Datenbanken und SQL

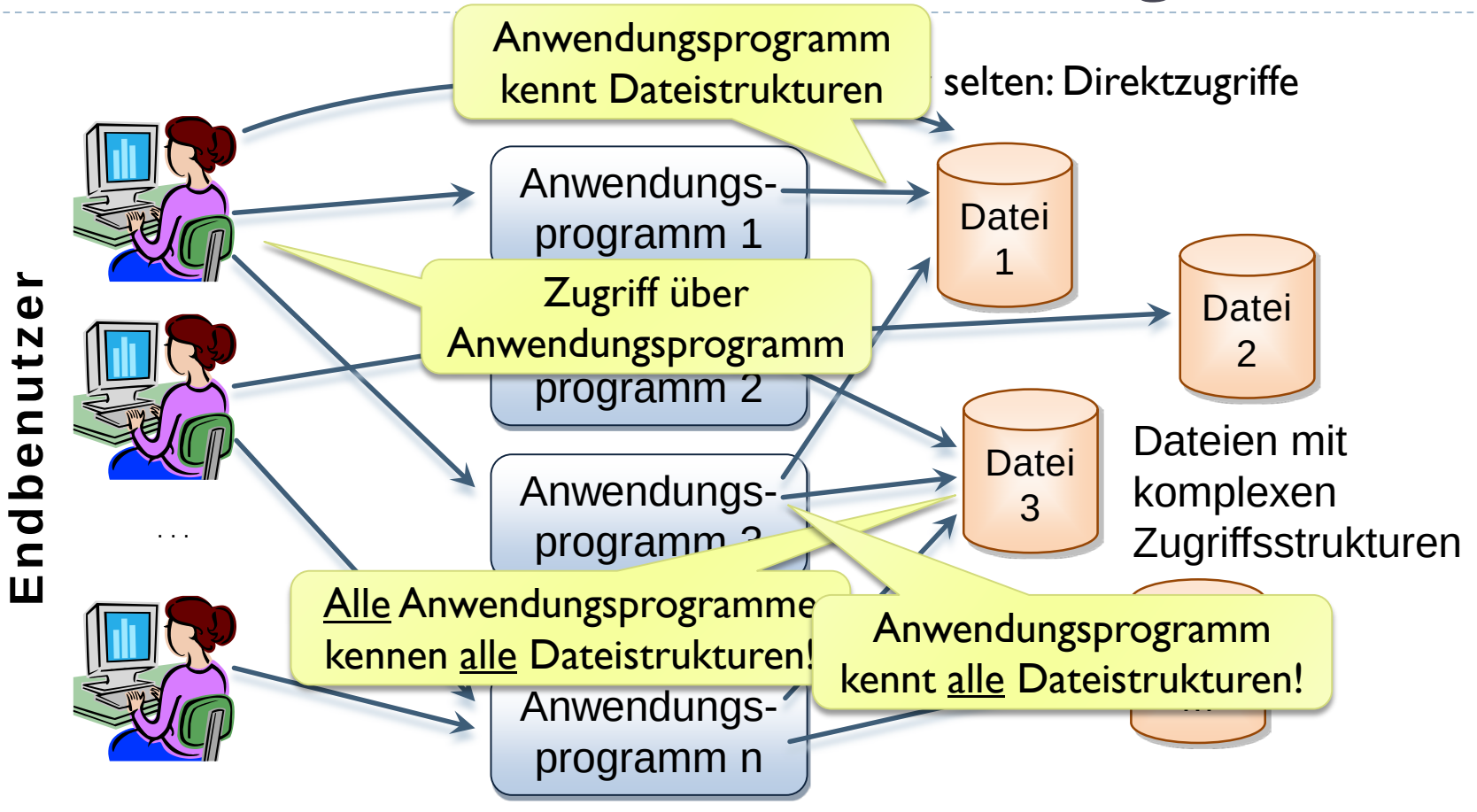
Kapitel 1

Übersicht über Datenbanken

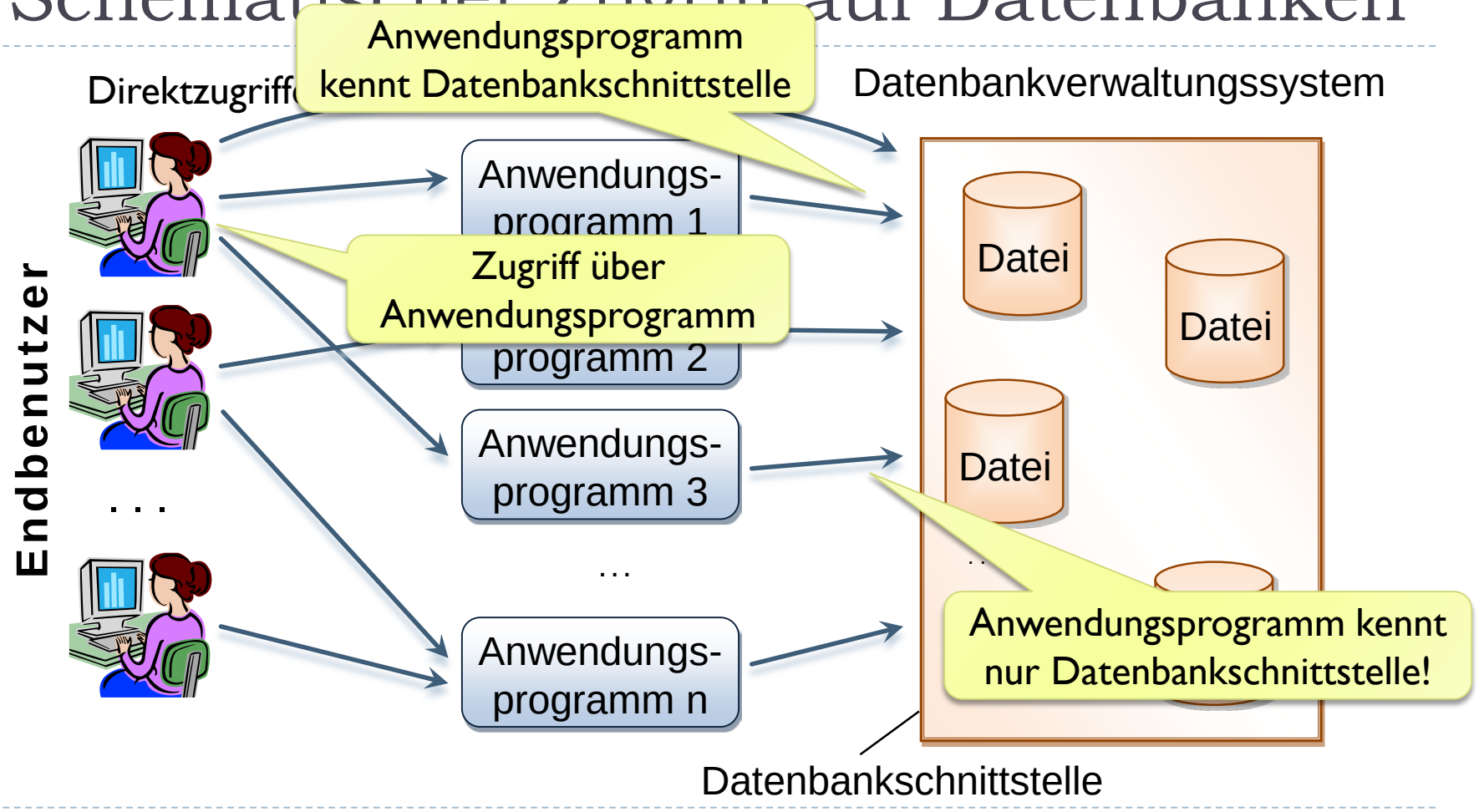
Übersicht über Datenbanken

- Vergleich: Datenorganisation versus Datenbank
- Definition einer Datenbank
- Bierdepot: Eine Mini-Beispiel-Datenbank
- Anforderungen an eine Datenbank
- Der Datenbankadministrator
- Relationale Datenbanken
- Nicht relationale Datenbanken
- Transaktionen

Mehrbenutzerbetrieb mit Datenorganisation



Schematischer Zugriff auf Datenbanken



Definition einer Datenbank

- **Definition (Datenbank):**

- Eine Datenbank ist eine Sammlung von Daten,
  - die untereinander in einer logischen Beziehung stehen und
  - die von einem eigenen Datenbankverwaltungssystem (DBMS) verwaltet werden.

