Zusammenfassung Datenbanken

Komponenten eines Datenbanksystems

- Datenbank → Sammlung von Daten
- Datenbankmanagementsystem → Vowalten; Verbeiten; Auswerten - Software
- Datenbanksystem → DB & DBMS = DBS

Aufgaben eines Datenbanksystems

- Beschreibung
- Speicherung & Pflege
- Wiedergewinnung

Anforderungen an ein Datenbanksystem

- Integration → Einheitliche Vorwaltung & redundanzfreie Datenhaltung
- Operationen → Möglichkeit Daten zu suchen & manipulieren
- Date Dictonary → Katalog für Zugriff auf Details
- Benutzersichten → unterschiedliche Sichten nach Anwendungszwecke
- Konsistenzüberwachung → Überwachung der Korrektheit
- Zugriffskontrolle → Ausschluss unautorisierter Zugriffe
- Transaktionen → Speicherung von erfolgreichen Änderungen
- Synchronisation → vermeidet Beeinflussung von mehreren Nutzen
- Datensicherung → Rekonstruktion von Daten

charakteristika von DBS

- Anfragesprachen
  - ↳ Data Definition Language (DDL) → z.B CREATE Table
  - ↳ Data Manipulation Language (DML) → z.B SELECT..
- Daten & Metadaten
  - ↳ Daten in Tabelle ↳ Struktur (z.B Header)
- Datenmodelle
  - ↳ konzeptionell → gesamtkonzept (Übersicht) der DB
    - ↳ Objekte & ihre Beziehungen (ER-Diagramm)
  - ↳ Logisch → Brücke zwischen konzept & Physisch
    - ↳ Relationsmodell / XML / NOSQL
  - ↳ physisch → realisierung der DB-Objekte
    - ↳ konkrete Implementation
- Datenabstraktion
  - ↳ Externe Ebene → Sicht auf Teilmenge des logischen Schemas
  - ↳ logische Ebene → gesamtes DB-Schema (konzeptionelle Ebene)
  - ↳ physische Ebene → Internes Schema → Interne Speicherung (Interne Ebene)
- Datenunabhängigkeit
  - ↳ logisch → Änderung am logischen Schema → Anpassung externe Ebene
  - ↳ pysisch → Änderung an internem Speicher → keine Auswirkung auf andere Ebenen

konzeptionelle Datenmodellierung

Für das Design stellen wir uns folgende Fragen:

- Welche Entitäten (objekte) gibt es in der Organisation?
- Welche Beziehungen existieren zwischen den Entitäten?
- Welche Informationen (Attribute) möchten wir über Entitäten & Beziehungen speichern?
- Welche Regeln gelten in der Orgenisation?
- Welche Integritätsbedingungen ergeben sich daraus?

Bausteine eines Entity-Relationship (ER) Modell

- Entitätstypen (Entities) →
- Attribute →
- Beziehungen →

z.B

Entitätstypen

↳ Eine Sammlung von Entitäten

↳ Personen ist ein Entitätstyp, Bob ist eine Entität davon

- ein Enitätstyp besitzt eine Menge von Attributen
- ein Attribut ist der Primärschlüssel (eindeutige Identifikation)

⇒ Entitätstypen können abhängig sein von einander

↳ starke & schwache Entitätstypen

↳ schwache übernimmt Id des starken

Im ER Modell:

Primärschlüssel → unterstrichen → <u>Id</u>

Existenzbeziehung → Existenz von Entität A hängt von Entität B ab

↳ 2. umrandung des schwachen

↳ 

Primärschlüssel bei Existenzeziehung

↳ Zusammensetzung aus beiden Ids lässt die genaue Id der Abhängigen Entität identifizieren

↳ Id der schwachen Entität wird gestrichelt unterstrichen

Attribute

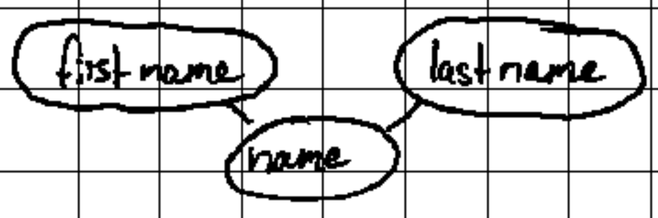
↳ Eigenschaften von Entitätstypen

verschiedene Arten:

