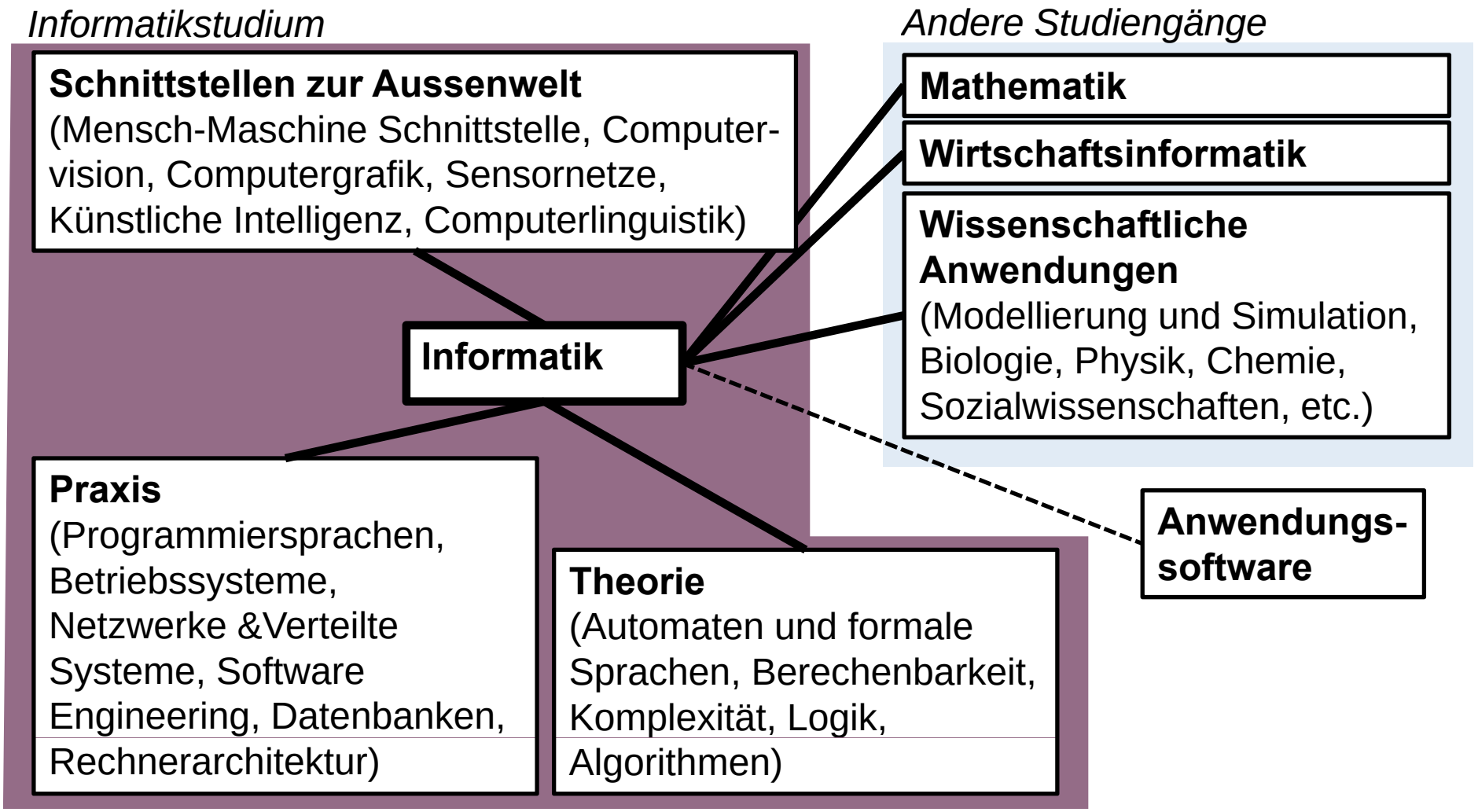
# Einführung in die Informatik:

