Kapitel 1: Datenbankdienste

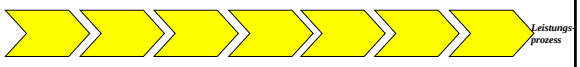
Einführung: Dienste
Dienstfunktionalität
Dienstmerkmale
Datenbanksysteme

viele Folien: © Prof. Lockemann, IPD, Uni Karlsruhe

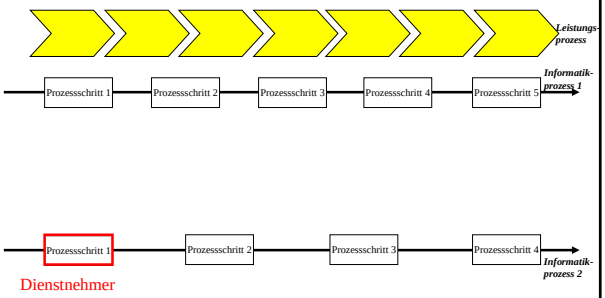
Kapitel 1: Datenbankdienste

Einführung: Dienste
Dienstfunktionalität
Dienstmerkmale
Datenbanksysteme

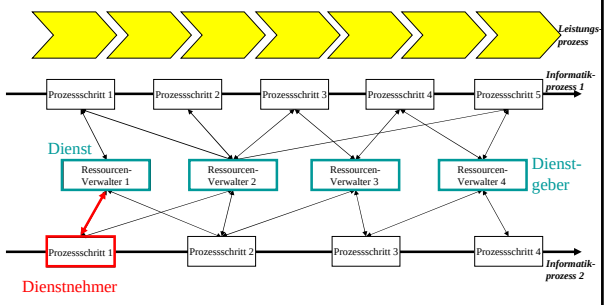
Leistungsprozesse und Ressourcen (1)



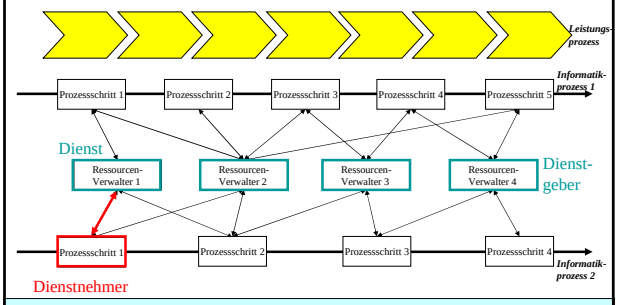
Leistungsprozesse und Ressourcen (1)



Leistungsprozesse und Ressourcen (1)



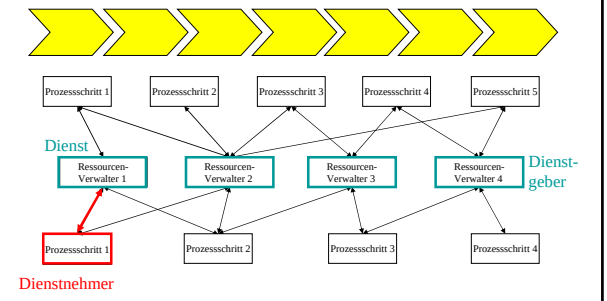
Leistungsprozesse und Ressourcen (1)



Dienst: Nach Form und Inhalt wohl definiertes Leistungsangebot eines Ressourcen-Verwalters, das Gegenstand einer Dienstleistungsvereinbarung ist.

Beispiel

Finden Sie konkrete Beispiele aus unserem Bibliotheksszenario!

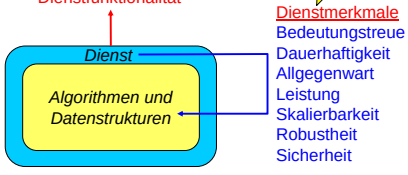


Dienstfunktionalität und Dienstmerkmale (1)

Ressourcen-Verwalter

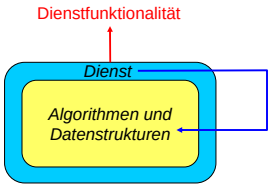
Sammlung von Funktionen, die in einem erkennbaren Zusammenhang stehen, aber einzeln abgerufen werden können

Von der Dienstfunktionalität in ihrer Gesamtheit zu beachtende Randbedingungen



Dienstfunktionalität und Dienstmerkmale (2)

