# Grundlagen

## basics

|  |
| --- |
| **Anforderungen an DBMS Lösung:** Redundanzfreiheit, Datenintegrität, Datenkonsistenz,  Datensicherheit, Datenschutz  **Wichtige Funktionen:** Transaktion, Mehrbenutzerbetrieb, Sicherheit, Backup / Recovery,  Datenkatalog, Query Sprachen  **Vorteile einer Datenbank:** Kontrolierbare Datenintegrität, Entkoppel und  Daten-Applikation, Mehrbenutzer, Kapselung der Daten, Abfrage Sprache (SQL), Sicherheit DB-Entwurfsprozess **Datenverarbeitungs-anforderungen**  **Informations**  **-anforderungen**  Anforderungsanalyse  Anforderungs  -spezifikation  Konzeptueller  DB-Entwurf  **UML-Klassendiagramm**  Text-Notation, zB. Umlet  Konzeptuelles  Datenmodell  **DBMS-Charakteristika**  Logischer DB-Entwurf  **Relationale Schreibweise**  Grafik-Notation, zB. rel. Diagramm  Logisches  Datenmodell  Physischer Entwurf  **Hardware/BS**  **DB-Schema (PG SQL)**  Relationales, DB-spezifisches Diagramm  Physisches Datenbankmodell (Schema) datenbankmodelle (paradigmen) **Hierarchisches Datenmodell (legacy)**  Struktur: Informationen in einer Hierarchie, Beziehung mit Daten gespeichert  Nachteile: «Welt» nicht immer als Hierarchie modellierbar, Keine Trennung zwischen  Anwendungen (extern) und Daten (internes Schema)  **Netzwerkmodell**  Struktur: Informationen in einer vernetzenHierarchie, Beziehung mit Daten gespeichert, Mehrfachbeziehung möglich  Nachteile: Effiziente Anwendung nur mit Kenntnis der komplexen Netzwerk-Datenstruktur  = Starke Abhängigkeit Anwendung (extern) und Daten (internes Schema)  **Relationen Modell**  Struktur: Unstrukturiere Daten (1. NF) in Tabellen (=Relationen) gespeichert,  Sehr flexibel und klare Trennung Anwendung/Daten  Nachteile: Impedence Mismatch, Performance Probleme bei komplexen Abfragen  **Objektrelationales Modell**  Struktur: Erweiterung rel. Datenmodell mit benutzerdef. Typen und Tabellenvererbung  Verschachtelte Tabellen und nebst Daten auch Methoden speicherbar |

## begriffe

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | UML-Klassendiagramm | ERM | Rel. Modell | SQL | | Objekt, Instanz | Entität | Tupel | Zeile | | Klasse, Objektmenge | Entitätmenge | Relation | Tabelle | | Attribut | Attribut | Attribut | Feld/Spalte/Kolonne | | Datentyp | Wertebereich | Wertebereich | Datentyp |   **Complete / Incomplete:** alle Unterklassen sind (nicht) definiert  **Disjunkt / Disjoint:** Instanz von genau einer Unterklasse  **Overlapping:** Instanz mehrerer Unterklassen  **Assoziation:** Eine Assoziation verknüpft mehrere (in der Regel zwei)  Entitätstypen  **Attribut:** Beschreibung eines bestimmten Merkmales (property) der  Entitäten  **Entität:** Individuelles Exemplar von Elementen der realen Welt  Beispiel: Der Angestellte Fritz Müller  **Entitätsmenge:** Menge von Entitäten, die bezüglich bestimmter  Eigenschaften als gleichartig behandelt werden.  Beispiel Menge aller Angestellter  **Entitätstyp:** Zusammenfassung von Attributen, die auf einer  Entitätsmenge definiert sind  **Kardinalität:** Die Kardinalitäten beschreiben den Typ dieser  Abbildung. ansi-3-ebenenmodell **Externe Ebene:** Sicht einer Benutzerklasse auf eine Teilmenge der Datenbank Definition  durch externes Schema  **Logische Ebene:** Beschreibt die logische Struktur der gespeicherten Daten.  Bei relationalen DBMS = Tabellen-Strukturen  **Interne Ebene:** Speicherstrukturen der physischen Datenorganisation.  Definition durch internes Schema  **Mapping:** Zwischen den Ebenen ist eine mehr oder weniger komplexe Abbildung notwendig  Diagram  Description automatically generated |

## Krähenfussnotation

One

Many



One (and only one)



Zero or one



One or many



## aggregation und komposition

**Aggregation:** “Is-part of” Beziehung

**Komposition:** Spezialform der Aggregation, welche eine existenzielle Abhängigkeit ausdrückt

1

1..\*

Raum

Gebäude

Komposition

0..\*

Vorlesung

3..\*

Student

Aggregation

## Vererbung

Wird mit einem Pfeil mit leerem Dreieick gezeichnet. Klassen von denen der Pfeil aus geht erben vom Ziel: **is-a Beziehung**

**Je eine Tabelle pro Sub- und Superklasse:**

Superklasse wird auf eine Tabelle abgebildet, welche zusätzlich ein diskriminierendes Attribut enthält, das den jeweiligen Typ der Subklasse spezifiziert.

Redundanzfrei, overlapping möglich ABER Abfragen sind komplex da viele Tabellen und ein zusätzliches Typ-Attribut notwendig.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Fahrzeug |  |  |  |
|  | **FzID** | Marke | Gewicht | FzgTyp |
|  | 1 | VW | 1100 | 0 |
|  | 2 | Merc. | 950 | 0 |
|  | 3 | Scania | 8000 | 1 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | PKW |  |  |  | LKW |  |  |
|  | ***FzgId*** |  | AnzPlaetze |  | ***FzgId*** | Ladeflaeche |  |
|  | 1 |  | 4 |  | 3 | 6 |  |
|  | 2 |  | 5 |  |  |  |  |

**Je eine Tabelle pro Subklasse:**

Subklassen werden auf eigene Tabelle abgebildet und enthalten alle Attribute.