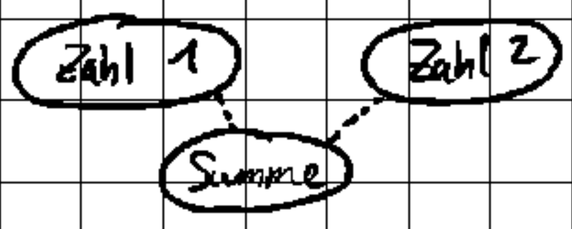
- Zusammengesetzte Attribute → bestehend aus 2 oder mehr
- Abgeleitete Attribute → errechnet aus anderen Attributen
- Mehrwertige Attribute → gleiches Attribut mehrmals in Entität
- Primärschlüssel → eindeutige Identifikation

Im ER-Modell

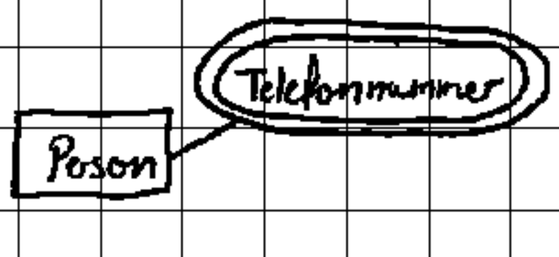
Zusammengesetzt



Abgeleitet



Mehrwertig



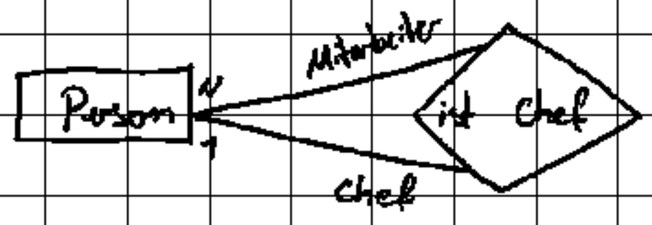
⇒ Zusammengesetzte Schlüssel

↳ mehrere Attribute bilden den Schlüssel

z.B

Beziehungen

- 1:N → Person hat N Termine, 1 Termin hat 1 Person
- N:M → Person hat M Termine, 1 Termin hat N Personen
- 1:1 → Person hat 1 Kreditkarte, 1 Kreditkarte hat 1 Person
- Rekursiv → Entität hat x-mal die gleiche Beziehung
  - ↳ Rollennamen zur Unterscheidung nötig
  - ↳

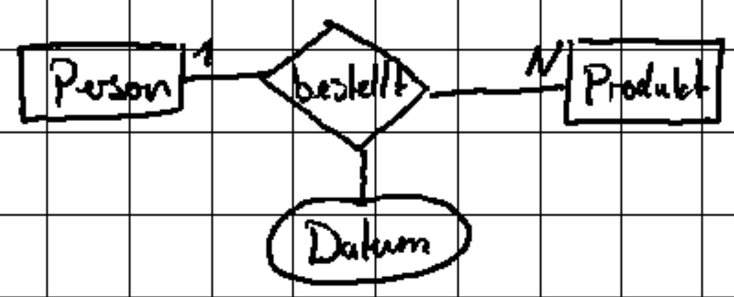


  - ↳ Chef hat N Mitarbeiter
    Mitarbeiter hat 1 Chef

- Mehrwertige Beziehungen
  - ↳ eine Beziehung kann zwischen mehr wie 2 Entitätstypen sein
    - ↳ Anzahl Entitätstypen = Grad der Beziehung

- Beziehungs Attribut
  - ↳ eine Beziehung kann einen Attributen haben

z.B Datum einer Bestellung



Integritätsbedingungen (Constraints)

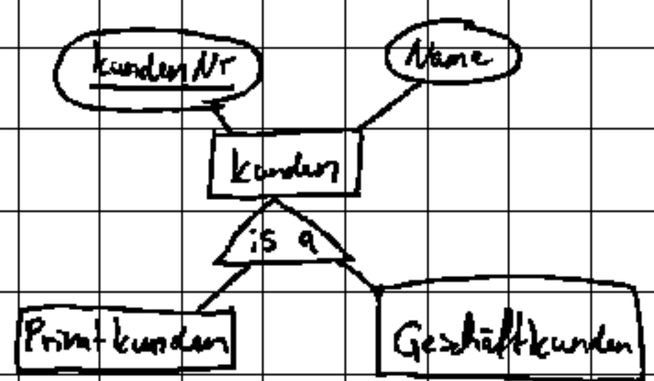
↳ Bedingung die erfüllt sein muss (z.B Person hat max. 1 Vater)

