Einführung in Datenbanken

Kapitel 1: Einführung

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

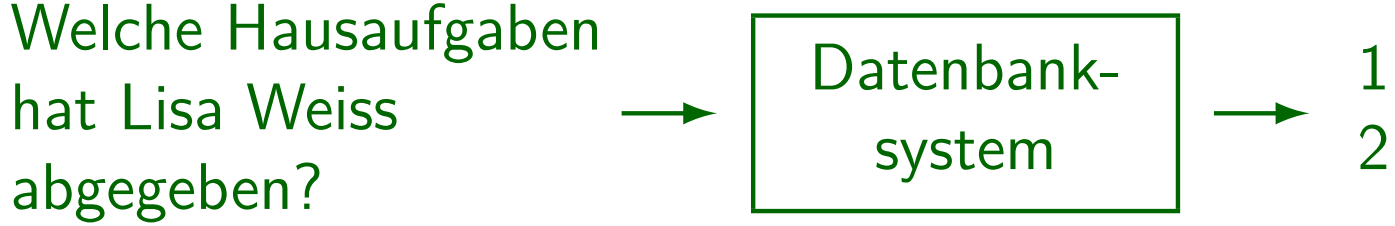
Wintersemester 2019/20

http://www.informatik.uni-halle.de/~brass/db19/

Inhalt

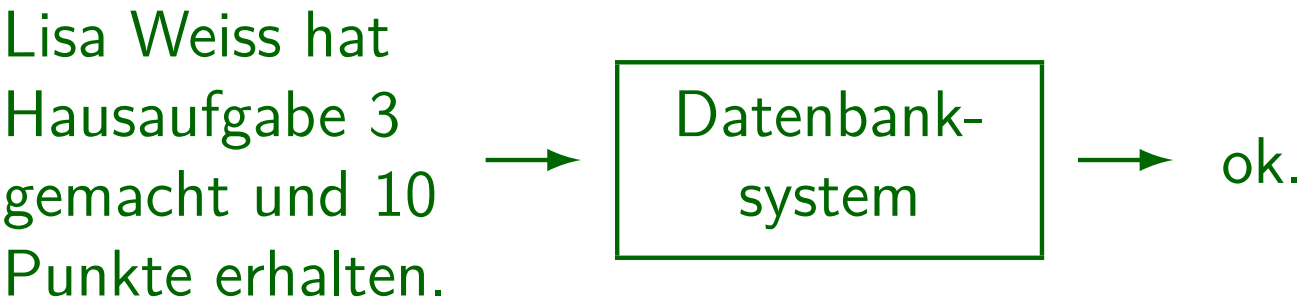
Aufgabe einer Datenbank (1)

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

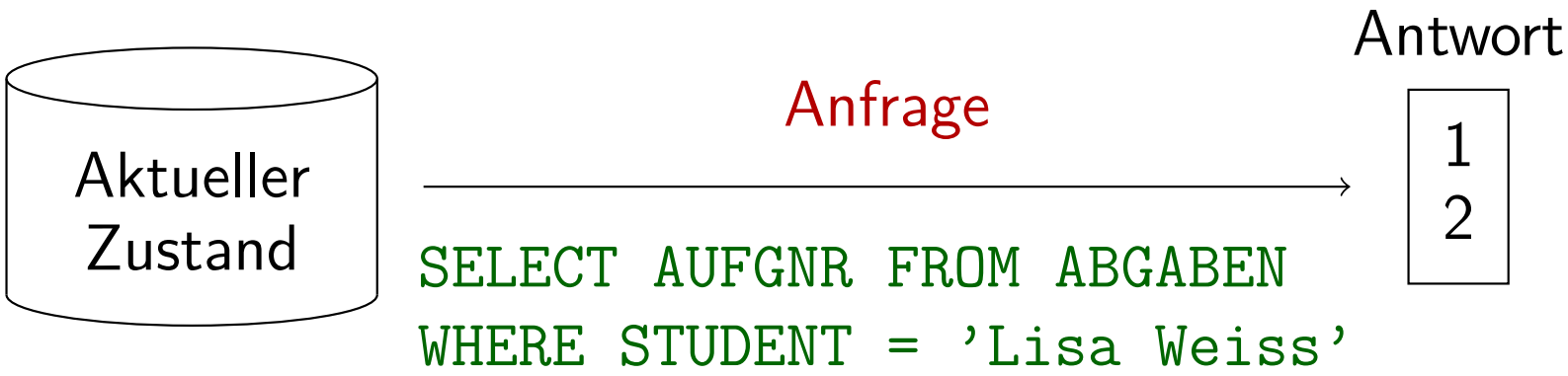
- Daher werden Fragen ("Anfragen") normalerweise in formaler Sprache gestellt, heute meist in SQL.

  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.

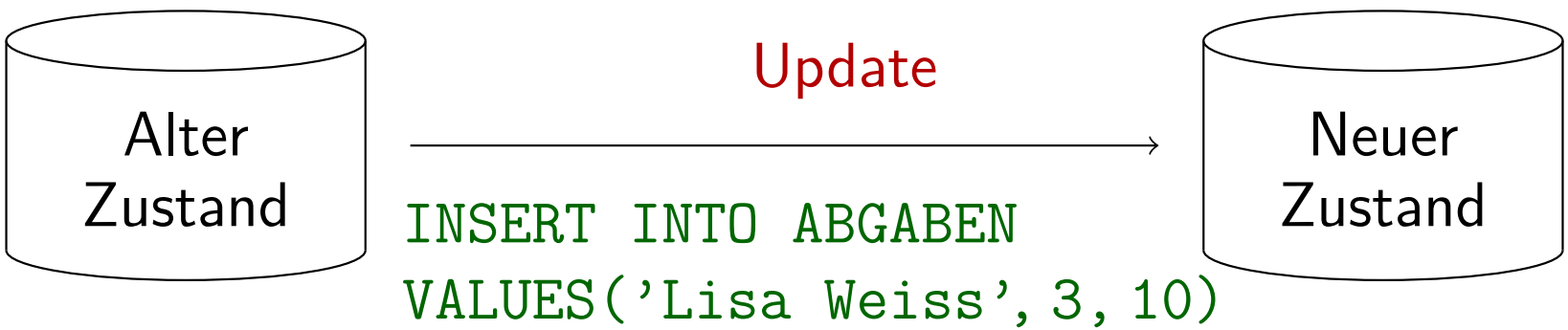
- Einige DBS erlauben natürlichsprachliche Anfragen.