Ermöglicht einfache Zugriffe auf Tablle ABER Semantikverlust, Overlapping nicht möglich und Schlüssel-Eindeutigkeit muss ggf. Seperat über mehrere Tabellen kontrolliert werden.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PKW |  |  |  |  | LKW |  |  |  |
| **FzID** | Marke | Gewicht | AnzPlätze |  | **FzID** | Marke | Gewicht | LadeFläche |
| 1 | VW | 1100 | 4 |  | 3 | Scania | 8000 | 6 |
| 2 | Merc. | 950 | 5 |  |  |  |  |  |

**Eine Superklasse-Tabelle:**

Superklasse-Tabelle speichert alle Attribute, auch die der Subklassen. Enthält auch ein diskriminierendes Attribut, welches den Typ der Subklasse spezifiziert.

Zugriffe sind sehr einfach ABER sehr einfach 3 NF verletzt, viele NULL Werte

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FzId** | Marke | Gewicht | FzTyp | AnzPlatz | Ladefläche |
| 1 | Vw Golf | 1100 | 0 | 4 | NULL |
| 2 | Mercedes | 950 | 0 | 5 | NULL |
| 3 | Scania | 8000 | 1 | NULL | 6 |

## anomalien

**Einfügeanomalie:** Tupel kann erst eingefügt werden, wenn andere Tupel existieren

**Löschanomalie:** Wenn Tupel gelöscht wird, gehen zusätzliche Informationen verloren

z.B: Beim Löschen von einem Mitarbeiter geht Info über Abteilung verloren

**Änderungsanomalie:** Bei Änderungen muss diese an x Orten nachgetragen werden

## Normalformen

**Grund:** Bei Redundanzen in Tabellen können Mutations-Anomalien auftreten

**1. Normalform:** Eine Tabelle befindet sich in der 1. NF, wenn die Wertebereiche der Attribute atomar sind.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| PersNr | Name |  | Persnr | Name | Vorname |
| 1001 | Marxer, Markus |  | 1001 | Marxer | Markus |

nach Anwendung der 1 NF:

**2. Normalform:** Eine Tabelle befindet sich genau dann in der 2. NF, wenn sie sich in der 1. NF befindet und jedes Nichtschlüsselattribut von jedem Schlüsselkandidaten voll funktional abhängig ist. Attribut A ist voll funktional abhängig, wenn es zu jedem A genau ein B existiert. Zusammengesetzte PK nicht erlaubt.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| PersNr | Name | Wohnort |  | ProjNr | PersNr | Name |
| 1001 | Marxer | Luzern |  | 27 | 1001 | Marxer |
|  |  |  |  | 25 | 1001 | Marxer |
|  |  |  |  |  |  |  |
|  |  |  |  | PersNr | ProjNr | Anteil |
|  |  |  |  | 1001 | 27 | 30 |
|  |  |  |  | 1001 | 25 | 20 |

**3. Normalform:** Eine Tabelle befindet sich in der 3. NF wenn sie sich in der 2. NF befindet und kein Nichtschlüsselattribut transitiv von einem Kandidatenschlüssel abhängt.

Transitive Abhängigkeit: Wenn A => B und B => C, dann folgt A => C

|  |  |  |
| --- | --- | --- |
| ProjNr | PLZ | PLName |
| 27 | 1001 | Marxer |
| 25 | 1001 | Marxer |
| 26 | 1100 | Widmer |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ProjNr | PersNr |  |  |  |  |  | PersNr | PLName |
| 27 | 1001 |  |  |  |  |  | 1001 | Marxer |
| 25 | 1001 |  |  |  |  |  | 1100 | Widmer |
| 26 | 1100 |  |  |  |  |  |  |  |

**Boyce-Codd-Normalform (BCNF):** Ist in der Relation nur ein Kandidatenschlüssel vorhanden oder es liegt keine Überlappung bei mehreren Kandidatenschlüsseln vor, befindet sich die Relation automatisch in der BCNF.

**4. Normalform:** Wenn BCNF erfüllt und nur noch einfache mehrwertige Abhängigkeiten in der Tabelle sind, d.h. nicht mehrere Relation 1:n oder m:n - Beziehungen zu einem Schlüsselwort.

**5. Normalform**: 4 NF erfüllt und die Tabelle sich nicht aus einfacheren Tabellen durch Verbundoperationen erschliessen lässt.

# Data Definition language (ddl)

## Create table

CREATE TABLE branch (  
 branch\_id SERIAL PRIMARY KEY NOT NULL,  
 street\_name VARCHAR(50) NOT NULL,  
 street\_nr INTEGER NOT NULL,  
 plz INTEGER NOT NULL  
);

## constraints / ref. integrity

ALTER TABLE employee  
 ADD CONSTRAINT fk\_person  
 FOREIGN KEY (*person\_id*)  
 REFERENCES person (*person\_id*)  
 ON DELETE CASCADE;  
ALTER TABLE employee  
 DROP CONSTRAINT fk\_person;

**PRIMARY KEY:** Attribut als PK und immer UNIQUE und NOT NULL

**NOT NULL:** Eintrag müssen einem vordefiniertem Wert entsprechen **UNIQUE:** Werte aller Tupel in einer Tabelle mit diesem Tag müssen einzigartig sein. UNIQUE definiert einen Sekundärschlüssel, wenn es mit NOT NULL zusammen verwendet wird. Mit NOT NULL besagt es, dass alle Werte ausser NULL definiert sein müssen

**REFERENCES:** Attribut gilt als Fremdschlüssel

**Default:** Setzt einen Wert der gilt, falls beim INSERT Command kein Tupel eingefügt wird

**CHECK**: Ermöglicht weitere Einschränkungen hinzuzufügen:

CREATE TABLE persons (  
 id INTEGER PRIMARY KEY NOT NULL,  
 lastname VARCHAR(255) NOT NULL,  
 age INTEGER;  
 CHECK (age >= 18)  
);

**ON DELETE CASCADE:** Tupels in B die eines der zu löschenden Tupels in A referenzieren, werden auch gelöscht.

**RESTRICT:** Falls ein Tupel in B eines der zu löschenden Tupels in A referenziert wird die Löschoperation nicht ausgeführt.

