) Bsp Studierende (= Tabelle der DB mit Schlüsseln, Relation), hat Attribute (Oval, PK ist unterstrichen (allerdings nur wenn etw. davon abhängt))

Entität: Zeilen der Tabelle, hat Attributwerte

Beziehungstyp: (= Rhombus), existentiell abhängig von anderen,*erbt* nur PK der Zugehörigen Entitstypen, kann aber noch zusätzlich eigene Attribute enthalten. Hat soviele Schlüssel wie er 1er Beziehungen hat. Ein Beziehungstyp kann auch mehrmals der gleichen Entität angehängt sein. (Zbsp Hierarchie von Mitarbeitenden) damit können aber Zyklen modelliert werden die nicht kontrolliert werden können (einmal mit m und einmal mit 1)

- Wenn Beziehungstyp abhängig von mehr als 2 Entitätstypen, kann man pro n-1 über n eine Aussage machen ⇒ weiter nichts.
- Allgemeine Aussage über Beziehungstypen siehe Figure...
- Besser ist es mehere Binäre Beziehungen zu verwenden.

Kardinalitäten: m oder 1 (= Pfeilbezeichnungen) um Beziehung zu bezeichnen

Unabhängiger Entitätstyp: Normaler Entittätstyp, hängt nicht von Beziehung ab, kann auch alleine existieren.

ISA-abhängiger Entitätstyp: (is a = ist ein), Generalisierungs-/Spezialisierungsmuster, existentielle Abhängigkeit ⇒ Generalisierung macht Sinn, wenn sich Entitäten unterscheiden aber Generalisierung gemeinsame Attribute auffängt (Alle PK aus Ursprung kommen in Abhängige)

ID-abhängiger Entitätstyp: Weitere Attribute Benötigt um eine Entität zu beschreiben ⇒ Entität ist ID-abhängig von noch mehr Attributen als ursprüngliche Entität (mind. 1 PK aus Ursprung kommt in Abhängige)

Zusammengesetzter Entitätstyp: Anhängen von Entitäten an Beziehungstypen ⇒ zwingend ID-abhängig emph(in Beziehung sind alle PKs und evt noch eigene Attribute, nicht alle nötig als PK für Abhängige)

5 Durchführung Design Datenbank mit Entity Relationship Prinzip: Vorlesung 5

—> Evtl mit Praktikas uusbaue.

5.1 Tips für ER-Diagramm

- keine Zyklen Modellieren (Bsp. nicht sich selbst vorgesetzt sein)
- Für mehrere Dinge (bsp Versionen) desselben: ID

- Allgemein können wir sagen:
- Es sei R(E_1, L_1, E_2, L_2, E_3, L_3, ..., E_n, L_n) ein Beziehungstyp
- Dieser hängt ab von den Entitätstypen E_1 ... E_n, wobei die Pfeile je mit L_1 ... L_n markiert seien.
- Mit M_j bezeichnen wir die Menge der Fremdschlüsselattribute von R, welche dem Primärschlüssel des Entitätstypen E_j entspricht. (M_j nicht leer, sowie M_i, M_j paarweise elementfremd)
- $M = M_1 \cup M_2 \cup M_3 \cup ... \cup M_n$
- Ist für mindestens ein j $L_j = 1$, so gilt für jedes dieser j mit $L_j = 1$: Die Menge M \ M_j ist ein Schlüssel von R
- Sind alle $L_j = m$, so ist M ein Schlüssel von R (wir wollen Relationen, nicht Bags)

18

Abbildung 6: Allgemein Beziehungstyp

x	y	z	Schlüssel
m	m	m	{A1, A2, A3}
m	m	1	{A1, A2}
m	1	m	{A1, A3}
m	1	1	{A1, A2} und {A1, A3}
1	m	m	{A2, A3}
1	m	1	{A2, A3} und {A1, A2}
1	1	m	{A2, A3} und {A1, A3}
1	1	1	{A2, A3} und {A1, A3} und {A1, A2}

Abbildung 7: *Schlüssel in Beziehungstyp*

- ID und ISA gehen beide Pfeile zum Objekt hin von dem sie abhängen
-

Durchführungstipps

- Brainstorming mit Fachleuten. Alles als Eigenständige Relationen aufschreiben
- Dabei können vorest auch illegale Relationen entstehen (zusammengesetzte, mehrfach vorhandene Attribute), die in der Nachbearbeitung in ID abgeändert werden

- Bei Nachbearbeitung bei vielen Gemeinsamkeiten Neue Entitätstypen erstellen (Bsp ISA Beziehung Person) ⇒ Spart Speicherplatz, ist Übersichtlicher, erzwingt gewisse Vorgaben)

- Darauf Achten, dass Beim Brainstorming nicht zu viele Daten abgespeichert werden wollen. (Handyhüllen müssen keine Infos über Apps enthalten) ⇒ System abgrenzen

- Kein Alter ⇒ Datum abspeichern

6 Korrektes ER-Diagramm: Vorlesung 6

Alle in der Praxis vorkommenden Anforderungen können durch korrektes ER-Diagramm abgedeckt werden. Wird aus einem leeren Diagramm durch eine Folge von Regeln erzeugt.