Zustand, Anfrage, Update

- Menge der gespeicherten Daten = DB-Zustand:



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



Strukturierte Information (1)

- Jede Datenbank kann 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 Studenten 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 Studenten 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.
- Jede Tabelle hat: Name, Folge von benannten Spalten (Attribute) und Menge von Zeilen (Tupel)

ABGABEN		
STUDENT	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(
        STUDENT VARCHAR(40)  NOT NULL,
        AUFGNR  NUMERIC(2)   NOT NULL,
        PUNKTE  NUMERIC(3,1) NOT NULL,
        PRIMARY KEY(STUDENT, 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 Studenten 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  STUDENT = '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?

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 $\mathcal{S} \in \mathcal{SCH}$ ein Menge $\mathcal{ST}(\mathcal{S})$ möglicher DB-Zustände.

    Die Menge der möglichen DB-Zustände ist die Semantik eines Schemas.
- 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 ≥ 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 einführen (wie z.B. Tabellennamen), 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><STUDENT>Lisa Weiss</STUDENT><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.

Datenmodell (7)

NoSQL-Datenbanken (aktueller Hype), typisch auch ohne Schema:

- 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). Dieses Datenmodell heißt "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.

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 Studenten.

    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.

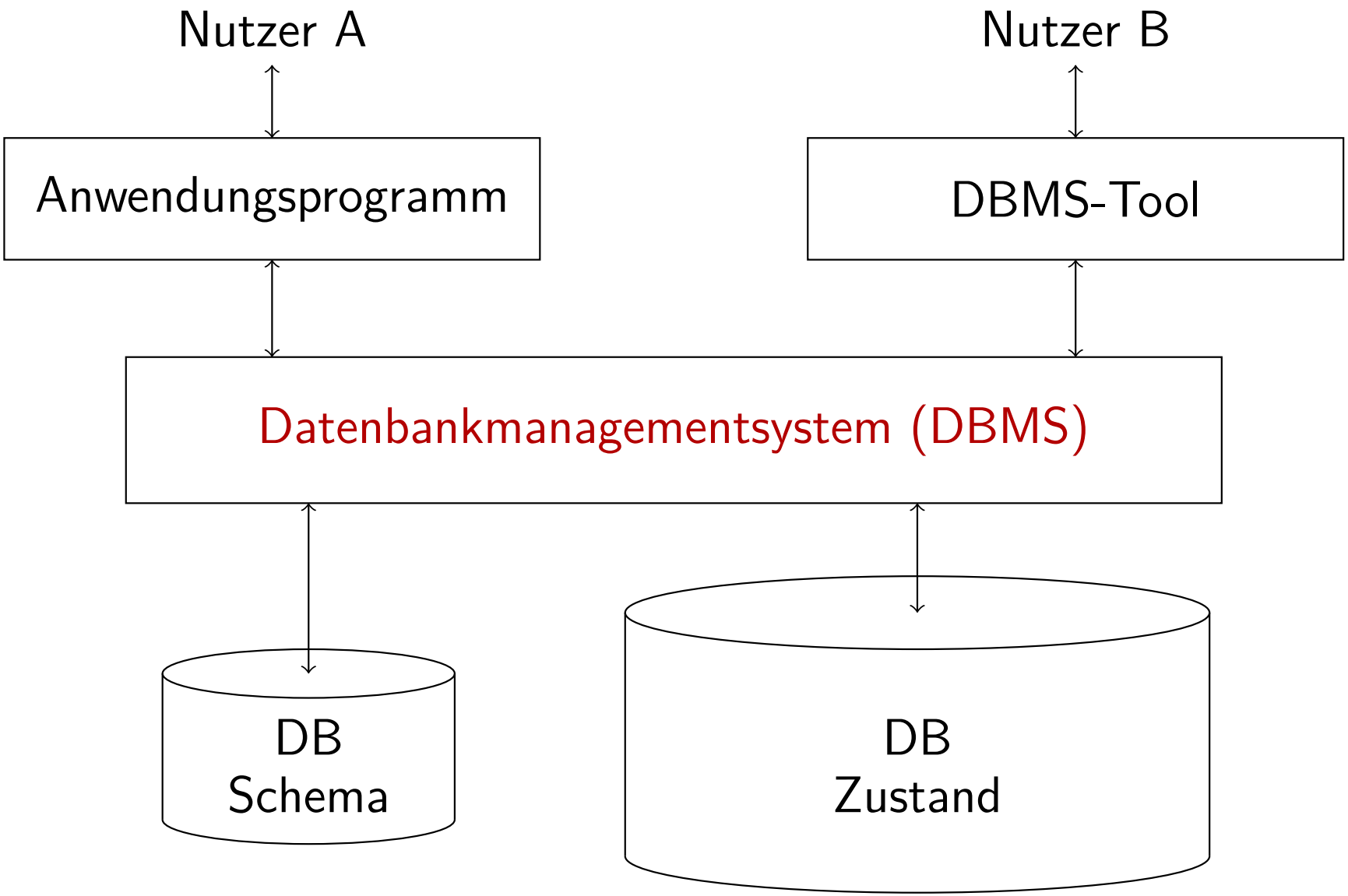
DBMS (4)