# Datenbanken

Thomas Studer

UNIVERSITÄT
BERN

# Übersicht


Informatikstudium
Andere Studiengänge
Schnittstellen zur Aussenwelt
(Mensch-Maschine Schnittstelle, Computer-vision, Computergrafik, Sensornetze, Künstliche Intelligenz, Computerlinguistik)
Mathematik
Wirtschaftsinformatik
Wissenschaftliche Anwendungen
(Modellierung und Simulation, Biologie, Physik, Chemie, Sozialwissenschaften, etc.)
Informatik
Anwendungs-software
Praxis
(Programmiersprachen, Betriebssysteme, Netzwerke &Verteilte Systeme, Software Engineering, Datenbanken, Rechnerarchitektur)
Theorie
(Automaten und formale Sprachen, Berechenbarkeit, Komplexität, Logik, Algorithmen)

# Datenbanken sind überall




amazon.de


Google™ Deutschland

# Vorschau



- Aufbau von Datenbanken
- Abfragen, SQL
- Konsistenz
- Effizienz
- Datenschutz
- Q & A

# Tabellen

UNIVERSITÄT BERN

**Konten:**

| KontoNr | Stand | Kunde | Ort |
|---|---|---|---|
| 1 | 20000 | Meier | Bern |
| 2 | -10 | Studer | Köniz |
| 3 | 250 | Müller | Thun |
| 4 | 50 | Meier | Bern |
| 4 | 50 | Müller | Thun |

In dieser Tabelle ist pro Konto angegeben:

- wie hoch der Kontostand ist,
- welche Personen auf das Konto zugreifen dürfen,
- wo diese Personen wohnen.

# CRUD

UNIVERSITÄT BERN

Grundlegenden Datenbankoperationen:

- **C**reate: Datensatz anlegen
- **R**ead: Datensatz lesen
- **U**pdate: Datensatz aktualisieren
- **D**elete: Datensatz löschen

# Abfrage (Read)



| KontoNr | Stand | Kunde | Ort |
|---|---|---|---|
| 1 | 20000 | Meier | Bern |
| 2 | -10 | Studer | Köniz |
| 3 | 250 | Müller | Thun |
| 4 | 50 | Meier | Bern |
| 4 | 50 | Müller | Thun |

- KontoNr, für die **Stand < 0** ist
- KontoNr, für die **Name >= L** und **Name < R** ist
- Kunden, für die **Ort = Bern** oder **Ort = Köniz** ist
- Kunden und deren Wohnort, für die **Ort = Bern** oder **Ort = Köniz** ist

# Abfrage: SQL

UNIVERSITÄT BERN

- KontoNr, für die **Stand < 0** ist
- KontoNr, für die **Name >= L** und **Name < R** ist
- Kunden, für die **Ort = Bern** oder **Ort = Köniz** ist
- Kunden und deren Wohnort, für die **Ort = Bern** oder **Ort = Köniz** ist

```
SELECT KontoNr FROM Konten WHERE Stand < 0
SELECT KontoNr FROM Knoten WHERE Name >= „L“ AND Name < „R“
SELECT Kunde FROM Konten WHERE Ort = „Bern“ OR Ort = „Köniz“
SELECT Kunde, Ort FROM Konten WHERE Ort = „Bern“ OR Ort = „Köniz“
```

# Konsistenz



| KontoNr | Stand | Kunde | Ort |
|---|---|---|---|
| 1 | 20000 | Meier | Biel |
| 2 | -10 | Studer | Köniz |
| 3 | 250 | Müller | Thun |
| 4 | -50 | Meier | Bern |
| 4 | 500000 | Müller | Thun |


Wo wohnt Meier?

Wieviel Geld ist auf dem Konto 4?

# Funktionale Abhängigkeiten

UNIVERSITÄT BERN

- KontoNr -> Stand
- Kunde -> Ort

| KontoNr | Stand | Kunde | Ort |
|---|---|---|---|
| 1 | 20000 | Meier | Bern |
| 2 | -10 | Studer | Köniz |
| 3 | 250 | Müller | Thun |
| 4 | 50 | Meier | Bern |
| 4 | 50 | Müller | Thun |

- Eine Tabelle erstellen, welche jede KontoNr genau einmal enthält zusammen mit dem Kontostand.
- Zweite Tabelle erstellen, welche jeden Kunden genau einmal enthält zusammen mit dem Ort
- Dritte Tabelle erstellen mit der Relation zwischen KontoNr und Kunde

# Normalform

| KontoNr | Kunde |
|---|---|
| 1 | Meier |
| 2 | Studer |
| 3 | Müller |
| 4 | Meier |
| 4 | Müller |

