Datenbanksysteme I

Sommersemester 2003

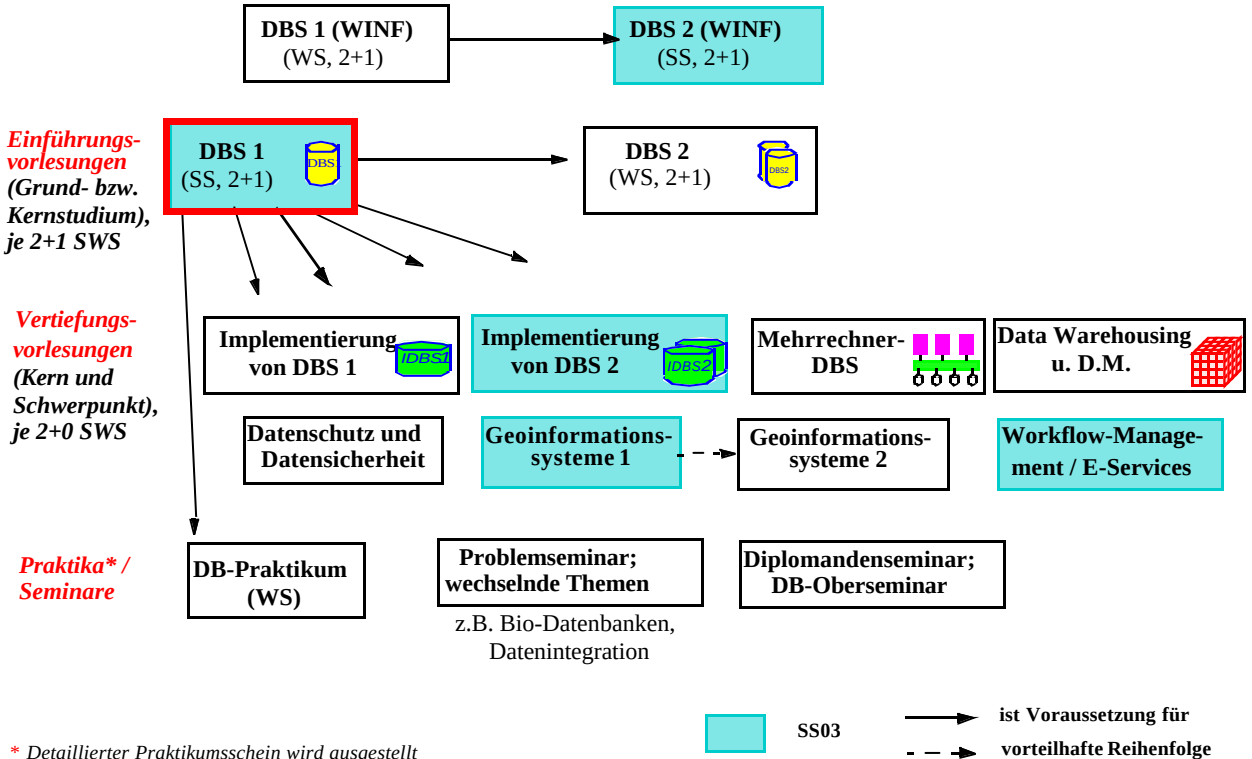
http://dbs.uni-leipzig.de

Prof. Dr. E. Rahm

Universität Leipzig

Institut für Informatik

Lehrveranstaltungen zu "Datenbanken"



* *Detaillierter Praktikumsschein wird ausgestellt*

Zur Vorlesung allgemein

- Vorlesungsumfang: 2 + 1 SWS
- Vorlesungsskript
  - im WWW abrufbar unter *http://dbs.uni-leipzig.de* (PDF, PS und HTML)
  - ersetzt keinesfalls die Vorlesungsteilnahme !
  - ersetzt nicht zusätzliche Benutzung von Lehrbüchern
- Übungen
  - Durchführung in zweiwöchentlichem Abstand
  - selbständige Lösung der Übungsaufgaben wesentlich für Lernerfolg
  - Übungsblätter im Web; Übungen zu SQL am Rechner (SQL-Trainer)
  - Ausgabe der Übungsblätter: 14.4., 5.5., 19.5., 2.6., 23.6.
  - keine Abgabe von Lösungen
- Übungsscheinvergabe (obligatorisch für Vordiplom Informatik)
  - Zwischenklausur am 26. Mai 2003 +
  - Abschlussklausur im Juli 2003

Zur Vorlesung (2)

- Übungsgruppen (Eintragung im Web)

Nr.	Termin	Woche	Hörsaal	**Termine**
1	Mo, 15:15	A	HS 21	**5.5., 19.5., 2.6., 16.6., 30.6.**
2	Mo, 15:15	B	HS 21	**28.4., 12.4., 26.4., 23.6., 7.7.**
3	Di, 17:15	A	SG 3-11	**6.5., 20.5., 3.6., 17.6., 1.7.**
4	Di, 17:15	B	SG 3-11	**29.4., 13.5., 27.5., 24.6., 8.7.**
5	Do, 11:15	A	SG 3-11	**8.5., 22.5., 5.6., 19.6., 3.7.**

- Ansprechpartner DBS1
  - Prof. Dr. E. Rahm: während/nach der Vorlesung,
    Sprechstunde (Donn. 14-15 Uhr), HG 3-56 , rahm@informatik.uni-leipzig.de
  - Wissenschaftliche Mitarbeiter: Timo Böhme, boehme@informatik.uni-leipzig.de, HG 3-01
  - Web-Angelegenheiten: S. Jusek, juseks@informatik.uni-leipzig.de, Raum HG 3-02

Vorlesungsziele

- Vermittlung von Kenntnissen, Fähigkeiten und Fertigkeiten
  - in der Nutzung von Informations- und Datenmodellen, insbesondere
    - Entity/Relationship-Modell und Erweiterungen
    - Relationenmodell und SQL
    - objektorientierte /objekt-relationale DBS und XML-DBS (-> Vorl. DBS2)
  - in der Modellierung von anwendungsbezogenen Realitätsausschnitten (Miniwelten, Diskursbereiche)
  - in der Programmierung von DB-Anwendungen (Vorl. DBS2)
  - im Entwerfen, Aufbauen und Warten von Datenbanken
- Voraussetzung für Übernahme von Tätigkeiten:
  - Entwicklung von datenbankgestützten Anwendungen
  - Nutzung von Datenbanken unter Verwendung von (interaktiven) Datenbanksprachen
  - Systemverantwortlicher für Datenbanksysteme, insbesondere Datenbank-, Datensicherungs-, Anwendungs- und Unternehmensadministrator
- DBS-Grundkenntnisse in fast allen IT-Berufen erforderlich