- 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

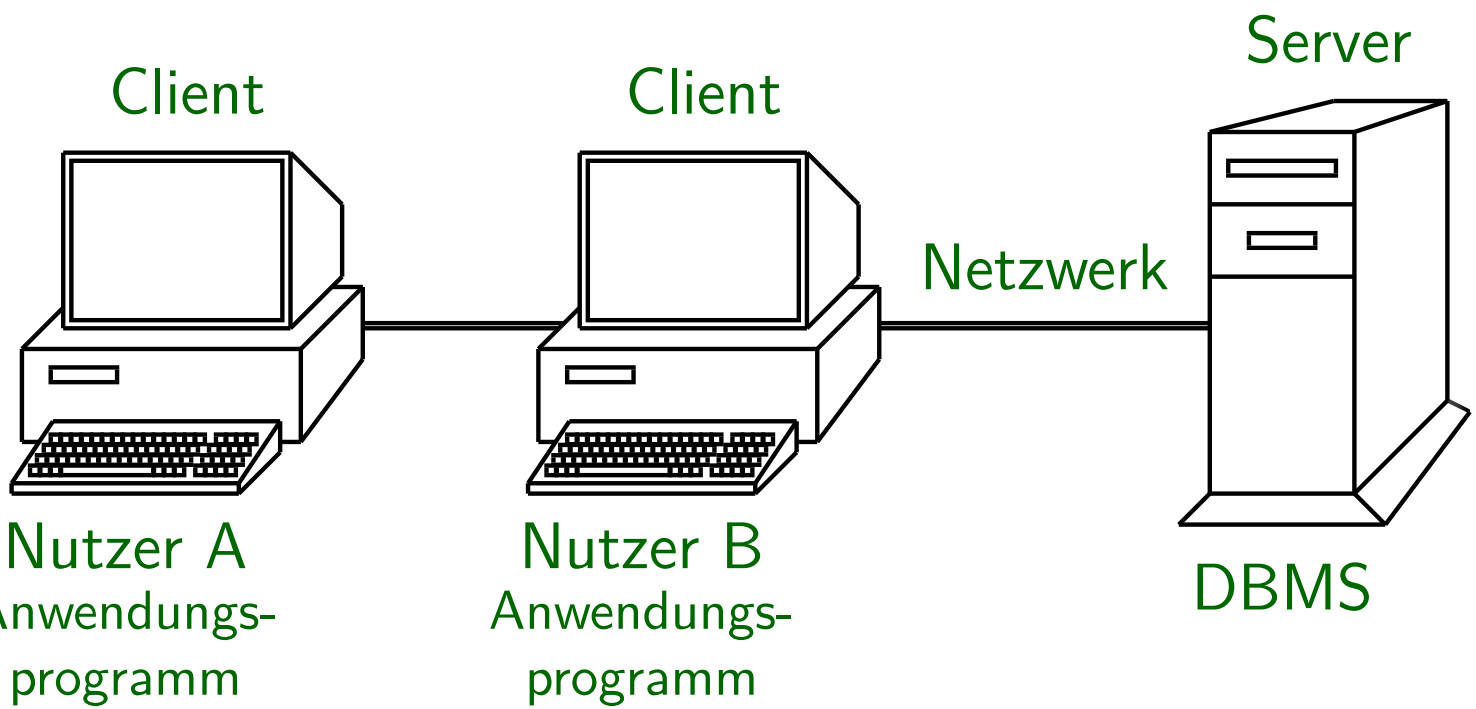
DB-Anwendungssysteme (1)

- 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).

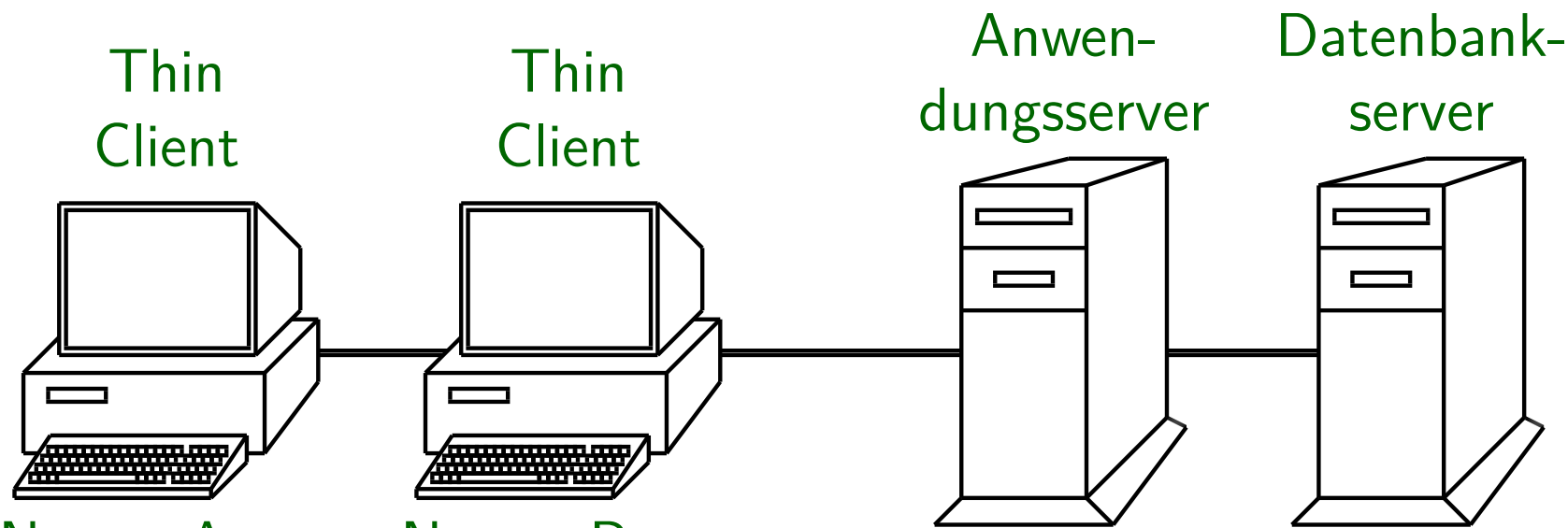
DB-Anwendungssysteme (2)