1. Definiere unabhänigen (vernünftigen) Entitätstyp: ⇒ ein neues Rechteck

2. Definiere Beziehungstyp; ($E_j fr 1 \leq j \leq n \ und \ 1 \leq n$) gegebene Rechtecke oder Rechteckumschlossene Rhomben (1 genügt!) ⇒ neuer Rhombus mit 1 oder m zu E_j Rechtecken

3. Definiere Attribut: (Muss bereits Rechteck/Rhombus/Rechteckumschlossener Rhombus existieren) ⇒ neues Oval mit innerhalb E eindeutigem Namen

4. Umwandlung Beziehungstyp in Zsmgesetzen Entitätstyp: (vorausgesetzt D ist ein Rhombus) ⇒ D wird durch Rechteck umschlossen

5. ID abhängiger Entitätstyp: (F ist Rechteck / Rechteckumschlossener Rhombus) ⇒ Neues Rechteck mit ID markiertem Pfeil zu F

6. ISA abhängiger Entitätstyp: (F ist Rechteck / Rechteckumschlossener Rhombus) ⇒ Neues Rechteck mit ISA markiertem Pfeil zu F

Falls diese Regeln eingehalten werden wird **Normalisierung** überflüssig.

Erstellen von SQL abfragen

6.1 Tips aus Praktika

- Finden Sie die Projektnummern aller Projekte, welche mindestens ein Teil geliefert bekommen, das auch von Sulzer (irgendwohin) geliefert wird.
```
SELECT DISTINCT x.PNr
FROM LTP AS x, LTP AS y, L
WHERE x.TNr = y.TNr AND y.LNr = L.LNr
AND L.LName ='Sulzer';
```

- Finden Sie die Lieferantennummern aller Lieferanten, welche das Teil mit der Nummer ‚T1' in einer Menge liefern, die grösser ist als die durchschnittliche Menge von Teilen ‚T1', welche an dasselbe Projekt geliefert werden.
```
SELECT DISTINCT x.LNr
FROM LTP AS x
WHERE x.TNr ='T1' AND x.Menge > (
SELECT AVG(y.Menge)
FROM LTP AS y
WHERE x.PNr = y.PNr AND y.TNr ='T1'
);
```

7 SQL DDL, DML: Vorlesung 7

Aus dem ER-Schema ergibt jedes Kästchen (jeder Entitäts und jeder Beziehungstyp ein Relationenformat, die Reihenfolge der Attribute ist beliebig (aber einmal festgelegt ist sie wichtig), die Fremd und Primärschlüssel müssen übernommen werden, Reihenfolge des erstellens ist gleich wie die des Schemas. **SQL:**

- Programmiersprache zur Bearbeitung von Datenbanken
- nicht Turing-complete

- sehr mächtig bei Behandlung von Mengen (eigentlich Relationale BAGs (arbeitet nicht one record at the time).
- Abfragen können also Duplikate liefern
- nicht Case-Sensitive
- SQL hat Tabellen anstelle von Relationen (was eine Reihenfolge impliziert... aber nicht so ist.)

7.0.1 Eine DB enthählt:

- Tabellen
- Constraints (Einschränkungen auf DB-Ebene /Tabellen-Ebene / Attributs-Ebene
- Indizes und Code Fragmente (nicht behandelt)

7.1 SQL DDL (Data Definition Language)

7.1.1 Behandeln die Struktur einer DB nicht den Inhalt:

- Erzeugen und Löschen von Datenbanken / Tabellen / Beziehungen
- DBMS (database management system) kann mehrere Datenbanken unterhalten und pflegen
- eine DB ist ein Schema (ein Relationales Modell einer Problemstellung)

7.1.2 Befehle:

Generell Für SQL DDL

CREATE : Erzeugt Element

ALTER : ändert Element

DROP : Löscht Element

CONSTRAINT : ist Optional, wird empfohlen zu bennennen, denn nur benannte Constraints können nachträglich geändert/gelöscht werden

- Check Klausel
- UNIQUE (attributeName, s) = Schlüssel (mehrere Mögliche)
- PRIMARY KEY (attributeName, s) = Primärschlüssel
- FOREIGN KEY (attributeName, s) REFERENCES tableName (attributeName, s) | default Primary Key
  + evtl. **Implizierter Foreign Key Trigger** (ON DELETE, ON UPDATE, NO ACTION, SET NULL, SET DEFAULT, CASCADE)

grundlegende Datentypen :

- CHAR(n)/CHARACTER(n): Zeichenkette, fixe Länge)
- CHAR VARYING(n)/VARCHAR(n): Zeichenkette variable Länge
- INT/INTEGER: Ganzzahl
- REAL: Fliesskommazahl
- NUMERIC(p,s)/DECIMAL(p,s): Festkommazahl