Vorläufiges Inhaltsverzeichnis DBS1

1. Einführung / Grundlagen von DBS

- DBS vs. Dateisysteme
- Eigenschaften von DBS
- Datenmodelle
- Transaktionskonzept (ACID)
- Aufbau von DBS: ANSI/SPARC-Modell, Schichtenmodell
- historische Entwicklung
- Einsatzformen: OLTP, OLAP, Data Mining

2. Informationsmodellierung: Entity-Relationship-Modell / UML

- Stufen des DB-Entwurfs
- Grundkonzepte des ER-Modells
- Beziehungstypen, Kardinalitätsrestriktionen
- Generalisierung und Aggregation
- UML (Klassendiagramme)

Vorläufiges Inhaltsverzeichnis (2)

3. Grundlagen des Relationalen Datenmodells
- Relationale Invarianten
- Relationenalgebra

4. Einführung in die Standardsprache SQL
- Befehlsübersicht
- Anfragemöglichkeiten (SELECT)
- Vergleich SQL - Relationenalgebra

5. Datenmanipulation, -definition, und -kontrolle
- SQL-Änderungsoperationen
- Datendefinition, Sichtkonzept (Views)
- Integritätsbedingungen und Trigger
- Zugriffskontrolle

6. Entwurf eines relationalen DB-Schemas (Normalformenlehre)

Lehrbücher (Auswahl)

Autoren	Titel	Verlag	Auflage	Jahr
Kemper, A.; Eickler, A.	Datenbanksysteme	Oldenbourg-Verlag	4	2001
Heuer, A.; Saake, G.	Datenbanken	mitp	2	2000
Elmasri, R.; Navathe, S.B.	Fundamentals of Database Systems	Addison-Wesley	3	1999
Ramakrishnan, R.; Gehrke, J.	Database Management Systems	McGraw Hill	3	2002
Ullman, J.D.; Widom, J.	A First Course in Database Systems	Prentice Hall	2	2001

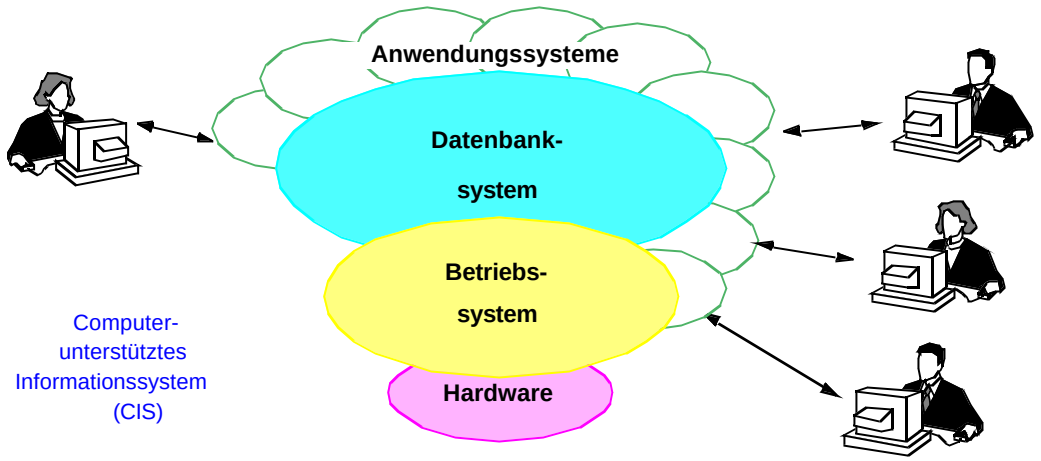
Lernziele Kapitel 1

- Begriffsdefintionen: Datenbank, Datenbanksystem, Datenbankverwaltungssystem
- Vergleich DBS - Dateiverwaltung
- Merkmale von DBS
- Erläuterung des Transaktionskonzepts
- Erläuterung der 3-Schema-Architektur nach ANSI/SPARC und ihrer Bedeutung im Hinblick auf Datenunabhängigkeit
- Darlegung des internen DBS-Aufbaus
- Historische Entwicklung von DBS
- Erläuterung der wichtigsten Datenmodelle mit ihren Vor- und Nachteilen
- Merkmale wichtiger DBS-Einsatzgebiete:
  - Transaktionsverarbeitung (OLTP)
  - Entscheidungsunterstützung (OLAP)

Persistente Datenhaltung

- Programmiersprachen
  - übliche Datenstrukturen: Arrays, Records, Listen, Bäume, Graphen ...
  - Speicherung im Hauptspeicher, d.h. Bestand nur für die Dauer einer Programmausführung ("transiente" Daten)
- persistente Datenspeicherung: innerhalb von Dateien oder Datenbanken
  - Nutzung von Hintergrundspeicher (Magnetplattenspeicher)
  - Daten bleiben über Programmende, Rechnereinschaltung etc. hinaus erhalten
  - andere Arten des Zugriffs: Lese- und Schreiboperationen auf Einheiten von Blöcken und Sätzen
  - inhaltsbasierter Zugriff auf Daten vielfach erforderlich

DBS als Kern von Informationssystemen



- IS = DBS + Anwendungssysteme + Benutzerschnittstellen
- DBS = DB + Datenbankverwaltungssystem (DBVS, DBMS)
  - DB: Menge der gespeicherten Daten
  - Datenbankverwaltungssystem (DBVS): Software-System zur Definition, Verwaltung, Verarbeitung und Auswertung der DB-Daten. Es kann mittels geeigneter Parametrisierung an die speziellen Anwendungsbedürfnisse angepaßt werden.