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?
  - _____
  - _____
  - _____
  - _____

DB-Anwendungssysteme (5)

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.

Inhalt

Persistente Daten (1)

Heute:

Morgen:

$\Rightarrow$ Die Ausgabe ist nur eine Funktion der Eingabe.

Persistente Daten (2)

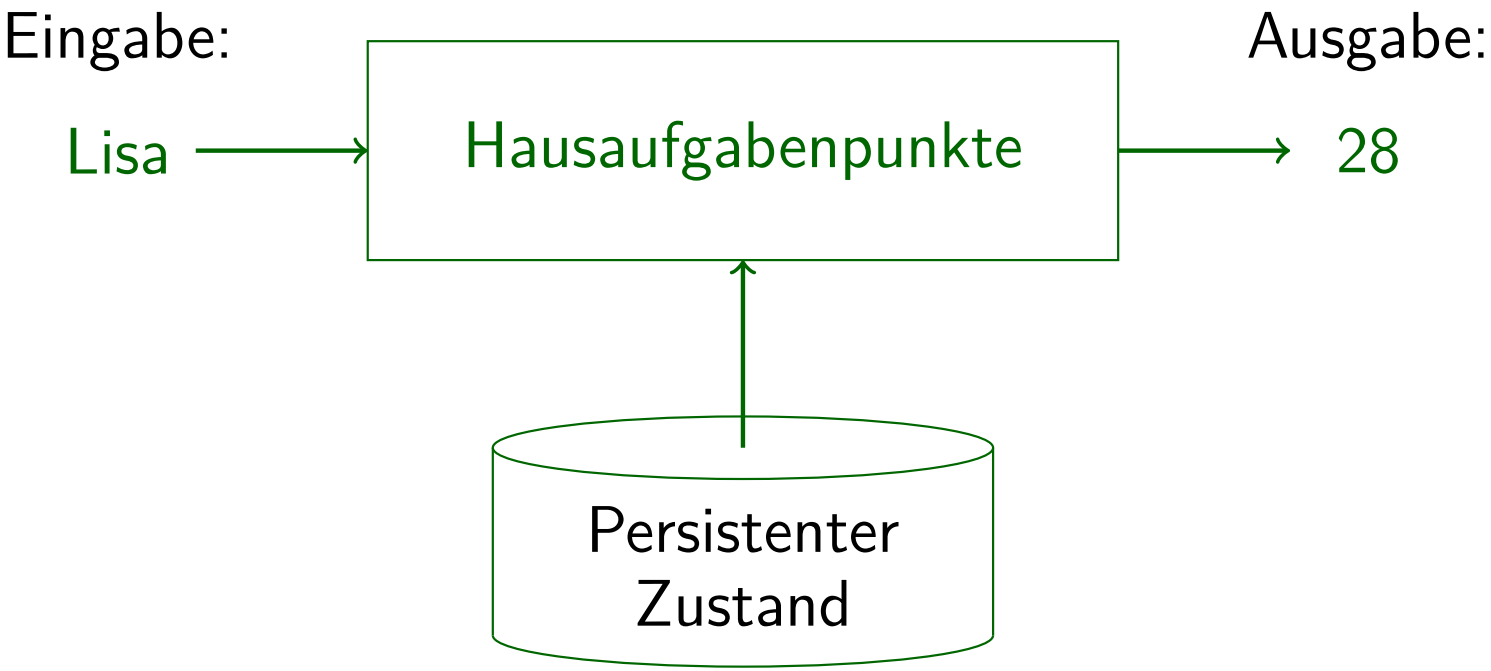
Heute:

Morgen:

$\Rightarrow$ Die Ausgabe ist eine Funktion der Eingabe und eines persistenten Zustands.

Daten heissen persistent (dauerhaft), wenn sie länger leben als ein Prozess, also z.B. auch nach einem Neustart des Betriebssystems noch vorhanden sind.

Persistente Daten (3)



Persistente Speicherung (1)

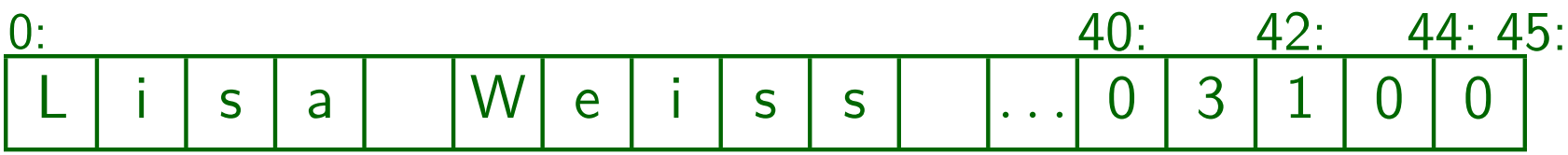
Klassischer Weg, Persistenz zu implementieren:

- Informationen, die in zukünftigen Programmläufen benötigt werden, werden in einer Datei gespeichert.
- Das Betriebssystem speichert die Datei auf einer Festplatte.
- Festplatten sind persistente Speicher: Der Inhalt geht nicht verloren, wenn der Rechner ausgeschaltet oder das Betriebssystem neu gestartet wird.
- Dateisysteme sind Vorgänger moderner DBMS.