- Erzeugen einer DB: CREATE SCHEMA/DATABASE dbname (evtl: AUTHORIZATION username);
  - dbname muss eindeutig sein
  - username kann als einziger wieder löschen
- Löschen einer DB: DROP SCHEMA dbname (evtl: CASCADE username);
  - löscht Schema nur wenn es keine Elemente enthält ausser bei cascade, dann mit allen Elementen
- CREATE DOMAIN domainName AS (dataType domain) (attribute);
- DROP DOMAIN domainName
  - Bsp: CREATE DOMAIN ssn_type AS CHAR(9
  - Eine Domain (Constraint auf DB Ebene), mit Wertebereich für das Ganze Schema (kann in jeder Tabelle Referenziert werden

  - v.a. geeignet für Schlüssel Attribute und ausführliche Attributsformate ⇒ da 1x zentral definiert und gepflegt

- CREATE TABLE tableName (attributeName attributeType attributeConstraint, attributeName ...)
  - (LIKE tableName1) (erstellt eine Neue Tabelle mit gleichen Attributen wie tableName1 aber ohne die Constraints von TableName1)
  - AS (query) => Speichert Resultate der Query in einer Neuen Tabelle, übernimmt keine Constraints/Schlüssel
  - attributeConstraint: CONSTRAINT: constraintName, DEFAULT: defaultValue, NOT NULL, CHECK (bsp > 0)

7.2 SQL DML (Data Manipulation Language)

7.2.1 Behandelt den Inhalt nicht die Struktur

Daten: Einfügen Ändern, Löschen

INSERT: Fügt immer ganze Tupel in Tabelle ein

UPDATE: Ändert Element

DELETE: Löscht Tupel oder Ganze Tabelle (wenn keine Search Condition)

- INSERT INTO tableName (attrList /query) VALUES (valueList) | query;
  - attrList und valueList muss übereinstimmen
- UPDATE tableName SET attrName = attrWert++ (evtl. WHERE); => Achtung: Nicht so ändern das Primary Key nicht mehr erfüllt ist. (gibt Fehler.)
- DELETE FROM tableName (WHERE searchCondidtion);

7.3 EBNF Erweiterte Backus Naur Form

Formale Syntax Beschreibung einer Programmiersprache.

Symbol	Bedeutung
""	Bezeichnen Symbole der Sprache, die wörtlich zu übernehmen sind
|	Alternativen werden mit einem senkrechten Strich getrennt
()	Runde Klammern dienen lediglich der Gruppierung.
[]	Eckige Klammern stehen für einen optionalen Inhalt, der Null oder einmal vorkommt.
{}	Geschweifte Klammern stehen für eine beliebige Wiederholung des Inhalts: 0-mal, 1-mal, 2-mal, ...
<>	Spitze Klammern stehen für Nichtterminale/Variablen.
::=	Definition / Produktionsregel (z.b. a ::= b)

Abbildung 8: EBNF Regeln

8 SQL Query: Vorlesung 8

- SQL ⇒ Daten abfragen
- Wenig Befehle aber Komplexe Verschachtelungen möglich
- Ähnlich wie Relationale Algebra aber:
  - teilweise nicht autom. Duplikatseliminierung (Widerspruch Mengendefinition)
  - man erhält Relationale Bags
  - Nullwerte haben dreiwertige Logik: *true, false, unknown*
    wobei unknown bedeutet dass auch das Gegenteil nicht stimmt.
    ⇒ Nullwerte vermeiden (DB-Design anwenden)

¬ unknown = unknown
false ∧ unknown = false
unknown ∧ unknown = unknown
true ∧ unknown = unknown
false ∨ unknown = unknown
unknown ∨ unknown = unknown
true ∨ unknown = true

Abbildung 9: Unknown Bedeutung

- SQL Abfragen (=Query) sind flüchtig

8.1 Query

Query :: = <subquery> {(„UNION“| „INTERSECT“(durchschnitt) | „EXCEPT“(differenz)) [„ALL“| „DISTINCT“] <subquery>};
subquery ::= „SELECT“[„ALL“| „DISTINCT“] <attributelist> „FROM“<tableName>;

- DISTINCT enstpricht einer Duplikate Elimination (sparsam Verwenden)
- Select kann einer Projektion enstsprechen
- AS = Rename
- Operationen als Selektions / Projektionsbedinung = WHERE oder subquery WHERE kann auch mit _ und % umgehen als Platzhalter mit LIKE und BETWEEN ... AND, IS, NOT, WHERE NOT IN
- Kommentar: –
- Subquery kann auch als JOIN dienen (NATURAL) (LEFT, RIGHT, FULL) (OUTER, CROSS) JOIN tableName ON joincondition tableName, MIT NATURAL: ohne Join Bedinung (standard)
- Join Kann immer auch durch Where dargestellt werden (From, beide Tabellen, where und join bedienung, weniger offensichtlich, aber ebenso richtig)
- SQL ist Mengensprache, Tupel haben keine Ordnung, können sortiert werden mit ORDER BY (... DESC) auch nur eineindeutig falls mit PK in ORDER BY
  - Lexikographische Ordnung: Chars können auch mit <>= geordnet werden
- WHERE EXISTS (SELECT 1) => eine art Boolean gibt zurück ob gefunden oder nicht

9 SQL Queries: Vorlesung 9 + 10 (T.1)

UNION, INTERSECT, EXCEPT, Aggregatfunktion, IN, ALL, SOME, ANY

9.1 Mengenoperationen (Bag Operationen), auf gleichen Tabellenformaten

DISTINCT ist Default.

