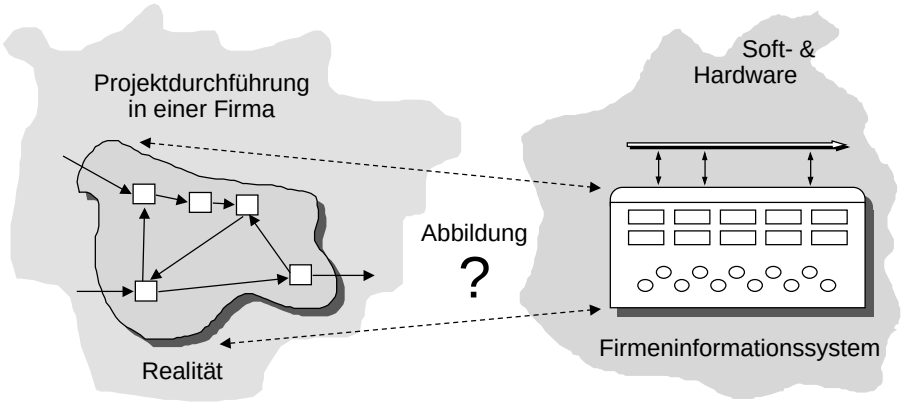
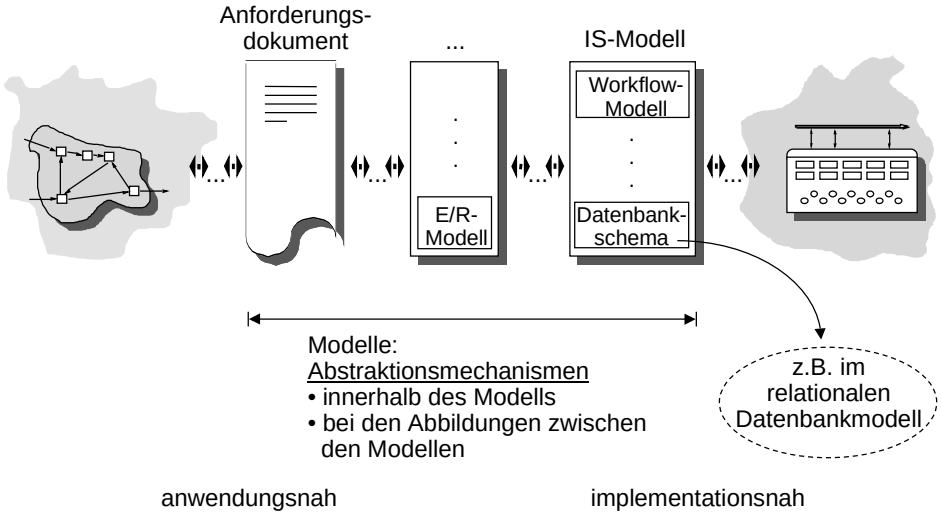
Kapitel 2: Grundlagen der Datenmodellierung

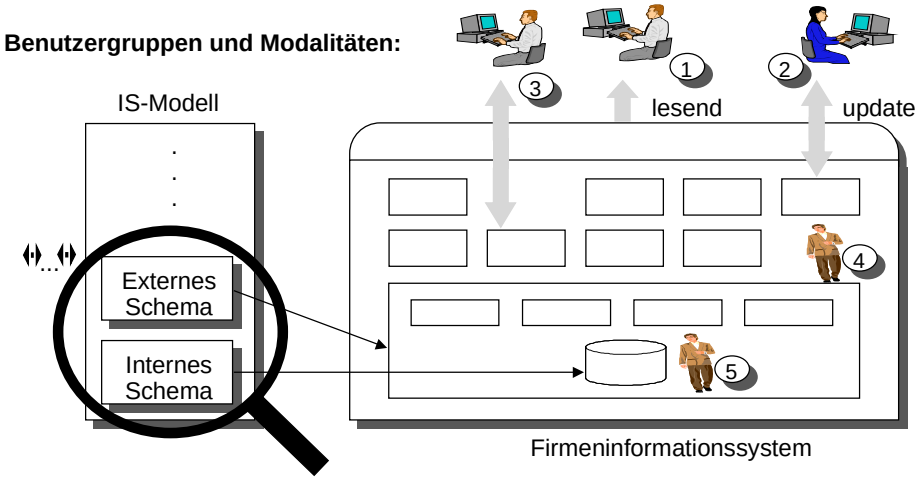
Motivation: Wozu Modelle?



Modelle und Abstraktion



Konkret: Datenbankmodelle



Verwendung von Modellen (1)

"Die Behauptung, daß Computersysteme ein Modell der Welt oder eines Realitätsausschnitts (slice of reality) darstellen können, scheint im Falle von Informationssystemen besonders angebracht."

"Damit die Realisierung solcher Systeme erleichtert wird und auch die Kommunikation mit ihren späteren Benutzern, sollte dieses Modell die Konzeptbildung des Benutzers im Anwendungsbereich (universe of discourse) in möglichst direkter und natürlicher Weise widerspiegeln."

Anwendungsgebiete:

- ❏ Analyse von Geschäftsvorgängen
- ❏ Informationssysteme: Entwurf, Dokumentation, Analyse (→ *Reverse Engineering*)

Nutzer:

- ❏ Menschen: Benutzer, Programmierer, ...
- ❏ Systeme: Generatoren, Repositories, Programmiersprachen, ...