⇛ Teil des DB-Schemas ⇛ Teil des Modellierungsprozess

Generalisierung

↳ Vererbung von Attributen des Super-Typen an Sub-Typen

Im ER Modell



Domain

↳ Werkbereich, der für einen Attribut zulässig ist

Kartesisches Produkt

↳ kreuzprodukt von k Mengen

↳ alle möglichen kombinationen

Relationen

↳ Eine Teilmenge des kartesischen Produktes

↳ Beziehungen zwischen Entitäten und ihren Attributen

⇒ im Relationsmodell (vom ER Modell) werden aus Entitätstypen Tabellen, aus Attributen Spalten und aus Beziehungen werden Verknüpfungstabellen, die die Schlüssel der Tabellen verlinken

Schlüssel

↳ Wert der eine Zeile einer Tabelle identifiziert

↳ muss folgende kriterien erfüllen:

- Eindeutigkeit → durch den Schlüssel wird eine Zeile einzigartig
- Minimalität → Schlüssel sollte nur notwendige Attribute enthalten
  ↳ z.B Studenten ID & Geburtsdatum ist nicht minimal, da die Studenten ID bereits einzigartig ist und somit ausreicht

Superschlüssel

↳ Menge von Spalten (Attributen) die die Zeilen (Tupel) eindeutig identifizieren in einer Tabelle (Relation)

↳ muss nicht minimal sein

↳ triviale Superschlüssel wären also alle Spalten der Tabelle

Schlüsselkandidat

↳ kleinst möglichen Schlüssel die eine Zeile eindeutig identifizieren

↳ können mehrere sein (z.B Studenten ID, AHV Nr ⇒ Beide minimal & eindeutig)

Formal: minimale Teilmenge eines Superschlüssel

Transformation ER Modell → Relationsmodell

- Entitätstypen → Tabellen
- Attribute → Spalten der Tabelle
- Sub-Attribute → einzelne Spalten (Super Attribut keine Spalte)
- Beziehungen → Verknüpfungstabellen → Schlüssel der Tabellen verlinken

- mehrwertige Attribute → eigene Tabelle mit Fremdschlüssel der Entity Tabelle
- 1:N Beziehungen → Fremdschlüssel
  ↳ N-Tabelle hat Schlüssel der 1-Tabelle
- rekursive Beziehung → Fremdschlüssel auf Schlüssel in eigener Tabelle
- N:M Beziehungen → Verknüpfungstabelle

- Generalisierung → folgende Möglichkeiten
  - volle Redundanz → jede Sup-Typ hat eigene Tabelle
    ↳ jede Sub-Typ hat auch spalten des Super Typ
  - Hausklassenmodell → jede Sub-Typ hat eigene Tabelle, inkl allen Spalten
    ↳ Werte werden nur in Spezifische Tabelle eingetragen
  - Vertikale Partitionierung → jeder Sub-Typ hat eigene Tabelle, jedoch nur mit den zusätzlichen Attributen als Spalten
    → Super-Typ ID ist Fremdschlüssel
  - Hierarchierelation → eine einzige Tabelle mit Spalten von allen Sub-Typen + einem Typ-Tag für Sub-Typ-identifizierung

Abhängigkeit

↳ eine Spalte hängt von einer anderen Spalte ab, wenn der Wert in einer Spalte immer das gleiche Gegenstücke in einer Anderen hat.

Z.B Alter & Geburtsdatum (G → A)

Voll Funktional Abhängig:

↳ eine Spalte A ist voll Funktional Abhängig von einer Spalte oder Gruppe von Spalten B, wenn sie vom gesamten B abhängt und nicht nur von einem Teil davon.