- Bag Concatenation $\sqcup$ = UNION ALL
- Durchschnitt $\cap$ = INTERSECT ALL
- Differenz $\setminus$ = EXCEPT ALL

9.2 Aggregatfunktionen

COUNT, MAX, MIN, SUM(1), AVG(1) (bei 1 nur bei Attrubuten mit Zählbarer Domain)
Bsp: Select Count(name) **as anzName** From abc Group By kdNr;
Count(*) Zählt alle Tupel
$\Rightarrow$ COALESCE innerhalb der Klammer um Default Wert anzugeben (ebeso wie DISTINCT)
HAVING kommt nach Group by und schliesst noch alle Gruppen aus die Nicht Bedingung erfüllen.

9.3 IN / ALL / SOME / ANY

- WHERE (name, vorname) (NOT) IN (select ...) (kann UNKNOWN geben)
- WHERE frequenz > ALL (select ...) (bei Null gibt es nichts)
- WHERE (NOT) EXISTS (select 1 from ...) (für exists bei Null false)
- WHERE frequenz > ANY (select ...) (SOME heisst exakt das gleiche und (=ANY) = (IN)

9.4 CASE

SELECT aba, ababba CASE
WHEN (bedinung1) THEN 'ist blöd'

WHEN (bedinung2) THEN 20
ELSE 'sdhfddafh' END
FROM afdaf

9.5 Sichten

Vereinfachung bei Mehrfachen Abfragen
CREATE VIEW sicht AS
Select aba, abab FROM aba, abab WHERE aba, abab, abab....
SELECT * FROM sicht => DA meist unklar was zu UPDATE/DELETE ist sind Sichten **READONLY** => VIEW widerspiegelt immer die Aktuellen Daten...

- Verbergung von Komplexität
- Logische Datenunabhängigkeit
- Vereinfachung (schrittweiser Aufbau von komplexen Abfragen)
- Sicherheit (Zugriffsrechte)
- Nachteile: Performance. (wird bei jeder Verwendung neu berechnet)
- Nachteile: kann nicht Sortiert werden
- Nachteile: Es entsteht oft ein Wildwuchs für Views...

10 Probleme bei NULL und UNKNOWN: Vorlesung 11

- Where mit Potentiell Unknown =nicht mal mehr NULL. (Auch wenn eigentlich immer etwas herauskommen sollte.
- Vergleiche von Leeren Mengen ergibt NULL nicht 0
- Summe Horizontal mit NULL gibt NULL, Vertikal wird NULL ignoriert
- bei IN kann UNKNOWN geben, bei Exists nicht
- COUNT ignoriert NULL nicht (SUM und AVG schon)
- Constraints sind auch bei UNKNOWN erfüllt
- Unterschiede ja nach DB System

10.1 Common Table Expressions (CTE's

WITH queryname1 (column_list) AS
(SELECT ...),
SELECT ... JOIN queryname 1 ...;
=> Eigentlich dasselbe wie geschachteltes query, aber einfacher zu lesen (besonders bei mehrfach geschachtelten) und Rekursiuve Queries sind auch erstellbar.

11 Integritätsbedinungen, Datenbankprogrammierung: Vorlesung 12

11.1 Massnahmen zur Integritätssicherung

Integrität: Annahmen über den Zustand der Daten
Dazu werden Regeln definiert $\Rightarrow$

11.1.1 Integritätsbedingungen = Konsitzenzregeln = Konsitzenzbedingungen ($\neq$ Korrektheit!)

Daten die sicher Falsch sind werden aus der DB ferngehalten.

Von Relationalen DBS durchgesetzt

Bereichsintegrität: (Wert muss in bestimmtem Wertebereich liegen) $\Rightarrow$ sichergestellt Durch Domänen und NULL/NOT NULL

Entitätsintegrität: Primärschlüssel eindeutig und immer vorhanden (NOT NULL) $\Rightarrow$ sichergestellt durch Definition einer nicht leeren Attributmenge als PK

Referentielle Integrität: Inhalt des FK muss NULL sein oder genau ein Tupel mit Schlüsselwert mzss in Referenzierter Tabelle vorhanden sein

Constraints (Einschränkungen) :

UNIQUE - Constraints: Andere Schlüssel (nebst PK und FK)

CHECK - Constraints: Regeln die aussagen über Attribute eines Tupels festlegen (Bsp grösser als und etc)

DEFAULT - Constraints: Werte als Vorgabewerte falls für Attribut kein Wert geliefert.