Ressourcen-Verwalter



Dienstmerkmale

Bedeutungstreue
Gemeinsames Verständnis der Daten bei allen am Austausch beteiligten Partnern

Dauerhaftigkeit
Zugriff auf Daten nach ihrer Entstehung zu jeder Zeit in der Zukunft

Allgegenwart
Datenzugriff zu jeder Zeit von jedem Ort

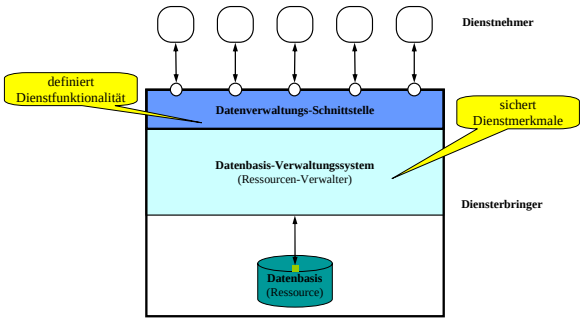
Leistung
- Effizienz: Minimaler Verbrauch innerer Ressourcen
- Verfügbarkeit: Rechtzeitiger Zugang zu den Ressourcen

Skalierbarkeit
Stetes Wachstum ohne Einbußen bei Funktionalität und Leistung

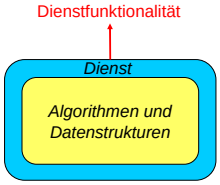
Robustheit
Zuverlässiges Erbringen der Dienste unter Störungen und Fehlern

Sicherheit
Keine Verluste, Störungen, Ausfälle durch unbedachte oder böswillige Eingriffe

Datenverwaltungssystem



Dienstfunktionalität der Datenverwaltung



Dienstfunktionen für das
- Entgegennehmen,
- Abspeichern,
- Ändern,
- Löschen,
- Auswählen,
- Bereitstellen

von Daten.

- Dienstfunktionen besitzen allgemeingültigen Charakter,
- Wirtschaftlichkeitsgründe diktieren „breitbandige“ Einsatzfähigkeit

⇒ **Generisches Vorgehen**.

Kapitel 1: Datenbankdienste

Datenmodell

Aspekte der Dienstfunktionalität

- Beschreibung der zulässigen Datenbasiszustände

  ⇒ Generisch: Mittel zur inhaltsneutralen Strukturierung der Datenbasis
- Beschreibung der zulässigen Zustandsübergänge, im wesentlichen in Form der anwendbaren Operatoren

  ⇒ Generische Operatoren

=: **Datenmodell**

Zustände einer Datenbasis (1)

Grundgedanke: Datentyp = Menge zulässiger Zustände

- **Typ**: Universum von Objekten gleicher mathematischer Struktur.
- **Ausprägung** (**Instanz**): Element eines Typs.
- **Polymorphes Typsystem**: Universum von Typen, i.d.R. beschrieben durch:
  - die Festlegung gewisser **atomarer Typen**, z.B.
    - int = Menge der ganzen Zahlen,
    - bool = {true, false},
    - date = Menge der Daten des Gregorianischen Kalenders;
  - die Angabe von **Typkonstruktoren**, mit denen Typen zu neuen Typen kombiniert werden können, z.B.
    - record(t_1,t_2,...,t_n): Menge der Tupel mit Komponenten aus t_1,t_2,...,t_n,
    - set(t) = Menge der Mengen mit Elementen aus t,
    - list(t) = Menge der Listen mit Elementen aus t.

Zustände einer Datenbasis (2)

Beispiel für ein einfaches polymorphes Typsystem:

Atomare Typen `int, float, string, time`

Typkonstruktoren

`datensatz` `::= [`*`sel`*`:`**`atomarerTyp`**`,`
`..., `*`sel`*`:`**`atomarerTyp`**`]`

`menge` `::= {`**`datensatz`**`}`

Selektor (Feldname)

set-Konstruktor

Typvariable

record-Konstruktor

Zustandsbeschränkungen

- Typkonstruktoren geben an, wie Zustände zu neuen Zuständen zusammengestellt werden können.
- Konsistenzbedingungen: Einschränkungen auf den konstruierbaren Zustandsmengen.
- **Polymorphe Konsistenzbedingung**: Generische Vorschrift für Konsistenzbedingungen als Teil des Datenmodells.