**SET NULL:** Fremdschlüsselattribut aRef eines Tupels aus B wird NULL gesetzt falls das referenzierte Tupel in A gelöscht wird.

**SET DEFAULT:** Fremdschlüsselattribut aRef eines Tupels aus B wird auf Defaultwert gesetzt, falls das referenzierte Tupel in A gelöscht wird.

**NO ACTION:** Fremdschlüssel abhängige Tupel werden nicht gelöscht

# Data manipulation language

## group by / having / order by

**Group By:** Gruppierung einer Ergebnismenge mit (SUM,COUNT, MAX etc). Die Gruppierung fasst mehrere Tupel der Ausgangstabelle zu einem einzigen Tupel im Ergebnis zusammen

SELECT *COUNT*(*\**) FROM branch GROUP BY country;

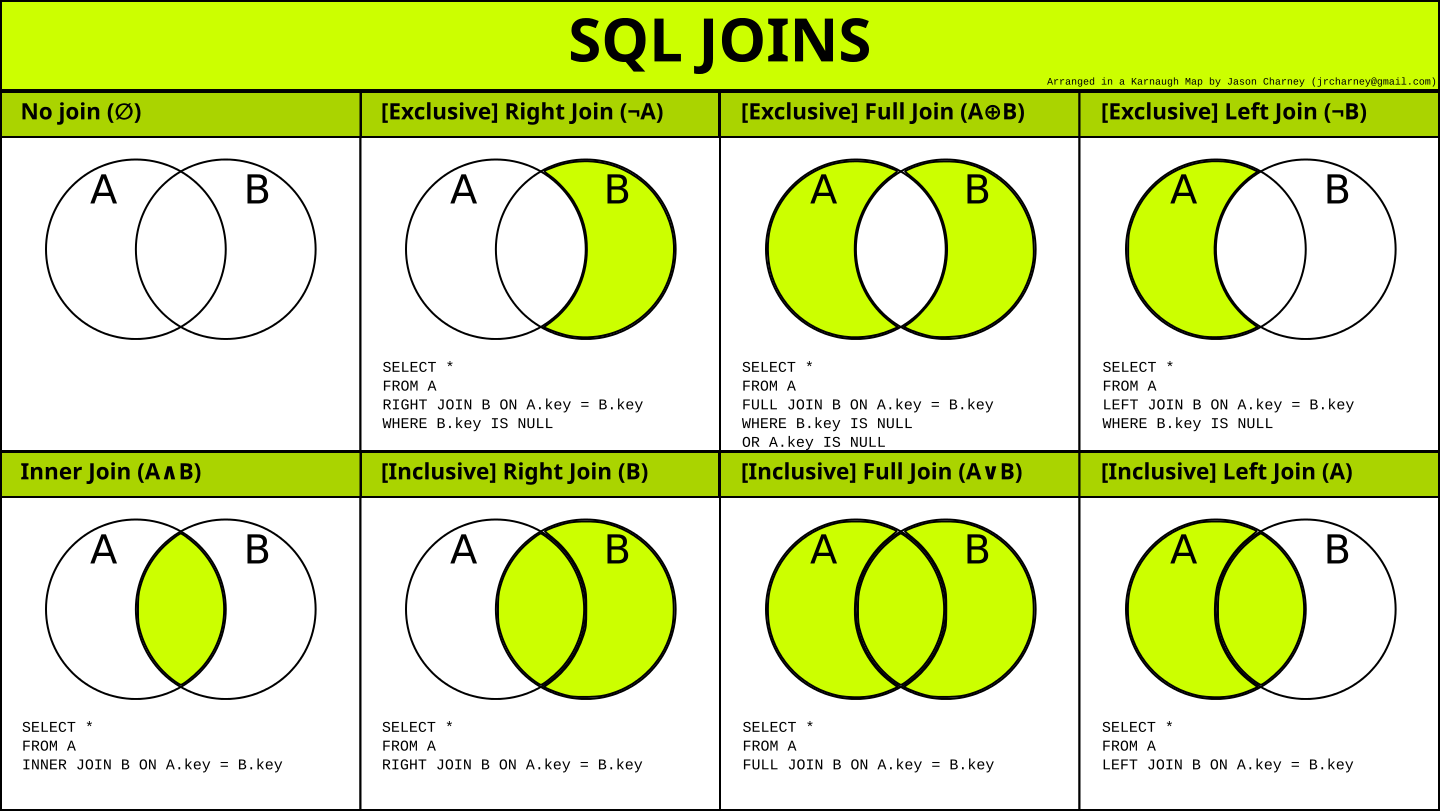
**Having:** Having wird nach GROUP BY benutzt um die Menge die durch GROUP BY entstehen weiter zu filtern.

SELECT *\** FROM Ang GROUP BY *Abt* HAVING *COUNT*(*\**)>5;

**Order By:** Sortiert Resultate nach Attribut auf- oder absteigend. Kann auch Reihenfolge der Kolumnen ändern

ORDER BY 2 DESC 1, 3

## joins



**Theta Join:** Join mit willkürliche Vergleichsoperatoren (zB. ≥)

**Equi-Join:** Theta Join mit einem =

**Natural Join:** Join mit Vergleich auf Kolumnenname (Bad practice)

Natural-Join

Outer-Join

Equi-Join

Theta-Join

**Lateral Join:** Im Query den Inhalt einer Tabelle mit Angaben aus einer anderen Tabelle ergänzen.FROM PERSON AS p  
INNER JOIN p ON p.*ID* = CUSTOMER.PERSON  
JOIN LATERAL (  
 SELECT *POINTS* FROM LOYALTY\_CARD AS lc  
 WHERE lc.*BELONGS\_TO\_PERSON* = CUSTOMER.PERSON  
) AS POINTS ON TRUE;

## aggregat- / skalarfunktionen

|  |  |
| --- | --- |
| **Aggregat** | **Skalar** |
| AVG()  COUNT()  FIRST()  LAST()  MIN()  MAX()  SUM() | UCASE()  LCASE()  MID()  LEN()  ROUND()  NOW()  FORMAT() |

## like

Alle Mitarbeiter mit Wohnort beginnend mit ‚LU‘:

SELECT *id* from employees WHERE *wohnort* LIKE 'LU%'

Alle Namen mit mehr als 2 Zeichen:

SELECT *id* from employees WHERE *wohnort* LIKE '\_\_%'

## subqueries

Eine Unterabfrage kein ein Suchargument aus der äusseren Abfrage enthalten, sie wird dann als korreliert bezeichnet. Ansonsten als unkorreliert.