Verwendung von Modellen (2)

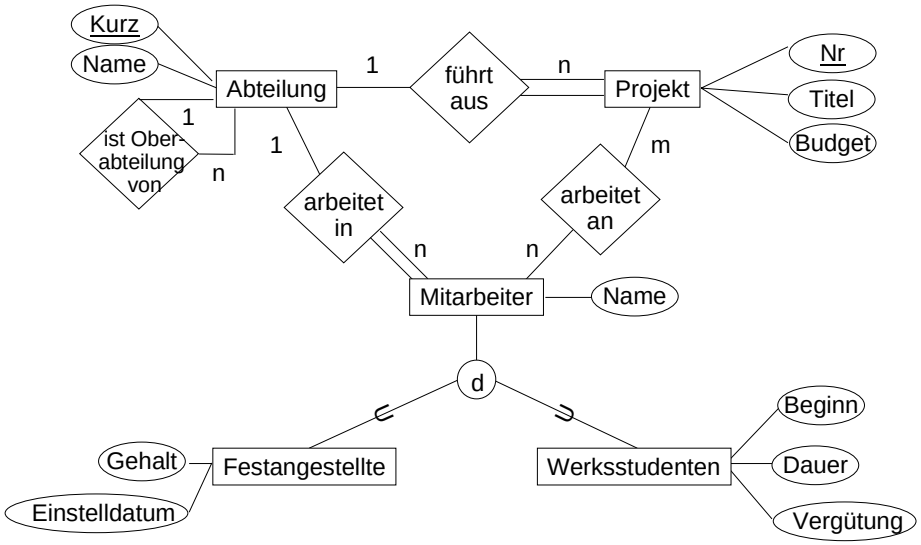
Modelle liefern wichtige Metainformationen über Informationssysteme:

- ❑ Speicherung über die Zeit:
  - Systementwurf 1996
  - Systemnutzung 1996 ... 2040
- ❑ Kommunikation zwischen Personen und Systemen:
  - Anwender, Systementwickler, Datenbanksystem, ...

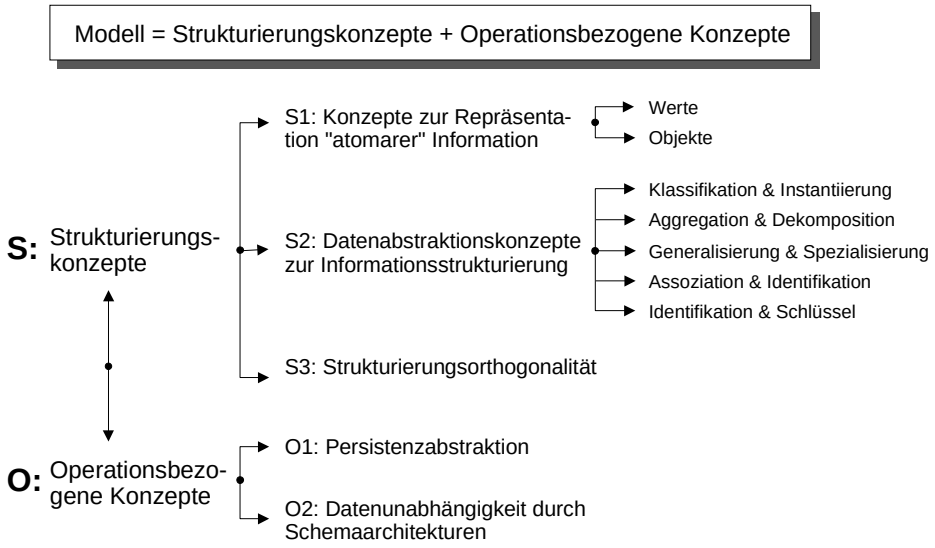
Entscheidend:

Das richtige Modell für die richtige Person zum richtigen Zeitpunkt.

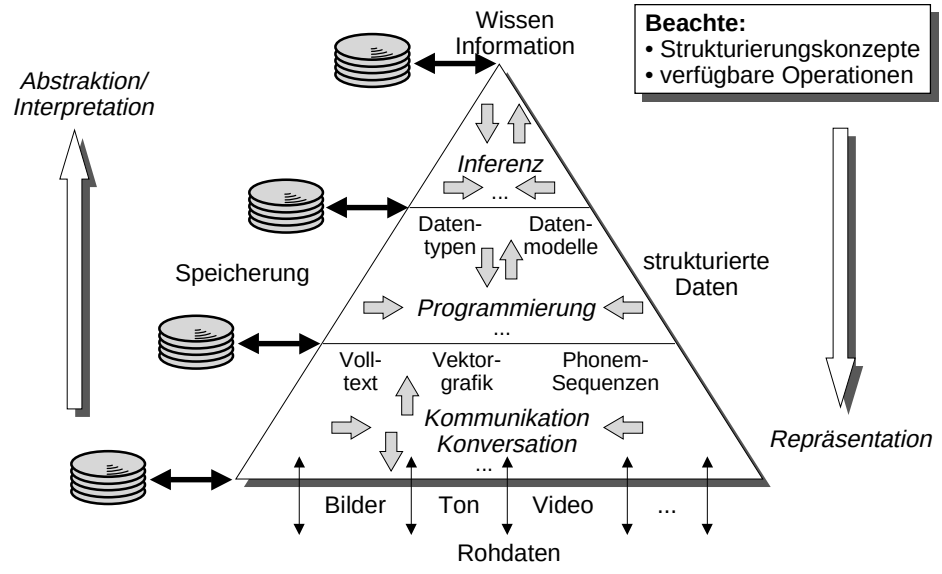
Beispiel: ER-Diagramm für das Firmen-IS



Datenbankmodellierungskonzepte: Überblick



Exkurs: Verhältnis zw. Daten und Information



S1: Repräsentation "atomarer" Information (1)