Wenn ein Datenbestand alle diese Regeln erfüllt heisst er **konsistent**

11.2 Grundkonzepte der Datenbankprogrammierung kennen

Vorteile

- Reduktion Datenverkehr (Client - DBMS, Kapselung) -> Sicherheit (Ausführungsrechte)
- Realisierung s. komplexer Abfragen die allein mit SQL nicht möglich sind
- Parametrisierung von Abfragen
- Erweiterbar/Änderbar ohne Anwendungen anzupassen die Prozeduren und Funktionen nutzen.

Nachteile

- Syntax/ Semantik nicht Standartisiert in allen DBMS
- Verwaltungsaufwand -> müssen wie Softwareobjekte verwaltet werden, aber in Betriebsumgebung (an 2 Orten -> Umständliche Fehlerbehandlung)
- Keine höheren Strukturierungsmöglichkeiten (Prozeduren, Module, Klassen)
- weniger gute Entwicklungsumgebung

11.3 PL/pgSQL

PL/pqSQL = Procedural Language/ PostgreSQL ist SQL-Erweiterung von PostgreSQL zur Datenbankprogrammierung.
Sollte eingesetzt werden wenn Datenzugriffe (lesend/schreibend) das main Goal der DB sind.
Wiederverwendbarkeit von Code

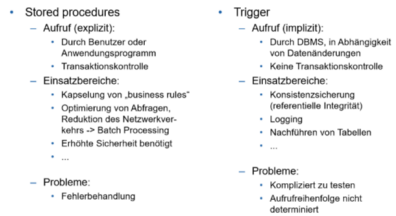

Abbildung 10: Trigger vs Procedures

11.3.1 Stored Pocedures = Funktionen (wissen was, wozu) (explizit)

- Konfigurationsfunktionen: (Zustand des Systems)
- Datum- & Zeitfunktionen: Funktionen zur Bearbeitung von Daten und Zeiten
- Mathematische Funktionen:
- Metadaten-Funktionen
- Sicherheitsfunktionen
- Zeichenfolgefunktionen
- Konversions- / Datentypprüfungsfunktionen
- Rangfolgefunktionen
- ...

Menge von SQL-Anweisungen die unter gemeinsamem Namen gespeichert und vom DBMS als Einheit ausgeführt werden.
Stored procedures werden in DB abgelegt und ausgeführt und können von Anwendungen / Benutzern aufgerufen werden.
CREATE/ALTER/DROP PROCEDURE|FUNCTION tadaa ... RETURN void AS
$body$
BEGIN
(if ... else | elsif, case ... when ... else ..end | loop... exit, while ..loop end loop, for.. in loop... end loop) EXCEPTION
END;
$body$

11.3.2 Trigger (wissen was, wozu) (implizit)

Entlastung der Anwendungsprogramme, einfacher als Rollout
ECA (EVENT CONDITION ACTION): ON ereignis IF bedingung DO action
CREATE/ALTER/DROP TRIGGER tadaa ... etc siehe oben

- BEFORE (vor DML-Operation)
- AFTER (nach DML-Operation)
- INSTEAD OF (anstelle von (DML-Operation)) **nicht in PostgreSQL**

Vorteile / Nachteile

- Sinnvoll: bei oft ausgeführter Funktion, SQL-Constraints reichen nicht, und stark vereinfacht wird
- Sinnvoll: falls Logik von der DB und nicht von Anwendungsprogrammen durchgeführt wird
- nicht Sinnvoll: Fehlerbehandlung/Testen/Debuggen schwierig
- nicht Sinnvoll: Unübersichtlich
- nicht Sinnvoll: Da Ergebnis evtl. abhängig von Aufrufreihenfolge
- nicht Sinnvoll: da geschachtelte Trigger nicht gut gehandelt (Terminierung)

11.3.3 Variablen

name [CONSTANT] type [NOT NULL] [(DEFAULT | :=) expression];

11.3.4 Cursor

'Schleifenvariablen' zur Abfrage eines einzelnen Tupels
DECLARE cursor_name CURSOR
FOR select_anweisung
[FOR UPDATE];

Zum Gebrauch: (zBsp mit LOOP OPEN cursor_name;
FETCH cursor_name INTO @variable1, @variable2, @variable3, ...;
CLOSE cursor_name;

12 Aufwand und Optimierung: Vorlesung 13

- Hardware Optimieren? $\Rightarrow$ hilft meist nichts, da Aufand nicht linear.

12.1 Speicherhierachie

12.1.1 Primärspeicher (RAM, Cache)

sehr schnell teuer, volatil (nur zur laufzeit des Systems vorhanden

12.1.2 Sekundärspeicher (Harddisk, SSD)

Faktor $10^3 - 10^6$ langsamer als Primärspeicher, günstig- billig, persistent (Inhalt über die laufzeit des Systems hinaus vorhanden)
$\Rightarrow$ Zugriffsücke!

