Wiederholung

→ Primärschlüssel

→ Fremdschlüssel

→ Referenzielle Integrität

→ Atomarität

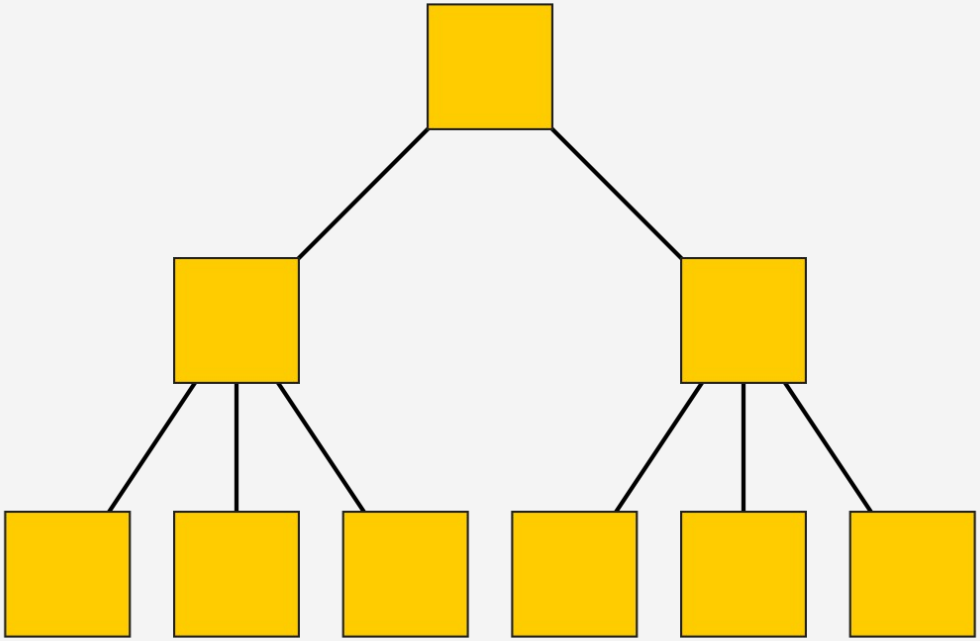
Datenmodelle

→ Hierarchisch (historisch)

→ Netz (historisch)

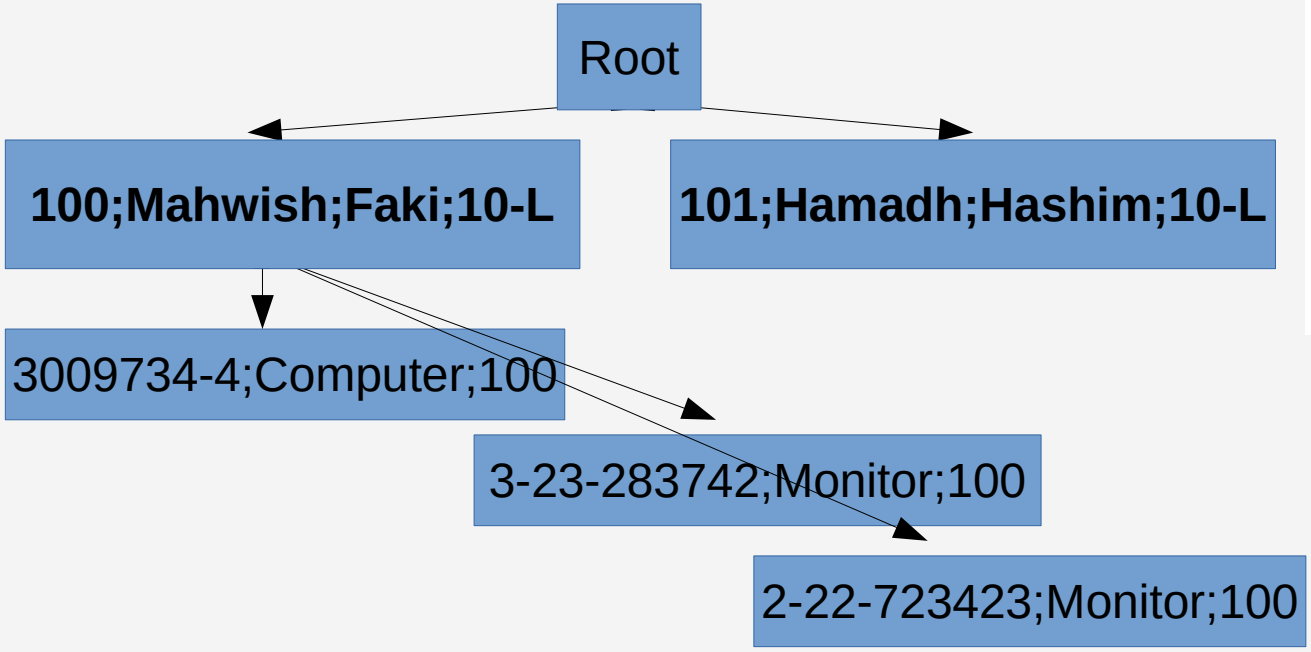
→ Relational (aktuell)