Database Management System

Bierdepot

Nr	Sorte	Hersteller	Typ	Anzahl
1	Hell	Lammsbräu	Kasten	12
3	Roggen	Thurn und Taxis	Kasten	10
4	Pils	Löwenbräu	Kasten	22
8	Export	Löwenbräu	Fass	6
11	Weißbier	Paulaner	Kasten	7
16	Hell	Spaten	6er Pack	5
20	Hell	Spaten	Kasten	12
23	Hell	EKU	Fass	4
24	Starkbier	Paulaner	Kasten	4
26	Dunkel	Kneitinger	Kasten	8
28	Märzen	Hofbräu	Fass	3
33	Pils	Jever	6er Pack	6
36	Alkoholfreies Bier	Löwenbräu	6er Pack	5
39	Weißbier	Erdinger	Kasten	9
47	Alkoholfreies Pils	Clausthaler	Kasten	1

Lesezugriff auf das Bierdepot

```
SELECT    Sorte, Hersteller, Anzahl
FROM      Bierdepot
WHERE     Sorte = 'Weißbier' ;
```

Nr	Sorte	Hersteller	Typ	Anzahl
	Weißbier	Paulaner		7
	Weißbier	Erdinger		9
11	Weißbier	Paulaner	Kasten	7

Schreibzugriffe auf das Bierdepot

```
INSERT
INTO      Bierdepot
VALUES    (43, 'Dunkel', 'Kaltenberg', 'Kasten', 6) ;

UPDATE    Bierdepot
SET       Anzahl = Anzahl - 1
WHERE     Nr = 11 ;

DELETE
FROM      Bierdepot
WHERE     Nr = 47 ;
```

Fügt eine neue Zeile hinzu

Reduziert die Anzahl zu Artikel 11

Löscht die Zeile zu Artikel 47

Datenbankhersteller

- Angaben zu Marktanteilen streuen (je nach Quelle)

Hersteller	Marktanteil geschätzt
Oracle	33%-48%
DB2	20%-30%
Microsoft	16%-20%
SAP	3%-4%

- Open Source Datenbanken:

MySQL
PostgreSQL

Anforderungen an eine Datenbank (1)

- Sammlung logisch verbundener Daten
- Speicherung der Daten mit möglichst wenig Redundanz
  - Beispiel für Redundanz:
    - Einkauf ←→ Lagerhaltung ←→ Verkauf
    - da: Lager = Einkauf – Verkauf
- Abfragemöglichkeit und Änderbarkeit

Anforderungen an eine Datenbank (2)

- Logische Unabhängigkeit der Daten von der Speicherung
- Zugriffsschutz
- Integrität und Korrektheit
- Mehrfachzugriff (Concurrency)
- Zuverlässigkeit und Audit
- Ausfallsicherheit
- Funktionalität zur Kontrolle der Datenbank

Datenbankadministrator

- **Aufgaben des Administrators:**
  - Einrichten einer Datenbank, Zugriffsschutz
  - Betrieb und Kontrolle der Datenbank
- **Datenbank-Schnittstellen:**
  - **DDL** Data Description Language
  - Kontrollsprache (in DDL integriert)
  - **DML** Data Manipulation Language
- **Aufgabenteilung:**
  - Anwender: DML (Select, Insert, Update, Delete)
  - Administrator: DDL (Create Table, Create View, Grant, …)

Datenbankmodelle im Überblick

- Relationale Datenbanken
  - Oracle, DB2, MS SQL Server, MySQL, PostgreSQL, Sybase
- Objektorientierte und objektrelationale Datenbanken
  - Oracle, PostgreSQL
- Hierarchische Datenbanken
  - IMS
- Netzwerkartige Datenbanken
  - IDMS, UDS
- Neue Datenbanksysteme
  - NOSQL: MongoDB