Beispiele für Informationssysteme / Anwendungen

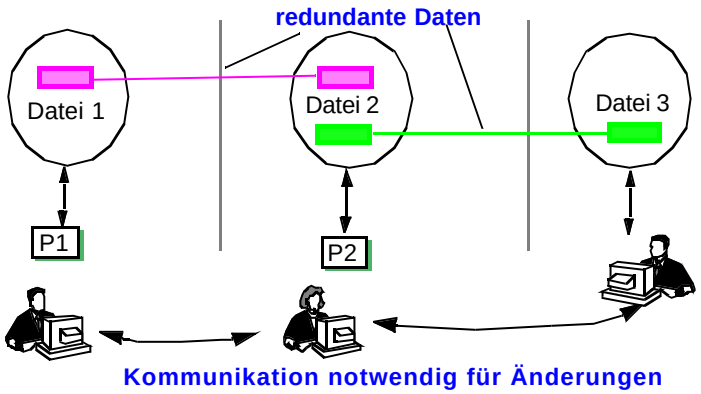
- Universitätsdatenbank
  - Verwaltung von Fakultäten, denen sowohl Studenten als auch Professoren zugeordnet sind
  - Studenten belegen Vorlesungen von Professoren und legen bei ihnen Prüfungen ab
  - Anwendungen sind z.B.: Immatrikulation, Rückmeldung, Exmatrikulationen, Stundenplanerstellung und Planung der Raumbelegung, Ausstellen von (Vor)diplomzeugnissen, Statistiken über Höhrerzahlen / Prüfungsergebnisse, etc.
- Datenbank einer Bank
  - Eine Bank gliedert sich gewöhnlich in mehrere Zweigstellen, denen die Angestellten und Bankkunden zugeordnet sind
  - Verwaltung verschiedenartiger Konten der Bankkunden, z.B. Girokonten, Sparkonten, Hypothekenkonten, Kleinkreditkonten, Wertpapierkonten, etc.
  - Anwendungen sind z.B.: Kontoeinrichtung / -auflösung, Buchung von Zahlungsvorgängen, Kreditgewährung, Zinsberechnung und -verbuchung, Personalverwaltung (Gehaltsabrechnung etc.)

Beispiele (2)

- Datenbank eines Produktionsbetriebes
  - Verwaltung verschiedener Abteilungen und deren Beschäftigte
  - Die Angestellten arbeiten an verschiedenen Projekten mit. Jedes Projekt benötigt für seine Durchführung bestimmte Teile. Jedes Teil kann von Lieferanten bezogen werden.
  - Die in einem Betrieb hergestellten Endprodukte setzen sich i.a. aus mehreren Baugruppen und Einzelteilen zusammen.
  - Typische Anwendungen sind z.B.: Personalverwaltung (Einstellung / Entlassung, Lohn- und Gehaltsabrechnung), Bestellung und Lieferung von Einzelteilen, Verkauf von Fertigprodukten, Lagerhaltung, Bedarfsplanung, Stücklistenauflösung, Projektplanung
- Datenbank einer Fluggesellschaft
  - Verwaltung von Flugstrecken / Flugzeugen / Personal (Piloten, Bord- und Bodenpersonal)
  - Auf Flugstrecken werden Flugzeuge bestimmter Typen mit dafür ausgebildetem Personal eingesetzt
  - Typische Anwendungen sind z.B.: Flugbuchungen von Passagieren, Erstellung von Passagierlisten, Personaleinsatzplanung, Materialeinsatzplanung, Flugplanerstellung, Überwachung der Wartefristen, Gehaltsabrechnung

Nutzung von Dateisystemen

- einfach nutzbarer Ansatz / weit verbreitet, aber erhebliche Probleme
- wiederholte Speicherung gleicher Daten => Redundanz
  - erhöhter Speicherplatzbedarf
  - Konsistenzprobleme !
- enge Bindung von Datenstrukturen an Programmstrukturen (geringe "Datenunabhängigkeit")



- Verantwortung des Programmierers für
  - Aufbau/Inhalt der Dateien, Effizienz des Zugriffs
  - Integrität der Daten, Sicherheit der Daten
- Lösung gleicher Aufgaben in allen Anwendungsprogrammen
  - Speicherverwaltung, Änderungsdienst, Retrieval, Schutzfunktionen
- Annahmen: Alles bleibt stabil ! Alles geht gut !

Aufgaben/Eigenschaften von Datenbanksystemen

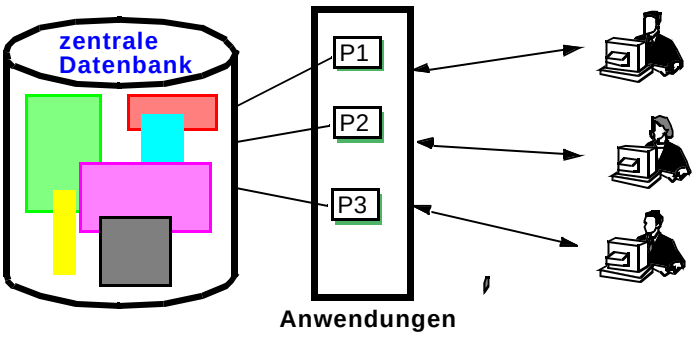
Generell: effiziente und flexible Verwaltung großer Mengen persistenter Daten (z.B. GBytes - T Bytes)

1. Zentrale Kontrolle über die operationalen Daten
2. Hoher Grad an Datenunabhängigkeit
3. Hohe Leistung und Skalierbarkeit
4. Mächtige Datenmodelle und Anfragesprachen / leichte Handhabbarkeit
5. Transaktionskonzept (ACID), Datenkontrolle
6. Ständige Betriebsbereitschaft (hohe Verfügbarkeit und Fehlertoleranz)
   - 24-Stundenbetrieb
   - keine Offline-Zeiten für DB-Reorganisation u.ä.

Zentrale Kontrolle der Daten

- Alle (operationalen) Daten können/müssen gemeinsam benutzt werden
  - keine verstreuten privaten Dateien
  - Querauswertungen aufgrund inhaltlicher Zusammenhänge
- Eliminierung der Redundanz
  - Vermeidung von Inkonsistenzen
  - keine unterschiedlichen Änderungsstände
- Datenbankadministrator (DBA) hat zentrale Verantwortung für Daten
- einfache Entwicklung neuer Anwendungen auf der existierenden DB; Erweiterung/Anpassung der DB (Änderung des Informationsbedarfs)

zentrale DB
statt verteilter Dateien



Datenunabhängigkeit

- Datenunabhängigkeit = Maß für die Isolation zwischen Anwendungsprogrammen und Daten
- Konventionelle Anwendungsprogramme (AP) mit Dateizugriff
  - Nutzung von Kenntnissen der Datenorganisation und Zugriffstechnik
  - kann gutes Leistungsverhalten ermöglichen, aber .....
- Datenabhängige Anwendungen sind äußerst unerwünscht
  - verschiedene Anwendungen brauchen verschiedene Sichten auf dieselben Daten
  - Änderungen im Informationsbedarf sowie bei Leistungsanforderungen erfordern Anpassungen bei den Speicherungsstrukturen und Zugriffsstrategien