Hierachisches Datenmodell



Ein typischer Baum

Hierachisches Datenmodell



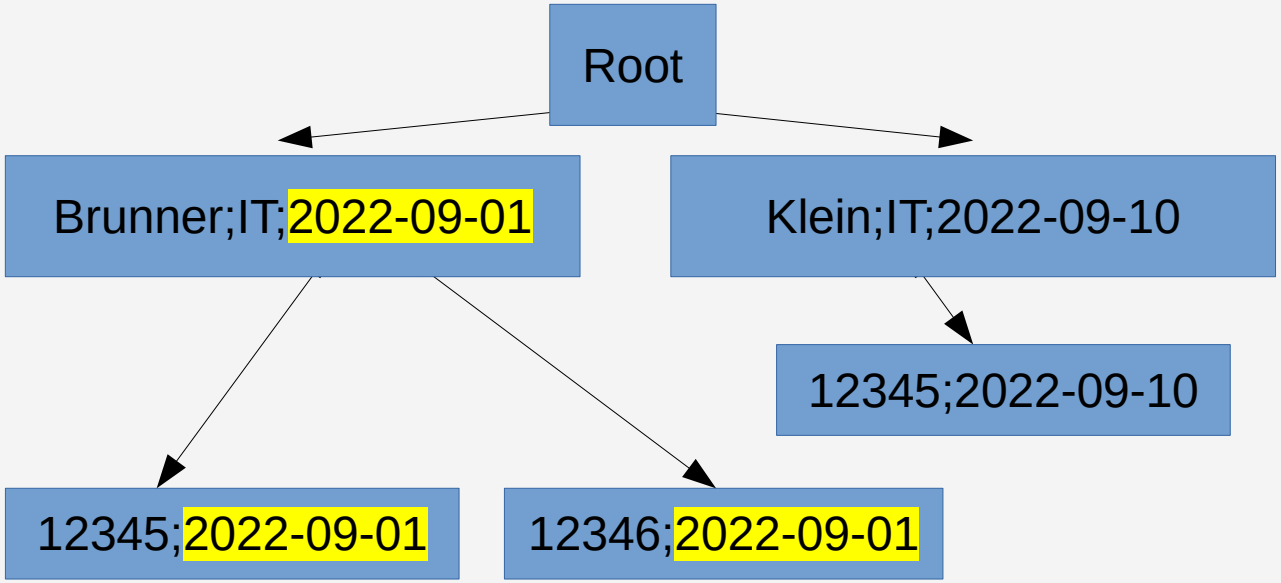
`employee` table

EmpNo	First Name	Last Name	Dept. Num
100	Mahwish	Faki	10-L
101	Hamadh	Hashim	10-L
102	darshan	Ar	20-B
103	Chaaya	Sandakelum	20-B

`computer` table

Serial Num	Type	User EmpNo
3009734-4	Computer	100
3-23-283742	Monitor	100
2-22-723423	Monitor	100
232342	Printer	100

Hierachisches Datenmodell

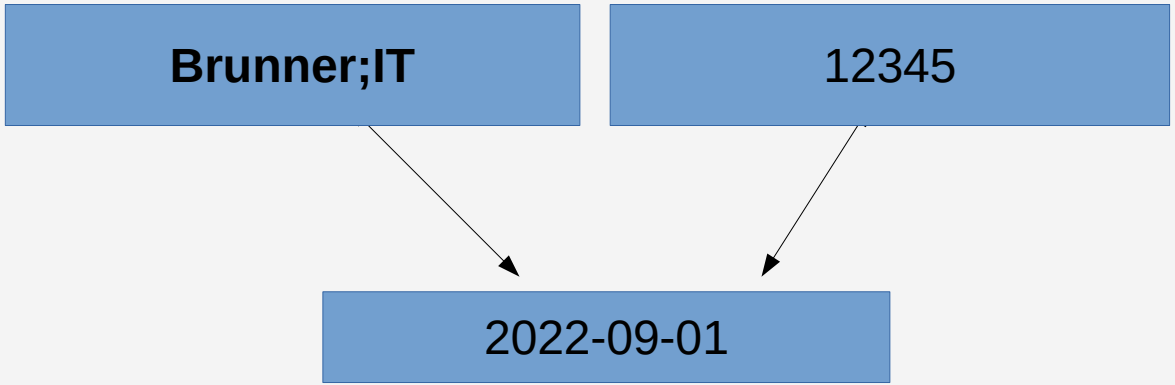


Prof	Department	Prüfungstermin
Brunner	IT	2022-09-01
Klein	IT	2022-09-10

Stud	Prüfungstermin
12345	2022-09-01
12345	2022-09-10
12346	2022-09-01

Problem: Redundanzen
Hier treten z.B. Prüfungsterminen mehrfach auf.