SELECT mat.*material\_nr*,   
 mat.*materialname*FROM material AS mat  
WHERE mat.*material\_nr* NOT IN (  
 SELECT matz.*material\_nr* FROM materialzuteilung AS matz);

|  |  |
| --- | --- |
| IN | WHEREexpression IN(subquery) |
| EXISTS | WHEREEXISTS(subquery) |
| ANY | WHERE expression operatorANY (subquery) |
| ALL | WHERE expression operatorALL (subquery) |

## views

Ermöglicht verschiedene Ansichten auf die Daten. Mit Views kann man Abfragen einfacher formulieren, da diese sich oft über mehrere Tabellen erstrecken. Irrelevante Daten können für bestimmte Benutzer versteckt werden.

CREATE VIEW branchInfo AS   
 SELECT branch.city,  
 branch.street\_name,  
 person.first\_name,  
 person.last\_name  
FROM branch

**Materialized View: G**espeicherte Views, die nicht automatisch aktualisiert werden

**Updatable View (Rule):** Können nur 1 Attribut im View enthalten, Kein DISTINCT, GROUP BY, Gruppen Funktionen (min, max)

CREATE RULE KURS\_UPDATE AS ON UPDATE TO KURSÜBERSICHT  
 DO INSTEAD UPDATE KURS  
 SET *BESSCHREIBUNG* = *"neue beschreibung"* WHERE *KURSNUMMER* = *"neue kursnummer"*;

**Row-Level Security (RLS):** Eine Art «System-Views», nur User mit entsprechendem Lese- und Schreibrechte (Policy)

**Temporary Table:** Am Ende der Session/Transaction gedroppt, Andere “permanente” Tabellen mit gleichem Namen nicht sichtbar

## window function

SELECT *persnr*, *abtnr*, *MAX*(*salaer*)  
OVER (PARTITION BY *abtnr*)

FROM angestellter

SELECT  
 product\_name,  
 price,  
 group\_name,  
 *AVG* (price) OVER (  
 PARTITION BY group\_name  
 )  
FROM products

**row\_number():** number of the current row within its partition

**rank():** rank of the current row with gaps

**percent\_rank():** evaluate the relative standing of a value within a set of values

**ntile(num buckets integer):** integer ranging from 1 to the argument val

**lag(value any):** value at the row that is offset rows before the current

**lead(value any):** value at row that is offset rows after the current

**first\_value(value any):** value at the row that is the first of the window frame.

## common table expression (cte)

Mit CTE kann zeilenweise ausgegeben werden. CTE erlaubt Abfragen quasi als Parameter und können rekursiv sein. Existieren nur während dem Statement.

Sämtliche Untergebene von Steiner (1010) finden:

WITH RECURSIVE unter (name, persnr, chef) AS  
 (SELECT A.*name*, A.*persnr*, A.*chef* FROM angestellter A  
 WHERE A.*chef* = 1010  
 UNION ALL  
 SELECT A.*name*, A.*persnr*, A.*chef* FROM angestellter A  
 INNER JOIN unter B ON B.persnr = A.*chef* )  
SELECT *\** FROM unter ORDER BY chef, name;

## Casting

**Methoden:**

SELECT ID,  
 BIRTHDAY AS BDAY\_YMD,  
 *TO\_CHAR*(BIRTHDAY, 'dd.mm.yyyy') AS BDAY\_DMY,  
 *AGE*(*NOW*(), BIRTHDAY) AS AGE,  
 *DATE\_PART*('year', *AGE*(*NOW*(), BIRTHDAY))  
FROM (  
 VALUES (1, DATE '1954-01-01'),  
 (1, DATE '1961-05-11'),  
 (1, DATE '1971-12-31')  
) AS TMP(ID, BIRTHDAY);

**Operator:**

Casting von Attributen mit :: Operator (betrag ist Integer):

betrag::VARCHAR(20)

## mengenoperationen

UNION / UNION ALL : Kombiniert Tabellen und löscht Duplikate

EXCEPT (MINUS): Differenz zwischen zwei Tupel

INTERSECT: Schnitt zweier Tupel

# **relationales algebra**

**Operation Projektion (π PI):** Auswahl von Kolonnen einer Relationstabelle

π(A1, A2) = Select A1, A2 from Table;

**Operation Selection (σ Sigma):** Auswahl von Zeilen einer Tabelle aufgrund von Kriterien, alle Zeilen die der Selektion nicht entsprechen, werden gestrichen.

σ(A1 > 30) = Select \* from Table where A1 > 30;

**Operation Umbennung (ρ Rho):** Umbennenung ρ von Relationen

**Kartesisches Produkt:** Seien zwei Relationen R1(A1, … An) und R2 (B1, … B2), dann ist das kartesische Produkt die Menge aller Paare aus Tupeln der ersten Relation R1 verknüpft (= multipliziert) mit der Menge aller Tupel der zweiten (R2).