12.1.3 Tertiärspeicher

langsam, billig, hohe Kapazität, nicht permanent zugriefbar, Persistent (Inhalt über Systemlaufzeit hinaus vorhanden)

12.1.4 Speicher RDBMS

Kobmbination von Primär und Sekundärspeicher (Ram + (meist) Harddisk), soviel als möglich Ram und Zugriff auf Sekundärspeichermedien so selten wie möglich.

12.2 Magnetplatten, Zugriff auf Magnetplattenspeicher

- Besteht aus Platte/Plattenstapel mit Zylindern welche aus übereinanderliegenden Spuren (= Tracks), bestehen, diese haben Blöcke und Sektoren (kleinste Teile)
- Eine Magnetplatte ist aufgrund ihrer mechanischen komponenten langsam.
- Der Zugriff aud den Magnetspeicher passiert immer Seitenweise, (eine Seite wird gelesen und geschrieben) dh. das Suchen eines Tupels innerhalb einer Seite erfolgt sehr rasch im RAM.
- Zw. Magnetspeicher und Hauptspeicher werden immer ganze Seiten (= Blöcke) übertragen. (bei PostgreSQL 8192 Bytes)

12.3 Abbildungen Sql auf Plattenspeicher

Die logische Datenorganisation wird durch das RDBMS und Betriebssysteme als eine Folge von Seiten abgebildet (= verkettete Liste)

- Einzelzugriff, Mehrfachzugriff, sequentieller Durchlauf Effizienz
- verschiedene Anfragetypen (exact match, partial match, range queries (Bereichsanfragen))
- Dynamisches Verhalten d.h. Anpassung an sich ändernde Datenmengen

12.4 Datenorganisation

12.4.1 HEAP-Datei

Unsortierte Aufreihung (sequentiell) der Datensätze in den Tables

- Vorteil: Aufreihung ist einfach
- Nachteil: Tablescan benötigt schon zur Suche von 1 Tupel
- s. Schlecht für viele Abfrageoptionen, geht schnell zum Anfügen von Daten

12.4.2 HASH-Verfahren

Datensätze werden unterschiedlichen Bereichen zugeordnet (Hashfunktion), innerhalb welchen die Tupel wieder untergeordnet sind

- Vorteil: Aufreihung ist einfach, durchsuchung nach PK ist sehr schnell
- Nachteil: einzelne Table müssen sequentiell durchsucht werden, von/bis nicht geeignet.
- Einfaches einfügen v. Datensätzen, löschen u. lesen deutlich schneller, sortierte Ausgabe nicht unterstützt.

⇒ **Es gibt keine für alle Zwecke optimale Datenorganisation**

12.5 Indexarten

Redundante Datenstrukturen, die dazu dienen DQL-Anweisungen zu beschleuningen.

12.5.1 Indexe Erstellung

Geclustert Indexe wenn PK angegeben, Für Sonstige Schlüssel nicht geclusterte Indexe, sonst selbstständig keine weiteren Indexe.
Mit EXPLAIN kann man sehen was DBMS macht.

```
CREATE INDEX name_index
ON table(column, colum2);
```

12.5.2 Wann lohnt sich ein Index

Attribute die oft abgefragt werde, Fremdschlässel, Attribute die oft gejoint werden, Attribute mit niedrigster Kardinalität sollten nicht indexiert werden.
Abschätzung:
s = Anz. Seiten, r = Anz. Tupel beides unbekannt.
n = Anz. Tupel die gelesen werden
Beispielannahmen: r/s = Sätze pro Seite, Baumtiefe 3 => für jedes Tupel müssen für 4 Seiten gelesen werden (3 Index + 1 Datenseiten)

12.5.3 ISAM-Verfahren (Index Sequential Access Method)

Enthählt neben den eigentlichen Datendatei (die Datensätze in sortierter Reihenfolge) auch noch die Index Datei (ermöglicht beschleunigten Zugriff auf die Daten)

- Vorteil: Richtiger Table wird über Index bestimmt, Sortierung erleichtert suche dort
- Nachteil: Aufreihung wegen Sortierung sehr aufwändig, Indexdatei muss immer angepasst werden
- Lesen wird deutlich effizienter, einfügen und löschen aber deutlich aufwändiger

12.5.4 Verschiedene Arten von Indexen:

Nicht / Geclusterte Indexe (Daten nach Indexkriterium physisch sortiert gespeichert)

Primär- / Sekundärindexe (Geclusterte Indexe nach Primärschlüssel, ungeclusterte nach Sekundär)

Dicht-besetze /dünn-besetzte Indexe (Jedes/nicht jedes Tupel hat ein Eintrag in Indexdatei (bsp geclusterte)))

Abdeckende Index (Index enthält nebst Suchattributwerte noch weitere Attributwerte)

Ein-Attribut, Mehr-Attribut-Indexe

12.5.5 B*-Baum-Verfahren

Index Dateien für Index Dateien (Verschachtelung für Indexdateien) $\Rightarrow$ Baumartige Struktur (Knoten enthalten nur Verweise, nicht die eigentlichen Daten)

- Vorteil: keine übergrossen Indexdateien
- Nachteil: Relativ aufwändige Einfüge-und Löschoperationen
- optimale Lösung für Anwendungen mit überwiegend lesendem Zugriff

12.5.6 Sekundärindex

ein Index für ein zusätzliches Merkmal (not PK) zum Bsp.