Z.B Alter ist abhängig von Geburts- tag -monat -jahr und nicht nur vom Jahr (B ⇒ A)

Normalisierungen

↳ minimierung von Redundanz & Anomalien

1. Normalform

↳ keine Zusammengesetzte, mehrwertige oder relationswertige Attribute (Verlinkung zu anderer Tabelle)

⇛ nur Atomare Werte & jede Zeile eindeutig identifizierbar

2. Normalform

↳ 1. Normalform & jedes Attribut hängt voll funktional vom ganzen Schlüssel ab.

3. Normalform

↳ 2. Normalform & keine nicht trivialen transitiven Abhängigkeiten Zwischen nicht - Schlüssel

↳ 2 → 1 + 3 → 2 = 3 → 1 ↯

⇛ nicht - Schlüssel dürfen nur vom Schlüssel abhängig sein, nicht aber von anderen nicht - Schlüsseln

Relationale Algebra

↳ formales Modell als Grundlage für Anfragesprachen

Grundoperationen

- Vereinigung $R = S \cup T$ (alles)
- Schnitt $R = S \cap T$ (gemeinsames)
- Differenz $R = S \setminus T$
- kartesisches Produkt (kreuzprodukt) $R = S \times T$
- Selektion $R = \sigma_f(S)$
- Projektion $R = \pi_{A,B,..}(S)$

Vereinigung ∪

↳ alle Zeilen aus beiden Tabellen

Schnittmenge ∩

↳ Zeilen, welche in beiden Tabellen vorkommen

Differenz $A \setminus B$

↳ Zeilen aus A ohne die Zeilen, die auch in B sind

kartesisches Produkt x

↳ Spalten aus zwei Tabellen "verschmelzen"

Selektion $\sigma_p(R)$

↳ Auswahl von Zeilen der Tabelle R, welche Kriterium p erfüllen

Projektion $\pi_{A_1 A_2 .. A_n}(R)$

↳ Beschränkung der Tabelle R auf die Zeilen $A_1 - A_n$

Join ⋈

↳ Kreuzprodukt mit anschliessender Selektion

↳ $R \bowtie_p S = \sigma_p(R \times S)$

Verschiedene Arten von Join:

- Inner Join ⋈ → Nur Zeilen mit Join Partner sind im Ergebnis
- left outer Join ⟕ → Alle Zeilen der linken Tabelle befinden sich im Ergebnis, auch ohne Join Partner
- right outer Join ⟖ → Alle Zeilen der rechten Tabelle sind im Ergebnis
- full outer Join ⟗ → Alle Zeilen beider Tabellen sind im Ergebnis
- natürlicher Join → Verbund mit Kriterium dass gleichnamige Attribute ausgewählt werden.
- self Join → Verbund eigener Tabelle bei rekursiven Beziehungen
- Semi Join → Verbund aus Zeilen mit Beziehungen

Umbenennungsoperator ρ

↳ Umbenennung von Attributen bei einer Anfrage, für das Resultat

z.B $\rho_{(\text{StudentenID} \rightarrow \text{ID},\ \text{createdAt} \rightarrow \text{ErstellDatum})}(\text{Studenten})$

Spaltenname → neuer Name

Tabellen Name

Anfrage optimierung

↳ Anfragen an Datenbanken können so geschrieben werden, dass sie möglichst schnell und effizient sind

Grundsätzlich gilt:

- Menge der Daten so früh wie möglich reduzieren durch
  - → frühe Selektion
  - → frühe projektion
- Join anstelle des kreuzproduktes
  - ↳ meist genügen kreuzprodukte aus Zeilen mit Beziehungen
- nichts doppelt berechnen