    -> möglichst starke Isolation der Anwendungsprogramme von den Daten

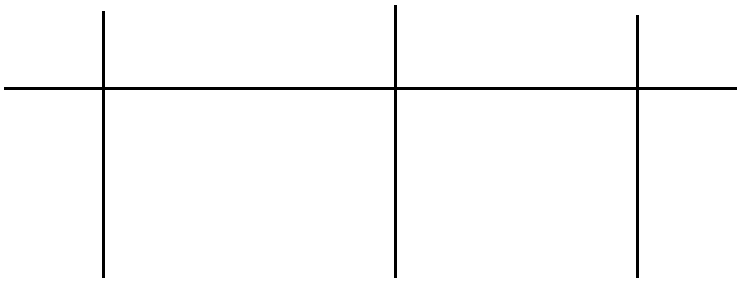
    sonst: extremer Wartungsaufwand für die Anwendungsprogramme
- Minimalziel: *physische Datenunabhängigkeit*
  - Unabhängigkeit gegenüber Geräteeigenschaften, Speicherungsstrukturen, Indexstrukturen, ...
- *logische Datenunabhängigkeit*
  - Unabhängigkeit gegenüber logischer Strukturierung der Daten
  - i.a. nur teilweise erreichbar

Hohe Leistung und Skalierbarkeit

- hoher Durchsatz / kurze Antwortzeiten für DB-Operationen auf großen Datenmengen
  - „trotz“ hoher Datenunabhängigkeit, d.h. möglichst loser Bindung der Programme an die Daten
- Leistungsverhalten
  - DBS-Problem, nicht Anwendungsproblem
  - Zugriffsoptimierung für DB-Anfragen durch das DBS (Query-Optimierung)
  - Festlegung von Zugriffspfaden (Indexstrukturen), Datenallokation etc. durch den DBA (idealerweise durch das DBS)
  - automatische Nutzung von Mehrprozessorsystemen, parallelen Plattensystemen etc. (-> Parallele DBS)
- Hohe Skalierbarkeit
  - Anpassung an gewachsene Leistungsanforderungen (wachsende Datenmengen und Anzahl der Benutzer)
  - Nutzung zusätzlicher/schnellerer Hardware-Ressourcen

Mächtige Datenmodelle

- Datenmodell/DBS-Schnittstelle
  - Operationen zur Definition von Datenstrukturen (Data Definition Language, DDL), Festlegung eines **DB-Schemas**
  - Definition von Integritätsbedingungen und Zugriffskontrollbedingungen (Datenschutz)
  - Operationen zum Aufsuchen und Verändern von Daten (Data Manipulation Language DML)
- Datenstrukturierung
  - Beschreibung der logischen Aspekte der Daten, neutral gegenüber Anwendungen
  - Anwendung erhält logische auf ihren Bedarf ausgerichtete Sicht auf die Daten
  - **formatierte** Datenstrukturen, feste Satzstruktur
  - Beschreibung der Objekte durch Satztyp, Attribute und Attributwerte ($S_i/A_j/AW_k$)
  - jeder Attributwert AW_k wird durch Beschreibungsinformation (Metadaten) A_j und S_i in seiner Bedeutung festgelegt.

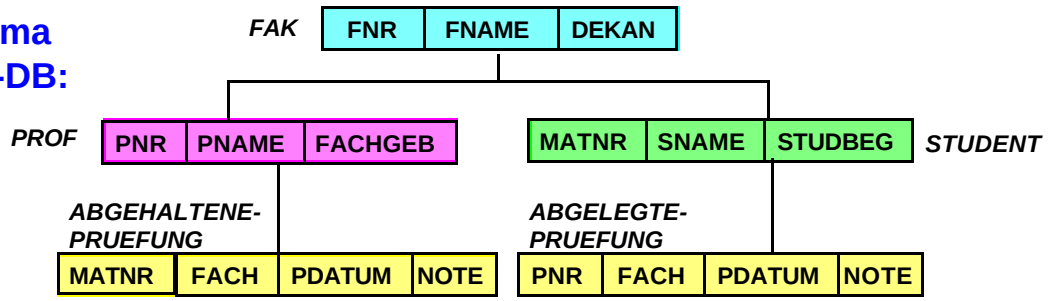


Mächtige Anfragesprachen

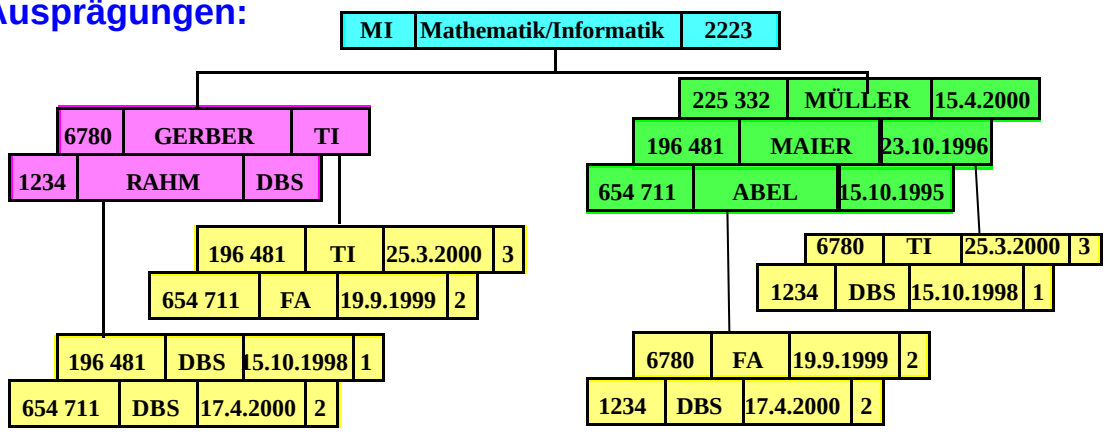
- Art der Anfragesprache (query language)
  - formale Sprache
  - navigierend oder deskriptiv, abhängig von Datenmodell
  - satz- oder mengenorientiert
  - einfache Verknüpfung mehrerer Satztypen („typübergreifende“ Operationen)
- Strukturierung ermöglicht Einschränkung des Suchraumes für Anfragen sowie effiziente Indexunterstützung
- Wünschenswert
  - deskriptive Problemformulierung
  - leicht erlernbar <-> hohe Auswahlmächtigkeit
  - DB-Zugrif im Dialog und von Programmen aus
  - Standardisierung (SQL)
- Erweiterung der Benutzerklassen
  - Systempersonal
  - Anspruchsvolle Laien
  - Gelegentliche Benutzer
  - Anwendungsprogrammierer
  - Parametrische Benutzer