- Vorteil: Effizient von lesenden Zugriffen bez. Merkmal weden erheblich gesteigert ohne Primärindex unverändert.
- Nachteil: Relativ aufwändige Einfüge-und Löschoperationen noch komplizierter
- optimale Lösung für Anwendungen mit nur lesendem Zugriff

13 Transaktionsverarbeitung: Vorlesung 14

13.1 ACID-Eigenschaften eines Transaktionssystems

Atomarität (Atomicitiy) / Unit of Work: Zusammengehörige Folge von Lese und Schreibzugriffen (= 1 Arbeitseinheit), muss als ganzes ent-

weder erfolgreich abgeschlossen (= **Commit**) oder rückgängig gemacht werden können (=**Rollback**)

Konsitstenz (Consitstency) : Alle Operationen hinterlassen die DB in einem konsistenten Zustand

Isolation / Nebenläufigkeit (Isolation): Gleichzeitiger Zugriff mehrerer Benutzer ermöglichen, so dass diese Transaktionen keinen unerwünschten Einfluss aufeinander haben (auch nur bei 1 CPU mehrere Zugriffe möglich gleichzeitig)

Dauerhauftigkeit / Recovery (Durability): Automatische Behandlung von Fehlern und schnellstmöglicher Wiederanlauf nach schwerwiegerendern Fehlern, Wiederherstellung (**Rollforward**) verlorener Daten und Rücksetzen (**Rollback**) fehlerhafter Daten.

13.2 Probleme mit konkurrenter Transaktionen beschreiben können

- Lost-Update (Nebenläufigkeitsproblem):
  - Überschreiben bereits getätigter Updates, durch dass keine Isolation herrscht. (Beide Updaten denselben Wert). Das 1. Update geht verloren.
  - Lösung: Durch andere Transaktion gelesene Daten dürfen bis zur Beendigung nicht verändert werden.
- Dirty-Read (Nebenläufigkeitsproblem)
  - Lesen von Uncommitted Changes. Problem bei Rollback dieser Changes.
  - Lösung: Nur Committed Changes werden gelesen.
- Non-Repeatable-Read (Nebenläufigkeitsproblem)
  - Lesen von 2 Verschiedenen Werten wenn Ben. 2 inzw. was geändert hat.
  - Lösung: Nur DB Zustand bei Beginn wird gezeigt.
- Phantom-Read (Nebenläufigkeitsproblem)
  - Zwischenresultate werden verändert wenn transaktion von Ben.2 etw. an Tabl. geändert.
  - Lösung: Nur DB Zustand bei Beginn wird gezeigt.

13.3 Aspekte von Nebenläufigkeit und Transaktionen in der Praxis kennen

Isolationsebenen vs. Phänomene:

Isolationsebene	Dirty Read	Non-Repeatable Read	Phantom Read	Lost Update
READ UNCOMMITTED	~~möglich~~ (nicht in Postgres)	möglich	möglich	~~möglich~~ (nicht in Postgres)
READ COMMITTED (Häufig in der Praxis)	verhindert	möglich	möglich	verhindert
REPEATABLE READ	verhindert	verhindert	~~möglich~~ (nicht in Postgres)	verhindert
SERIALIZABLE	verhindert	verhindert	verhindert	verhindert

In Postgres: READ UNCOMMITTED = READ COMMITTED

Abbildung 11: Lost Update kann nie in Transaktionssystem toleriert werden, der Rest zu gewissem Ausmass tolerierbar.

13.3.1 Transaktionen in der Praxis

Anwendungsentwickler muss Transaktionsgrenzen festlegen, COMMIT, ROLLBACK, ABORT
BEGIN/COMMIT/ROLLBACK TRANSACTION: falls durch Fehler beendet die Sitzung abbrechen, Systemabsturz machen ist Inkonsistenter Zustand => Recovery!

Eine DDL Anweisung wird in eine laufende Transaktion eingebettet. In eigene Transaktion gekapselt.

- Falls Constraint Verletzung sofort Rollback (wird am Ende der Transaktion überprüft)

Zustand bei offener Transaktion

- Vorhegender Zustand wird in **Before-Images** festgehalten (um wieder hergestellt werden zu können)
- Betroffene Sätze sind durch Schreib/Lese-Sperren blockiert; sie können in anderen Transaktionen nicht gelesen /geändert werden
- Andere Transaktionen sehen geänderte Daten nicht.