Kardinalitätsschätzung

↳ Kosten schätzung der Operationen

- Selektion → $|\sigma_p R| = sf_p \cdot |R|$

  sf_p: prozent Annahme; $|R|$: Anzahl Zeilen in Tabelle

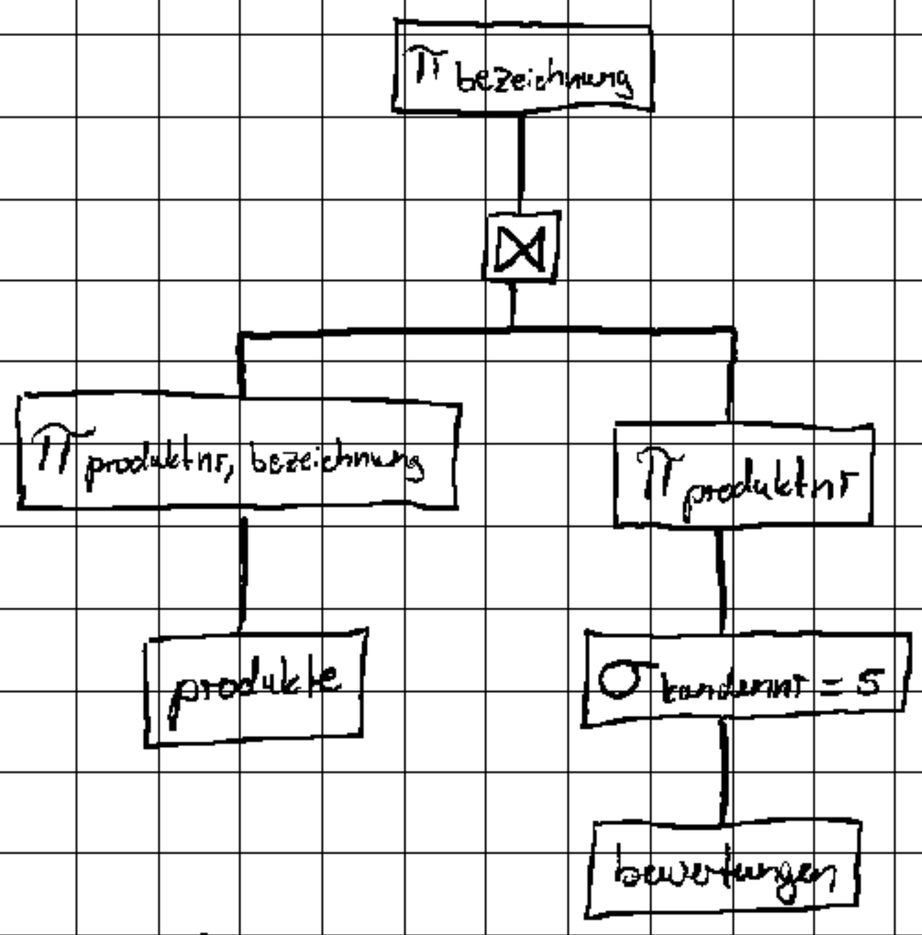
- Kreuzprodukt → $|R \times S| = |R| \cdot |S|$

- Join → $|R \bowtie S| = 0 < X < |R| \cdot |S|$

Operatoren Baum

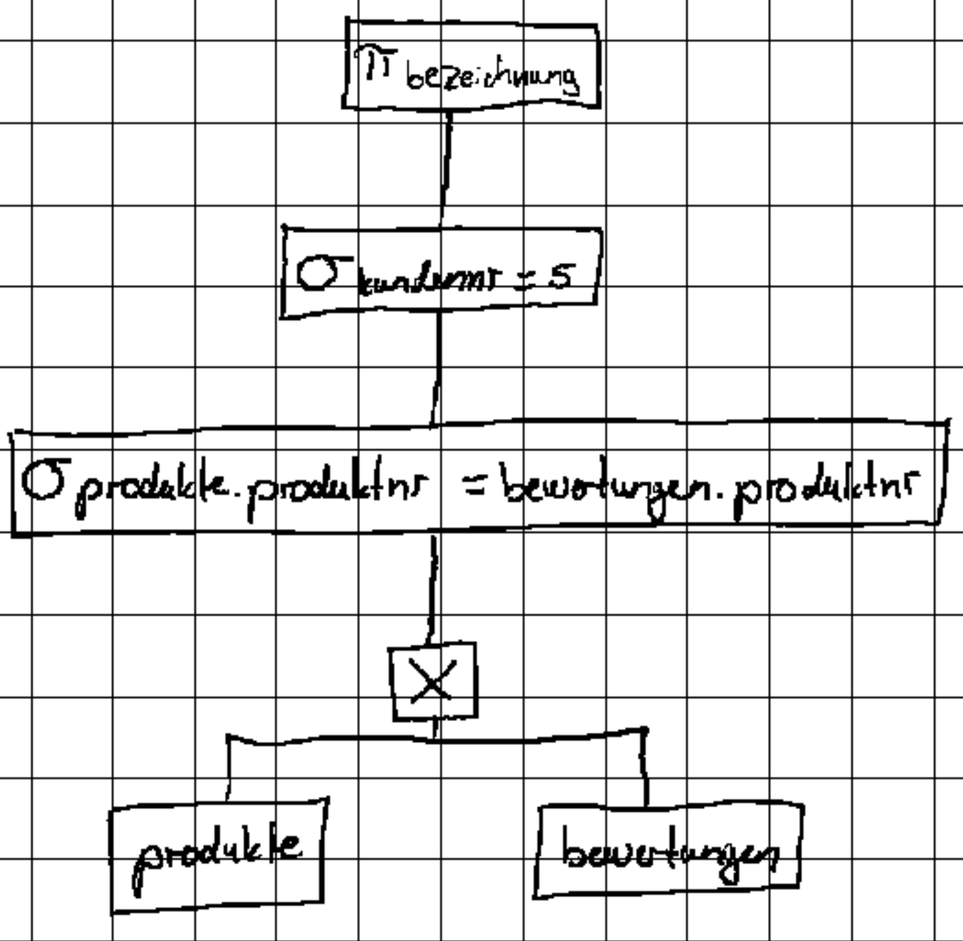
↳ grafische Darstellung der Auswirkungen von Operationen auf das Ergebnis

