# Einführung in Datenbanken

## Kapitel 1: Grundlegende Begriffe

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2020/21

http://www.informatik.uni-halle.de/~brass/db20/

# Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- Die folgenden Begriffe erklären und richtig gebrauchen:
  - Datenbank-Zustand
  - Anfrage
  - Update
  - Datenbank-Schema
  - Datenmodell (mit einigen Beispielen)
  - DBMS (Datenbank-Managementsystem)
  - DDL, DML, Anfragesprache
- Ein einfaches Beispiel für ein DB-Schema und einen DB-Zustand im relationalen Datenmodell geben.
- Ein DBMS mit einem Webserver vergleichen (Client-Server).

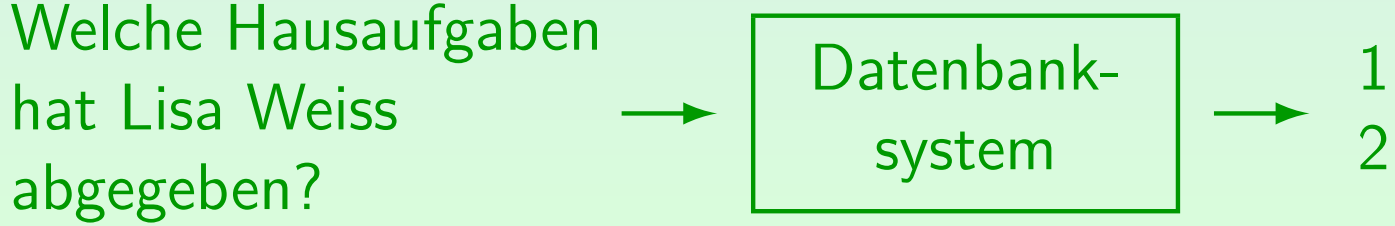
# Inhalt

# Aufgabe einer Datenbank (1)

- Was ist eine Datenbank? Das ist eine schwierige Frage.
  Es gibt keine präzise und allgemein anerkannte Definition.

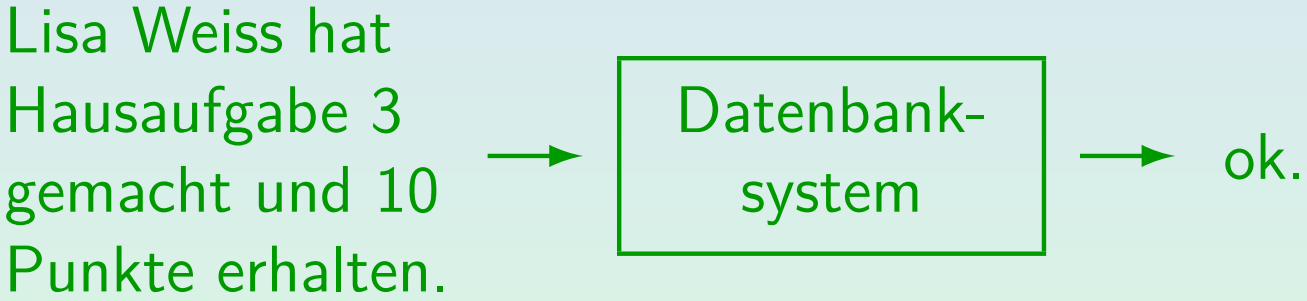
  Elmasri/Navathe (2000): „A database is a collection of related data.
  By data we mean known facts that can be recorded and habe an implicit meaning.“

- Naive Näherung: Die Hauptaufgabe eines Datenbanksystems (DBS) ist die Beantwortung gewisser Fragen über eine Teilmenge der realen Welt, z.B.



# Aufgabe einer Datenbank (2)

- Das DBS dient nur als Speicher für Informationen.
  Die Informationen müssen zuerst eingegeben und dann immer aktualisiert werden.




- Ein Datenbanksystem ist eine Computer-Version eines Karteikastens (nur mächtiger).
- Eine Tabellenkalkulation ist (fast) ein kleines DBS.

# Aufgabe einer Datenbank (3)

- Normalerweise machen Datenbanksysteme keine besonders komplizierten Berechnungen auf den gespeicherten Daten.

  Aber es gibt z.B. Wissensbanken, Data Mining-Werkzeuge, „Big Data Analytics“.

- Sie können jedoch die gesuchten Daten schnell in einer großen Datenmenge finden (Gigabytes oder Terabytes – größer als der Hauptspeicher).

  Im Unterschied zu Suchmaschinen und Information Retrieval Werkzeugen gibt man präzise Bedingungen für die gesuchten Daten an. Bei Suchmaschinen kann dagegen ein Text mehr oder weniger gut auf die Anfrage passen.

- Sie können auch Daten aggregieren/kombinieren, um eine Antwort aus vielen Einzeldaten zusammenzusetzen (z.B. Summe der Punkte von Lisa Weiss).