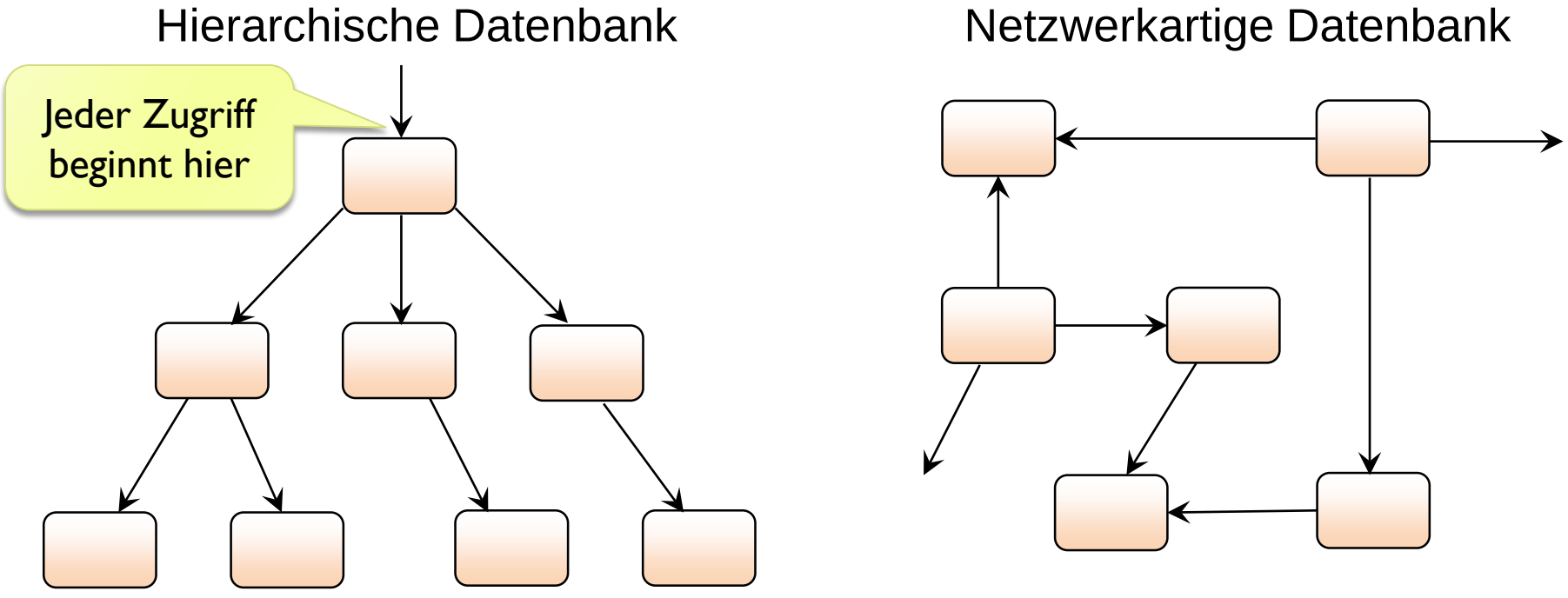
Relationale Datenbanken

	Relationale Datenbanken
Vorteile	Leichte Änderbarkeit des Datenbankaufbaus, mathematisch fundiert, leicht programmierbar und zu verwalten
Nachteile	Häufig viele Ein-/Ausgaben notwendig, erfordert hohe Rechnerleistung, erzeugt Redundanz

Objektorientierte Datenbanken

	Objektorientierte und objektrelationale Datenbanken
Vorteile	Objektorientierter Ansatz, universell einsetzbar, noch relativ einfach programmierbar und zu verwalten, (meist) aufwärtskompatibel zu relationalen Datenbanken
Nachteile	Relativ viele Ein-/Ausgaben notwendig, erfordert eine relativ hohe Rechnerleistung, teilweise recht komplexer Aufbau

Hierarchische und Netzwerk-Datenbanken



Hierarchische und Netzwerk-Datenbanken

	hierarchisch	netzwerkartig
Vorteile	sehr kurze Zugriffszeiten, minimale Redundanz	kurze Zugriffszeiten, geringe Redundanz
Nachteile	Strukturänderung kaum möglich, komplexe Programmierung	Strukturänderung nicht einfach, relativ komplexe Programmierung

NoSQL Datenbanken

- NoSQL = Not Only SQL
- NoSQL Datenbanken gliedern sich in
  - Key/Value und dokumentenbasierte Datenbanken
    - z.B. CouchDB, MongoDB
  - Spaltenorientierte Datenbanken
    - z.B. Google Big Table, Simple DB von Amazon
    - HBase, Cassandra
  - Graphenorientierte Datenbanken
    - z.B. Sones, Neo4j

NoSQL Datenbankmodelle

- **Key/Value und dokumentenbasierte Modelle**
  - Schemafreie Modelle, daher sehr flexibel
  - Seit 1979 im ersten Einsatz
  - Lotus Notes ist dokumentenbasiert
- **Spaltenorientierte Modell**
  - Die Daten werden spaltenweise gespeichert!
  - Bei Anfragen nach wenigen Eigenschaften extrem performant
- **Graphen Modelle**
  - In Navigationsgeräten
  - Wie finde ich den besten Weg von A nach B?

Transaktionen

- **Abfrage**
  - Lesezugriff: Select
  - Query
  - Retrieval
- **Mutation**
  - Schreibzugriff: Insert, Update, Delete
- **Transaktion**
  - Konsistenzerhaltende Operation
  - Atomare Operation

Konsistenz und Redundanz

- **Definition (Konsistenz):**
- Eine Datenbank heißt in sich **konsistent**, wenn alle gespeicherten Daten untereinander widerspruchsfrei sind.

- **Definition (Redundanz):**
- Daten heißen **redundant**, wenn sie mehr als einmal in einer Datenbank abgespeichert werden, also an sich überflüssig sind.