R1 x R2 :={ (a1, …, an, b1, …, bn) | a in R1, b in R2 }

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| R1 |  | R2 |  |  |  | R1 | R2 |
| 77 |  | 11 |  | Kart. Produkt |  | 77 | 11 |
| 25 |  | 22 |  | => |  | 77 | 22 |
| 13 |  |  |  | (R1 x R2) |  | 25 | 11 |
|  |  |  |  |  |  | 25 | 22 |
|  |  |  |  |  |  | 13 | 11 |
|  |  |  |  |  |  | 13 | 22 |

**Theta-Verbund (Θ)** Allgemeiner Verbund:

R [a Θ b] S := { < r1, ... rm, s1, ..., sn > ¦ r[a] r[b] }

Wobei Θ für = , > , < , <> , <= oder >= stehen kann

zB.: SELECT *\** FROM R JOIN S ON R.a < S.b

**Equi Join**: Theta-Join mit Gleichheitsoperation

R [a = b] S := { < r1, ... rm, s1, ..., sn > ¦ r[a]=r[b] }

zB.: SELECT *\** FROM R JOIN S ON R.a = S.b;

|  |  |
| --- | --- |
| ⋈ | Natural Join |
| ⟕ | Left Outer Join |
| ⟖ | Right Outer Join |
| ⟗ | Outer Join |
| ⋉ | Semi-Join |
| ▷ | Anti-Join |

# **data control language (dcl)**

## sicherheit

**System-Sicherheit:** Authentisierung mit Login, Autorisierung auf Tabellen durch Rechte, Kontrolle vom System-Ressourcen, Auditing, Transportsicherheit

**Daten-Sicherheit (Datenschutz DBMS):** Zugriffskontrolle zur Verhinderung von nicht autorisierten Zugriffen auf Datenobjekten, Auditing

**Massnahmen gegen Exploits:** User Input prüfen und escapen, Datenzugriff abstrahieren (Stored Procedures), Keine Superuser anlegen

## rechte vergeben

**Rolle erstellen / löschen:**

CREATE ROLE username WITH LOGIN PASSWORD 'pw';

DROP ROLE username;

**Privilege:** Wenn ein User ein Datenbankobjekt (Schema, Tabelle, Funktion) erstellt dann hat nurer selbst und Admin user Zugriff auf diese. Um andern user Zugriff zu erteilen muss erdiesen explizit mit GRANT geben.

GRANT INSERT, SELECT ON TABLE BRANCH TO USERNAME;

**Grant:** Grant kann Rechte für Table, column, view, sequence, database, function, schema etc. vergeben.

**Revoke:** Wenn P1 P2 Rechte gibt und die Rechte von P1 werden entzogen, verliert P2 alle Rechte, die er von P1 erhalten hat.

REVOKE ALL ON branch FROM username;

# **transaction**

|  |  |
| --- | --- |
| **A**tomicity | Vollständig oder gar nicht durchgeführt |
| **C**onsistency | Von einem konsistenten Zustand in einen andern,  Vor und nach der Transaktion sind die Daten konsistent |
| **I**solation | So ausführen als ob isoliert  Transaktionen beinflussen sich gengenseitig nicht |
| **D**urability | Auswirkungen auf Dateien bleiben dauerhaft bestehen und nicht durch Fehler verloren |

**Fault Tolerance:** Bei Servercrash Operation wiederholen oder rückgängig gemacht

**Concurrency:** Isolation der Transaktion, Paralellität wird ermöglicht durch Verzahnung

**Resultate von COMMIT:**

**Success:** Alle Änderungen atomar und durable gespeichert

**Failure:** Alle (temporären) Änderungen verworfen

## serialisierung

|  |  |  |
| --- | --- | --- |
|  | **RA(x)** | **WA(x)** |
| **RB(y)** | Kein Konflikt | Konflikt  Wenn x == y |
| **WB(y)** | Konflikt  Wenn x == y | Konflikt  Wenn x == y |

A screenshot of a computer

Description automatically generated with low confidence

Shape, arrow

Description automatically generatedDiagram, text

Description automatically generated with medium confidenceWenn der Serialisierbarkeitsgraph keinen Zyklus aufweist, kann der Schedule serialisert werden. Commit Reihenfolge wird auch beschränkt, da ein Commit nur möglich ist, falls keine Konfliktkante von andere nicht commited Transaktionen kommt.

## isolation levels

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Isolation Level** | **Dirty Read** | **Fuzzy Read** | **Phantom Read** | **Read Skew** | **Write Skew** | **Deadlock** | **Casc.**  **Rollback** |
| **Read uncomm.** | Möglich | Möglich | Möglich | Möglich | Möglich |  |  |
| **Read**  **comm.** | Unmöglich | Möglich | Möglich | Möglich | Möglich |  |  |
| **Repeatable Read** | Unmöglich | Unmöglich | Möglich | Unmöglich | Möglich | Möglich |  |
| **Serializable** | Unmöglich | Unmöglich | Unmöglich | Unmöglich | Möglich | Unmöglich | Unmöglich |

**Dirty Read:** Lese Daten von anderer, nicht committed Transaktionen

**Fuzzy Read:** Lese gleiche Daten mehrmals, sehe andere Werte. Gelesene Daten ändern sich plötzlich durch Commit von anderen nebenläufigen Transaktionen

→ inconsistent Analysis Problem

**Phantom Read:** Select in Tabelle, entdecke plötzlich neue / gelöschte Rows durch Commit in andere Transaktion (tritt in Postgres nicht auf wegen Snapshot Isolation)

**Read Skew:** Updates von T1 können von T2 gelesen warden, bevor T1 commited hat, somit wird Konsistenz gebrochen

**Write Skew:** T1 und T2 überschreiben das gleiche Feld während beide T noch aktiv sind.

**Lost Update:** Wenn 2 Transkationen das gleiche Feld überschreiben, geht eins verloren.

**Isolation Level pro Transaktion:**

BEGIN TRANSACTION ISOLATION LEVEL ...;

**Isolation Level pro Session:**

SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL ...;

## locking (sperrprotokolle)

Locks auf verschiedene Granulitäten (Table, Table Range, Row, Item). Desto kleiner, desto grössere Parallelität. Psql meist ROW

**Pessimistisches Verfahren:** Sperren (Locking) jedes DB-Objekt vor Zugriff. Sperrprotokolle können Seriablisierbarkeit garantieren. Two Phase Locking, Strict Two Phase Locking, Preclaiming Two Phase Locking

**Optimistisches Verfahren:** Vermeide Konflikte nicht, sondern erkenne sie nur. Commit prüfen ob Änderung okay sonst Rollback.