Persistente Speicherung (2)

Implementierung von Persistenz mit Dateien:

- Dateien sind normalerweise nur Folgen von Bytes.
- Die Struktur der Datensätze (Records) muss wie in Assembler-Sprachen festgelegt werden.



- Die Information über die Dateistruktur existiert nur in den Köpfen der Programmierer.
- Das System kann nicht vor Fehlern schützen, da es die Dateistruktur nicht kennt.

Persistente Speicherung (3)

Implementierung von Persistenz mit einem DBMS:

- Die Struktur der zu speichernden Daten wird so definiert, dass sie das System versteht:

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

- Somit ist die Dateistruktur formal dokumentiert.
- Das System kann Typfehler in Anwendungsprogrammen erkennen.
- Vereinfachte Programmierung, höhere Abstraktion

DBMS als Unterprogrammbibliothek (1)

- Die meisten DBMS nutzen Dateien des Betriebssystems (z.B. Windows oder Linux), um Daten zu speichern.

  Manche nutzen aus Effizienzgründen direkten Plattenzugriff.

- Man kann ein DBMS als Unterprogrammbibliothek für Dateizugriffe verstehen.

- Verglichen mit dem Dateisystem bietet ein DBMS Operationen auf höherer Ebene an.

  Man arbeitet z.B. mit Punktzahl statt Bytes an bestimmer Position.

- Es enthält bereits viele Algorithmen, die man sonst selbst programmieren müsste.

DBMS als Unterprogrammbibliothek (2)

- Zum Beispiel enthält ein DBMS Routinen zum
  - Sortieren (Mergesort)
  - Suchen (B-Bäume)
  - Freispeicherverwaltung in Datei, Pufferverwaltung
  - Aggregationen, statistische Auswertungen
- Optimiert für große Datenmengen (die nicht in den Hauptspeicher passen).
- Es unterstützt auch mehrere Nutzer (automatische Sperren, Locks) und Sicherheitsmaßnahmen, um die Daten bei Abstürzen zu schützen (siehe unten).

Inhalt

Datenunabhängigkeit (1)

- DBMS = Software-Schicht über dem Dateisystem.
  Zugriff auf die Dateien nur über DBMS.
- Die Indirektion erlaubt es, interne Veränderungen zu verstecken.
- Idee abstrakter Datentypen:
  - Implementierung kann geändert werden,
  - aber die Schnittstelle bleibt stabil.
- Hier ist die Implementierung die Dateistruktur,
  die aus Effizienzgründen geändert werden muss.
  Das Anwendungsprogramm muss nicht angepasst werden:
  Schnittstelle zum Datenzugriff ist stabil.

Datenunabhängigkeit (2)

Typisches Beispiel:

- Anfangs nutzte ein Professor die Hausaufgaben-DB nur für seine Vorlesungen im aktuellen Semester.

  Zur Vereinfachung lässt die obige Tabelle "ABGABEN" nur eine Vorlesung zu. Aber es könnte eine zusätzliche Spalte geben, um verschiedene Vorlesungen zu unterscheiden.

- Da die DB klein war und es wenige Zugriffe gab, genügte es, die Daten als "heap file" zu speichern.

  D.h. die Zeilen der Tabelle (Records) werden ohne bestimmte Reihenfolge gespeichert. Um Anfragen auszuwerten, muss das DBMS die ganze Tabelle durchsuchen, d.h. jede Zeile der Tabelle lesen und überprüfen, ob sie die Anfragebedingung erfüllt ("full table scan"). Für kleine Tabellen ist das kein Problem.

Datenunabhängigkeit (3)

- Später nutzte die ganze Uni die DB und Daten vergangener Vorlesungen mussten aufbewahrt werden.
- Die DB wurde viel größer und hatte mehr Zugriffe.

  Das "Lastprofil" hat sich geändert.

- Für schnelleren Zugriff wird ein Index (typischerweise ein B-Baum) benötigt.

  Ein Index eines Buches ist eine sortierte Liste mit Stichwörtern und den zugehörigen Seitenzahlen, wo die Wörter vorkommen. Ein B-Baum enthält im Wesentlichen eine sortierte Liste von Spaltenwerten zusammen mit Verweisen auf die Tabellenzeilen, die diese Werte enthalten.
  Indexe werden in der Vorlesung "Datenbank-Programmierung" behandelt, und ganz ausführlich in "Datenbanken II B: DBMS-Implementierung".

Datenunabhängigkeit (4)

Fußnote: Indexe vs. Indices

- Als Plural von "Index" ist in der DB-Literatur (wenn es um B-Bäume etc. geht) "Indexe" üblich, obwohl "Indices" auch vorkommt (selten).

  Z.B. verwenden Kemper/Eickler, Vossen, Härder/Rahm und die deutsche Übersetzung von Elmasri/Navathe den Plural "Indexe". In meinem deutschen Wörterbuch steht als Plural nur "Indices" bzw. "Indizes", aber das deckt sich nicht mit meinem persönlichen Sprachempfinden. In der Wikipedia stehen "Indexe", "Indices", "Indizes" als (offenbar gleichberechtigte) Alternativen (speziell für Datenbanken). Als Genitiv sind "des Indexes" und "des Index" beide möglich.