Beispiel: Jeder Datensatz in einer Menge besitzt einen **Schlüssel**, d.h. einen Wert, der ihn eindeutig unter allen Datensätzen der Menge identifiziert. Formal: Es existiert Funktion

`key: `**`menge`**` × `**`atomarerTyp`**` → `**`datensatz`**

Zustandsübergänge mittels Operatoren

- **Operatoren**: mathematische Funktionen, die auf Elemente von Typen angewendet werden können, z.B.
  - Gleichheitstest (x = y): anwendbar auf beliebige Typen
  - Anordnung (x < y): anwendbar auf Zahlen, Daten, Zeichenketten,...
  - arithmetische Operationen (+, –, ×, /): anwendbar auf Zahlen
  - logische Operationen (*and, or, not*): anwendbar auf boolesche Werte
  - Mengenoperationen (∪, ∩, –): anwendbar auf Typen, die durch Anwendung des set-Typkonstruktors entstanden sind
- **Monomorphe** Operatoren können nur auf Elemente bestimmter Typen angewendet werden (z.B. *not*).
- **Polymorphe** Operatoren können auf Elemente unterschiedlicher (wenn auch nicht aller) Typen angewendet werden (z.B. =, < oder ∪).

Zustandsübergänge einer Datenbasis

- Ein polymorphes Typsystem hat zwingend polymorphe Operatoren, erfüllt also die Forderung nach generischen Operatoren.
- Beispiele : Abspeichern, Löschen von Ausprägungen von Typen in Ausprägungen von Mengentypen.
- Auswählen und Bereitstellen von Elementen der Datenbasis lässt sich als der identische Zustandsübergang erfassen.

Erweiterung unseres Beispiels:

Polymorphe Operatoren

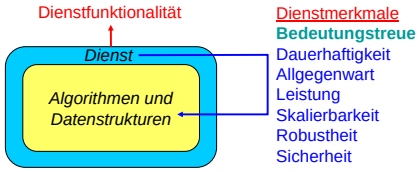
```
union: menge × menge → menge
intersect: menge × menge → menge
insert: menge × datensatz → menge
deleteByKey: menge × atomarerTyp → menge
findByKey: menge × atomarerTyp → datensatz
```


Zusammenfassung Datenmodell

- Pragmatische Definition Datenmodell:
  - System aus atomaren Typen, Typkonstruktoren, polymorphen Konsistenzbedingungen und polymorphen Operatoren.

Kapitel 1: Datenbankdienste

Bedeutungstreue in der Datenverwaltung



- Daten müssen interpretierbar sein (**Information**)
- Daten müssen bei allen am Austausch beteiligten Partnern dieselbe Information besitzen
- Interpretation muss über die Zeit dieselbe bleiben
- Konventionen zur Interpretation begleiten die Daten ⇒ Aufgabe des Datenverwaltungssystems

Gewinnung der Konventionen

- Interpretation durch Bindung an eine auf bestimmte Aspekte beschränkte Anwendungswelt, die **Miniwelt**.
- Informationen sind somit gedankliche Abstraktionen (**Modelle**) der Miniwelt.
- Daten sind daher Repräsentationen von Modellen.

- **Eine Datenbasis ist bedeutungstreu, wenn ihre Elemente Modelle einer gegebenen Miniwelt repräsentieren. (Datenbasiskonsistenz)**
- Datenbasis beschreibt Zustand der Miniwelt durch mathematische Strukturen (Mengen, Tupel, Funktionen, ...).

⇒ Datenbasis ist damit **formales Modell** der Miniwelt.

Beispiel Realwelt: Flugwesen

- Reale, anfassbare Gegenstände:
  - Flugzeuge mit Flugzeugtyp, Besatzung, Sitzplätzen, Laderäumen, Farbanstrichen;
  - Flughäfen mit Bezeichnung, Flugsteigen, Passagieraufkommen, Rollbahnen;
  - Passagiere mit Namen, Anschriften, Telefonnummern, Größe, Gewicht, Haarfarbe, Religionszugehörigkeit;
  - Flugscheine mit Vordrucken und Aufdrucken.
- Gedankliche, in ihren Auswirkungen beobachtbare Phänomene:
  - Flüge mit Abflugs- und Ankunftszeiten, Flugzeugtyp, Startgeschwindigkeit, Fluggeschwindigkeit, -höhe und -route;
  - Reservierungen mit Flugnummer und -datum, Sitzplatz, Passagier.