Hierarchisches Datenmodell

Schema Uni-DB:



Ausprägungen:



Hierarchisches Datenmodell (2)

- Beispielanfragen mit IBM IMS (Anfragesprache DL1)

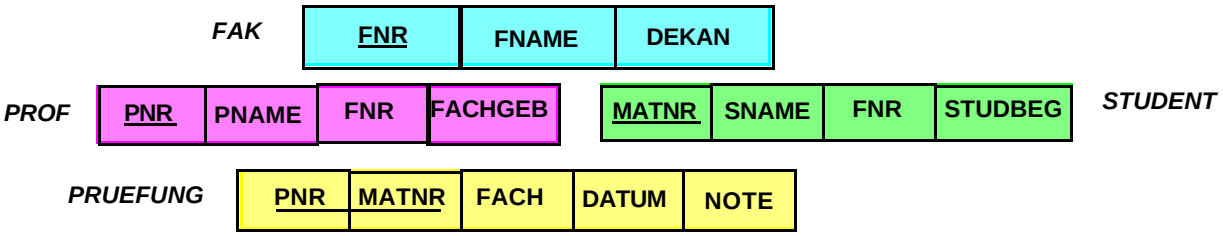
Finde alle Studenten der Fakultät MI, die ihr Studium vor 1995 begonnen haben:

```
        GU FAK (FNR = 'MI');                          { GET UNIQUE }
NEXT:   GNP STUDENT(STUDBEG < '1.1.1998');            {GET NEXT WITHIN PARENT }
        IF END_OF_PARENT THEN EXIT;
        PRINT STUDENT RECORD;
        GOTO NEXT;
```

Finde alle Studenten der Fakultät MI, die im Fach DBS eine Note 2 oder besser erhielten:

```
        GU FAK (FNR = 'MI');
NEXT:   GNP STUDENT;
        IF END_OF_PARENT THEN EXIT;
        GNP ABGELEGTE_PRUEFUNG (FACH = 'DBS') AND (NOTE ≤ '2');
        IF END_OF_PARENT THEN GOTO NEXT;
        PRINT STUDENT RECORD;

        GOTO NEXT;
```

Relationenmodell



FAK

FNR	FNAME	DEKAN
MI	Mathematik/ Informatik	2223

STUDENT

MATNR	SNAME	FNR	STUDBEG
654 711	ABEL	MI	15.10.1995
196 481	MAIER	MI	23.10.1996
225 332	MÜLLER	MI	15.4.2000

PROF

PNR	PNAME	FNR	FACHGEB
1234	RAHM	MI	DBS
2223	GÜNTHER	MI	AN
6780	GERBER	MI	TI

PRUEFUNG

PNR	MATNR	FACH	DATUM	NOTE
6780	654 711	FA	19.9.1999	2
1234	196 481	DBS	17.4.1998	4
1234	654 711	DBS	17.4.2000	2
6780	196 481	TI	25.3.2000	3

Relationenmodell (2)

- Beispielanfragen mit SQL

Finde alle Studenten der Fakultät MI, die ihr Studium vor 1995 begonnen haben:

```
SELECT    *
FROM      STUDENT
WHERE     FNR = 'MI' AND STUDBEG < '1.1.1998'
```

Finde alle Studenten der Fakultät MI, die im Fach DBS eine Note 2 oder besser erhielten:

```
SELECT    S.*
FROM      STUDENT S, PRUEFUNG P
WHERE     S.FNR = 'MI' AND P.FACH = 'DBS' AND P.NOTE ≤ 2 AND S.MATNR = P.MATNR
```

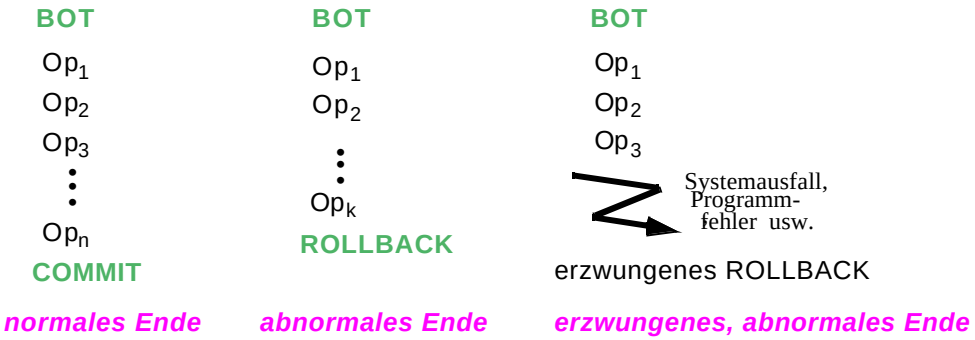
Transaktionskonzept

- Kontrollstruktur: Transaktionen mit den vier ACID-Eigenschaften

  Eine Transaktion besteht aus einer Folge von DB-Operationen, für die das DBS folgende Eigenschaften garantiert

  - **A**tomicity: Alles-oder-Nichts
  - **C**onsistency: Gewährleistung der Integritätsbedingungen
  - **I**solated Execution: "logischer Einbenutzerbetrieb"
  - **D**urabiliy: Persistenz aller Änderungen

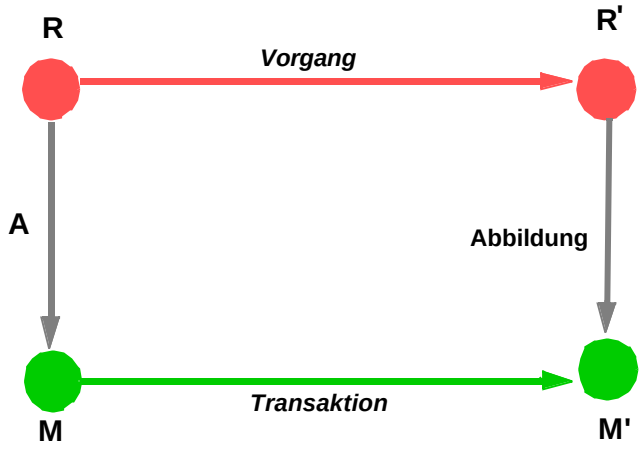
- Atomarität: mögliche Ausgänge einer Transaktion