- Dagegen besteht Einigkeit, dass die kleinen tiefgestellten Buchstaben (wie i in x_i) "Indices" sind.

Datenunabhängigkeit (5)

Ohne DBMS (oder mit Prä-Relationalem DBMS):

- Die Verwendung eines Indexes für den Datenzugriff muss explizit in der Anfrage verlangt werden.
- Somit müssen Anwendungsprogramme geändert werden, wenn sich die Dateistruktur ändert.
- Vergisst man, ein selten verwendetes Anwendungsprogramm zu ändern, wird die DB inkonsistent.

  Das kann jedoch nur passieren, wenn man direkt mit Betriebssystem-Dateien arbeitet. Schon ein DBMS für das Netzwerk-Datenmodell (z.B.) machte Index-Updates automatisch. Die Anfrage-Programme mussten sich jedoch explizit auf den Index beziehen.

Datenunabhängigkeit(6)

Mit Relationalem DBMS:

- Das System versteckt die Existenz von Indexen an der Schnittstelle.
- Anfragen und Updates müssen (und können) sich nicht auf Indexe beziehen.
- Folgendes geschieht automatisch durch das DBMS:
  - Aktualisierung des Indexes bei einem Update,
  - Nutzung des Indexes zur Anfrageauswertung, wenn dies von Vorteil (d.h. schneller) ist.

Datenunabhängigkeit (7)

Konzeptuelles Schema ("Schnittstelle"):

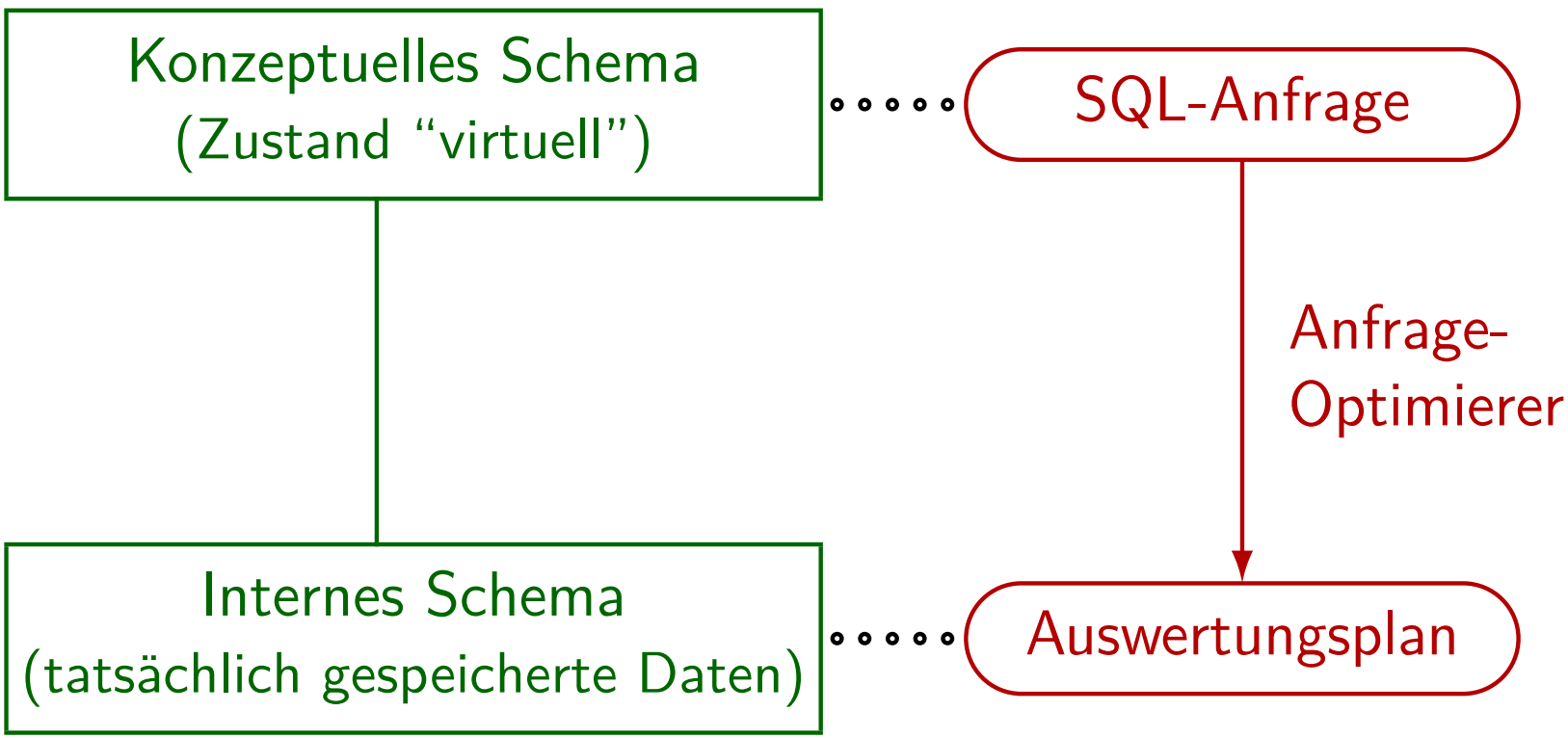
- Nur der logische Informationsgehalt der DB
- Vereinfachte Sicht: Speicherdetails sind versteckt.