Werte (Literale):

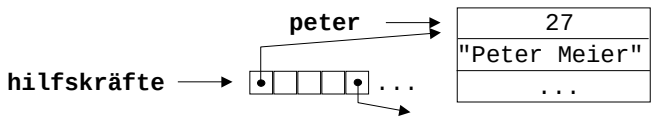
- ❏ Sematikunabhängig vom Datenbankzustand
- ❏ Es wird unterschieden in
  - Basiswerte (z.B. Zahlen, Zeichen, Zeichenketten)
  - `3.1415; "Z"; "Otto"`
  - Zusammengesetzte Werte: heterogene Strukturen (Tupel, Rekords, Strukturen) und homogene Strukturen (Arrays, Listen, Mengen, Multimengen)
  - `{1,2,3,4}; record age = 31 name = "Peter" end`
- ❏ Operationen auf Werten haben Kopiersemantik (z.B. Addition, Feldzugriff)
  - `menge + {1,2,3}`
- ❏ Verlustfreie Darstellung als lineare textuelle Repräsentation

S1: Repräsentation "atomarer" Information (2)

Objekte:

- ❏ Semantikabhängig vom Datenbankzustand (→ Zustandsvariablen in imperativen Programmiersprachen)
- ❏ Der Zustand kann durch destruktive Zuweisung verändert werden.

  `peter.name := "Peter Meier"`
- ❏ Objektidentität (object identity, OID) bleibt unabhängig vom Zustand erhalten
- ❏ Objekte können mehrfach über die OID referenziert werden (sharing



- ❏ Änderungen des Objektzustands werden unmittelbar auf allen Pfaden sichtbar, z.B. die Änderung des Studentennamen (→ Referenzsemantik)
- ❏ Spezielle Notation zur textuellen Repräsentation der (zyklischen) Graphenstruktur

S2: Datenabstraktionskonzepte z. Informationsstrukturierung

In fast allen Datenbankmodellen findet man Konstrukte für die folgenden Abstraktionskonzepte:

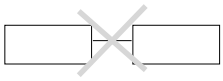
- ❏ Klassifikation und Instantiierung
- ❏ Aggregation und Dekomposition
- ❏ Generalisierung und Spezialisierung
- ❏ Assoziation und Identifikation
- ❏ Identifikation und Schlüssel

In späteren Kapiteln werden diese Konstrukte beschrieben. Nachfolgend werden die Abstraktionskonzepte anhand einer populären grafischen Notation erklärt.

Entity-Relationship-Diagramme wurden von P.P.S. Chen vorgeschlagen (vgl. P.P.S. Chen. "The Entity Relationship Model - Toward a Unified View of Data. In: ACM Transactions on Database Systems, Vol. 1, No. 1, März 1976, S. 9 ff.) und mehrfach erweitert (→ extended E/R diagram, EE/R Modell).

Grundlegende Elemente von ER-Diagrammen

- ❏ Objekttyp 
- ❏ Werttyp 
- ❏ Beziehungstyp 
- ❏ Die Elemente von ER-Diagrammen bilden einen bipartiten Graphen:

 und 

- ❏ Verbindungen zwischen Symbolen der gleichen Typen sind nicht erlaubt.

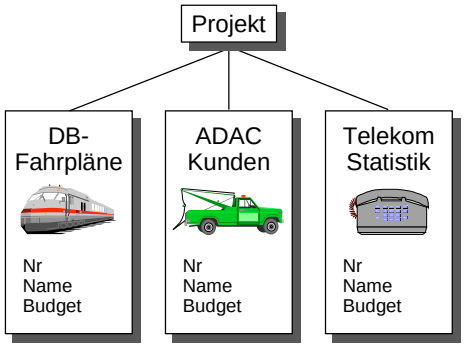
Klassifikation und Instantiierung (1)

- ❏ Objekte mit ähnlichen Eigenschaften können zu Klassen zusammengefaßt werden.
- ❏ Jedes Objekt ist die Instanz einer Klasse.
- ❏ Strukturelle Klassifizierung
- ❏ Dynamische Extension

Klassifikation und Instantiierung (2)

Beispiel:

- ❏ Unterschiedliche Projekte werden zur Klasse "Projekt" zusammengefaßt.