⇒ Oberste Knoten ist das Ergebnis, Blätter sind Tabellen und Knoten sind Operationen



- Zuerst minimierung der Daten, dann Join

⇒ weniger Daten insgesamt



- Zuerst grosser Datensatz, dann minimierung

⇒ mehr Daten insgesamt

SQL

↳ Anfragesprache für Datenbanken

Data Definition Language (DDL)

↳ Tabellen definition → create, drop, alter etc...

Create Syntax:

```
CREATE TABLE name (
Varname Type Spaltenoption,
Varname2 Type );
```

↳ Daten Typen:

- INT → ganze Zahl
- NUMERIC (x,y) → Dezimalzahl mit x Stellen vor und y nach dem Koma
- REAL → reelle Zahl
- CHAR(x) → Zeichenkette mit Länge x
- VARCHAR(x) → Zeichenkette mit maximaler Länge x
- DATE → Datum, Format: Jahr - Monat - Tag
- TIMESTAMP → Zeit & Datum, Format: yyyy-mm-tt h:m:S:ms

Spaltenoptionen

- PRIMARY KEY → Primärschlüssel (unique & not null)
- NOT NULL → Verbot von Null Werten
- CHECK → Bedingung für akzeptierte Werte
  z.B CHECK(vorname LIKE '%@%')
- UNIQUE → einzigartiger Wert (keine Duplikate)
- DEFAULT → definiert Standardwert
- REFERENCES → Referenz auf Fremdschlüssel

Zeilenoptionen

- ON DELETE ... → was passiert wenn Referenz gelöscht wird
  - CASCADE → alles was mit gelöschter Referenz verbunden ist auch löschen
  - SET NULL → null setzen
  - SET DEFAULT → Standardwert setzen
  - RESTRICT → Löschen verbieten
  - NO ACTION → abwarten
- ON UPDATE ... → analog zu on delete

Bedingungen

- CHECK (varname LIKE ('%@%'))
  irgendwas @ irgendwas
- CHECK (varname BETWEEN 1 AND 5)
- ... DATE CHECK (date1 > date2)

Tabelle erstellen mit LIKE & AS

↳ Tabelle mit dem Vorbild einer Anderen erstellen

```
CREATE TABLE t2 (LIKE t1);
CREATE TABLE t2 AS
    SELECT * FROM t1 WHERE x<10;
```

Tabelle modifizieren / löschen

- ALTER TABLE t1 ADD COLUMN date1 DATE;
- DROP TABLE t1;

Data Manipulation Language (DML)

↳ einfügen, ändern, löschen von Daten

Einfügen - INSERT

```
INSERT INTO t1 (col1, col2, col3 ...)
       VALUES ('1', 'b', 3.3, NULL ...);
```

⇒ Spaltennamen können ganz oder teilweise weggelassen werden, falls man nur einen Teil der Spalten befüllen möchte, oder die Struktur bekannt ist.