Transaktionskonzept / Zugriffskontrolle

- Consistency: Erhaltung der logischen Datenintegrität
- Erhaltung der physischen Datenintegrität
  - Führen von Änderungsprotokollen für den Fehlerfall (Logging)
  - Bereitstellen von Wiederherstellungsalgorithmen im Fehlerfall (Recovery)
- Kontrollierter Mehrbenutzer-Betriebs (Ablaufintegrität)
  - logischer Einbenutzer-Betrieb für jeden von n parallelen Benutzern (Leser + Schreiber)
  - Synchronisation / Isolation i.a. durch Sperrverfahren
  - wichtig: Lese- und Schreibsperren mit angepaßten Sperreinheiten (Sperrgranulate)
  - ***Ziel*:** möglichst geringe gegenseitige Behinderung
- Automatisierte Zugriffskontrollen (Datenschutz)
  - separat für jedes Datenobjekt
  - unterschiedliche Rechte für verschiedene Arten des Zugriffs
  - ***Idealziel*:** "least priviledge principle"

Modell einer Miniwelt: Grobe Zusammenhänge



R: Realitätsausschnitt (Miniwelt)

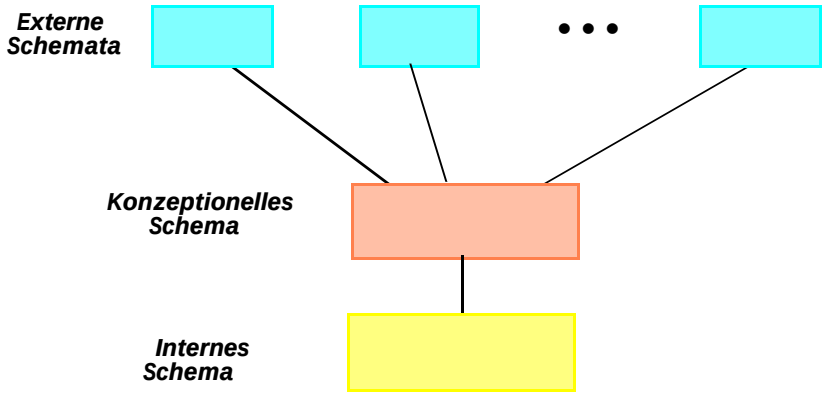
M: Modell der Miniwelt (beschrieben durch DB-Schema)

A: Abbildung aller wichtigen Objekte und Beziehungen (Entities und Relationships)

=> Abstraktionsvorgang

- Transaktion:
  - garantiert ununterbrechbaren Übergang von M nach M'
  - implementiert durch Folge von DB-Operationen
- Integritätsbedingungen:
  - Zusicherungen über A und M
  - Ziel: möglichst gute Übereinstimmung von R und M

3-Ebenen-Architektur nach ANSI-SPARC



Konzeptionelles Schema:

- logische Gesamtsicht auf die Struktur der Datenbank
- Beschreibungssprache: DDL (Data Definition Language)
- abtrahiert von internem Schema (-> physische Datenunabhängigkeit)

Externes Schema:

- definiert spezielle Benutzersicht auf DB-Struktur (für Anwendungsprogramm bzw. Endbenutzer)
- abtrahiert von konzeptionellem Schema (ermöglicht partiell logische Datenunabhängigkeit)

Internes Schema:

- legt physische Struktur der DB fest (physische Satzformate, Zugriffspfade etc.)
- Beschreibungssprache: SSL (Storage Structure Language)

Stark vereinfachtes Beispiel für die Daten-beschreibung in der 3-Schema-Architektur

Extern (PL/1)

```
DCL  1 PERSP,
       2 PERS# CHAR(6),
       3 SAL FIXED BIN(31)
```

Extern (COBOL)

```
01 PERSC.
   02 PNR PIC X(6).
   02 DEPTNO PIC X(4).
```

Konzeptionelles Schema:

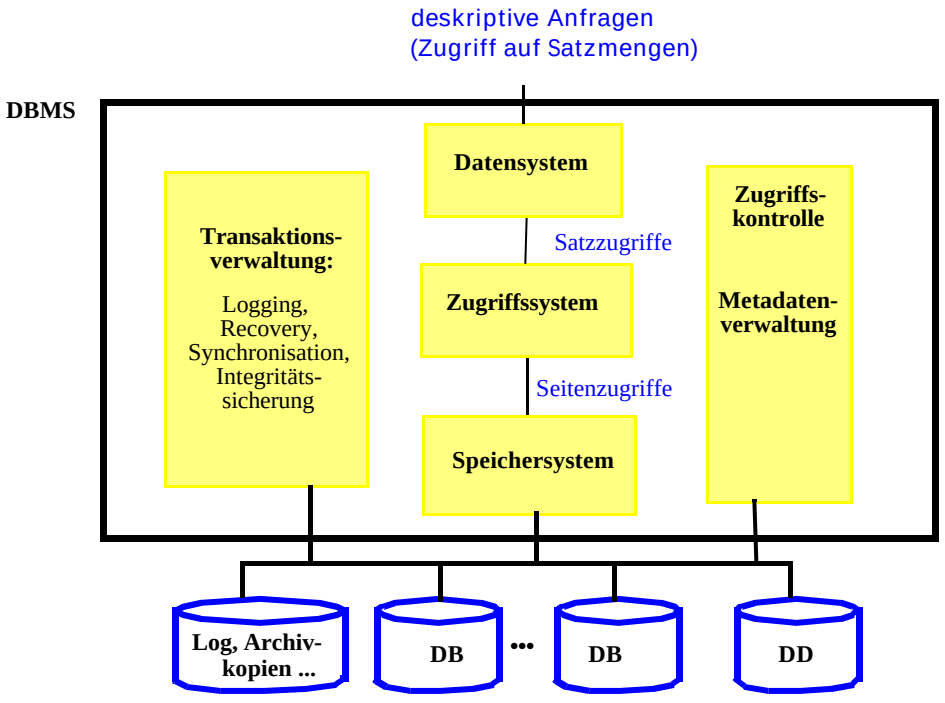
```
PERSONAL
        PERSONAL_NUMMER          CHARACTER   (6)
        ABTEILUNGS_NUMMER        CHARACTER   (4)
        SALARY                   NUMERIC     (5)
```

Internes Schema:

```
STORED_PERS     LENGTH=18
        PREFIX  TYPE=BYTE(6), OFFSET=0
        PNUM    TYPE=BYTE(6), OFFSET=6, INDEX=PNR
        ABT#    TYPE=BYTE(4), OFFSET=12
        PAY     TYPE=FULLWORD, OFFSET=16
```