Beispiel zu Redundanz und Konsistenz

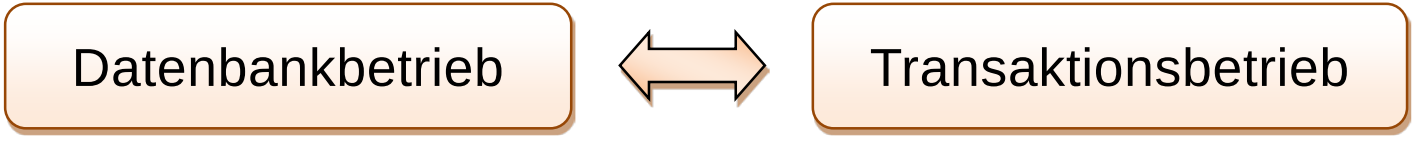
- **Es gilt:**
  - Warenbestand = Wareneingang – Warenausgang
- **In der Datenbank:**
  - Lagertabelle + Einkaufstabelle + Verkaufstabelle
- **Folgerung:**
  - Redundanz und Gefahr der Inkonsistenz
- **Frage:**
  - Welche der drei obigen Tabellen würden Sie entfernen?

Beispiel: Buchung

- **Bank speichert:**
  - Alle Kontostände S_i der n Kunden (i=1..n)
  - Summe der Kontostände S_{ges} aller Kunden (Redundanz!)
- **Szenario:**
  - Überweisung von 500 Euro von Kunde A nach Kunde B
  - 1. Schritt: Abbuchung von 500 Euro von Kunde A
  - 2. Schritt: Buchung von 500 Euro für Kunde B
  - Nach dem 1. Schritt: Absturz des Rechners
  - Datenbank ist nun inkonsistent ($\sum S_i \neq S_{ges}$) und fehlerhaft!
- **Folgerung: Transaktion muss atomar ablaufen!**

Transaktionen

- In betriebswirtschaftlichen Anwendungen und Buchungssystemen **zwingend** erforderlich, da
  - Inkonsistenzen nicht hinnehmbar sind
  - eine Buchung immer atomar ausgeführt werden muss
    - Atomare Ausführung heißt: Nichts oder alles wird ausgeführt
- Datenbanken garantieren auch im Fehlerfall die **atomare** Ausführung von Transaktionen
- Folgerung:
  - Datenbanksystem und Transaktionssystem sind Synonyme



A C I D

- Ein **Transaktionsbetrieb** muss folgende Bedingungen erfüllen:

- **A** **Atomarity** (Atomarität)
- **C** **Consistency** (Konsistenz)
- **I** **Isolation**
- **D** **Durability** (Dauerhaftigkeit)

A = Atomarität

- Eine Transaktion läuft immer **atomar** ab
- Eine noch laufende Transaktion kann **jederzeit**, insbesondere im Fehlerfall, zurückgesetzt werden

- In SQL:
  - **COMMIT**; Transaktion ist beendet, Daten sind gespeichert
  - **ROLLBACK**; Transaktion wird komplett zurückgesetzt

C = Konsistenz (Consistency)

- **Eine Transaktion ist konsistenzerhaltend**
- **Teiltransaktionen gibt es nicht:**
  - Eine Transaktion läuft komplett ab (Commit;) oder
  - Eine Transaktion wird nicht wirksam (Rollback;)

- **Folgerung:**
  - Eine Datenbank ist konsistent, wenn
    - alle Mutationen innerhalb von Transaktionen erfolgen
    - der Transaktionsmechanismus, insbesondere der Rollback, immer und jederzeit unterstützt wird (auch im Fehlerfall!)

I = Isolation

- Eine Transaktion läuft so ab, als sei sie allein im System
- Eine Transaktion ist vollständig isoliert von anderen parallel laufenden Transaktionen
- (Fast) gleichzeitige Zugriffe auf gleiche Daten müssen wegen Konsistenzverletzungen synchronisiert werden
- Beispiel zum Bierdepot bei 2 Verkaufsstellen:
  - 2 Kunden wollen das letzte Fass Pils von Bischofshof
  - Beide Kunden erfahren, dass noch Ware vorhanden ist
  - Aber: Nur ein Kunde bekommt das Fass

D = Dauerhaftigkeit

- **Die Daten werden dauerhaft gespeichert**
- **Ein Benutzer kann sich also auf Folgendes verlassen:**
  - Er erhält die Rückmeldung, dass seine Transaktion erfolgreich abgeschlossen wurde
  - Seine von dieser Transaktion manipulierten Daten sind daher dauerhaft und sicher gespeichert
- **Beispiel:**
  - Ein Kunde einer Versicherung verlässt sich darauf, dass seine vor 15 Jahren abgeschlossene Versicherung nicht verloren geht