Netzmodell



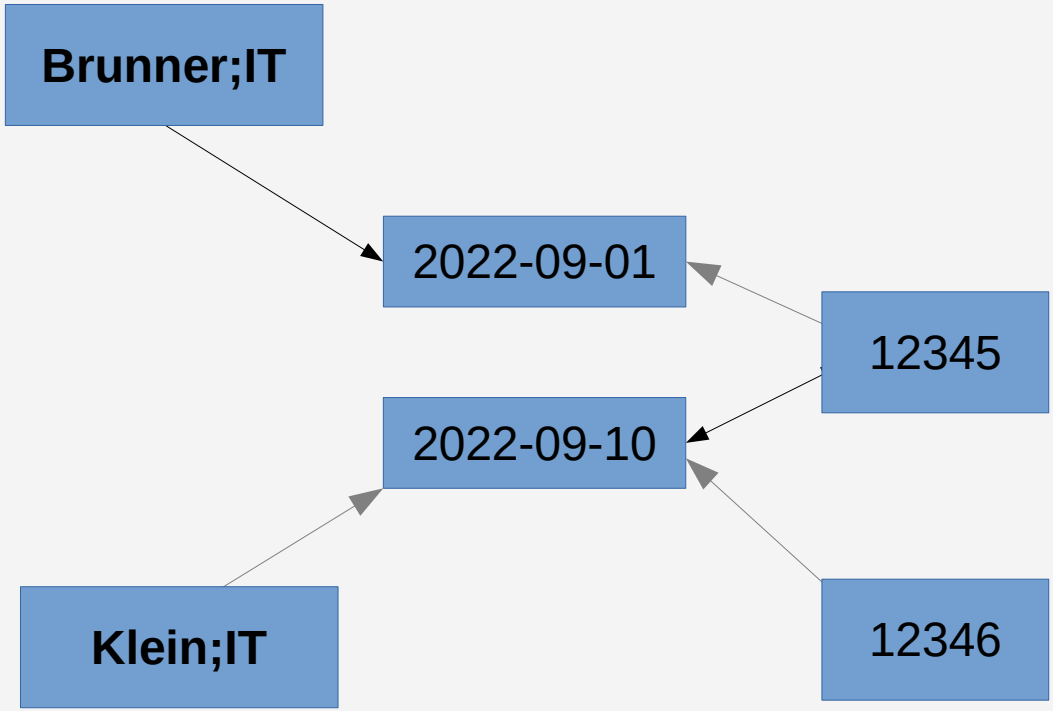
Prof	Department
Brunner	IT
Klein	IT

Stud
12345
12345
12346

Prüfung
2022-09-01
2022-09-10

Redundanzen vermeiden

Netzmodell



Prof	Department
Brunner	IT
Klein	IT

Stud
12345
12345
12346

Prüfung
20200901
20200910

Aber schnell unübersichtlich

Relationales Modell

- Tabellen

- Primärschlüssel & Fremdschlüssel

- Beziehungen über Schlüsselverknüpfung
  (SQL mit Primär- und Fremdschlüsseln)

Relationales Modell

Verwendung von Primär- und Fremdschlüsseln

Prof

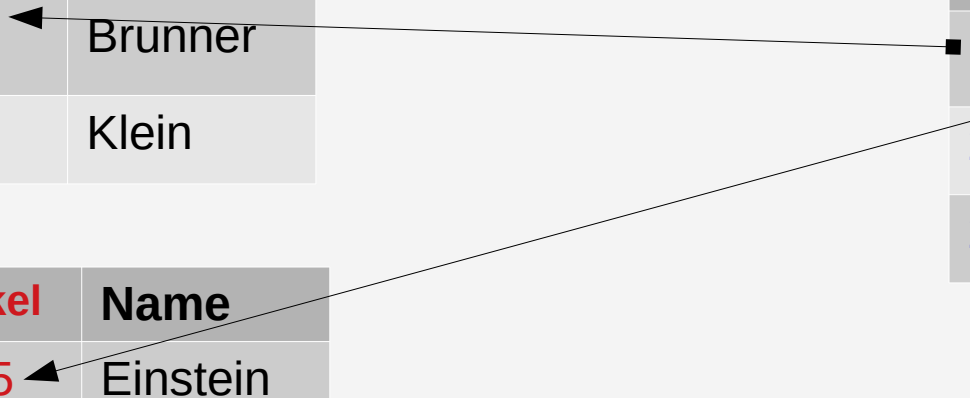
Prof-ID	Name
1	Brunner
2	Klein

Stud

Matrikel	Name
12345	Einstein
12346	Gödel

Prüfung

Prof-ID	Matrikel	Datum
1	12345	2022-09-01
2	12345	2022-09-10
2	12346	2022-09-10



Relationenalgebra

Relationen sind im mathematischen Sinne so etwas:

Eine Relation ist allgemein eine Beziehung, die zwischen „Dingen“ bestehen kann.

Die Relation ist dabei z.B. eine Teilmenge des kartesischen Produktes („Kreuzproduktes“).

Das kartesische Produkt
Mathematik

Datenmenge: $\mathbb{N}$ (natürliche Zahlen)

Kartesisches Produkt: $\mathbb{N}x\mathbb{N}$

Das kartesische Produkt von Mengen ist nichts weiter als die Kombination aller Elemente des einen Faktors mit denen des anderen.

Das Kreuzprodukt

Datenmenge: $\mathbb{N}$ (natürliche Zahlen)
$\mathbb{N} = \{1, 2, 3, 4, \ldots\}$

$\mathbb{N} \times \mathbb{N} = \{ (1,1), (1,2), (1,3), (1,4), \ldots,$
$(2,1), (2,2), (2,3), (2,4), \ldots \}$

Relationen als n-Tupel

- Beispiel: Relation „**kleiner als**“ für natürliche Zahlen

  $\mathbb{N}x\mathbb{N}$ = { (1,1), (1,2), (1,3), (1,4), …, (2,1), (2,2), (2,3), (2,4), … }

  $R \subseteq \mathbb{N}x\mathbb{N}$
  R = { (1,2), (1,3), (1,4),…}

  Nochmal:
  Eine Relation ist eine Untermenge, die nach bestimmten Regeln erstellt wird.