Schlüssel:
**<KontoNr, Kunde>**

Schlüssel:
**KontoNr**

| KontoNr | Stand |
|---|---|
| 1 | 20000 |
| 2 | -10 |
| 3 | 250 |
| 4 | 50 |

Schlüssel:
**Kunde**

| Kunde | Ort |
|---|---|
| Meier | Bern |
| Studer | Köniz |
| Müller | Thun |

# DB Schema



Das Datenbank **Schema** beschreibt die Tabellen mit den Attributen und den Schlüsseln:

Konten = (KontoNr, Stand)
KoKu = (KontoNr, Kunde)
Kunden = (Kunde, Ort)

Ein DB Schema kann mit einem **Entity-Relationship Diagramm** dargestellt werden.

Eine DB **Instanz** besteht aus allen in der DB abgespeicherten Datensätzen.

# Daten Design (1)



- Welche Tabellen braucht es?
- Mit welchen Attributen (Spalten)?
- Welche funktionalen Abhängigkeiten gibt es?

- Kann man alle Daten abspeichern und auch wieder abfragen?
- Ist die Datenkonsistenz gewährleistet?

# Abfrage: SQL (2)

Schema:
Konten = (KontoNr, Stand)
KoKu = (KontoNr, Kunde)
Kunden = (Kunde, Ort)

Gesucht:
KontoNr auf die eine Person aus Bern zugreifen darf.

```
SELECT KoKu.KontoNr FROM KoKu, Kunden WHERE
                    KoKu.Kunde=Kunden.Kunde AND Kunden.Ort=„Bern“
```

# SELECT * FROM KoKu, Kunden

UNIVERSITÄT BERN

| KoKu.KontoNr | KoKu.Kunde | Kunden.Kunde | Kunden.Ort |
|---|---|---|---|
| 1 | Meier | Meier | Bern |
| 1 | Meier | Studer | Köniz |
| 1 | Meier | Müller | Thun |
| 2 | Studer | Meier | Bern |
| 2 | Studer | Studer | Köniz |
| 2 | Studer | Müller | Thun |
| 3 | Müller | Meier | Bern |
| 3 | Müller | Studer | Köniz |
| 3 | Müller | Müller | Thun |
| … | … | … | … |

# SELECT * FROM KoKu, Kunden WHERE KoKu.Kunde=Kunden.Kunde

UNIVERSITÄT BERN

| KoKu.KontoNr | KoKu.Kunde | Kunden.Kunde | Kunden.Ort |
|---|---|---|---|
| 1 | Meier | Meier | Bern |
| ~~1~~ | ~~Meier~~ | ~~Studer~~ | ~~Köniz~~ |
| ~~1~~ | ~~Meier~~ | ~~Müller~~ | ~~Thun~~ |
| ~~2~~ | ~~Studer~~ | ~~Meier~~ | ~~Bern~~ |
| 2 | Studer | Studer | Köniz |
| ~~2~~ | ~~Studer~~ | ~~Müller~~ | ~~Thun~~ |
| ~~3~~ | ~~Müller~~ | ~~Meier~~ | ~~Bern~~ |
| ~~3~~ | ~~Müller~~ | ~~Studer~~ | ~~Köniz~~ |
| 3 | Müller | Müller | Thun |
| … | … | … | … |

# Abfrage: SQL (3)

Schema:
Konten = (KontoNr, Stand)
KoKu = (KontoNr, Kunde)
Kunden = (Kunde, Ort)

Gesucht:
Kunden welche am meisten Geld auf einem Konto haben.

```
SELECT KoKu.Kunde FROM KoKu, Konten WHERE
                    KoKu.KontoNr = Konten.KontoNr AND
                    Konten.Stand => ALL (SELECT Stand FROM Konten)
```

# Transaktion (1)



Überweise CHF 1000.- vom Konto 1 zu Konto 4

Ablauf:

1. Betrag = Betrag – 1000 bei Konto 1
2. Betrag = Betrag + 1000 bei Konto 4

Diese beiden Schritte sollen eine Einheit bilden. Entweder werden beide ausgeführt oder keiner. D.h. sie werden als eine Transaktion ausgeführt.

Falls bei Schritt 2 ein Problem auftritt wird ein sogenanntes **Rollback** ausgeführt, d.h. der Zustand vor Beginn der Transaktion wird wieder hergestellt.