Miniwelt Platzbuchung

- Reale, anfassbare Gegenstände:
  - Flugzeuge mit Flugzeugtyp, Besatzung, Sitzplätzen, Laderäumen, Farbanstrichen;
  - Flughäfen mit Bezeichnung, Flugsteigen, Passagieraufkommen, Rollbahnen;
  - Passagiere mit Namen, Anschriften, Telefonnummern, Größe, Gewicht, Haarfarbe, Religionszugehörigkeit;
  - Flugscheine mit Vordrucken und Aufdrucken.
- Gedankliche, in ihren Auswirkungen beobachtbare Phänomene:
  - Flüge mit Abflugs- und Ankunftszeiten, Flugzeugtyp, Startgeschwindigkeit, Fluggeschwindigkeit, -höhe und -route;
  - Reservierungen mit Flugnummer und -datum, Sitzplatz, Passagier.

Datenbasisschemata (1)

- **Vorgehen**: Einsetzen konkreter (aus dem Modell der Miniwelt gewonnener) Typen und Bezeichner für die Typvariablen bzw. Selektoren in den Typkonstruktoren und polymorphen Konsistenzbedingungen.
- Das polymorphe Typsystem wird zu einem monomorphen Typsystem instantiiert.
- Erst jetzt ist die Datenbasis definierbar.
  - Erst jetzt liegt ein formales Modell der Miniwelt vor.
- **Datenbasistyp**: Ergebnis der Konkretisierung.
- **Datenbasisschema** (kurz: **Schema**): Beschreibung eines Datenbasistyps.

Beispiel

Zugrundeliegender polymorpher Typ (Schablone für monomorphe Typen)

monomorphe Konsistenzbedingung

```
Datenbasistyp  Buchungsdatenbasis

Typ    Flughafen ::= datensatz  [
       FlughCode: string, Stadt: string, Land: string,
       Name: string, Zeitzone: int ]
Typ    Flugzeugtyp ::= datensatz  [
       FtypId: string, Name: string, First: int,
       Business: int, Economy: int ]
Typ    Flug ::= datensatz  [
       FlugNr: string, von: string, nach: string,
       FtypId: string, Wochentag: string,
       Abflugszeit: time, Ankunftszeit: time,
       Entfernung: int ]
Typ    Flughäfen ::= menge  {Flughafen} key FlughCode
Typ    Flugzeugtypen ::= menge  {Flugzeugtyp} key FtypId
Typ    Flüge ::= menge  {Flug} key FlugNr
```


Weitere denkbare Konsistenzbedingungen

Land → Zeitzone

Wert unter *von* kommt als Wert unter *FlughCode* vor

(von,nach) → Entfernung

```
Datenbasistyp  Buchungsdatenbasis

Typ    Flughafen ::= datensatz  [
       FlughCode: string, Stadt: string, Land: string,
       Name: string, Zeitzone: int ]
Typ    Flugzeugtyp ::= datensatz  [
       FtypId: string, Name: string, First: int,
       Business: int, Economy: int ]
Typ    Flug ::= datensatz  [
       FlugNr: string, von: string, nach: string,
       FtypId: string, Wochentag: string,
       Abflugszeit: time, Ankunftszeit: time,
       Entfernung: int ]
Typ    Flughäfen ::= menge  {Flughafen} key FlughCode
Typ    Flugzeugtypen ::= menge  {Flugzeugtyp} key FtypId
Typ    Flüge ::= menge  {Flug} key FlugNr
```

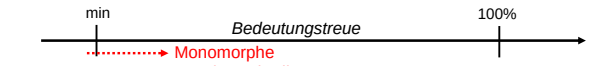

Beispiel

- Wie könnten Schema und Konsistenzbedingungen im Bibliotheksbeispiel aussehen?

Zustandsübergänge (1)

- Ein Datenverwaltungssystem gewährleistet (Schema-) Konsistenz, wenn seine Dienstfunktionen stets einen (schema-)konsistenten Zustand seiner Datenbasis wieder in einen (schema-)konsistenten Zustand überführen.
- Gehorcht die Datenbasis einer wie immer gearteten Schemakonsistenz vor Ausführen einer Dienstfunktion, so gehorcht sie ihr auch zum Abschluss der Ausführung.