Relationen & SQL-Tabellen

Genau betrachtet ist auch jede SQL-Tabelle eine Relation, da sie ja Beziehungen zwischen Daten dastellt.
Auch nochmals am Beispiel für „kleiner“

$R \subseteq \mathbb{N} \times \mathbb{N}$
$R = \{ (1,2), (1,3), (1,4)\}$ (Diesmal endliche Datenmenge)

A	B
1	2
1	3
1	4

- Wir haben das kartesische Produkt berechnet und die Relation als Teilmenge gebildet

- Wir haben das Ergebnis als Tabelle dargestellt

- Unterschied zur Mathematik: SQL-Tabellen sind immer endlich.

Relationenalgebra
Relationale Operationen mit SQL-Bezug

→ Project

→ Restrict

→ Product / Join

Relationenalgebra
Relationale Operationen ohne SQL-Bezug

- → Union
- → Intersection (Durchschnitt / Schnittmenge)
- → Difference
- → Division

Project - Projektion

- Eine Projektion projiziert eine Relation und macht sie dabei „schmaler“.
- Am besten kann man sich das vorstellen, dass **Spalten** ausgewählt werden.
- Notation : $\pi_{Relation}(<Attributliste>)$ oder $PROJ_{Relation}(<Attributliste>)$

Project – Projektion
Beispiele

$R = \{ (1, 2), (1, 4), (1, 6), (1, 8), (1, 10), ... \}$

$\pi_R(\text{Komponente Nr. 2}) = \{ (2), (4), (6), (8), (10), ... \}$

Projektion in SQL
„Spalten auswählen“

Studierende

MatrikelNr	Nachname	Vorname
4001000	Wolkowitsch	Andrea
4001001	Adler	Tim
4001002	Goldstein	Sarah

Vorname
Andrea
Tim
Sarah

$\pi_{\text{Studierende}}(\text{Vorname}) =$

{ Andrea, Tim, Sarah }

Dies macht „select“.

Restriktion / Restrict

Restrict ist eine Einschränkung, um aus einer Relation Tupel (oder „Zeilen“) auszuwählt.

Notation: $R^*_{\text{Relation}}(\text{Bedingung})$

Restriktion / Restrict
Beispiel

Relation „kleiner als“ für natürliche Zahlen
Datenmenge: $\mathbb{N}$

$R_{<}$ = { (1,1), (1,2), (1,3), (1,4),…,(2,1), (2,2), (2,3),(2,4),…}

- Nun sei die Restriction:
  Betrachte nur Tupel, die Zifferen unter 4 enthalten:
  $R^{*}_{R<}$ (kleiner_4) = { (1,2), (1,2), (1,3), (2,1), (2,2), (2,3),... }

Restriction in SQL
„Zeilen auswählen“

Studierende

Matrikelnr	Nachname	Vorname
4001000	Wolkowitsch	Andrea
4001001	Adler	Tim
4001002	Goldstein	Sarah

Matrikelnr	Nachname	Vorname
4001000	Wolkowitsch	Andrea
4001001	Adler	Tim

$$R^*_{\text{Studierende}}(\text{Matrikelnr} < 4001002) =$$

{
(4001000, Wolkowitsch, Andrea),
(4001002, Adler, Tim)
}

Dies macht „where“.

Product

Hier handelt es sich im nichts weiter als das kartesische Produkt.

Product

Beispiel für endliche Mengen

{1, 2} x {Eins, Zwei} = { (1, Eins), (1, Zwei), (2, Eins), (2, Zwei) }

Hier sehen wir:

- Es bilden sich Tupel (im Beispiel Zwei-Tupel, also Paare)

- Die Anzahl dieser Tupel ist gleich dem Produkt der Anzahl der Elemente der einzelnen Komponenten, hier 2 * 2 = 4

→ Ein Produkt wird also schnell vom Datenumfang groß.

Product bei SQL

In SQL führt eine Anweisung wie **select * from tabelle1, tabelle2** (kein where) zum kartesischen Produkt.

Matrikelnr	Nachname
4001000	Wolkowitsch
4001001	Adler

x

VorlesNr	VorlesName
1000	Datenbanken
1001	BWL

STUDENT = { (4001000, Wolkowitsch), (4001001, Adler)}

VORLESUNG = { (1000, Datenbanken), (1001, BWL) }

STUDENT x VORLESUNG = {
((4001000, Wolkowitsch), (1000, Datenbanken)),
((4001000, Wolkowitsch), (1001, BWL)),
((4001001, Adler), (1000, Datenbanken)),
((4001001, Adler), (1001, BWL))
}

Product bei SQL

STUDENT x VORLESUNG = {
((4001000, Wolkowitsch),(1000, Datenbanken)),
((4001001, Adler), (1000, Datenbanken)),
((4001000, Wolkowitsch),(1001, BWL)),
((4001001, Adler), (1001, BWL))
}

MatrikelNr	Nachname	VorlesNr	VorlesName
4001000	Wolkowitsch	1000	Datenbanken
4001001	Adler	1000	Datenbanken
4001000	Wolkowitsch	1001	BWL
4001001	Adler	1001	BWL

Product bei SQL
Zwischenfazit

MatrikelNr	Nachname
4001000	Wolkowitsch
4001001	Adler

x

VorlesNr	VorlesName
1000	Datenbanken
1001	BWL

=

MatrikelNr	Nachname	VorlesNr	VorlesName
4001000	Wolkowitsch	1000	Datenbanken
4001001	Adler	1000	Datenbanken
4001000	Wolkowitsch	1001	BWL
4001001	Adler	1001	BWL

Die Resultate von **select * from Tabelle1 , Tabelle2** werden sehr schnell sehr groß.
Lösung: Filtern mit Projektion und Restriktion.