# Transaktion (2)

UNIVERSITÄT BERN

Schema:
Konten = (KontoNr, Stand)
KoKu = (KontoNr, Kunde)
Kunden = (Kunde, Ort)

Aktion: Lösche Konto 1

1. Lösche Eintrag mit KontoNr 1 in Konten
2. Lösche alle Einträge mit KontoNr 1 in KoKu

Auch hier müssen beide Schritte ausgeführt werden. Falls es bei Schritt 2 ein Problem gibt, muss ein Rollback ausgeführt werden.

# Transaktion (3)

UNIVERSITÄT BERN

***Transaktion*:** Folge von Operationen, die logische Einheit bilden.

***ACID – Eigenschaften:***

**Atomarität**: Eine Transaktion wird entweder ganz oder gar nicht ausgeführt. Transaktionen ist „unteilbar“. Wenn eine atomare Transaktion abgebrochen wird, ist das System unverändert.

**Konsistenz**: Nach Ausführung ist Datenbestand konsistent.

**Isolation**: Keine gegenseitige Beeinflussung von Transaktionen.

**Dauerhaftigkeit**: Die Auswirkungen einer Transaktion müssen im Datenbestand dauerhaft bestehen bleiben.

# Effiziente Abfragen: Binäre Suche

UNIVERSITÄT BERN

Wie ist der Kontostand von „Studer“?

Wenn man bei „A“ beginnt und alle Namen durchgeht dauert es zu lange!

Besser: In der Mitte anfangen und entweder nach oben oder unten suchen

Noch besser: In die Mitte gehen, dann entweder die obere oder untere Hälfte wieder teilen, den gefundenen Viertel wieder teilen, etc.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

↑

1

# Effiziente Abfragen: Binäre Suche

Wie ist der Kontostand von „Studer“?

Wenn man bei „A“ beginnt und alle Namen durchgeht dauert es zu lange!

Besser: In der Mitte anfangen und entweder nach oben oder unten suchen

Noch besser: In die Mitte gehen, dann entweder die obere oder untere Hälfte wieder teilen, den gefundenen Viertel wieder teilen, etc.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

↑ ↑

1 2

# Effiziente Abfragen: Binäre Suche

Wie ist der Kontostand von „Studer“?

Wenn man bei „A“ beginnt und alle Namen durchgeht dauert es zu lange!

Besser: In der Mitte anfangen und entweder nach oben oder unten suchen

Noch besser: In die Mitte gehen, dann entweder die obere oder untere Hälfte wieder teilen, den gefundenen Viertel wieder teilen, etc.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

↑ ↑ ↑

1 3 2

# Effiziente Abfragen: Binäre Suche

Wie ist der Kontostand von „Studer“?

Wenn man bei „A“ beginnt und alle Namen durchgeht dauert es zu lange!

Besser: In der Mitte anfangen und entweder nach oben oder unten suchen

Noch besser: In die Mitte gehen, dann entweder die obere oder untere Hälfte wieder teilen, den gefundenen Viertel wieder teilen, etc.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

↑ ↑ ↑ ↑

1 3 4 2

# Effiziente Abfragen: Binäre Suche

Wie ist der Kontostand von „Studer“?

Wenn man bei „A“ beginnt und alle Namen durchgeht dauert es zu lange!

Besser: In der Mitte anfangen und entweder nach oben oder unten suchen

Noch besser: In die Mitte gehen, dann entweder die obere oder untere Hälfte wieder teilen, den gefundenen Viertel wieder teilen, etc.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

↑ ↑ ↑ ↑ ↑

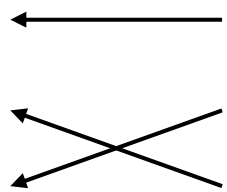
1 3 4 5 2

Tabelle muss nach Namen sortiert sein!

# Index anlegen

In der Praxis wird nicht die Tabelle sortiert, sondern es wird ein sogenannter **Index** erstellt.

| Kunde | Ort |
|---|---|
| Meier | Bern |
| Müller | Thun |
| Studer | Köniz |



| Index: Ort |
|---|
| Bern |
| Köniz |
| Thun |

Zu einer Tabelle sind auch **mehrere Indexe** möglich. So kann nach verschiedenen Attributen gesucht werden.