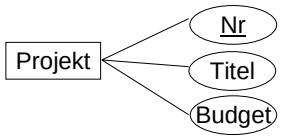
Aggregation und Dekomposition (1)

- ❏ Eine Instanz umfaßt (aggregiert) meist mehrere Eigenschaften (Attribute).
- ❏ Ein Attribut besteht aus einem Attributnamen und einem Attributwert.
- ❏ Durch Dekomposition kann über den Attributnamen auf den Attributwert zugegriffen werden (→ Selektion von Rekord-Komponenten in Programmiersprachen).
- ❏ In Programmiersprachen werden Tupel oder Rekords benutzt, um die Attribute einer Instanz zu aggregieren.
- ❏ Objekte können zu übergeordneten Objekten aggregiert werden:
  - Beziehungen zwischen Komponenten und übergeordnetem Objekt
  - Übergeordnetes Objekt kann wiederum an Beziehungen teilnehmen.

Aggregation und Dekomposition (2)

Beispiele:

- ❏ Ein Projekt wird beschrieben durch
  - eine Nummer
  - einen Titel
  - das Budget



- ❏ Eine Stückliste aggregiert die auf der Liste enthaltenen Artikel (im E/R-Modell ist die "part of"-Beziehung nicht direkt darstellbar).

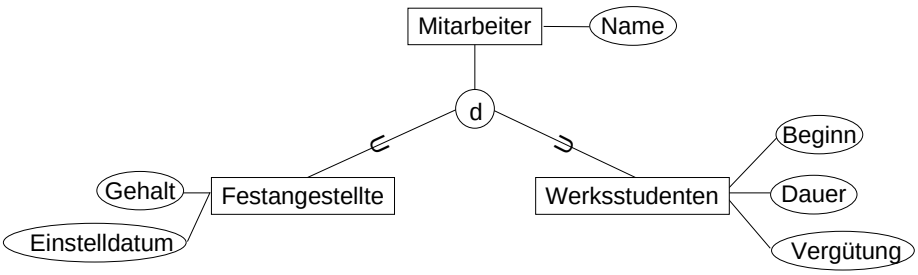
Generalisierung und Spezialisierung (1)

- ❏ Spezialisierung bezeichnet die Verfeinerung einer Klasse.
- ❏ Generalisierung ist die Vergröberung einer Klasse.
- ❏ Spezielle Klassen (Subklasse) und allgemeine Klassen (Superklasse) bilden eine Subklassenhierarchie (→ Subtypisierung, Typhierarchie).
- ❏ Instanzen einer Klasse sind auch Instanzen der Superklasse.
- ❏ Spezialisierungen können disjunkt oder überlappend sein.
- ❏ Subklassen erben die Eigenschaften der Superklasse und fügen evtl. neue hinzu (→ Vererbung).
- ❏ Bei Operationen auf Klassen können auch Instanzen von Subklassen verwendet werden (→ Subtypisierung).

Generalisierung und Spezialisierung (2)

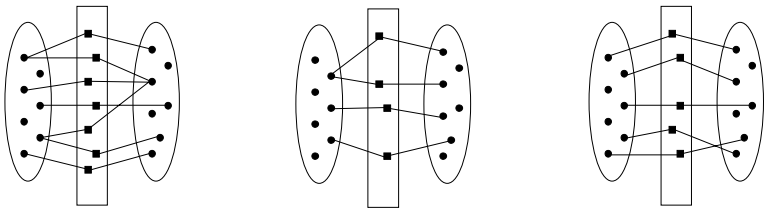
Beispiel:

- ❏ Festangestellte und Werksstudenten sind Mitarbeiter.
- ❏ Festangestellte haben die zusätzliche Eigenschaften Gehalt und Einstelldatum.
- ❏ Werksstudenten haben die zusätzliche Eigenschaften Beginn, Dauer und Vergütung.



Assoziation und Identifikation (1)

- ❏ Objekte können miteinander in Beziehung gesetzt (assoziiert) werden:
  - Binäre Beziehungen assoziieren zwei Klassen oder Objekte.
  - Ternäre Beziehungen assoziieren drei Klassen.
  - Allgemein: n-äre Beziehungen zwischen n Klassen, wobei n der Grad der Beziehung ist.
- ❏ Kardinalitätsbeschränkungen legen die genaue Zahl oder ein Intervall für die Anzahl der in Beziehung stehenden Instanzen fest.



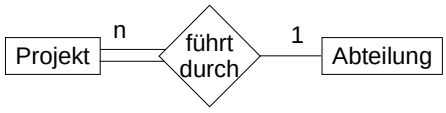
Assoziation und Identifikation (2)

- ❏ **Totale Partizipation**: Jede Instanz einer Klasse muß mit einer Instanz der zweiten Klasse in Beziehung stehen.
- ❏ **Partielle Partizipation**: Eine Instanz einer Klasse kann in Beziehung zu einer Instanz der zweiten Klasse stehen (s. Beispiel auf der nächsten Folie).
- ❏ Rollennamen identifizieren die Menge der Instanzen, die mit einer anderen Instanz in Beziehung stehen.
- ❏ Rollen können als abgeleitete Attribute verstanden werden, die die Menge der Instanzen als Attributwerte besitzen.

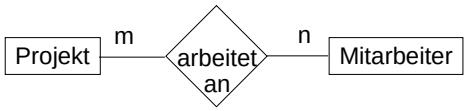
Assoziation und Identifikation (3)

Beispiele:

- Projekte werden von Abteilungen durchgeführt. Jedes Projekt muß einer Abteilung zugeordnet sein. Eine Abteilung kann mehrere Projekte ausführen.