# Aufgabe einer Datenbank (4)

- Obige Frage

  „Welche Hausaufgaben hat Lisa Weiss abgegeben?“

  war in natürlicher Sprache (Deutsch).

- Das Verstehen natürlicher Sprache durch Computer ist ein schwieriges Problem (Missverständnisse).

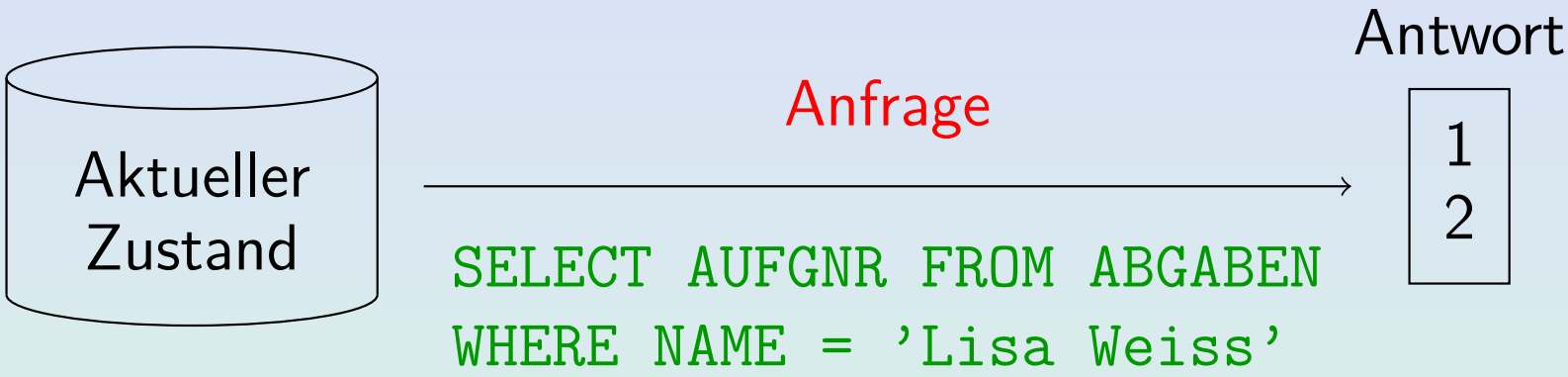
- Daher werden Fragen („Anfragen“) normalerweise in formaler Sprache gestellt, heute meist in SQL.

  Man kann SQL als spezielle Programmiersprache für Anfragen an Datenbanken verstehen. Im Gegensatz zu Sprachen wie Pascal, C oder Java kann man in SQL aber keine beliebigen Algorithmen notieren.

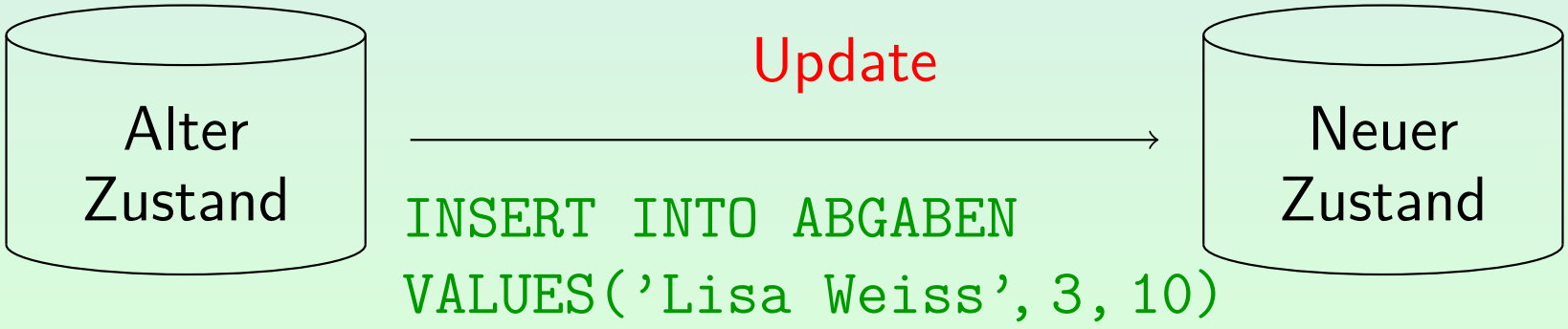
- Einige DBS erlauben natürlichsprachliche Anfragen.

# Zustand, Anfrage, Update

- Menge der gespeicherten Daten = DB-Zustand:




- Eingabe, Modifikation oder Löschung von Daten ändert den Zustand:



# Inhalt

# Strukturierte Information (1)

- Normalerweise kann eine Datenbank nur Informationen einer vorher definierten Struktur speichern:


- Da die Daten strukturiert sind (nicht nur Text), sind komplexere Auswertungen möglich, z.B.:

  Gib für jede Hausaufgabe aus, wie viele Studierende sie bearbeitet haben, sowie die durchschnittlich erreichte Punktzahl.