MVVC, Snapshot Isolation

**X-Lock:** Für Write/Read-Zugriffe, Nur 1 X-Lock pro Transaktion pro Objekt

**S-Lock:** Für Read-Zugriffe, Mehrere Transaktion ein S-Lock auf Objekt

**Details:** Bei Update werden Tupel mit xLocks versehen (Deadlocks möglich). Keine Locks beim Lesen. Schreiber blockieren keine Leser, aber Schreiber blocken Schreiber.

**Two-Phase Locking (2PL):**

1. Phase (Growing Phase): Jedes Objekt vor Zugriff in Transaktion gesperrt (x-/s-Lock)

2. Phase (Shrinking Phase): Sobald Transaktion Lock freigibt, keine neuen Locks erlaubt

**Strict Two-Phase Locking (S2PL):**

Alle Locks warden erst nach Ende der Transaktion freigegeben

Vorteile: Keine Cascading Rollbacks und unklarer Beginn der Shrinking Phase

Nachteile: Deadlocks noch möglich und kann Parallelität unnötig einschränken

**Preclaiming Two-Phase Locking:**

Tranaktion muss alle Locks auf einmal (atomar) am Anfang belegen

Vorteile: Garantiert Deadlock-Freiheit

Shape, rectangle

Description automatically generatedShape

Description automatically generated with medium confidenceShape

Description automatically generatedNachteil: Transaktion muss wissen, welche Zugriffe/Sperren notwendig sind = unrealistisch

Two-Phase Locking Strict Two-Phase Locking Preclaiming Two-Phase Locking

**Deadlock Erkennung/Auflösung:**

**Behebung:** Transaktionasabbruch nach max. Wartezeit (Poor mans’s solution), Abbruch einzelner Transaktion bei Deadlock

Diagram

Description automatically generatedDiagram

Description automatically generated**Erkennung:** Analyse mit Betriebsmittelgraph und Wartegraph

**Auflösung:** Scheduler erkennt Zyklus und sucht Opfer (Transaktion),

Minimierung der Rücksetzung (T mit min Sperren),

Maximierung der Ressourcen (T mit max Sperren),

Vermeidung von Starvation durch verschiedene T rollbacken

**Snapshot Isolation (SI):**

Jede Transaktion sieht Snapshot zu seinem Start-Zeitpunkt. Bei Änderungen im Commit prüfen, dass Objekte unverändert wie zum Snapshot-Zeitpunkt sind, sonst Rollback.

Vorteile: Effiziente Implementation, «Lesen blockiert Schreiben und umgekehrt nicht»

Nachteile: Keine Garantie für Serialisierbarkeit und Write Skew/Read-only Transaction möglich

Timeline

Description automatically generated

**Multiversion Concurrency Control (MVCC):**

Mehrere Versionen xt pro DB-Objekt x. Jede Transaktion T hat Timestamp TS(T) des Startzeitpunkts.

Schreiben: Erzeuge neue Version xt mit t = TS(T)

Lese: Sehe letze Version xt mit grösstem t ≤ TS(T)

Bei Updates von Tupel werden xLocks gesetzt, keine Locks beim Lesen. DBMS verwaltet mehrere Versionen eines Tupels. «Schreiber blocken keine Leser und umgekehrt».

In Read Commited liest Statement nur Daten mit selben Version. In Repeatable Read oder Serialisable list die ganze Transaktion nur Daten mit derselben Version.

Diagram

Description automatically generated with medium confidence

## isolationlevelvergleich

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Garant.  Serializability | Keine Deadlocks | Keine Cascading Rollbacks | Keine Konflikt-Rollbacks | Hohe Parallelität | Realistisch (ohne Voranalyse) |
| Two-Phase Locking | **✓** | **✕** | **✕** | **✓** | **✕** | **✕** |
| Strict 2PL | **✓** | **✕** | **✓** | **✓** | **✕** | **✓** |
| Preclaiming 2PL | **✓** | **✓** | **✓** | **✓** | **✕** | **✕** |
| Validation-Based | **✓** | **✓** | **✕** | **✕** | **✓** | **✓** |
| Timestamp-Based | **✓** | **✓** | **✕** | **✕** | **✓** | **✓** |
| Snapshot-Isolation | **✕** | **✓** | **✓** | **✕** | **✓** | **✓** |
| SSI | **✓** | **✓** | **✓** | **✕** | **✓** | **✓** |

## write ahead log (wal)

Alle Änderungen von Transaktionen zuerst in Log festhalten und dann ausführen. Nach einem Absturz können alle committed Transaktionen replayed oder undoed werden.

**Struktur:** [LSN, TransaktionsID, PageID, Redo, Undo, PrevLSN]

**LSN (Log Sequence Number):** Eindeutig, monoton ansteigend

**TransaktionsID:** Transaktionskennung TA der Transaktion, die die Änderung durchgeführt hat.

**Page:** Die Kennung der Seite, auf der die Änderungsoperationen vollzogen wurde.

**Redo:** Information gibt an, wie die Änderung nachvollzogen werden kann. Absoluter Wert nach der Änderung.

**Undo:** Information beschreibt, wie die Änderung rückgängig gemacht werden kann. Absoluter Wert vor Änderung.

**PrevLSN:** Einen Zeiger auf den vorhergehenden Log-Eintrag der jeweiligen Transaktion. Für effizientes Undo(Rollback)

# **index**

CREATE INDEX indexname ON tablename (*columnname*)

**B-Tree:** Geeignet für Hintergrundpseicher/Index-Bedürnisse, Fast optimal für Query/Insert

**HASH:** Ordnet Keys zu Records, Keine NULL-Werte, Overflow möglich

**GIST:** Baumartige Struktur, Für geometrische Datentypen (Map)

**GIN:** Liste von Worten auf Dokumente zeigen, Schneller Zugriff

**BRIN:** Min/Max Werte als Block, Gut für Range Search

**ISAM:** Historisch zuerst, Schnell Query/Insert, Aktualisieren schlecht (nicht in PG)

**Partieller Index:** Index welcher mit einem WHERE limitiert wird

**Funktionaler Index:** Index mit Funktionsausdruck

**Zusammengesetzte Index:** Index über mehrer Kolumnen

## b-tree

CREATE INDEX index ON table USING *BTREE*(*column*)