- An Projekten arbeiten Mitarbeiter. Ein Mitarbeiter kann an mehreren Projekten arbeiten. Jedes Projekt wird von beliebig vielen Mitarbeitern bearbeitet.



- Bemerkung: In der Literatur findet man auch andere Beschriftungsregeln.

Identifikation und Schlüssel (1)

Zur **Identifikation** existieren zwei grundlegende Ansätze in Datenbankmodellen:

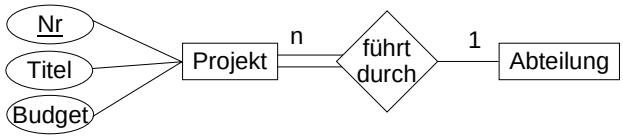
- **Referentielle Identifikation** bezeichnet direkte Verweise auf Objekte (→ Zeiger in Programmiersprachen).
- **Assoziative Identifikation** verwendet die Werte von Attributen oder Attributkombinationen, um sich eindeutig auf Objekte zu beziehen (→ Schlüssel: Ausweisnummer, Fahrgestellnummer, ...).
- In der Praxis benötigt man häufig beide Formen der Identifikation.

Schlüssel:

- Schlüssel sind Attribute oder Attributkombinationen mit innerhalb einer Klasse eindeutigen Werten und eignen sich deshalb zur Identifikation.
- Es kann mehrere Schlüsselkandidaten geben (Primärschlüssel, Sekundärschlüssel).
- Schlüssel stellen als Attributwerte Beziehungen zu anderen Objekten her (Fremdschlüssel).
- Durch Fremdschlüssel referenzierte Objekte müssen existieren (→ referentielle Integrität).

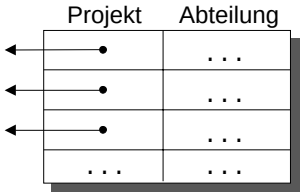
Identifikation und Schlüssel (2)

Beispiel: Projekte können durch eine Nummer eindeutig identifiziert werden.



Dabei existieren zwei Möglichkeiten zur Identifikation von Projekten innerhalb der Assoziation "führt durch":

Referentielle Identifikation



Assoziative Identifikation

Projekt	Abteilung
4711	. . .
4712	. . .
4713	. . .
. . . .	. . .

S3: Strukturierungsorthogonalität

- ❏ Wiederholte Anwendung von Abstraktionsmechanismen:
- ❏ Beispiel:

  Ein Buch wird durch wiederholte Anwendung der Aggregation beschrieben:

  Kapitel = Liste von Unterkapiteln
  Buch = Liste von Kapiteln

- ❏ Datenbankmodelle, die eine uneingeschränkte, wiederholte Anwendung von Abstraktionsmechanismen gestatten, besitzen Strukturierungsorthogonalität.
- ❏ Strukturierungsorthogonalität erhöht die Ausdrucksmächtigkeit von Datenmodellen.
- ❏ **Gegenbeispiele**:
  - Keine wiederholte Aggregation im E/R-Modell
  - Keine wiederholte Aggregation im relationalen Modell

O1: Persistenzabstraktion

- ❏ Direkte Mechanismen zur Benennung und Manipulation langlebiger Daten ohne explizite Lade- und Speicherungsoperationen beim Zugriff sind verfügbar.
- ❏ Ziel: Algorithmen des Informationssystems arbeiten ohne komplexe Details des Datentransfers zwischen Primär- und Sekundärspeicher.
- ❏ **Orthogonale Persistenz**: Jede Datenstruktur für transiente Daten (z.B. sequentielle Dateien, relationale Tabellen) ist auch für persistente Daten (z.B. Rekords, Vektoren, Listen, Zeiger) verfügbar und umgekehrt.
  - Resultat: Der Anwendungsprogrammierer muß keine Datenstrukturkonvertierungen zwischen persistenter und transienter Speicherung durchführen.
- ❏ **Persistenzunabhängigkeit**: Zusätzlich zur orthogonalen Persistenz können Prozeduren, Funktionen und Anfragen uniform auf transiente und persistente Daten als Argumente angewendet werden.

Alternativen zur Definition d. Datenlebensdauer (1)

1. Persistente Typen:

- ❏ Definition der Lebensdauer eines Datums explizit zusammen mit seiner Struktur (→ persistente Datenbanktabellen und transiente Ergebnistabellen)

```
relation of person;     pointer to person;
```

- ❏ Nachteil: Inkompatibel mit dem Konzept der Persistenzunabhängigkeit

2. Persistente Sichtbarkeitsbereiche:

- ❏ Ableitung der Definition der Lebensdauer eines Datums implizit aus dem statischen Kontext, in dem das Datum deklariert wird.
  - Transientes Datum: Eine lokal in einer Anfrage definierte Variable.

```
select * from Mitarbeiter
  into oldPerson
  where age > 80
```

  - Persistentes Datum: Eine global in einem Datenbankschema definierte Variable.