# Strukturierte Information (2)

- Eigentlich speichert ein DBS nur Daten (Zeichenketten, Zahlen, etc.), und keine Informationen.
- Daten werden durch Interpretation zu Information.
- Begriffe wie „Studierende“ und „Aufgaben“ müssen
  - dem System formal bekanntgemacht (deklariert) werden,
  - sowie für den Nutzer erklärt/definiert werden.

Natürlich ist es möglich, eine Datenbank zu entwickeln, in der beliebige Texte gespeichert werden können. Dann kann aber das DBS nur nach Substrings suchen und keine komplexere Anfragen bearbeiten. Je mehr ein DBS über die Struktur der Daten „weiß“, desto besser kann es den Nutzer bei der Suche und Auswertung unterstützen. Beispiel: Bücher von Goethe von Büchern über Goethe unterscheiden (braucht AUTOR und TITEL-Feld).

# Zustand vs. Schema (1)

Datenbank-Schema:

- Formale Definition der Struktur des DB-Inhalts.
- Legt mögliche Datenbankzustände fest.
- Nur einmal definiert (wenn die DB erstellt wird).

  In der Praxis kann es manchmal notwendig sein, das Schema zu verändern. Das passiert jedoch selten und ist problematisch.

- Entspricht der Variablendeklaration (Informationen über Datentypen).

  Z.B. wenn eine Variable `i` als `short int` (16 Bit) deklariert ist, sind die möglichen Zustände von `i` normalerweise `-32768 .. +32767`.

# Zustand vs. Schema (2)

Datenbank-Zustand (Ausprägung eines Schemas):

- Beinhaltet die aktuellen Daten,
  dem Schema entsprechend strukturiert.
- Ändert sich oft
  (wenn Datenbank-Informationen geändert werden).
- Entspricht dem Inhalt/Wert der Variablen.

  Z.B. hat `i` im aktuellen Zustand $s$ den Wert `5`. Ein Update, z.B.
  `i := i + 1`, verändert den Zustand zu $s'$, in dem `i` den Wert `6` hat.

# Zustand vs. Schema (3)

- Im relationalen Datenmodell sind die Daten in Form von Tabellen (Relationen) strukturiert (oft mehrere/viele).
- Schema einer Tabelle: Tabellen-Name, Folge von benannten Spalten („Attribute“), jeweils mit Datentyp.
- Zustand: Menge von Zeilen („Tupel“)

| ABGABEN | | |
|---|---|---|
| NAME | AUFGNR | PUNKTE |
| Lisa Weiss | 1 | 10 |
| Lisa Weiss | 2 | 8 |
| Daniel Sommer | 1 | 9 |
| Daniel Sommer | 2 | 9 |

DB-Schema

DB-Zustand

# Zum Ausprobieren (1)

- Wenn Sie ein System wie PostgreSQL installiert haben, können Sie die obige Tabelle so anlegen:

```
CREATE TABLE ABGABEN(
       NAME    VARCHAR(40)   NOT NULL,
       AUFGNR  NUMERIC(2)    NOT NULL,
       PUNKTE  NUMERIC(3,1)  NOT NULL,
       PRIMARY KEY(NAME, AUFGNR));
```

`VARCHAR(40)` ist ein String mit bis zu 40 Zeichen, `NUMERIC(2)` eine ganze Zahl mit zwei Dezimalstellen (`-99..+99`). `NUMERIC(3,1)` hat drei Dezimalstellen, davon eine nach dem Komma (bis `99.9`). `NOT NULL` verbietet leere Einträge. Der `PRIMARY KEY` stellt sicher, dass es zu einem/r Studierenden und einer Aufgabennummer nur einen Eintrag gibt. Alles wird noch genau erklärt! [Code]

- Daten einfügen:

```
INSERT INTO ABGABEN VALUES('Lisa Weiss', 1, 10);
```

# Zum Ausprobieren (2)

- Einfache SQL-Anfragen haben die Form:

```
SELECT ⟨Spalten⟩
FROM   ⟨Tabelle⟩
WHERE  ⟨Bedingung⟩
```

- Beispiel:

```
SELECT AUFGNR, PUNKTE
FROM   ABGABEN
WHERE  NAME = 'Lisa Weiss';
```

Die Zeilenaufteilung ist egal. SQL ist eine formatfreie Sprache wie Java. Groß-/Kleinschreibung ist nur für die Daten (wie `'Lisa Weiss'`) relevant, z.B. nicht für `SELECT`. Das „;“ am Ende gehört eigentlich nicht dazu.

- Anzeigen der ganzen Tabelle: `SELECT * FROM ABGABEN;`
- Tabelle löschen: `DROP TABLE ABGABEN;`

# Übung

- Der Professor möchte ein Programm entwickeln, das an jeden Studenten eine E-Mail folgender Art verschickt:

```
Sehr geehrte Frau Weiss,
folgende Bewertungen sind über Sie gespeichert:
    - Aufgabe 1: 10 Punkte
    - Aufgabe 2:  8 Punkte
Melden Sie sich bitte, falls ein Fehler vorliegt.
Mit freundlichen Grüßen, ...
```

- Muss er dazu obige Tabelle erweitern?
  Sollte er vielleicht die Daten auf mehrere Tabellen verteilen?