Product bei SQL: Filter
Projektion und Restriktion

select **Nachname, VorlesName** from Studierende, Vorlesung where **VorlesNr = 1000**

Matrikelnr	Nachname
4001000	Wolkowitsch
4001001	Adler

VorlesNr	VorlesName
1000	Datenbanken
1001	BWL

Matrikelnr	Nachname	VorlesNr	VorlesName
4001000	Wolkowitsch	1000	Datenbanken
4001001	Adler	1000	Datenbanken
4001000	Wolkowitsch	1001	BWL
4001001	Adler	1001	BWL

Ohne Filter

Product bei SQL: Filter
Projektion und Restriktion

select **Nachname, VorlesName** from Studierende, Vorlesung where **VorlesN = 1000**

MatrikelNr	Nachname
4001000	Wolkowitsch
4001001	Adler

VorlesNr	VorlName
1000	Datenbanken
1001	BWL

MatrikelNr	Nachname	VorlesNr	VorlesName
4001000	Wolkowitsch	1000	Datenbanken
4001001	Adler	1000	Datenbanken
4001000	Wolkowitsch	1001	BWL
4001001	Adler	1001	BWL

Die Attributliste in Select ist die Projektion und filtert Spalten.

Product bei SQL: Filter
Projektion und Restriktion

select **Nachname, VorlName** from Studierende, Vorlesung **where**
VorlesNr **= 1000**

MatrikelNr	Nachname
4001000	Wolkowitsch
4001001	Adler

VorlesNr	VorlName
1000	Datenbanken
1001	BWL

MatrikelNr	Nachname	VorlesNr	VorlesName
4001000	Wolkowitsch	1000	Datenbanken
4001001	Adler	1000	Datenbanken
4001000	Wolkowitsch	1001	BWL
4001001	Adler	1001	BWL

Die Restriktion ist die WHERE-Klausel und filtert Zeilen.

Product bei SQL: Filter
Projektion und Restriktion

select **Nachname, VorlName** from Studierende, Vorlesung where **VorlesNr = 1000**

MatrikelNr	Nachname
4001000	Wolkowitsch
4001001	Adler

VorlesNr	VorlesName
1000	Datenbanken
1001	BWL

Nachname	VorlesName
Wolkowitsch	Datenbanken
Adler	Datenbanken

Product bei SQL: Filter
Projektion und Restriktion

select **Nachname, VorlName** from Studierende, Vorlesung where **VorlesNr = 1000**

MatrikelNr	Nachname
4001000	Wolkowitsch
4001001	Adler

VorlesNr	VorlesName
1000	Datenbanken
1001	BWL

Nachname	VorlesName
Wolkowitsch	Datenbanken
Adler	Datenbanken

Wichtig in der Praxis:

Schon vor dem Bilden des Produkts filtern, sonst bekommen wir die Zwischenergebnisse nicht gspeichert.

Weitere Operationen
(in MariaDB ohne SQL-Entsprechung)

Union

Intersection

Difference

Division

Union, Intersection, Difference

- Union ist die Vereinigung von Mengen / Relationen

  $\text{Union}(M_1, M_2) = M_1 \cup M_2$

- Intersection ist die Schnittmenge zweier Mengen / Relationen

  $\text{Intersection}(M_1, M_2) = M_1 \cap M_2$

- Difference: Mengen / Relationen subtrahieren:
  {1,2,3,4,5} - {4,5} = {1,2,3}

Dazu gibt es in MySQL/MariaDB keine direkte Entsprechung.

Division

In der Relationenalgebra eigentlich ein Operator alá **„für alle“**

Wir haben eine Relation R und eine Relation S

R / S = t

wenn für alle Tupel aus S ein um t erweitertes Tupel existiert, welches aus R ist. Dabei ist t Element der Relation R.

Division
Beispiel

→ R beschreibt, welcher Lieferent (LNR) welche Projekte (PNR) mit welchen Teilen (TNR) beliefert.
→ S beschreibt, welches Projekt welche Teile benötigt.

Frage:
Welche Lieferanten (LNR) beliefern **alle** Projekte (PNR) mit **allen** benötigten Teilen (TNR)?

R

LNR	PNR	TNR
L1	P1	T1
L1	P2	T1
L2	P1	T1
L2	P1	T2
L2	P2	T1

S

PNR	TNR
P1	T1
P1	T2
P2	T1

Division

LNR	PNR	TNR
L1	P1	T1
L1	P2	T1
L2	**P1**	**T1**
L2	**P1**	**T2**
L2	**P2**	**T1**

/

PNR	TNR
P1	T1
P1	T2
P2	T1

R DIV S = L2
Weil: ALLE Tupel aus S – erweitert um L2 – in R vorkommen:

(L2, P1, T1) ist aus R
(L2, P1, T2) ist aus R
(L2, P2, T1) ist aus R

Division

LNR	PNR	TNR
L1	P1	T1
L1	P2	T1
L2	**P1**	**T1**
L2	**P1**	**T2**
L2	**P2**	**T1**

/

PNR	TNR
P1	T1
P1	T2
P2	T1

Mit L1 geht es nicht:

(L1, P1, T2) $\notin$ R

Datenunabhängigkeit im DBS

Physische Datenunabhängigkeit

Zur Erinnerung

Binäres Schreiben in eine Datei

```
void foo() {
  long int x = 1;
  fwrite(&x, f);
}
```

H**exdump der entstandenen Datei**

arm32: **0x01** 0x00 0x00 0x00
mips32: 0x00 0x00 0x00 **0x01**

Physische Datenunabhängigkeit

Physische Datenunabhängikeit verbirgt, wie „konkret“ gespeichert wird.