Ändern - UPDATE

```
UPDATE t1 SET col1 = 2 WHERE col2 = 3;
UPDATE t1 SET col1 = col1 + 2
```

Löschen - DELETE / TRUNCATE

```
DELETE FROM t1 WHERE col2 = 3;
DELETE FROM t1 WHERE col2 = 3 AND col3 = 'b';
DELETE FROM t1;          → kann rückgängig gemacht werden
TRUNCATE TABLE t1;       → löscht Tabelle direkt
```

Data Query Language (DQL)

↳ lesen von Daten

SELECT

- SELECT col1, col2, ... FROM t1;
- SELECT * FROM t1;
- SELECT DISTINCT col1 FROM t1;
- SELECT * FROM t1 WHERE col1 $\leq$ 3;
- SELECT * FROM t1 WHERE col1 LIKE 'M%';
- SELECT * FROM t1 WHERE col1 IS / IS NOT Null;

Kreuzprodukt

- SELECT * FROM t1, t2

Join

- SELECT * FROM t_1, t2 WHERE t_1.col1 = t_2.col2;
- SELECT * FROM Tabelle1 t1 JOIN Tabelle2 t2
  WHERE t1.col1 = t2.col2;

Aggregatfunktionen

↳ Funktionen die auf Mengen angewendet werden können

- COUNT → Anzahl
- SUM → Summe
- AVG → Durchschnitt
- MIN / MAX → minimaler / maximaler Wert
- GROUP BY col → Ergebnis nach Wert gruppieren
- ORDER BY col → Ergebnis nach Spalte sortieren
- HAVING → nur Werte die der Bedingung entsprechen

⇒ z.B. SELECT COUNT(*) FROM t1;

Sub - Anfragen

↳ SQL Anfragen innerhalb einer anderen Anfrage

z.B SELECT * FROM t1 WHERE col1 =
(SELECT MAX(col1) FROM t2);

Mergen operationen

↳ kombination der Resultat mehrer Anfragen

- UNION → SELECT ... UNION SELECT ...
- UNION ALL → UNION mit Duplikaten
- INTERSECT → Schnittmenge
- EXCEPT → Differenz
- (NOT) EXISTS → Werte existieren is Selektion
- IN → Wert ist in Selektion Spalte enthalten
- CAST → Typ umwandlung
  ↳ SELECT CAST(col1 AS INT) FROM t1;

Views in SQL

↳ virtuelle Tabellen, die keine Daten selber speichern, sondern nur die Daten aus anderen Tabellen dynamisch anzeigen nach vor definierten Mustern.

Erstellen

- CREATE VIEW BMWs AS
  SELECT * FROM Autos WHERE brand='BMW';
  ↳ aus dieser View kann wie gewohnt auch selektiert werden
  ↳ CHECK OPTION → stellt sicher dass eingefügte Werte der View Anforderung entsprechen
  ⇒ SELECT... WHERE brand = 'BMW' WITH CHECK OPTION;

Materialisierte Views

↳ Views, die die Daten abspeichern → müssen aktualisiert werden

- CREATE MATERIALIZED VIEW BMWs...
- REFRESH MATERIALIZED VIEW BMWS;

Data Control Language (DCL)

↳ Benutzer & Rollen Vowaltung

- CREATE USER u_1 WITH PASSWORS 'p_1';
- CREATE ROLE r_1;
- GRANT r_1 TO u_1;
- GRANT `SELECT, INSERT, UPDATE, REVOKE ALL PRIVALEGES` ON t_1 TO u_1/r_1;

⇒ user / Rollen sollen in der Lage sein Rechte zu vergeben:

.... with GRANT OPTION;

Transaction Control Language (TCL)

Commit & Rollback

↳ Speichere Daten nur bei erfolgreicher Ausführung sonst mache es rückgängig

Z.B

```
BEGIN TRY
BEGIN TRANSACTION
  INSERT INTO t1 ... ;
  INSERT INTO t1 ... ;
      ⋮
COMMIT
END TRY
BEGINN CATCH
 ROLLBACK
 END CATCH
```

ACID Transaktionen

- **Atomicity** → ganz oder gar nicht
- **Consistency** → von konsistentem Zustand zu konsistentem Zustand
  ↳ DB bleibt intakt
- **Isolation** → parallel laufende Transaktionen beeinflussen sich nicht
- **Durability** → Transaktion bleibt auch bei Systemausfall erhalten