**Balancierte Bäume** sind eine geeignete Datenstruktur um einen Index anzulegen.

# Daten Design (2)



- Welche Abfragen kommen vor?
- Nach welchen Attributen wird dabei gesucht?
- Welche Indexe müssen gebildet werden?

# Das Data Privacy Problem

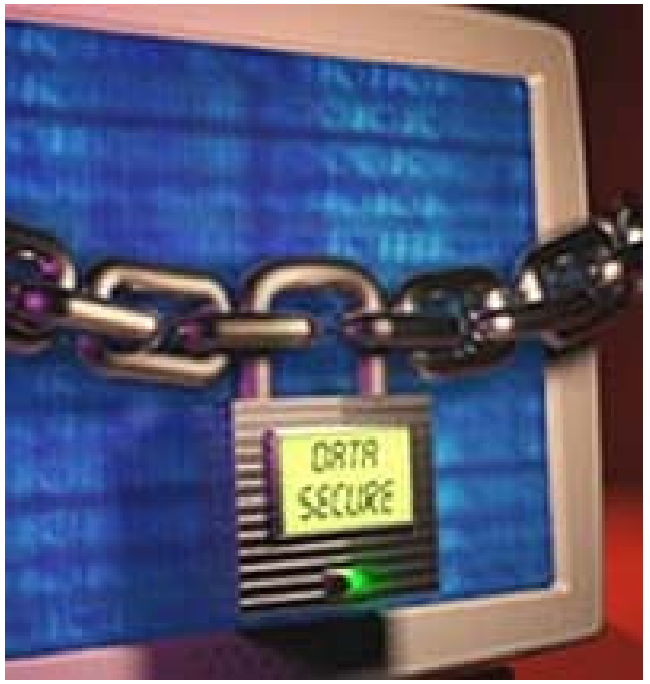


- Oft müssen Informationssysteme einen Teil der gespeicherten  Daten öffentlich zur Verfügung stellen.
- Dies kann durch direkten Zugriff oder durch generierte Reports geschehen.
- Datenschutz-Probleme entstehen, wenn das System gleichzeitig sensible Daten geheim halten soll.
- Deshalb soll das System automatische Test zur Verfügung stellen, um zu verifizieren, dass die sensiblen Daten tatsächlich nicht öffentlich zugänglich sind.

# Fragen



- Was bedeutet Data Privacy?
- Wie testet man, ob Data Privacy gewährleistet ist?
- Wie kann man Data Privacy sicherstellen?



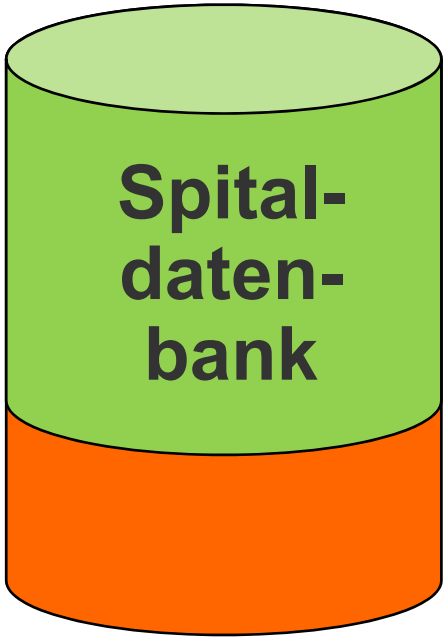
# Setting: Spitalinformationssystem



Privatsphäre wird mit Hilfe von Abfragen spezifiziert

**Öffentliche Information:**
Welche Diagnosen wurden gestellt?

**Vertrauliche Information:**
Mit welcher Diagnose war Max im Spital?



# Was bedeutet Data Privacy?



- **Provable Privacy:**
  Vertrauliche Information kann nicht hergeleitet werden

- **k-Anonymität:**
  Es gibt zu jeder vertraulichen Anfrage mindestens k mögliche Antworten

- **Perfect Privacy:**
  A priori Wahrscheinlichkeiten über vertrauliche Informationen werden nicht verändert

# Beispiel



**Annahme:** aus den öffentlichen Informationen folgt, dass Max mit der Diagnose **Blinddarm** oder mit der Diagnose **Beinbruch** im Spital war.

| | |
|---|---|
| Provable Privacy ☺ | 3-Anonymität ☹ |
| 2-Anonymität ☺ | Perfect Privacy ☹ |

Im Folgenden betrachten wir nur Provable Privacy.