- Sichtenbildung durch das Externe Schema
  - Anpassung der Datentypen an die der Wirtssprache (DBS wird "multi-lingual")
  - Zugriffsschutz: Isolation von Attributen, Relationen, ...
  - Reduktion der Komplexität: Anwendung sieht nur die erforderlichen Daten

Grobaufbau eines DBS



Schichtenmodell

deskriptive Anfragen
Zugriff auf Satzmengen

Übersetzung und Optimierung von Anfragen

Datensystem

Satzzugriffe

Verwaltung von physischen Sätzen und Zugriffspfaden

Zugriffssystem

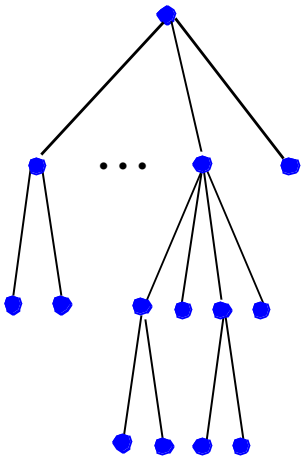
Seitenzugriffe

Systempuffer- und Externspeicher-Verwaltung

Speichersystem

DB+DD

Dynamischer Kontrollfluß einer DB-Operation



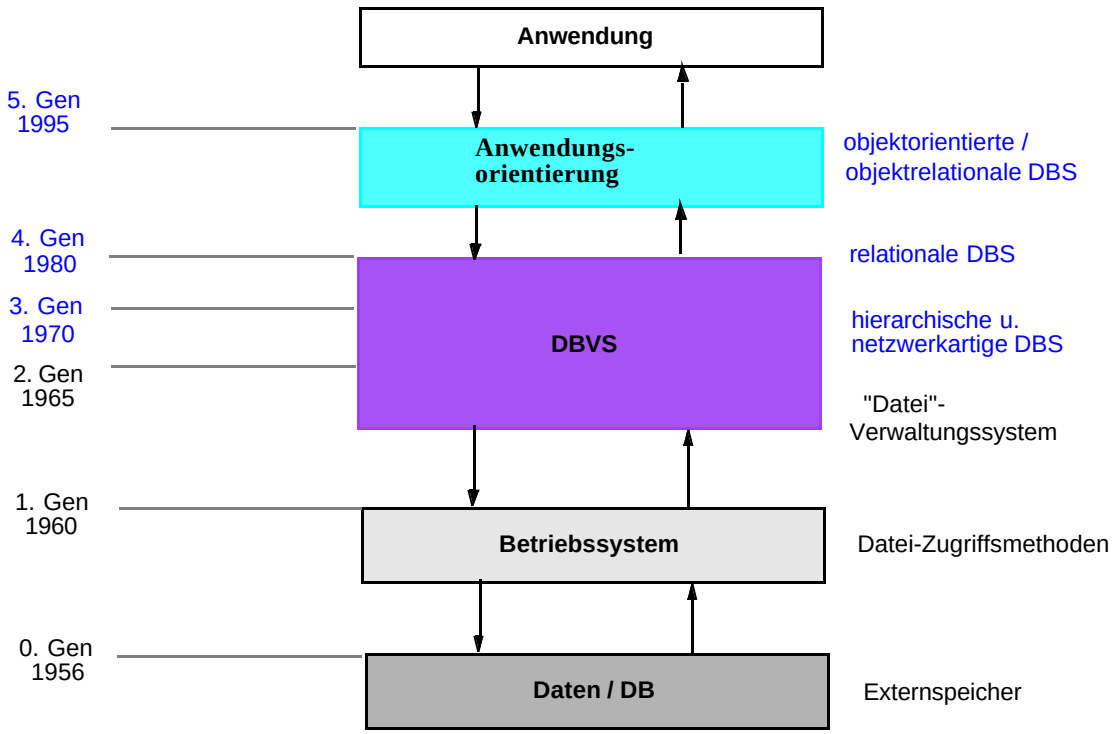
DML-Operationen

Einfüge Satz;
Modifiziere Index (B*-Baum)