**B+ Bäume:** Bäume die Elemente ausschliesslich in den Blättern speichern. Blätter sind verkettet zB. Als LinkedList

**B\* Bäume:** Bäume deren Knoten min. zu 2/3 gefüllt sein müssen.

**Einfügen:**

1. Führe eine Suche nach dem Schlüssel durch; diese endet an einem Blatt an der Einfügestelle

2. Füge den Schlüssel dort ein

3. Ist der Knoten (Blatt) überfüllt? teile ihn wie folgt:

1. Lege einen neuen Knoten an und belege ihn mit den Schlüsseln, die rechts vom mittleren Eintrag des überfüllten Knotens liegen

2. Verwende als Vaterknoten den nächst kleineren (oder niedrigsten) des überfüllten Knotens ein

3. Verbinde den Verweis rechts des neuen Eintrags im Vaterknoten mit dem neuen Knoten («Rechtsbewegung»)

4. Ist der Vaterknoten (jetzt) überfüllt?

1. Handelt es sich um die Wurzel, lege eine neue Wurzel an (Baum wächst)

2. Wiederhole den Schritt «Knoten überfüllt?» oben mit dem Vaterknoten

**Löschen:**

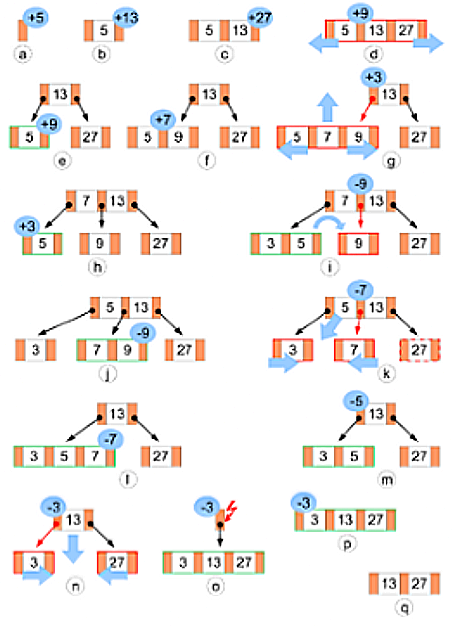
1. Führe eine Suche nach dem Schlüssel durch; diese endet an der Löschstelle falls er gefunden wird, ansonsten => Abbruch.

2. Liegt Schlüssel auf einem Blatt, kann er entfernt werden.

3. Liegt der Schlüssel auf einem Knoten und [m/2] wird unterschritten Unterlauf: Es muss ein Ersatz-Knoten vom „Bruder“ vom Vater geholt werden («Linksbewegung»).

4. Ist der Bruderknoten jetzt unterlaufen, muss der Vater-Knoten-Schlüsselheruntergeholt und die Brüder verschmolzen werden (=> merge): Lösche leeren Bruder-Knoten. Handelt es sich um die Wurzel, bestimme neue Wurzel (Schrumpfung).

5. Ist der Vaterknoten jetzt unterlaufen? Wiederhole Schritt 3 mit Vaterknoten.



# **backup and recovery**

**Logischer Backup:** SQL Dump, Blockiert keine schreibende oder lesende T, für mittelgrosse Datenmengen

**Physischer Backup:** File System Ebene, DB muss gestoppt werden, schneller als log. Backup

**Cloud Backup:** unterschiedliche Angebote

**Loosers:** Transaktionen die zum Zeitpunkt des Absturzes noch aktiv → UNDO

**Winners:** Transaktionen die nach dem Checkpoint Okay → REDO

**Klassifikation 1:** Lokale Fehler einer Transaktion → Rollback oder lokales undo

**Klassifikation 2:** Hauptspeicherverlust → Globales Undo und Redo

**Klassifikation 3:** Hintergrundspeicherverlust → Backup und Log File Techniken

## recovery tools

**Verlust der DB:** Rückspielen des letzten Backups und Redo der Transaktionen seit dem letzten Backup mit dem Log-File

**Verlust des Log-Files:** Sofortiger Backup der DB

# **optimierungen**

Indexe werden bei Spalten verwendet, die häufig in Abfragen (WHERE) vorkommen, viele unterschiedliche Werte haben und in JOINS vorkommen.

Weniger geeignet sind Indexe bei Spalten, die sich nicht häufig ändern, wenig unterschiedliche Werte haben (Boolean, Enum) und viele Daten auf einmal eingefügt werden da es schneller ist, den Index zu löschen und später neu zu erzeugen.

Text

Description automatically generated with medium confidence

**Logische Optimierung:** Abfrage ändern, sodass bei einem gleichen Resultat effizienter berechnet werden kann. Zwischenergebnisse klein halten (WHERE-Statement)

**Physische Optimierung:** Effiziente Algorithmen auswählen, um die Operationen der relationen Algebra zu implementieren, Für jeden Ausführungsplan gibt es Kommunikations-, Berechnungs-, I/O-, Speicherkosten

**Planer/Optimierer:** Erzeugung des optimalen Ausführungsplan durch Generierung aller möglichen Ausführungspläne (versehen mit Kosten) und Wahl des «günstigsten»

**Table Scans:**

|  |  |
| --- | --- |
| Full Table Scan | Heisst in Postgres Seq Scan |
| Index Scan | Falls es in WHERE ein Attribut ist, zu dem es ein Index gibt |
| Bitmap Index Scan | Ein «Bitmap Index/Heap Scan» lädt alle Tupel-Zeiger auf einmal aus dem Index, erzeugt ad-hoc einen internen Bitmap-Index, um sie dann im Hauptspeicher zu sortieren, und lädt die Tabellen-Tupel entsprechend der physischen Speicherreihenfolge |
| Index Only Scan | Falls alle Attribute der Projektion bereits im Index vorhanden  Wegen MVCC müssten alle Datenblocks konsultiert werden ausser «Visibility Map»-Bit ist gesetzt in der Indexstruktur |