Internes/Physisches Schema ("Implementierung"):

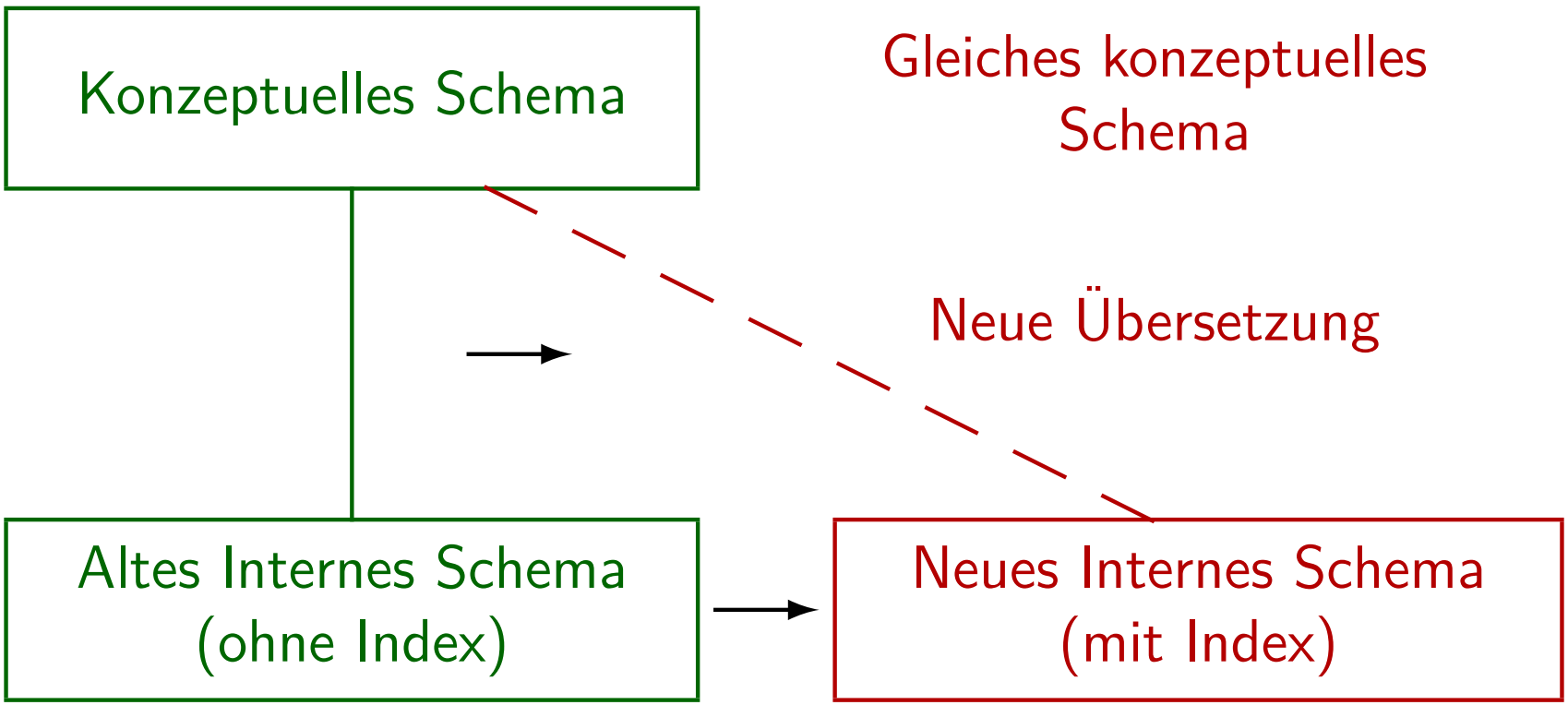
- Indexe, Aufteilung der Tabellen auf Festplatten
- Parameter für Speicher-Management, wenn Tabellen wachsen oder schrumpfen
- Physische Plazierung neuer Zeilen in einer Tabelle.

  Z.B. Zeilen mit gleichem Spaltenwert in den gleichen Plattenblock.

Datenunabhängigkeit (8)



Datenunabhängigkeit (9)



Datenunabhängigkeit (10)

Zur Bezeichnung "Datenunabhängigkeit":

- Ziel ist die Entkopplung von Programmen und Daten.
- Daten sind eine unabhängige Ressource.

  Früher hatten sie nur im Zusammenhang mit den Anwendungsprogrammen eine Bedeutung, mit denen sie ursprünglich erfasst wurden.
- Physische Datenunabhängigkeit:
  - Programme sollten nicht von Speichermethoden (Datenstrukturen) abhängen.
  - Umgekehrt werden die Dateistrukturen nicht durch die Programme festgelegt.

  $\Rightarrow$ Schützt Investitionen in Programme und Daten.

Deklarative Sprachen (1)

- Physische Datenunabhängigkeit verlangt, dass sich die Anfragesprache nicht auf Indexe beziehen kann.
- Deklarative Anfragesprachen gehen noch weiter:
  - Anfragen sollen nur beschreiben,
    was (welche Information) gesucht wird,
  - aber sollen keine spezielle Methode vorschreiben,
    wie diese Information berechnet werden soll.
- Algorithmus = Logik + Kontrolle (Kowalski)

  Imperative/Prozedurale Sprachen: Kontrolle explizit, Logik implizit.

  Deklarative/Deskriptive Sprachen: Logik explizit, Kontrolle implizit.

Deklarative Sprachen (2)

- SQL ist eine deklarative Anfragesprache.
  Man gibt nur Bedingungen für die gesuchten Daten an:

```
SELECT X.PUNKTE
FROM   ABGABEN X
WHERE  X.STUDENT = 'Lisa Weiss'
AND    X.AUFGNR = 3
```

- Häufig leichtere Formulierungen: Der Nutzer muss nicht über eine effiziente Auswertung nachdenken.
- Oft viel kürzer als imperative Programmierung.
  Daher ist z.B. die Programmentwicklung billiger.