# Inhalt

# Datenmodell (1)

- Ein Datenmodell definiert
  - eine Menge $\mathcal{SCH}$ möglicher DB-Schemata,

    Das ist die Syntax (ggf. auch abstrakte Syntax mit mathematischen Strukturen).
  - für jedes DB-Schema $\mathcal{S} \in \mathcal{SCH}$ eine Menge $\mathcal{ST}(\mathcal{S})$ möglicher DB-Zustände.

    Die Menge der möglichen DB-Zustände ist die Semantik eines Schemas.
  - Eine Menge $\mathcal{A}$ möglicher Antworten auf Anfragen.

    Datenstrukturen wie z.B. Tabellen. Anfragesprache: Siehe Folie 27.
- Oft (aber nicht immer) ist ein Datenmodell durch eine Menge von Basis-Datentypen parametrisiert.

  Z.B. für relationales Modell nicht wichtig, ob die Tabelleneinträge nur Zeichenketten oder auch Zahlen, Datums- oder Zeitwerte usw. sein können.

# Datenmodell (2)

- Ein Datenmodell definiert also normalerweise Typ-Konstruktoren, um komplexe Datenstrukturen aus elementaren Datentypen zu bilden.

  Z.B. `RECORD`/`STRUCT`, `SET`, `LIST`, `ARRAY`. Relationales Modell: `SET(STRUCT)`.

- Oft kann man im Schema auch mittels Integritätsbedingungen (engl. „Constraints“) die möglichen Zustände einschränken.

  Z.B.: „Die Punktzahl muss $\geq 0$ sein.“ Oder ein sogenannter Schlüssel: „Es kann keine zwei Einträge für den gleichen Studenten und die gleiche Aufgabe geben.“

- Es ist auch sehr typisch, dass DB-Schemata Symbole/Namen einführen (wie z.B. Tabellennamen, Spaltennamen), die der DB-Zustand auf Werte oder Funktionen abbildet.

  Diese Namen verwendet der Benutzer dann in Anfragen/Updates.

# Datenmodell (3)

- Es gibt keine allgemein anerkannte formale Definition für den Begriff „Datenmodell“.

  Obige Definition ist mein Vorschlag. Man könnte mehr ins Detail gehen, aber das würde darin enden, Oracle 8.1.3 als ein Datenmodell zu definieren. Die meisten Bücher behandeln den Begriff sehr ungenau.

- Man könnte speziell darüber streiten, ob die Anfragesprache zum Datenmodell gehört.

  Z.B. gab es am Anfang kleine PC-DBMS, die Daten in Tabellen strukturierten, aber keine Anfragen zuließen, die Daten mehrerer Tabellen kombinierten. Diese Systeme galten als nicht ganz relational. Um die Ausdrucksstärke verschiedener Anfragesprachen für ein Datenmodell zu vergleichen, muss man beides unterscheiden. Außerdem wird das ER-Modell in Vorlesungen meist ohne Anfragesprache gelehrt.

# Datenmodell (4)

- Data Definition Language (DDL):
  Sprache zur Definition von Datenbank-Schemata.
- Data Manipulation Language (DML):
  Sprache für Anfragen und Updates.

  SQL, die Standardsprache für das relationale Modell, kombiniert DDL und DML. Der Anfrage-Teil der DML wird Anfragesprache (QL) genannt.
  Er ist meist komplizierter als der Update-Teil. Aber Updates können auch Anfragen enthalten (um neue Werte zu ermitteln).
- Manchmal wird der Begriff „Datenmodell“ verwendet, wenn eigentlich „Datenbank-Schema“ korrekt wäre.

  Z.B. Unternehmens-Datenmodell: Schema für alle Daten einer Firma.

# Datenmodell (5)

## Beispiele für Datenmodelle:

- Relationales Modell
- Entity-Relationship-Modell (viele Erweiterungen)

  Klassen ohne Methoden („Entities“) und bidirektionale Beziehungen.
- Objekt-Orientiertes Datenmodell (z.B. ODMG)
- Objekt-Relationales Datenmodell
- XML (mit DTDs als Schema oder XML Schema Definitionen)

  Ähnlich wie HTML mit benutzerdefinierten „Tags“:

  `<DB><ABGABE><NAME>Lisa Weiss</NAME><AUFGNR>1</AUFGNR>...`
- Netzwerk-Modell (CODASYL) (historisch)
- Hierarchisches Modell (historisch)

# Datenmodell (6)

- Es gibt auch Datenmodelle (im weiteren Sinn), bei denen nicht explizit ein Schema deklariert wird.

  Formal hat $\mathcal{SCH}$ dann genau ein Element. D.h. im wesentlichen definiert das Datenmodell eine Menge möglicher Zustände.