Konsistente Zustände (1)

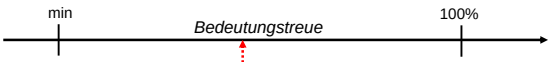


Legitime Zustände durch Ausführen der Operatoren mit Interpretation des Schemas
Schemakonsistenz

Idealfall: volle, aktuelle Übereinstimmung von Datenbasis und Miniwelt
Datenbasiskonsistenz

Datenmodell legt Regeln fest, nach denen Zustandsräume konstruiert werden können.
Schema legt bestimmten Zustandsraum fest.

Konsistente Zustände (2)



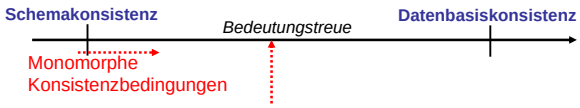
Legitime Zustände durch Ausführen der Operatoren mit Interpretation des Schemas
Schemakonsistenz

Idealfall: volle, aktuelle Übereinstimmung von Datenbasis und Miniwelt
Datenbasiskonsistenz

Ausführung von Transaktionsprozeduren

Konsistenz: Grad an Bedeutungstreue einer Datenbasis

Zustandsübergänge (2)



Transaktionsprozedur: Dienstprozedur als frei definierte Abfolge von Operationen, die Zustandsübergänge einschränkt: Prozedurale Kontrolle der Konsistenz

- Will man also Konsistenz durchsetzen: Als Dienstfunktionen sind ausschließlich Transaktionsprozeduren zulässig.
  - Die Bedeutungstreue ist an das Ende der Prozedur gebunden, während der Ausführung, also am Ende einzelner Operationen, muss sie nicht zutreffen.
  - Daher dürfen Konsistenzbedingungen erst am Ende durchgesetzt werden.

Beispiel

Mustert eine Fluggesellschaft einen Flugzeugtyp aus, so ist dessen Datensatz mit der entsprechenden FtypId aus der Menge der Flugzeugtypen zu entfernen, gleichzeitig sind alle Flug-Datensätze, die sich auf diese FtypId beziehen, auf den neuen Flugzeugtyp umzustellen.

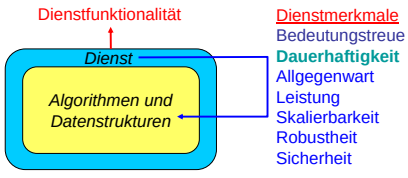
Zusammenfassung Bedeutungstreue

- Schema definiert Menge der im Betrieb zulässigen Zustände und Zustandsübergänge.
- Charakterisierung des Zustandsraumes durch:
  - Auswahl konkreter Typen aus dem Typsystem des Datenmodells,
  - weitere Eingrenzung durch Konsistenzbedingungen
  - und (implizit) durch Transaktionsprozeduren.
- Charakterisierung der Zustandsübergänge durch:
  - Spezialisierung der polymorphen Operatoren des Datenmodells auf die im Schema auftretenden Typen,
  - weitere Eingrenzung durch Transaktionsprozeduren.
- **Datendefinitionssprache** (DDL) dient zur Formulierung des Schemas.
- **Datenmanipulationssprache** (DML) dient zum nachfolgenden Umgang mit der Datenbasis.

Durchsetzen von Konsistenz (2)

- **Datenbasistransaktion** (auch kurz: Transaktion): Ausführung einer Transaktionsprozedur.
- Folgerung: Konsistenz ist erst am Ende einer Transaktion definiert.
- Somit wird auch das Durchsetzen von Konsistenzbedingungen einer Transaktionsverwaltung übertragen. Eine Transaktion überprüft also am Ende, ob sich eine Konsistenzbedingung erfüllen lässt bzw. erfüllt ist. Trifft dies nicht zu, so wird eine Korrekturaktion ausgelöst oder ein Fehler signalisiert.
- Beispiel: In einer Transaktion wird ein insert-Operator gezwungen, gegen die Schlüsselbedingung zu verstoßen.
- **Anwendungstransaktion**: Konsistente Ausführung eines Leistungsprozesses.