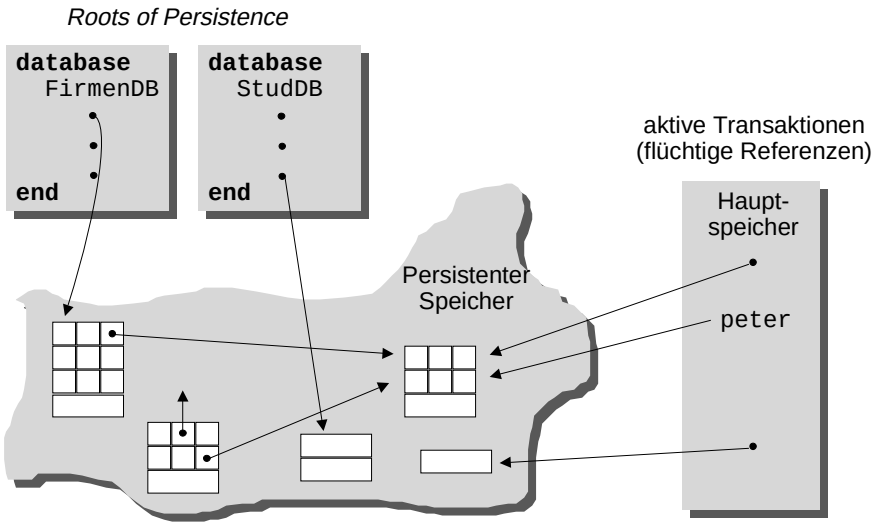
```
database FirmenDB
  relation Mitarbeiter
  relation Projekte
  ...
end
```

Alternativen zur Definition d. Datenlebensdauer (2)

3. Transitive Erreichbarkeit:

- ❏ Verfeinerung des Modells der persistenten Sichtbarkeitsbereiche
- ❏ Statische persistente Sichtbarkeitsbereiche existieren als Wurzeln der Persistenz (roots of persistence), z.B. die Datenbankschemata eines Datenbanksystems.
- ❏ Außerdem werden dynamische Abhängigkeiten zwischen Daten berücksichtigt, z.B. hängt die Semantik eines Datums A von der Semantik eines Datums B ab, wenn:
  - A besitzt B als Rekordfeld oder Mengenelement.
  - A ist ein Zeiger auf B.
  - A ist ein Fremdschlüssel auf B.
  - A ist eine Subklasse von B.
  - A ist eine Funktion oder Anfrage, die B benutzt.

Alternativen zur Definition d. Datenlebensdauer (3)



Alternativen zur Definition d. Datenlebensdauer (4)

- ❏ Die Persistenz des Datums A impliziert automatisch die Persistenz des Datums B (Ziel: Sicherstellung der → referentiellen Integrität).
- ❏ Also gelten die Daten als persistent, die von den Wurzeln der Persistenz aus transitiv erreichbar sind.
- ❏ Nutzen:
  - Dynamische Erreichbarmachung einer komplexen, transient erzeugten Datenstruktur von einer persistenten Datenstruktur aus durch eine einzelne Zuweisung.
  - Vermeidung subtiler Speicherlecks (memory leaks), die aufgrund fehlender Speicherfreigabeanweisungen nicht mehr erreichbarer persistenter Daten in Informationssystemen entstehen.
- ❏ Generalisierung des Konzepts der automatischen Freispeicherverwaltung (→ Garbage Collection).

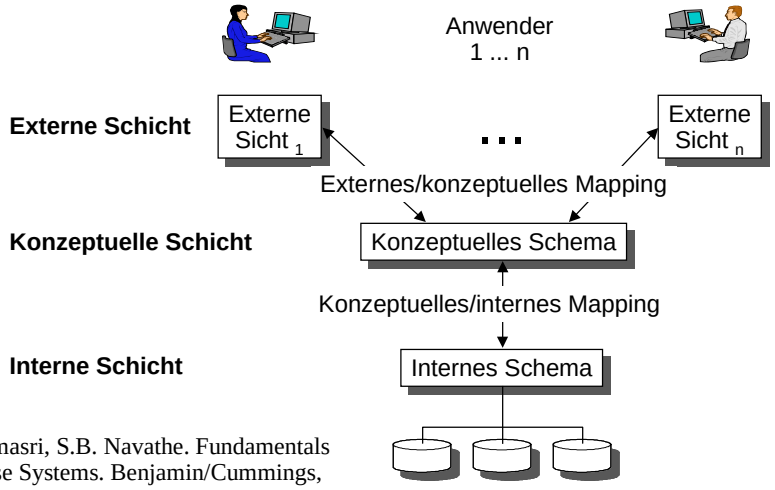
O2: Datenunabhängigkeit durch Schemaarchitekturen (1)

Drei-Schichten-Architektur von Datenbanksystemen:

- ❏ Auch "ANSI/SPARC Architektur"
- ❏ Ziel:
  - Unabhängigkeit zwischen Daten und Anwendungen
  - Unterstützung mehrerer Benutzersichten
- ❏ Definition verschiedener Schemata in den einzelnen Schichten
- ❏ Ablage der Schemata im Datenbank-Katalog
- ❏ Drei Schichten:
  - Interne Schicht
  - Konzeptuelle Schicht
  - Externe Schicht

O2: Datenunabhängigkeit durch Schemaarchitekturen (2)

Die Drei-Schichten-Architektur:



vgl. R. Elmasri, S.B. Navathe. Fundamentals of Database Systems. Benjamin/Cummings, Redwood City, CA. 1989.