**Join Strategien:**

|  |  |
| --- | --- |
| Nested loop Join | Rechte Tabelle wird in jede Zeile der linken gescannt  Einfach, kann aber zeitaufwändig sein |
| Merge Join | Tabelle werden nach Join-Attributen sortiert  Tabellen werden parallel gescannt |
| Hash Join | Rechte Tabelle in Hash-Tabelle, Join-Attribut als Hash-Key  Link Tabelle gescannt, jede gefundene Zeile als Hash-Key |

**Explain:** Gibt geplanten Ausführungsplan aus ohne Query auszuführen

**Explain Analyze:** Führt Query aus und gibt den Plan mit effektiven Ausführungskosten aus

# **the big d-dictionary**

**Daten:** Maschinell verarbeitbar, wahr/falsch, an Datenträger gebunden

**Datenbank:** Organisierte Kollektion von Daten und elektronisch zugreifbar

**Datenbankapplikationen:** Oracle, MySQL, Microsoft Access

**Datenbankschema:** Logische Konfiguration einer relationalen Datenbank

**Datenbanksystem (DBS):** Besteht aus einem DBMS und einer/mehrere DB

**Datenbasis:** Strukturiere Sammlung von Daten

**Datenkatalog:** Aus Metadaten bestehendes Datenverzeichnis und enthält Informationen über die Namen von Tabellen, deren Attributen, Datentypen, Wertebereiche und Indizes

**Datenkonsistenz:** Logische Widerspruchfreiheit

**Datensicherheit:** Sicherung vor phyischem Verlust

**Datenschutz:** Schutz vor unberechtigtem Zugriff

**Data Definition Language (DDL):** Für die Definition der logischen und externen Sichten

**Data Manipulation Language (DML):** Für die Abfrage und Modifikation der Datenbestände

**Data Query Language (SQL):** Für die Anpassung der Schemas

**Data Control Language (DCL):** Für die Verwaltung der Datenbank und für die Kontrolle von Transaktionen sowie für die Vergabe von Benutzerrechten

**COALESCE:** Ersetzt NULL durch den nächsten Parameter, der nicht Null ist

**Impedence-Mismatch:** Datentyp mismatch (Datentyp in Java/SQL stimmt nicht überein)

**Referentielle Integrität (RI):** Keine Referenz auf nicht existierende Primärschlüssel

**Surrogate Key: E**in künstlich erzeugter Key, der nichts mit der realen Welt zu tun hat. (28 ist Wisconsin)

**Truncate:** Löscht alle Tupel von einer Tabelle

# **beispiele**

## queries

*-- Select with Join as subquery*

SELECT "name", mgs.email, betrag  
FROM mitgliedschaft AS mg  
JOIN (SELECT *id*, "name", 20 AS betrag FROM einzelmitglied)

AS em ON em.id = mg.*id;*

*-- Create View*

DROP VIEW IF EXISTS AktuelleLeiter;  
CREATE VIEW AktuelleLeiter (leiter\_id, vorname, nachname) AS  
 SELECT *leiter\_id*,  
 *vorname*,   
 *nachname* FROM leiter;

## UML-Klassendiagramm

-Eine Reise hat eine ReiseID, eine Beschreibung, eine Dauer (in Tagen) und einen Preis (in Schweizer Franken, CHF).

- Eine Reise wird pro Jahr mind. einmal durchgeführt. Falls eine Reise mehrfach durchgeführt wird, dürfen sich die Zeitdauern der einzelnen Durchführungen nicht überschneiden.

- Ein Kunde (Name, Vorname) bucht eine oder mehrere Reisen zum gewünschten Datum. Eine Reise kann von beliebig vielen Kunden gebucht werden. Bei Überbuchung selektioniert das Datum der Anmeldung.

- Eine bestimmte Reise wird von einem Reiseleiter (Name, Vorname, Anstellungsdatum) geführt. Ein Reiseleiter kann beliebig viele Reisen desselben Typs führen. Falls eine Reise mehrfach durchgeführt wird, können die einzelnen Reisen vom gleichen oder von verschiedenen Reiseleitern geführt werden. Der Reiseleiter darf in der Regel nur eine Reise leiten, wenn er für diese Reise qualifiziert ist. Dazu muss er ein Assessment ablegen. Gespeichert wird das Datum des letzten erfolgreich abgelegten Assessments.

- Optional: Ein Reiseleiter kann auch als Kunde an Reisen teilnehmen. Personendaten müssen redundanzfrei erfasst werden..

Diagram

Description automatically generated

## konzeptionelles modell

- Jede Person wird identifiziert durch ihren Namen und ihre Adresse.

- Es können nur Bücher bestellt werden.

- Ein Buch besitzt einen Titel und eine ISBN-Nummer und einen Preis.

- Das Buch wurde von einer Person, dem Autor geschrieben.

- Personen können Bestellungen aufgeben.

- Bei den Bestellungen wird das Bestelldatum erfasst.

- Eine Bestellung kann beliebig viele Bücher umfassen. Pro bestelltes Buch werden der ak-tuell gültige Preis, sowie die Anzahl Exemplare gespeichert.

Diagram

Description automatically generated

## Logisches Modell

|  |  |
| --- | --- |
| Leiter (  leiter\_id SERIAL,  vorname VARCHAR(20),  nachname VARCHAR(20),  tel VARCHAR(10),  email VARCHAR(50) UNIQUE,  wohnort VARCHAR(50)  ) | Material (  material\_nr INTEGER,  materialname VARCHAR(255),  kondition ENUM('sehr gut', 'gut', 'abgenutzt', 'kaputt')  ) |
| Ausleihe (  ausleihe\_id INTEGER,  status ENUM('verfügbar', 'ausgeliehen'),  rückgabedatum DATE,  *leiter\_id* INTEGER NOT NULL REFERENCES Leiter,  )  Zelt (  anzPers INTEGER,  vorzelt BOOLEAN,  *material\_nr* INTEGER NOT NULL REFERENCES Material  ) | Materialzuteilung (  *ausleihe\_id* INTEGER NOT NULL REFERENCES Ausleihe,  *material\_nr* INTEGER NOT NULL REFERENCES Material,  ) |
|  |  |