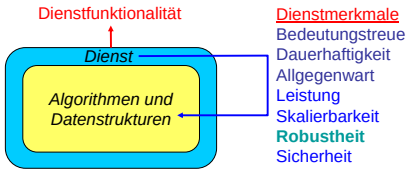
Dauerhaftigkeit in der Datenverwaltung



- Verbringen der Daten auf nichtflüchtiges Speichermedium genügt nicht.
- Dauerhafte Daten müssen zudem konsistent sein.
- Das sind sie, wenn sie von abgeschlossenen Transaktionen stammen.

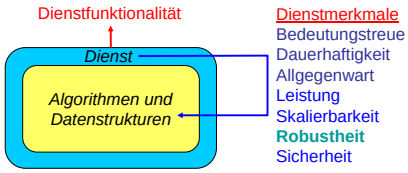
Dauerhaftigkeit ist die andauernde Nichtflüchtigkeit gewisser, in jedem Fall konsistenter Datenbasiszustände (Unverletzlichkeit der Datenbasis).

Robustheit 1: Persistenz



- Erreichen eines konsistenten nichtflüchtigen Zustands: **Persistenz**
- Folgerung 1: Dauerhaftigkeit ist potenziell unbegrenzte Lebensdauer eines persistenten Zustandes.
- Folgerung 2: Transaktionsabschluss ist ein Persistenz-Ereignis.

Robustheit 2: Resistenz



- Wahrung von Persistenz (und damit auch Konsistenz) auch unter dem Einfluss von Störungen: **Resistenz**
- Störung des Dienstnehmer-Dienstgeber-Verhältnisses: **Fehler-Resistenz**.
- Störung des Dienstnehmer-Dienstnehmer-Verhältnisses: **Konflikt-Resistenz**.

Transaktionen (1)

- Präzisierung von Robustheit:
  - Datenbasis führt — ungeachtet möglicher Störungen — nur solche Zustandsübergänge aus, die explizit durch Aufruf von Transaktionsprozeduren angefordert wurden.
- Impliziert zwei Garantien:
  - **Persistenz**: Effekte von abgeschlossenen Zustandsübergängen gehen nicht durch Störungen verloren, d.h., Datenbasiszustand ändert sich nicht „einfach so“.
  - **Resistenz**: Angeforderte Zustandsübergange werden entweder wie verlangt durchgeführt, oder wenn Zustandsübergang wegen Störung nicht abgeschlossen werden kann, strebt Datenbasis definierten konsistenten Zustand, z.B. Ausgangszustand, an.

Beispiel-Transaktion

- Gewünschter Zustandsübergang: Ausmusterung eines Flugzeugtyps.
- Transaktionsprozedur:
  - Entferne zugehörigen Datensatz aus Tabelle Flugzeugtypen.
  - Ersetze in Tabelle Flüge alle Vorkommen der alten FtypId durch FtypId des Ersatzmodells.
- Beachte: Datenbasis-Zustand ist zwischen Schritt 1 und 2 inkonsistent, davor und danach konsistent.
- Resistenz garantiert, dass bei Störung entweder Zustand vor Schritt 1 oder nach Schritt 2 erreicht wird.
- Persistenz garantiert, dass nach Abschluss von Schritt 2 Ergebnis nicht mehr verloren gehen kann.

Transaktionen (2)

- Transaktionsbegriff fasst Konsistenz, Persistenz und Resistenz zusammen:
- **Transaktion**: resistente Ausführung eines konsistenten Zustandsübergangs (Operator oder Transaktionsprozedur) mit Persistenz des Endzustands.

Fehler-Resistenz (1)

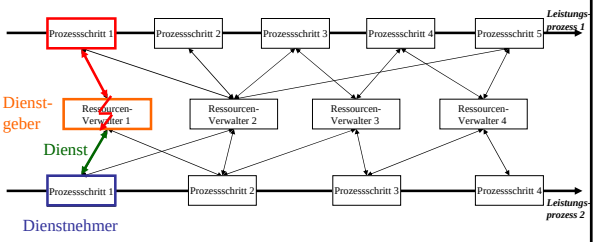
Beispiele für Störungen:

- Fehlerhafte Eingaben; Programmierfehler; unbeabsichtigte Wechselwirkung von Dienstleistungen für unterschiedliche Dienstnehmer; Programmfehler in der Datenverwaltungssoftware; Ausfall von Hintergrundspeichern, Datenträgern oder dem Rechner.