Wie eine Datenbank ihre Tabellen auf der Festplatte sichert, ist für den Nutzer transparent.

So ist es hier Aufgabe des Datenbanksystems, sich um Probleme alá LittleEndian vs BigEndian zu kümmern.

Logische Datenunabhängigkeit

Logische Datenunabhängikeit verbirgt, wie „konkret“ Daten verwaltet werden.

So ist es für den Nutzer „egal“, wie das Datenbanksystem die folgende Tabelle **intern** im Speicher darstellt.

Nachname	**Vorname**	**Land**	**Geburtsjahr**
Gurion	Ben	Israel	1886
Nasser	Gamal, Abdel	Ägypten	1918
Curie	Marie	Polen	1867
Gagarin	Juri	UdSSR	1934

Logische Datenunabhängigkeit
Beispiel Bäume

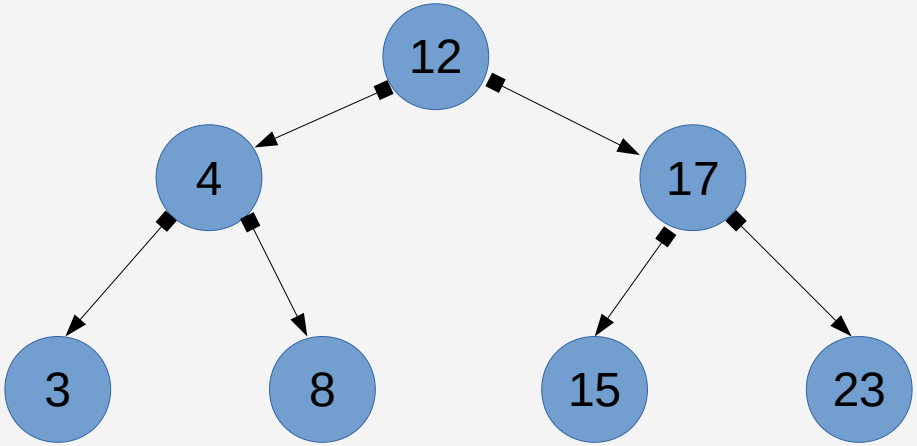
Wie kann ein DBS Daten schnell finden?

select * from Personen where ID=12

Lineare Suche dauert, daher u.U. intern Bäume.

Logische Datenunabhängigkeit
Bäume

- Einfacher zu erklären: Binärbaum
- Daten: 12, 4, 17, 15, 8, 23, 3
- Das kann man freilich **linear** durchsuchen, oder man schreibt es gleich als Baum:

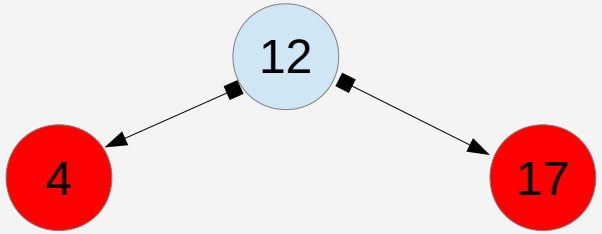


Logische Datenunabhängigkeit
Bäume erstellen

- Einfacher: Binärbaum
- Daten: 12, 4, 17, 15, 8, 23, 3
- Das kann man freilich **linear** durchsuchen, oder man schreibt es gleich als Baum:

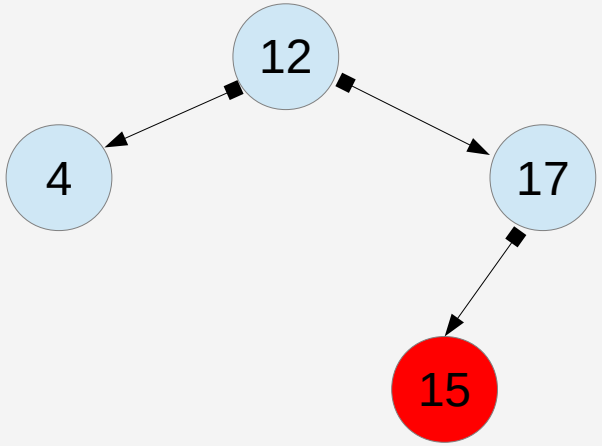
Logische Datenunabhängigkeit
Bäume erstellen

- Einfacher: Binärbaum
- Daten: 12, 4, 17, 15, 8, 23, 3
- Das kann man freilich **linear** durchsuchen, oder man schreibt es gleich als Baum:



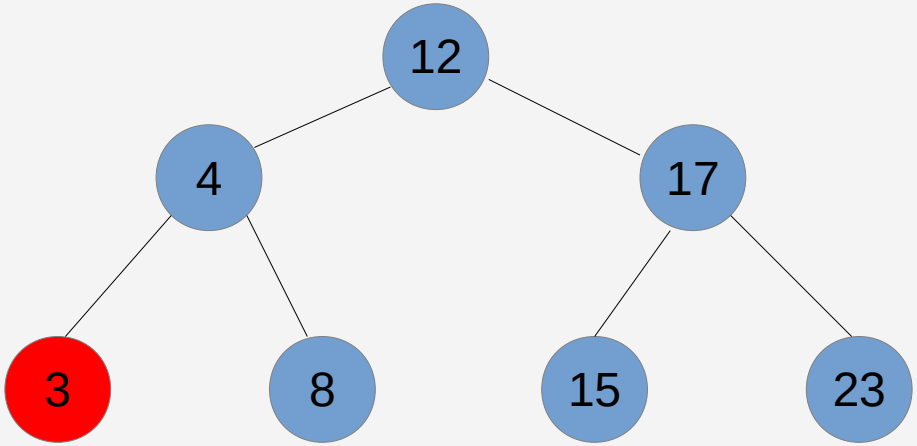
Logische Datenunabhängigkeit
Bäume erstellen

- Einfacher: Binärbaum
- Daten: 12, 4, 17, 15, 8, 23, 3
- Das kann man freilich **linear** durchsuchen, oder man schreibt es gleich als Baum:



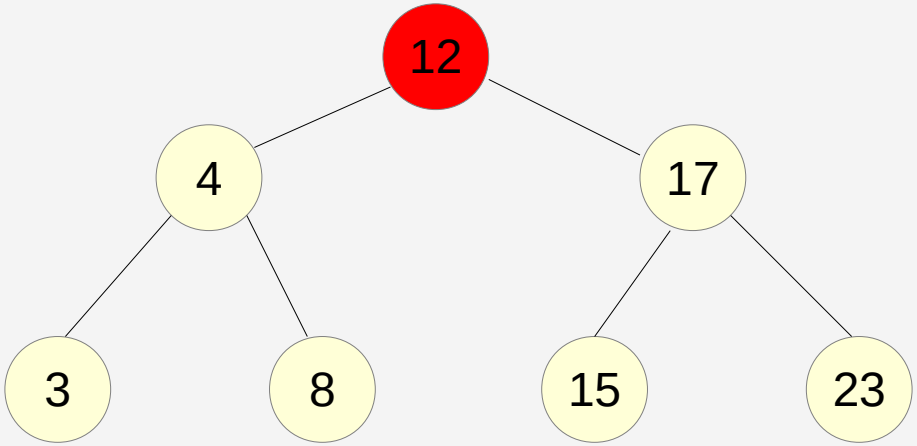
Logische Datenunabhängigkeit
Bäume erstellen

- Einfacher: Binärbaum
- Daten: 12, 4, 17, 15, 8, 23, 3
- Das kann man freilich **linear** durchsuchen, oder man schreibt es gleich als Baum:



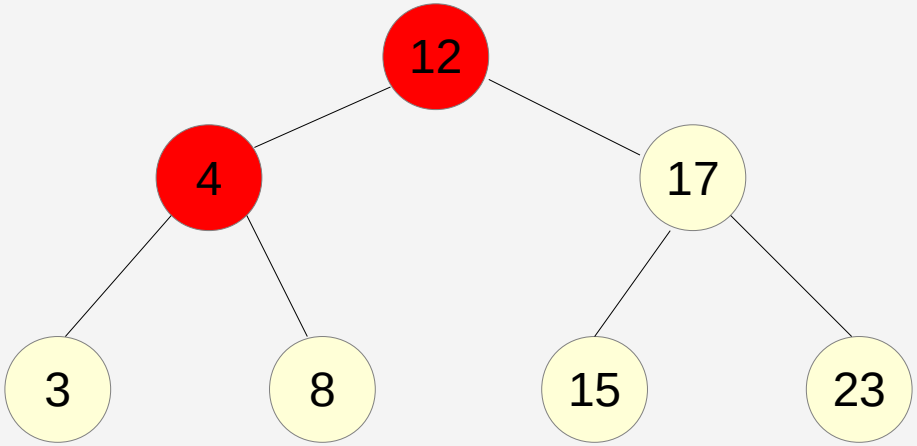
Logische Datenunabhängigkeit
Bäume durchsuchen

- Suche die 8 (bei 7 Elementen)
- Daten: 12, 4, 17, 15, 8, 23, 3



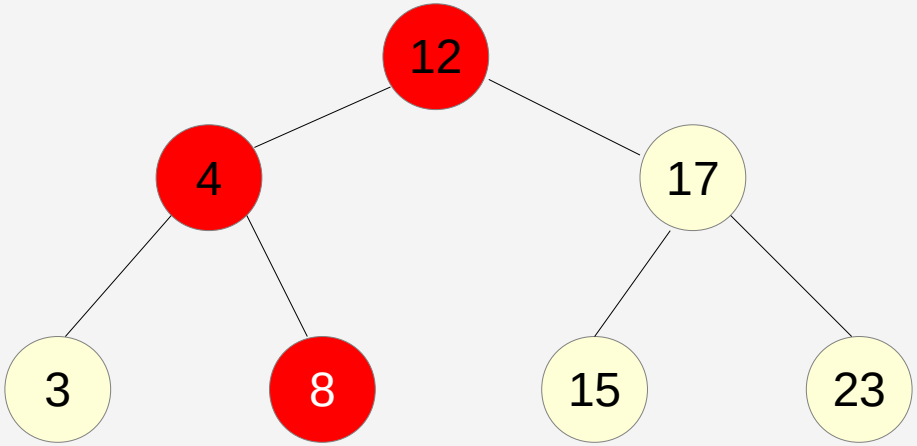
Logische Datenunabhängigkeit
Bäume durchsuchen