Deklarative Sprachen (3)

- Deklarative Anfragesprachen
  - erlauben leistungsstarke Optimierer

    weil sie keine Auswertungsmethode vorschreiben.
  - benötigen leistungsstarke Optimierer

    weil ein naiver Auswertungsalgorithmus ineffizient wäre.
- Größere Unabhängigkeit von aktueller Hardware/Software-Technologie:
  - Einfache Parallelisierung
  - Heutige Anfragen können zukünftige Algorithmen verwenden (bei neuer DBMS-Version).

Weitere DBMS-Funktionen (1)

Transaktionen:

- Folge von DB-Befehlen, die als atomare Einheit ausgeführt werden ("ganz oder gar nicht")

  Wenn das System während einer Transaktion abstürzen sollte, werden alle Änderungen rückgängig gemacht ("roll back"), sobald das DBMS neu startet. Stürzt das System nach einer Transaktion ab ("commit"), sind alle Änderungen garantiert im DB-Zustand gespeichert.

- Unterstützung von Backup und Recovery

  Daten vollendeter Transaktionen sollten Plattenfehler überleben.

- Unterstützung von gleichzeitigen Nutzern

  Nutzer/Programmierer sollen nicht über parallele Zugriffe anderer Nutzer nachdenken müssen (z.B. durch automatische Sperren).

Weitere DBMS-Funktionen (2)

Sicherheit:

- Zugriffsrechte: Wer darf was womit machen?

  Es ist sogar möglich, Zugriffe nur für einen Teil der Tabelle oder nur für aggregierte Daten zuzulassen.

- Auditing: Das DBMS speichert ab, wer was mit welchen Daten gemacht hat.

Integrität:

- Das DBMS prüft, ob die Daten plausibel bzw. konsistent sind. Es lehnt Updates ab, die definierte Geschäftsregeln verletzen würden.

Weitere DBMS-Funktionen (3)

Data Dictionary / Systemkatalog (Meta-Daten):

- Informationen wie Schema, Nutzer und Zugriffsrechte sind in Systemtabellen verfügbar, z.B.:

SYS_TABLES	
TABLE_NAME	OWNER
ABGABEN	BRASS
SYS_TABLES	SYS
SYS_COLUMNS	SYS

SYS_COLUMNS		
TABLE_NAME	SEQ	COL_NAME
ABGABEN	1	STUDENT
ABGABEN	2	AUFGNR
ABGABEN	3	PUNKTE
SYS_TABLES	1	TABLE_NAME
SYS_TABLES	2	OWNER
SYS_COLUMNS	1	TABLE_NAME
SYS_COLUMNS	2	SEQ
SYS_COLUMNS	3	COL_NAME

Dies ist nur ein Beispiel, die genauen Tabellen sind systemabhängig.

Inhalt

Zusammenfassung (1)

Funktionen von Datenbanksystemen:

- Persistenz
- Integration, keine Redundanz/Duplikatspeicherung
- Datenunabhängigkeit
- Weniger Programmieraufwand: Viele Algorithmen enthalten, besonders für Externspeicher (Platten).
- Ad-hoc-Anfragen

  D.h. wenn jemand eine neue Anfrage im Kopf hat, kann er sie sofort stellen. Früher benötigte man dafür ein neues Programm.

Zusammenfassung (2)

Funktionen von Datenbanksystemen, Fortsetzung:

- Hohe Datensicherheit (Backup & Recovery)
- Zusammenfassung von Updates zu Transaktionen

  Transaktionen werden ganz oder gar nicht ausgeführt (sind atomar).
- Mehrbenutzerbetrieb: Synchronisation von Zugriffen
- Integritätsüberwachung
- Sichten für verschiedene Nutzer (Nutzergruppen)
- Datenzugriffskontrolle
- System-Katalog (Data Dictionary)

Zusammenfassung (3)

- Hauptziel eines DBMS: dem Nutzer eine vereinfachte Sicht auf den persistenten Speicher geben.
- Der Nutzer muss nicht nachdenken über:
  - Physische Speicherdetails
  - Unterschiedlichen Informationsbedarf von verschiedenen Nutzern
  - Effiziente Anfrageformulierung
  - Möglichkeit von Systemabsturz/Plattenfehlern
  - Möglichkeit von störenden gleichzeitigen Zugriffen anderer Nutzer

Übungen (1)

Aufgabe:

- Angenommen, Sie sollen ein System zur Evaluation der Lehre an dieser Universität entwickeln:
  Studierende stimmen über die Vorlesungs-Qualität ab.

  Es gibt ein Formular im Internet, in das Studierende ihre Daten eingeben können. Diese werden auf dem Web-Server abgespeichert.
  Später werden die gesammelten Daten ausgewertet,
  z.B. Durchschnittswerte berechnet.

- Vorschlag: Die Daten werden in einer Datei gespeichert (z.B. im CSV-Format).

- Welche Argumente gibt es, stattdessen ein DBMS zu verwenden?

Übungen (2)

- Stellen Sie sich vor, die Hausaufgabenpunkte sind in einer Textdatei gespeichert mit dem Format

  Vorname:Nachname:Aufgabennummer:Punkte

  (d.h. ein Tupel der Tabelle `ABGABEN` pro Zeile).

- Welchen Aufwand schätzen Sie für die Entwicklung eines C-Programms, das die Gesamtpunktzahl je Student ausgibt (alphabetisch geordnet).

  Anzahl Zeilen: _____ (ohne Kommentare)

  Arbeitszeit: _____

- In SQL braucht man 4 Zeilen und 2 Minuten Zeit.