

Datenbanken und Datensicherheit

Prof. Dr. rer. nat. Peter Gerwinski

23. November 2023

Datenbanken und Datensicherheit

<https://gitlab.cvh-server.de/pgerwinski/dbs>

- 1 Einführung
- 2 Kurzeinführung Unix
- 3 Kurzeinführung TCP/IP
- 4 Relationale Datenbanken
 - 4.1 Einführung in DBMS
 - 4.2 Einführung in SQL
 - 4.3 Normalformen
 - 4.4 Verknüpfungen von Tabellen
 - 4.5 Sichten
 - 4.6 Schlüsselfelder
 - 4.7 Datensicherung

...



Änderungen
vorbehalten

4 Relationale Datenbanken

4.2 Einführung in SQL

Datenbank-Abfragesprache: Structured Query Language (SQL)

Literatur: z. B. https://de.wikibooks.org/wiki/Einführung_in_SQL

Wichtige SQL-Befehle:

- **CREATE** – Datenbanken, Tabellen usw. anlegen
- **DROP** – Datenbanken, Tabellen usw. löschen
- **SELECT** – Daten abfragen
- **INSERT INTO ... VALUES** – Daten eingeben
- **UPDATE** – Daten ändern
- **DELETE FROM** – Daten löschen

4 Relationale Datenbanken

4.3 Normalformen

Problem: Schlecht angelegte Datenbanken werden schnell inkonsistent.
Beliebte Fehler:

- Speichern von mehreren Daten in demselben Tabelleneintrag
- Speichern von denselben Daten in verschiedenen Tabelleneinträgen
- implizite Zusammenhänge
- voneinander unabhängige Zusammenhänge in derselben Tabelle

Literatur: z. B. [https://de.wikipedia.org/wiki/Normalisierung_\(Datenbank\)](https://de.wikipedia.org/wiki/Normalisierung_(Datenbank))

4 Relationale Datenbanken

4.3 Normalformen

Problem: Schlecht angelegte Datenbanken werden schnell inkonsistent.
Beliebte Fehler:

- Speichern von mehreren Daten in demselben Tabelleneintrag
→ 1. Normalform
- Speichern von denselben Daten in verschiedenen Tabelleneinträgen
→ 2. Normalform
- implizite Zusammenhänge
→ 3. Normalform und Boyce-Codd-Normalform
- voneinander unabhängige Zusammenhänge in derselben Tabelle
→ 4. und 5. Normalform

Lösung: Normalformen

Literatur: z. B. [https://de.wikipedia.org/wiki/Normalisierung_\(Datenbank\)](https://de.wikipedia.org/wiki/Normalisierung_(Datenbank))

4 Relationale Datenbanken

4.4 Verknüpfungen von Tabellen

Problem: Gut angelegte Datenbanken (→ Normalformen) sind stark aufgesplittet.

Wie kann man sie weiterhin effizient benutzen?

Lösung: Verknüpfungen von Tabellen

SQL-Befehl: JOIN

Literatur: z. B. <https://de.wikipedia.org/wiki/SQL>

4 Relationale Datenbanken

4.4 Verknüpfungen von Tabellen

Was machen wir mit Tabelleneinträgen,
bei denen die **ON**-Bedingung nicht erfüllt ist?

```
SELECT <Feld[er]> FROM <Tabelle1> [INNER] JOIN <Tabelle2>  
    ON <Tabelle1>.<Feld> = <Tabelle2>.<Feld>;
```

→ weglassen: **[INNER] JOIN**

```
SELECT <Feld[er]> FROM <Tabelle1> LEFT JOIN <Tabelle2>  
    ON <Tabelle1>.<Feld> = <Tabelle2>.<Feld>;
```

→ linke Tabelle trotzdem anzeigen (mit **NULL**-Einträgen): **LEFT JOIN**
(analog: **RIGHT JOIN** für rechte Tabelle)

```
SELECT <Feld[er]> FROM <Tabelle1> FULL JOIN <Tabelle2>  
    ON <Tabelle1>.<Feld> = <Tabelle2>.<Feld>;
```

→ beide Tabellen trotzdem anzeigen (mit **NULL**-Einträgen): **FULL JOIN**

4 Relationale Datenbanken

4.5 Sichten

```
SELECT <Feld[er]> FROM <Tabelle1> [INNER] JOIN <Tabelle2>  
    ON <Tabelle1>.<Feld> = <Tabelle2>.<Feld>;
```

→ Wir betrachten beide Tabellen zusammen als eine große Tabelle.

```
CREATE VIEW <Sicht> AS  
    SELECT <Feld[er]> FROM <Tabelle1> JOIN <Tabelle2>  
        ON <Tabelle1>.<Feld> = <Tabelle2>.<Feld>;  
SELECT <Feld[er]> FROM <Sicht> [WHERE ...];
```

→ Wir sprechen das Ergebnis genau wie eine Tabelle an.

→ Es ist möglich, ohne Verlust an Komfort alle Daten in Normalform zu halten.

4 Relationale Datenbanken

4.6 Schlüsselfelder

```
CREATE TABLE tabelle1 (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    ...  
);  
CREATE TABLE tabelle2 (  
    ...  
    tabelle1_id INT,  
    ...  
    FOREIGN KEY(tabelle1_id) REFERENCES tabelle1(id)  
);
```

- Dem DBMS mitteilen, welche Felder für **JOIN** vorgesehen sind.
- Das DBMS kann mit auf Konsistenz achten.

4 Relationale Datenbanken

4.7 Datensicherung

```
$ pg_dump --clean -h <Rechner> -U <User> -W <Datenbank>
```

- Ausgabe des gesamten Datenbankinhalts als SQL-Quelltext zur Standardausgabe

—> keine Probleme mit sich evtl. ändernden Binärformaten

- Es ist möglich, den Inhalt direkt in einer Pipe weiterzuverarbeiten (z. B. zu komprimieren).
- Zurückspielen: mit `psql`

```
$ psql -h <Rechner> -U <User> -W <Datenbank> \  
    < <Ausgabe von pg_dump>
```

- analog für **MariaDB**: `mariadb-dump`