Mehrbenutzeranomalien

↳ Probleme die auftreten wenn mehrere Personen eine Transaktion machen

- Dirty Read → user 1 liesst un-commited Änderungen von user 2
- Lost Update → gleichzeitiger write mehrerer user
- Non repeatable Read → user 1 liesst werte während user 2 ändert

Serialisierbarkeit

↳ Transaktionen bestehen aus read & write Operationen

⇒ read von user 1 auf Wert x : $r_1(x)$

write von user 1 auf Wert x : $w_1(x)$

commit von user 1 : c_1

Serialisierbar

↳ wenn Operationen keine Konflikte aufweisen

Konflikte

↳ wenn mehrere User Operationen ausführen

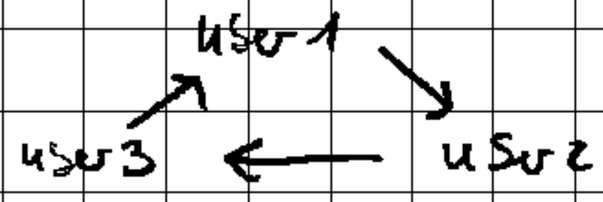
- read user 1 write user 2
- write user 1 read user 2
- write user 1 write user 2

Serialisierbarkeitsgraph

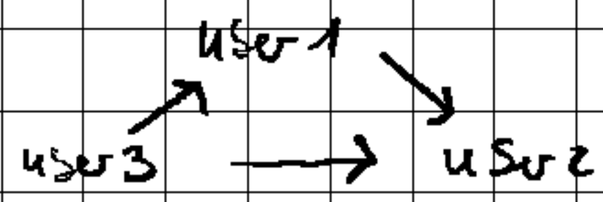
↳ User sind Knoten, Konflikte sind Kanten

z.B

nicht Serializierbar



Serializierbar



⇒ Zyklus im Graphen = nicht Serializierbar

kein Zyklus = Serializierbar

SX - Sperrverfahren

↳ Verwaltung von Datenzugriffen durch Locks:

- Shared Lock (S) → read only von mehreren
- Exclusive Lock (X) → read & write von Jemandem

Schwächen von Relationalen DBs

- viele Informationen in vielen Tabellen
  - ↳ viele Beziehungen
    - ↳ large, komplizierte Queries
- Homogenität
  - ↳ Jede Zeile hat die gleichen Spalten (horizontal homogen)
  - ↳ Jede Spalte hat den gleichen Typ Wert (vertikal homogen)
- Schwirige Verteilbarkeit auf mehrere Rechner

Herausforderungen von DBs

↳ 4 Vs

- Volume → grosse Daten
- Velocity → Real-Time Verarbeitung → viele Inserts
- Variety → Daten ohne fixes Schema
- Veracity → Unklar ob Daten korrekte oder falsche Infos beinhalten

↳ Vertikale Skallierung (scaling up)

↳ mehr Resourcen im gleiden Rechner

↳ Horizontale Skallierung (scaling out)

↳ mehr Rechner im Cluster

NoSQL

- verteilte Speicherung & Berechnung
- flexible Schemata
- nicht relational
- Unterstützung von Semi- & un-strukturierten Daten

4 Kategorien

- Key - Value - Stores → Schlüssel - Wert Paare
- Wide - Column - Stores → Spalten mit Sub-Spalten
- Dokumentendatenbanken → Kollektionen mit Dokumenten
- Graphendatenbanken → Knoten & Kanten

Vektordatenbanken

Lo Speichern Daten inform eines Vektors.

Dadurch können die Daten innerhalb des Vektorraumes eingeordnet werden.

Das ermöglicht es ähnliche Daten zu finden, da sie "nahe" bei einander liegen.

Abfrage:

- Query wird erstellt und vektorisiert (embedding)
- Datenbank vogleicht Vektor-Embedding der Anfrage mit den gespeicherten Daten und berechnet die Distanzen
- Datenbank returned sehr ähnlide Vektoren

⇛ effiziente Ähnlichkeitssuche auch bei viel Dimensionalen Werten

↳ geeignet für LLMs, MLMs, Geografische Daten