- Das macht die Datenbank sehr flexibel.

  Erleichtert Änderungen, z.B. für agile Software-Entwicklung.

- Es kann aber auch zu Chaos in den Daten führen.

  Schlechte Datenqualität („Data Quality“) kann zu falschen oder ungenauen bzw. unvollständigen Ergebnissen führen. Dann braucht man „Data Cleaning“.

- Z.B. „Well-formed XML“: DOM-Schnittstelle

  Dieses Beispiel zeigt, dass man ein Datenbank-Schema (auch allgemein) als zusätzliche Einschränkung auffassen kann, die eine Typprüfung erlaubt. In relationalen DBen kann man Daten erst nach Schema-Definition speichern.

# NoSQL-Datenbanken (1)

- Ohne Schema. Konzentration auf DBMS-Teilfunktionalität.
- Graphdatenbanken, z.B. Neo4J.

  Die Datenbank ist im Prinzip ein gerichteter Graph wie in der Mathematik. Die Knoten können mit beliebig vielen (auch 0) „Labels“ versehen werden, das entspricht Klassen oder Rollen. Ausserdem kann man in Knoten „Properties“ abspeichern (Name-Wert-Paare). Kanten muss genau ein Typ zugeordnet werden (Kanten-Markierung), und auch hier können Eigenschaften zugeordnet werden (Name-Wert-Paare, z.B. Distanzen). Datenmodell: „Labeled Property Graph“.
- Dokument-Datenbanken, z.B. MongoDB.

  Datenbank besteht aus „Collections“ (Verzeichnissen), deren Elemente beliebige JSON-Dokumente sind (mit einer eindeutigen, ggf. systemgenerierten ID).

  ```
  {"Student": "Lisa Weiss", "AufgNr": 1, "Punkte": 10}
  ```
- Key-Value-Stores, z.B. Redis.

  Abbildung von Schlüsselwerten (Strings) auf verschiedene Datenstrukturen.

# NoSQL-Datenbanken (2)

A history of databases in No-tation:

- 1970: NoSQL = We have no SQL
- 1980: NoSQL = Know SQL
- 2000: NoSQL = No SQL!
- 2005: NoSQL = Not only SQL
- 2013: NoSQL = No, SQL!

[https://blog.jooq.org/2013/11/15/a-history-of-databases-in-no-tation/]
Blog von Lukas Eder.
Idee von Mark Madsen [https://twitter.com/markmadsen]
Bild von Edd Wilder-James:
[https://twitter.com/edd/status/400190499585544192/photo/1]

# Anfragesprachen zu einem Datenmodell

- Anfragen sind Zeichenketten einer Anfragesprache.

  Genauer: Für jedes Schema $\mathcal{S}$ gibt es eine Menge möglicher Anfragen $\mathcal{Q}_\mathcal{S}$.

- Die Semantik einer Anfrage $Q \in \mathcal{Q}_\mathcal{S}$ ist eine Abbildung von Datenbank-Zuständen $\mathcal{ST}(\mathcal{S})$ in Antworten $\mathcal{A}$.

- Bei Hausaufgaben oder der Klausur reicht es nicht, wenn die Anfrage im Beispiel-Zustand die erwartete Antwort liefert. Sie muss in allen Zuständen korrekte Antworten liefern.

- Zwei Anfragen sind äquivalent, wenn sie in allen Zuständen jeweils die gleichen Antworten liefern.

- Zwei Anfragesprachen sind gleich mächtig, wenn man Anfragen zwischen ihnen übersetzen kann, so dass man jeweils die gleiche Funktion von Zuständen in Antworten erhält.

# Inhalt

# DBMS (1)

Ein Datenbankmanagementsystem (DBMS) ist ein anwendungsunabhängiges Softwarepaket, das ein Datenmodell implementiert, d.h. Folgendes ermöglicht:

- Definition eines DB-Schemas für eine konkrete Anwendung,

  Da das DBMS anwendungsunabhängig ist, speichert es das Schema normalerweise auf der Festplatte, oft zusammen mit dem DB-Zustand (in speziellen „System-Tabellen“).

- Speichern eines DB-Zustands, z.B. auf Festplatte,
- Abfragen an den aktuellen DB-Zustand,
- Änderung des DB-Zustands.

# DBMS (2)

- Natürlich verwenden normale Nutzer kein SQL für den täglichen Umgang mit einer Datenbank.
- Sie arbeiten mit Anwendungsprogrammen, die eine bequemere / einfachere Benutzerschnittstelle für Standard-Aufgaben bieten.

  Z.B. ein Formular, in dem Felder ausgefüllt werden können.

- Intern enthält das Anwendungsprogramm jedoch ebenfalls SQL-Befehle (Anfragen, Updates),
  um mit dem DBMS zu kommunizieren.

# DBMS (3)

- Oft greifen eine ganze Reihe verschiedener Anwendungsprogramme auf eine zentrale Datenbank zu.
- Beispiel: Anwendungsprogramme für Punkte-DB:
  - Web-Interface für Studierende.