# Informationssystem eines Spitals

UNIVERSITÄT BERN

| Name | Ort | Diagnose | Aufenthalts-dauer |
|---|---|---|---|
| Bob | Bern | 1 | 2 |
| Bob | Bern | 2 | 8 |
| Eva | Bern | 1 | 3 |
| Lea | Thun | 2 | 10 |
| Max | Berken | 3 | 23 |
| Zoe | Thun | 4 | 6 |

# Datenschutz durch Views

UNIVERSITÄT BERN

| Name | Ort |
|---|---|
| Bob | Bern |
| Eva | Bern |
| Lea | Thun |
| Max | Berken |
| Zoe | Thun |

# Kombination aus Views kann Privacy verletzen



| Name | Ort |
|---|---|
| Bob | Bern |
| Eva | Bern |
| Lea | Thun |
| Max | Berken |
| Zoe | Thun |

| Ort | Diagnose |
|---|---|
| Berken | 3 |
| Bern | 1 |
| Bern | 2 |
| Thun | 2 |
| Thun | 4 |

# Privacy durch Aggregation

UNIVERSITÄT BERN

| Diagnose | Anzahl | Durchschnitt Aufenthaltsdauer |
|---|---|---|
| 1 | 2 | 1.5 |
| 2 | 2 | 9 |
| 3 | 1 | 23 |
| 4 | 1 | 6 |

BFS Statistik der Krankenhäuser:
Von den 7399 verschiedenen Diagnosen wurden 797 nur einmal gestellt (Daten 2006)

# Privacy Tests



Wir haben ein Verfahren entwickelt, mit dem effizient überprüft werden kann, ob unter jeder möglichen Kombination der öffentlichen Views Privacy gewährleistet ist.

**Idee:** Konstruiere eine Datenbank, die explizit alle Informationen enthält, welche aus den öffentlichen Views folgen. Falls die Abfrage nach vertraulichen Daten in dieser Datenbank keine eindeutige Antwort liefert, so ist Privacy gewährleistet.

# Privacy Tests (2)



Falls die Daten nicht in Tabellen abgespeichert sind (sondern eine komplexere Struktur aufweisen), so kann es kein Verfahren geben, das in allen Fällen effizient Privacy testen kann.

**Idee:** Nehmen wir an, wir hätten so ein Verfahren. Dann könnte man damit effizient alle Probleme lösen, welche exponentiell viel Zeit zur Lösung benötigen.
Widerspruch.

# Privacy Tests (3)



Die vorhergehende Folie bezieht sich auf ein allgemeines Verfahren.

Falls aber die komplexe Struktur der Daten eine bestimmte Form hat, so ist Privacy garantiert. Man kann effizient überprüfen, ob diese Form gegeben ist.

**Idee:** Wir testen, ob die Struktur so ist, dass die privaten Daten lokal sind (d.h. sie sind in einem bestimmten Sinn unabhängig von den öffentlichen Daten). Solche Strukturtests sind effizient durchführbar.

# Data Privacy

UNIVERSITÄT BERN

- Es gibt verschiedene Begriffe von Data Privacy:
  Provable Privacy, k-Anonymität, Perfect Privacy
- Privacy kann durch Views oder Aggregation gewährleistet werden
- In relationalen DBs kann Provable Privacy effizient getestet werden
- In Knowledge Base Systems gibt es keinen allgemeinen Test der effizient ist.
- Aber es gibt ein Kriterium, das Privacy garantiert und das effizient überprüft werden kann.

# Daten Design (3)

UNIVERSITÄT BERN

- Wer braucht Zugriff auf welche Daten?
- Welche Views müssen definiert werden?
- Welche Aggregate müssen gebildet werden?

- Ist Datenschutz gewährleistet?

# Zusammenfassung

UNIVERSITÄT BERN

- Welche Tabellen braucht es (mit welchen Attributen)?
- Welche funktionalen Abhängigkeiten gibt es?
- Ist die Datenkonsistenz gewährleistet?

- Welche Abfragen kommen vor?
- Nach welchen Attributen wird dabei gesucht?
- Welche Indexe müssen gebildet werden?

- Wer braucht Zugriff auf welche Daten?
- Welche Views / Aggregate müssen definiert werden?
- Welche Aggregate müssen gebildet werden?

# Fragen