Fehler-Resistenz (2)

- So lange es zu keinen Störungen kommt, wird die Datenbasis einfach fortentwickelt. Tritt eine Störung auf, so wird ein früherer persistenter Zustand angestrebt.
- Wann ein persistenter Zustand vorliegt, muss dazu ausdrücklich dem Datenverwaltungssystem bekannt gemacht werden.
- Verbreitet: Da eine Transaktion ein persistentes Ergebnis bewirkt, ist der Abschluss einer Transaktion ein Persistenz-Ereignis. Die Transaktionsprozedur ist so zu entwerfen, dass man sie für beendet erklärt genau dann, wenn ihre Ergebnisse nicht mehr verloren gehen dürfen.
- Kommt die Transaktion nicht zum Abschluss, so ist der jüngste persistente Zustand der Zustand unmittelbar vor Ausführung der Transaktion. Man wird daher diesen Zustand wieder herstellen.

Konflikt-Resistenz (1)



Störfaktor für die Konsistenz: Bemühen sich mehrere Dienstnehmer-Transaktionen zur selben Zeit um dieselbe Ressource, so kann es zu einer unerwünschten Wechselwirkung zwischen ihnen kommen (**Konflikt**).

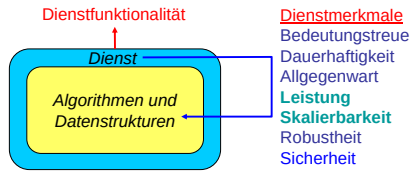
Konflikt-Resistenz (2)

- **Konkurrenz** um gemeinsame Daten (engl.: concurrency für Nebenläufigkeit): Konfliktverhalten von Transaktionen.
- **Benötigt**: Protokoll, das Konflikte zwischen *nebenläufigen* Dienstnehmer-Transaktionen eines Datenverwaltungssystems unterbindet (**Synchronisations-Protokoll**).
- **Korrektheitskriterium**: Wahrung der Konsistenz durch: Konkurrierende Datenbasistransaktionen sind dann korrekt synchronisiert, wenn jede so abläuft als ob sie ohne Konkurrenz wäre, insbesondere also dem Dienstnehmer keinen inkonsistenten Datenbasiszustand präsentiert und bei Abschluss einen persistenten Datenbasiszustand erreicht.

Persistenz und Dauerhaftigkeit

- Persistenz schafft die Voraussetzungen für Dauerhaftigkeit.
- Unbegrenzt lange Speicherung der Daten verschärft die Forderung nach Konsistenz, denn es steigt die Wahrscheinlichkeit, dass durch Störungen oder Fehler die Daten in ihrem Inhaltsbezug korrumpiert werden.
- Beispiele: Ausfall von Hintergrundspeichern, Datenträgern oder dem Rechner; externe Ereignisse wie Feuer, Wasser, Klima, Alterung mit physischer Vernichtung der Datenbasis.

Skalierbarkeit/Leistung in der Datenverwaltung



- Enge Wechselwirkung zwischen Skalierbarkeit und Leistung
- Daher gemeinsame Betrachtung: **Performanz**

Wechselwirkung Skalierbarkeit/Leistung

Aspekte der Leistung:
- **Effizienz**: Möglichst geringer Ressourcenverbrauch durch die Dienste.
- **Verfügbarkeit**: Bedarfsaktueller Zugang zu den Ressourcen.

Aspekte der Skalierbarkeit:
- Wachstum der Datenbasis ⇔ Wachstum der Transaktionslast

Skalierbarkeit

- Forderung:
  - Leistung des DBMS darf mit wachsender DB-Größe und wachsender Nutzungsintensität (Anzahl und Komplexität der angeforderten Transaktionen) nicht kollabieren.
- Maßzahlen für Leistung:
  - **Durchsatz** = Anzahl abgeschlossener Transaktionen pro Sekunde,
  - **Antwortzeit** = Zeitspanne zwischen Beginn und Ende einer Transaktion.
- Realisierung durch:
  - effiziente Datenstrukturen für Datenbasis-Zustand,
  - effiziente Algorithmen für Operatoren,
  - Verteilung des DBMS auf mehrere Rechner.

Kapitel 1: Datenbankdienste

Dienstfunktionalität und Dienstmerkmale