    Mit Funktionen zum Eintragen, Anschauen der Punkte, ...
  - Programm zum Eintragen von Punkten für Klausur und Hausaufgaben (für Übungsleiter/Tutor).
  - Programm, das einen Bericht (Übersicht) für den Professor ausdruckt, um Noten zu vergeben.

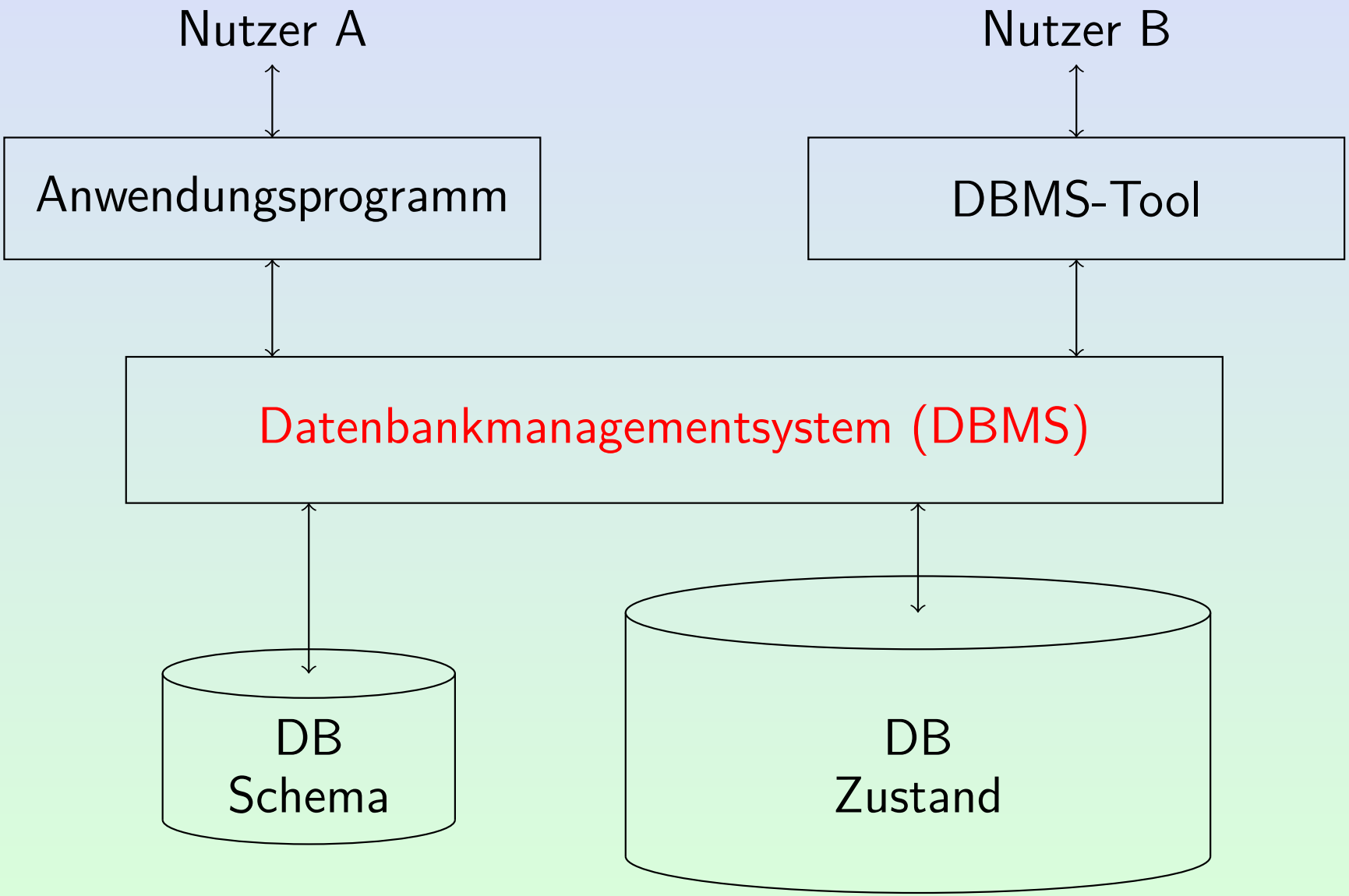
# DBMS (4)

- Mit einem DBMS wird normalerweise eine interaktive SQL-Schnittstelle mitgeliefert:
  - Man kann einen SQL-Befehl (z.B. Anfrage) eintippen
  - und bekommt dann das Ergebnis (Tabelle) auf dem Bildschirm angezeigt.
- Dieses Programm kommuniziert mit dem DBMS wie andere Anwendungsprogramme auch.

  Wenn man will, kann man so ein Programm auch selbst schreiben.

  Es schickt die SQL-Befehle als Zeichenketten an das DBMS.