- Suche die 8 (bei 7 Elementen)
- Daten: 12, 4, 17, 15, 8, 23, 3



Logische Datenunabhängigkeit
Bäume durchsuchen

- Suche die 8 (bei 7 Elementen)
- Daten: 12, 4, 17, 15, 8, 23, 3
- Gefunden: 3 Schritte
  $\log_2(7) = 2.8$ also rund 3



Logische Datenunabhängigkeit
Bäume durchsuchen

- Lineare Suche: O(n)

  (im Durchschnitt braucht man bei n Suchelementen n/2 Versuche)

- Baumsuche:
  → Binärbaum O(log n) .. O(n)

  Hinweis: $\log_2(n)$ errechnet im Prinzip die Anzahl der Ebenen, die ein Binärbaum hat, und im Grunde ist die die Anzahl der Suchanfragen. (Bsp war: 7 Elemente, 3 Ebenen, 3 Suchanfragen)

Datenunabhängigkeit **umsetzen**:
Datenbankaufbau

Arbeitsweise eines DBS
Schritt 1: Satzsystem

Auf Nutzerebene werden **beschreibende** Anfragen an das DBS gestellt, z.B:
„Suche alle Studenten, die bei Prof. Brunner eine Prüfung hatten.“

→ Diese Anfrage wird vom DBS **interpretiert** und **optimiert**.
→ Die Optimierung erfolgt daher, da es für die Performance eintscheidend ist, in welcher Reihenfolge das DBS die Daten durchsucht, um die Anfrage beantworten zu können.

Das Resultat sind **Satzzugriffe**.
Das Datenbanksystem weiß nun, in welcher Reihenfolge es Tabellen auswerten muss.

Arbeitsweise eines DBS: Schritte 2 und 3

Bisher: (1) Satzsystem
Wir haben nun Datensatzzugriffe (ergo: Datenzeilen aus Tabellen)

Dazu kommen nun noch folgende Schritte

(2) Zugriffssystem:
→ Dieses weiß, **wo** die Daten zu finden sind (z.B. in Dateien des DBS).
→ Das Resultat sind Seitenzugriffe (oder Blockzugriffe)

(3) Speichersystem:
→ Das Speichersystem kümmert sich um evtl. gepufferte Daten.
→ Dies kann Aufgabe des Betriebsystems sein oder auch eine des DBS selbst.

Komponenten eines DBS

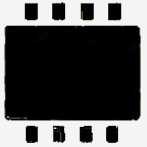
select ...

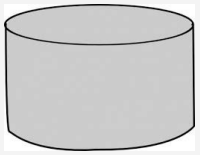
Analyse/Optimierung:
Lies Tabellen
„Studenten“ & „Prüfungen“

1

Zugriffssystem:
Liest Dateien
Student.ibd
&
Prüfungen.ibd

2





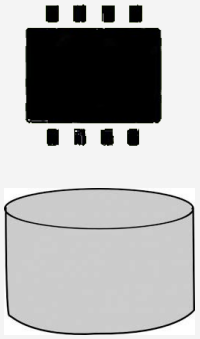
Speichersystem:

Student.ibd

Prüfungen.ibd

3

Speichersystem

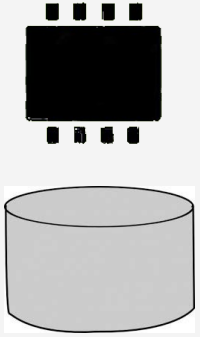


Speichersystem:

Student.ibd

Prüfungen.ibd

Das geht noch, da getrennte Daten



Speichersystem:

Student.ibd

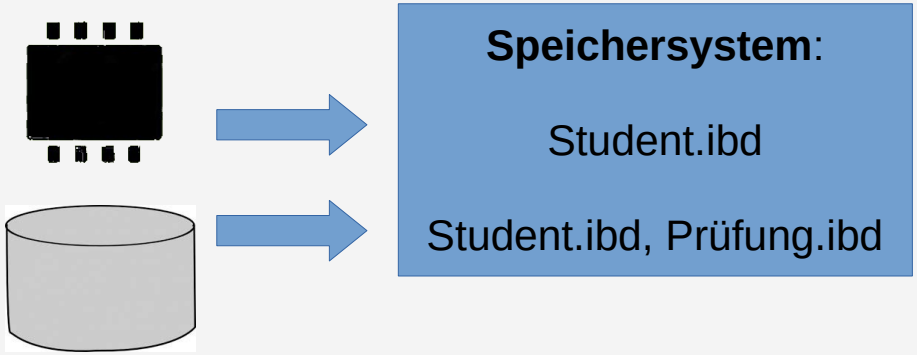
Student.ibd, Prüfung.ibd

Problematischer, da gemischte Daten mit Konsistenzproblemen

Es befinden sich Kopien von Student.ibd im Speicher und auch auf „Festplatte“.

Diese Inkonsistenzen mussen vom DBS aufgelöst werden.

Speichersystem



Stichwort Cacheverwaltung:

- Duplikate erkennen, Duplikate beseitigen

- So ist das Lesen einer Kopie unproblematisch, das Ändern aber nicht.

- Wann abgleichen? Performance-Problem.