Zustand nach DB Commit

- Geänderte Daten sind in DB festgeschrieben
- Alle Datenänderungen sind in **Transaktionslogs** protokolliert
- Vorhergehender Zustand kann nicht mehr mittels ROLLBACK wieder hergestellt werden
- Alle Sperren der Transaktion sind wieder frei gegeben
- Geänderte daten können mit READ COMMITED ISOLATION (oder in nachfolgenden Transaktionen gelesen werden)

Zustand nach Rollback

- Geänderte Daten vollständig zurückgesetzt
- Herstellen der alten Daten durch Before-Image
- Rollback wird auch im Transaktionslog aufgezeichnet
- Spreren der Transaktion wieder freigegeben

13.4 Schedules

Serielle Schedule: konsistenzerhaltend, schlechte Performance
Konflikteserialisierbar (Serielle Schedule): mit effizienten Algorithmen (immer möglich solange es keine Zyklen enthält (Schreib/Lese-Konflikt oder Lese/Schreib-Konflikt oder Schreib/Schreib-Konflikt))

13.4.1 Scheduler / Transaktionsmanager

Kann 3 Zustände einnehmen: Ausführen (execute), Verzögern (delay), Zurückweisen (reject)
Scheduler arbeitet

- aggressiv: wenn er Konflikte zulässt, dann versucht er aufgetretene Konflikte zu erkennen und aufzulösen. (bsp Postgers)
- konservativ: vermeidetet Konflikte aber nimmt Verzögerungen von Transaktionen in Kauf (bsp SQL Server).

13.5 Konzept der Sperren

Dienen der Erweiterung jeder Transaktion durch spezifische Operationen

- Lese-Sperre (Share Lock: andere Transaktionen können weiter lesendend auf das DB Objekt zugreiffen und Lesesprerren setzen aber keine Schreibsperren)
- Schreib-Sperre (Exclusive Lock: Das Betreffende DB Objekt alleine kann diese noch bearbeiten)
- Unlock (read und write unlock)

Regeln zur Sperrdisziplin

- Schreibzugriffe nur mit Schreibsprerre (nur wenn dort keine Sperre bereits)
- Lesezugriffe nur mit Lesesperre (nur wenn dort keine Schreibsperre bereits)
- Nach Unlock keine erneute Sperre direkt
- Transaktion darf nicht 2x auf gleiches Objekt angefordert werden
- bei Commit/Rollback müssen alle Sperren aufgehoben werden

13.6 Blocking, Livelock und Deadlock

Blocking: Eine gesperrte Ressource zwingt andere Prozesse zu warten bis diese freigegeben wird. => Keine Lösung Sperren sind nötig.

- Lange laufende Abfragen vermeiden, kurze Transaktionen
- Ineffiziente Abfragen optimieren
- Vernünftige indiziern und Indexe verwenden
- Keine Benutzereingaben innerhalb von Transaktionen
- Sperr-Timeouts verwenden

Livelock: Transaktion kommt nie dran weill immer andere vorher berücksichtigt werden => RDBMS (Relational Database Management System) muss geeignet (fair) auswählen.

Deadlock: Eine Menge von Transaktionen sperren sich gegenseitig, alle warten ewig. => RDBMS erkennt Deadlocks durch Zyklensuche

- Objekte immer in derselben Reihenfolge ansprechen
- Teure Objekte zuletzt ansprechen
- Angepassten Isolationslevel verwenden

13.7 Grundlagen von Recovery kennen

Massnahmen zur Wiederherstlelung verlorengegangener Datenbestände: DBMS besteht aus Transaktionsmanager und Scheduler für I und C und Recovery Manager für A und D

Recovery-Komponenten: Puffer-Manager Holt/Schreibt Daten von stablien Speicher in den/vom Puffer, ersetzt Daten im Falle eines Pufferüberlaufs

13.7.1 Fehlerklassifikation

- Transaktionsfehler
  - bricht jeweilige Transaktion ab
  - haben keinen Einfluss auf jeweiligen Speicher des Systems (lokaler Fehler)
  - Bsp: Fehler im Programm, Transaktionsabbruch durch Benutzer oder System
  - Behandlung: Zurücksetzung aller Änderungen der Transaktion (Rollback)
- Systemfehler
  - Zerstörung der Daten im Hauptspeicher (flüchtige DB) aber nicht im Hintegrundspeicher (permanente DB)
  - Bsp: DBMS Fehler (config), Betriebssystemsfehler, Harwarefehler
  - Behandlung: Zurücksetzung aller Änderungen der Transaktionen (REDO und UNDO)
- Mediafehler
  - Verlust von Daten der stabilen Datenbank
  - Bsp: Head-Crashes (Controller Fehler), Feuer, Erdbeben, Operator Fehler
  - Behandlung: Einspielen der verloren DB-Dateien aus Archiv, Anwenden aller Transaktionlogs aus Log-Archiv, bring in konsisten Zustand (REDO und UNDO)

13.7.2 Logging

WAL-Prinzip (Write Ahead Prinzip): Vor dem Commit einer Transaktion müssen alle zugehörigen Log-Einträge auf stabilen Speicher ausgelagert werden
Vor dem Auslagern einer modifizierten gepufferten Seite in persistente Datenbank müssen alle Log Einträge in stabilen Speicher ausgelagert werden.
Sicherungspunkte (checkpoint): Die geänderten Seiten werden in persistente DB geschrieben, und Pufferlogs in stabilen Speicher