# DBMS (5)


Nutzer A
Nutzer B
Anwendungsprogramm
DBMS-Tool
Datenbankmanagementsystem (DBMS)
DB
Schema
DB
Zustand

# Inhalt

# Etwas Datenbank-Vokabular

- Eine DB besteht aus DB-Schema und DB-Zustand.

  Z.B. sagt man „Hausaufgaben-Datenbank“. Es hängt vom Kontext ab, ob man den derzeitigen Zustand meint, oder nur das Schema und den Speicherplatz oder -ort (Netzwerk-Adresse des Servers).
  Es ist falsch, eine einzige Tabelle oder Datei Datenbank zu nennen, außer sie beinhaltet alle Daten des Schemas.

- Ein Datenbank-System (DBS) besteht aus einem DBMS und einer Datenbank.

  Aber „Datenbank-System“ wird auch als Abkürzung für DBMS genutzt.

- Ein Datenbank-Anwendungssystem besteht aus einem DBS und Anwendungsprogrammen.

  Ein Anwendungsprogramm unterstützt eine Aufgabe in der realen Welt und greift dazu auf die Datenbank zu (mit Anfragen und/oder Updates).

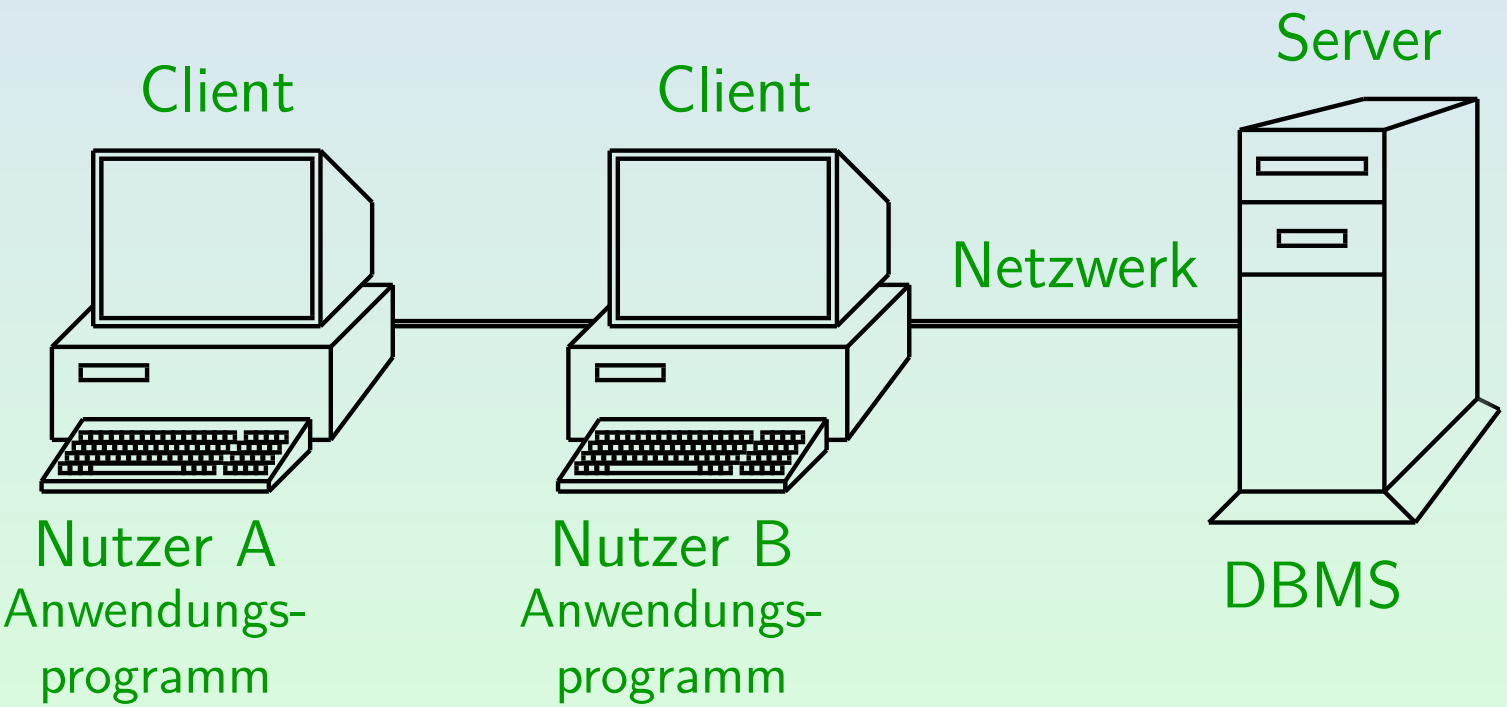
# DB-Anwendungssysteme (1)

- Oft greifen verschiedene Nutzer gleichzeitig auf die gleiche Datenbank zu.
- Das DBMS ist normalerweise ein im Hintergrund laufender Server-Prozess (ggf. mehrere), auf den über das Netzwerk mit Anwendungsprogrammen (Clients) zugegriffen wird.