O2: Datenunabhängigkeit durch Schemaarchitekturen (3)

Externe Schicht:

- ❏ Jedes externe Schema (Benutzersicht: Benutzer = Anwendungsprogramm) beschreibt die Sicht eines oder mehrerer Benutzer auf die Daten.
- ❏ Für den Benutzer nicht relevante Daten werden vor ihm verborgen.
- ❏ Beispiel: Das Schema eines Projektinformationssystems verbirgt das Gehalt der Mitarbeiter.

Konzeptuelle Schicht:

- ❏ Das konzeptuelle Schema legt die Strukturen der konzeptuellen Sicht der gesamten Datenbank für die gesamte Benutzergemeinde fest (Vereinigung aller Anwendersichten zu einer gemeinschaftlichen Sicht).
- ❏ Berücksichtigt werden Entitäten, Datentypen, Beziehungen und Integritätsbedingungen, die physikalischen Speicherstrukturen werden verborgen.
- ❏ Beispiel: Das Schema eines Firmeninformationssystems sammelt alle Informationen über die Mitarbeiter.

O2: Datenunabhängigkeit durch Schemaarchitekturen (4)

Interne Schicht:

- ❏ Internes Schema beschreibt die physikalischen Speicherstrukturen der Datenbank.
- ❏ Unter Benutzung eines physikalischen Datenmodells werden Details der Datenspeicherung und Zugriffspfade beschrieben.
- ❏ **Beispiele:**
  - Separate Speicherbereiche für Festangestellte und Werksstudenten
  - B-Tree: Zugriff auf Projekte über die Projektnummer

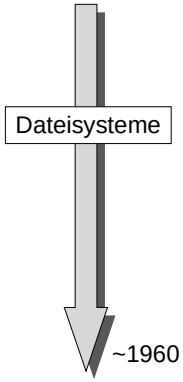
O2: Datenunabhängigkeit durch Schemaarchitekturen (5)

Datenunabhängigkeit:

- ❏ Schutz des Benutzers eines DBMS vor nachteiligen Auswirkungen im Zuge von Änderungen in der Systemumgebung (1996 ... 2040).
- ❏ Arten der Datenunabhängigkeit:
  - **Logische Datenunabhängigkeit:**
    - Das konzeptuelle Schema kann ohne Konsequenzen für das externe Schema geändert werden.
    - Beispiel: Erweiterung der Datenbank um eine neue Klasse oder Zusammenfassung mehrerer Klassen durch Generalisierung im konzeptuellen Schema.
  - **Physische Datenunabhängigkeit:**
    - Das interne Schema kann unabhängig vom konzeptuellen Schema geändert werden, ohne daß Funktionsänderungen in den Anwendungen auftreten.
    - Beispiel: Reorganisation der Daten oder Einrichtung neuer Zugriffspfade.

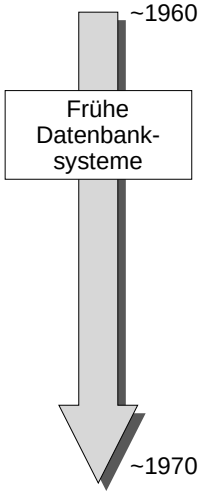
Datenbanksysteme und ihre Modelle: Historie (1)

1. Records fester Größe aggregieren Attribute verschiedener Datentypen.
2. Dateien fassen gleichartige Rekords zusammen, können dynamisch wachsen und schrumpfen, in ihrem Umfang die Kapazität des Hauptspeichers überschreiten und langlebig gespeichert werden.
3. Zugriffsstrukturen (z.B. Hashtabellen, ISAM-Dateien) ermöglichen es, Rekords basierend auf ihren Attributwerten effizient in einer Datei zu selektieren.
4. Explizite oder implizite Datei- und Rekordsperren werden auch zur Kontrolle des parallelen Zugriffs eingesetzt.



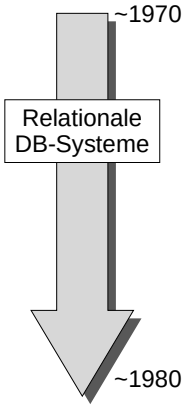
Datenbanksysteme und ihre Modelle: Historie (2)

5. Recordidentifikatoren, Adressen und Verweisattribute ermöglichen es, Rekords zu hierarchischen oder netzwerkartigen Datenstrukturen zu verketten.
6. Mehrere Indexstrukturen können gleichzeitig geöffnet sein und werden vom Datenbanksystem konsistent als eine gemeinsame Datenbank mit Verweisinformation gewartet.
7. Zugriffskontrollmechanismen beschränken Rekordzugriffe auf authorisierte Agenten.
8. Es existiert eine Trennung zwischen logischem und physischen Schema.
9. Transaktionen bieten automatische Fehlererholung, Verklemmungserkennung und Integrität beim Mehrbenutzerzugriff.



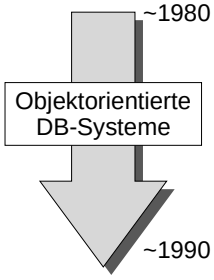
Datenbanksysteme und ihre Modelle: Historie (3)

10. Hochsprachliche, deklarative Anfragesprachen wie SQL ermöglichen eine abstrakte mengenorientierte Datenmodellierung und Datenmanipulation unabhängig von der Verfügbarkeit gespeicherter Verweisstrukturen.
11. Zusätzliche anwendungsorientierte Dienstschnittstellen unterstützen interaktive Anfragen, Formular- und Reportdefinitionen sowie die integrierte Beschreibung von Algorithmen und Anfragen in Sprachen der 4. Generation.
12. Es existiert eine Trennung zwischen konzeptuellem und logischem Schema.

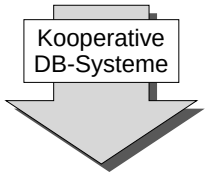


Datenbanksysteme und ihre Modelle: Historie (4)

- ❏ 13. Programmiersprachenanbindung
- ❏ 14. Persistenzabstraktion
- ❏ 15. Objektidentität
- ❏ 16. Verhaltensmodellierung
- ❏ 17. Vererbung von Struktur und Operationen
- ❏ 18. Versionsverwaltung



- ❏ Verteilungsabstraktion
- ❏ Kooperationsmodelle
- ❏ Multi-mediale Daten
- ❏ ...



Datenbanksprachen

Datenbanksprachen umfassen typischerweise Konstrukte:

- ❏ zur Datendefinition
- ❏ zur Schemaänderung

Datendefinitionssprache

- ❏ zur deskriptiven Formulierung von Anfragen
- ❏ zur Modifikation der Datenbank

Datenmanipulationssprache

Modellunabhängige Notation für DB-Zustände (1)

Beschreibung der Konzepte mit Hilfe einer **datenmodellunabhängigen Notation** für Datenbankzustände.

Literale:

- ❏ Beschreiben atomare Werte, die als Bausteine in komplexen Informationsstrukturen verwendet werden können.

```
30, "MFSW", NULL
```

Aggregat:

- ❏ Beschreibt einen zusammengesetzten Wert, der aus einer (ungeordneten) Folge von Paaren (Attributname X_i, Attributwert A_i) besteht.
- ❏ Die Attribute eines Aggregats können über ihre Namen angesprochen werden.
- ❏ Die meisten DB-Modelle unterstützen selektive Wertzuweisungen, jedoch nicht das Hinzufügen oder Löschen von Attributen.
- ❏ Analog: *Records* in Pascal, *Structures* in C

X_1	X_2	...	X_n
A_1	A_2	...	A_n

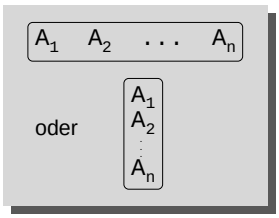
Modellunabhängige Notation für DB-Zustände (2)

Referenz:

- ❑ Beschreibt (gerichteten, logischen) Verweis auf ein anderes Objekte (hier: Verweis auf Objekt A).
- ❑ Ein Pfeil führt von einem referenzierenden zum referenzierten Objekt.
- ❑ Analog: *Pointer* in Programmiersprachen

Kollektion:

- ❑ Beschreibt einen zusammengesetzten Wert, der aus einer variablen Anzahl unbenannter Komponenten besteht (hier: $A_1, A_2, \ldots, A_n$).
- ❑ Dabei wird von einer dynamischen Klassifikation der Objekte (Werte) in der Kollektion ausgegangen.
- ❑ Typisch: homogene Struktur der Komponenten



Regularität und typisierte Datenbank-Schemata (1)

Die Regularität von Datenbankzuständen erlaubt die strukturierte Beschreibung *möglicher* oder *zulässiger* Datenbankzustände durch ein Datenbank-Schema (statische Klassifikation durch Typisierung).

Analogie: Schemaebene in DBMS ↔ Typebene in Programmiersprachen

Typisierung auf der Schemaebene:

- ❑ **Basisdatentyp:** (→ *Domänen*) beschreibt Wertebereiche. Er setzen sich aus einer Menge von Literalen zusammen.

```
Int:  {...,-1,0,1,...}
Char: {a,...,z,A,...,Z}
```

- ❑ **Aggregattyp:** beschreibt die Struktur von Aggregaten. Er setzt sich aus Paaren (Bezeichner, Attributtyp) zusammen. Daher können Operationen auf den Attributen auf ihre Zulässigkeit überprüft werden (`x.name = 3` ↯).

```
record
  name :String
  budget :Float
end
```

Regularität und typisierte Datenbank-Schemata (2)

- **Referenztyp:** Beschreibt die Struktur der referenzierten Objekte. In Datenbanken findet man häufig eine Kombination von Referenz und Kollektionstyp.

```
pointer to Abteilung
(einzelne Referenz)
pointer* to Abteilung
(mengenwertige Referenz)
```

- **Kollektionstyp:** (→ *Massendatentypen*) Beschreibt die Struktur von homogenen Kollektionen. Dabei wird die Art der Kollektion (Liste, Menge etc.) und der Typ ihrer Elemente festgelegt.

```
set of Abteilung
bag of Abteilung
list of Abteilung
```


Durchgängiges Beispiel die folgenden Kapitel

Die Projektdatenbank als Ausschnitt des Firmeninformationssystems (s. Folie 2.6).