  Einem Web-Server sehr ähnlich.
  Für einige kleine PC-DBMS gibt es nur ein einziges Programm, das als DBMS und als Interpreter für die Anwendungsprogramme dient.
  Es gibt auch „Embedded DBMS“, die als Bibliothek zum Programm hinzugebunden werden (ohne eigenen Server-Prozess). In diesen Fällen kann aber meist nur ein Nutzer gleichzeitig auf die Datenbank zugreifen.
- Man kann das DBMS als Erweiterung des Betriebssystems sehen (leistungsfähigeres Dateisystem).

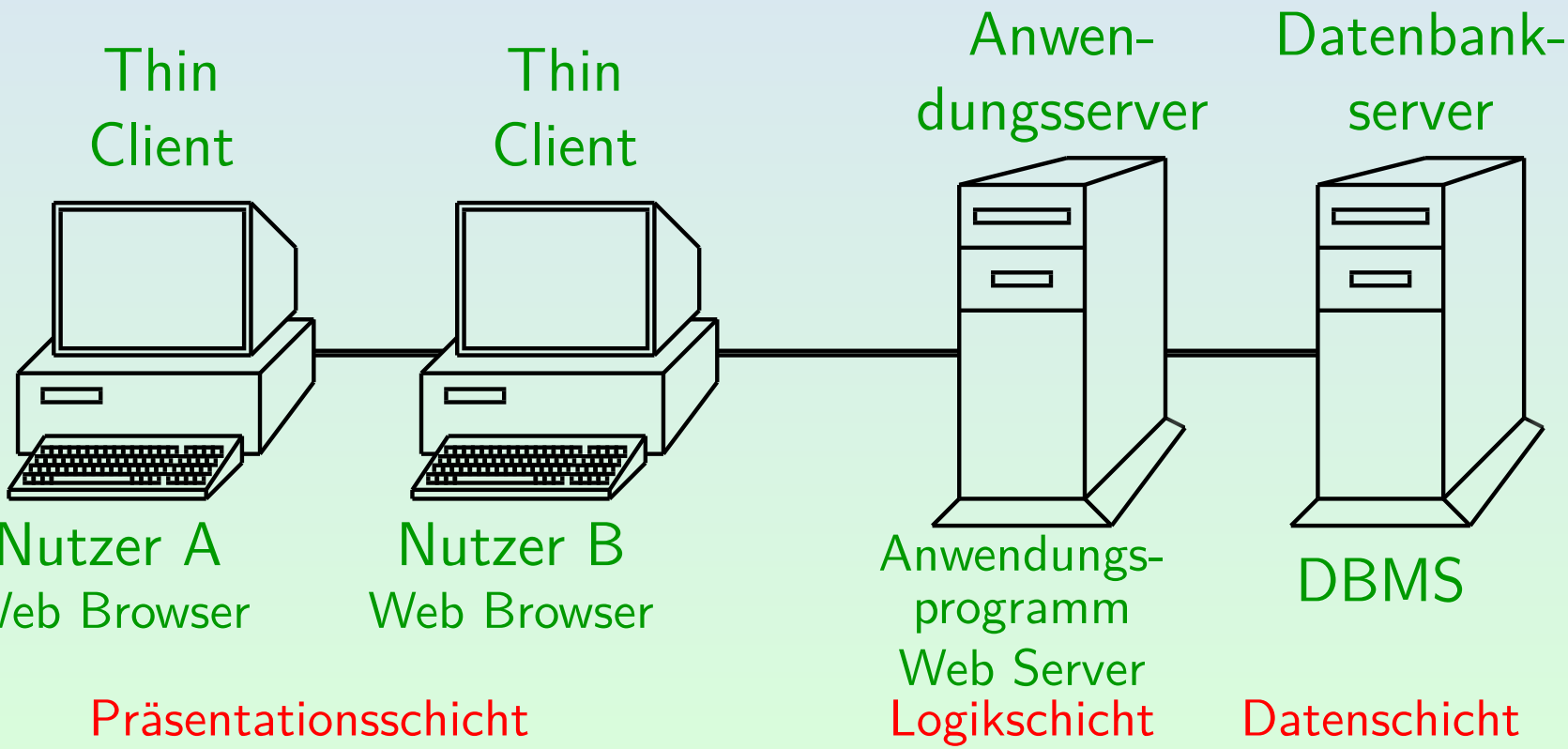
# DB-Anwendungssysteme (2)

Client-Server-Architektur:



# DB-Anwendungssysteme (3)

Dreischichten-Architektur (Three-Tier Architecture):



# DB-Anwendungssysteme (4)

Übung:

- Betrachten Sie die Datenbank einer Bank zur Verwaltung von Girokonten.
- Für welche Aufgaben benötigt die Bank Anwendungsprogramme, die auf die Datenbank zugreifen?
  - _____
  - _____
  - _____
  - _____