Stelle Seite bereit;
Gib Seite frei

Lese / Schreibe Seite

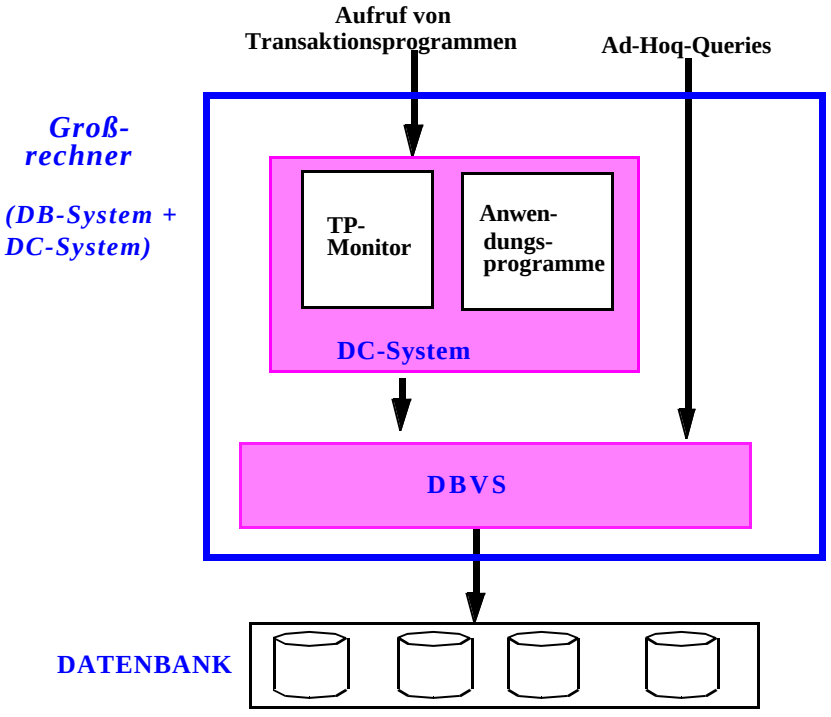
Historische Entwicklung



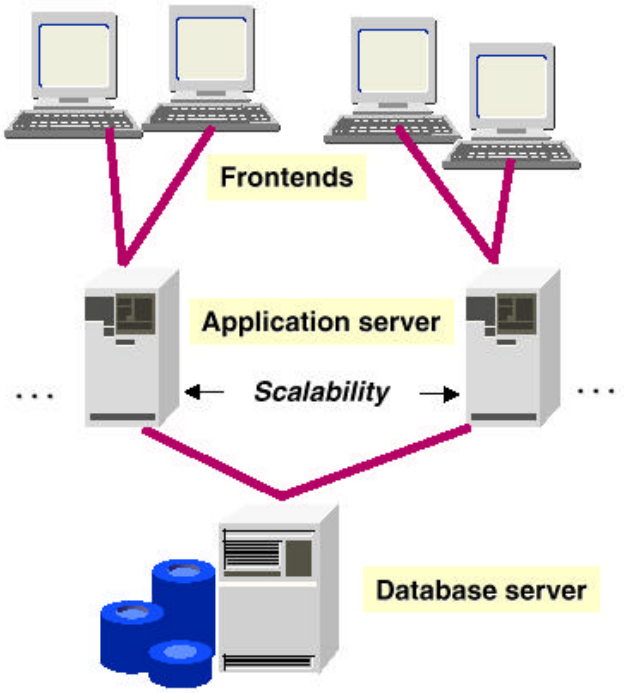
Einsatzformen von DBS

- dominierende DBS-Nutzung im Rahmen von **Transaktionssystemen** (**OLTP**, Online Transaction Processing) sowie E-Business: Ausführung vorgeplanter Anwendungen
- Eine Online-Transaktion ist die Ausführung eines Programmes, das mit Hilfe von Zugriffen auf eine gemeinsam genutzte Datenbank (DB) eine i.a. nicht-triviale Anwendungsfunktion erfüllt, z.B.
  - Bearbeiten einer Bestellung
  - Platzreservierung für einen Flug
  - Kontostandsabfrage; Abbuchen eines Geldbetrages; Überweisung
  - Anmelden eines Autos
  - Abwickeln eines Telefonanrufes, ...
- weitere DBS-Einsatzfelder / -Ausprägungen
  - Decision Support: OLAP (Online Analytical Processing), Data Warehousing, Data Mining
  - Multimedia-, Geo-, Volltext-DBS
  - Deduktive DBS, Wissensbankverwaltungssysteme
  - Architektur: zentralisierte DBS vs. Mehrrechner-DBS (Verteilte/Parallele/Heterogene DBS), ...

Grobaufbau eines zentralisierten Transaktionssystemes (ca. 1985)



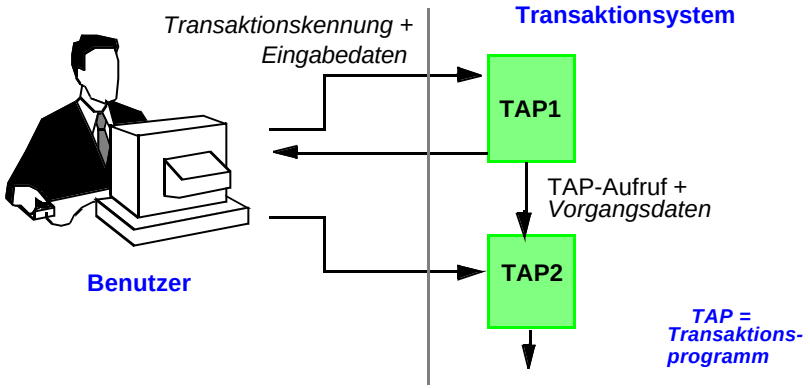
3-stufige Client/Server-Architektur zur Transaktionsverarbeitung



Anwendungen +
System-Middleware wie TP-Monitor
bzw. Applikations-Server

Transaktionsverarbeitung

- Prinzipieller Ablauf

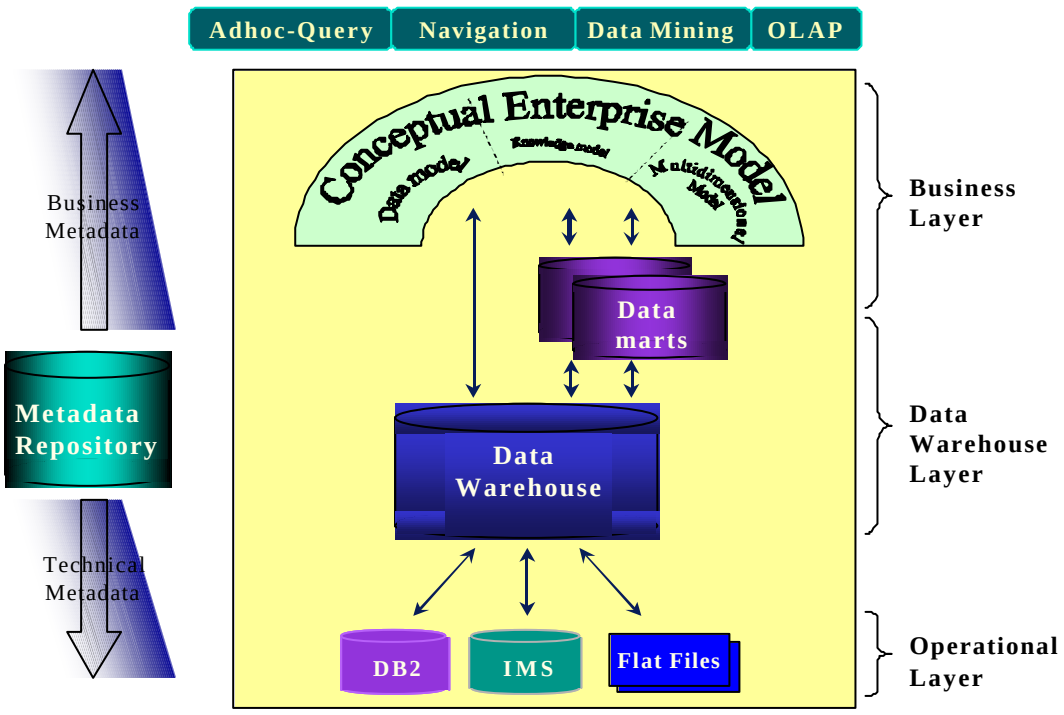


- Merkmale:
  - Benutzung im Dialog: “parametrischer” Benutzer
  - wenige kurze Transaktionstypen: hohe Wiederholrate
  - Vorgang besteht i.a. aus mehreren Dialogschritten
  - sehr viele Benutzer gleichzeitig
  - gemeinsame Datenbestände mit größtmöglicher Aktualität
  - kurze Antwortzeiten als Optimierungsziel vor Auslastung
  - stochastisches Verkehrsaufkommen
  - hohe Verfügbarkeit des Systems

Entscheidungsunterstützende Systeme (Decision Support Systems, DSS)

- **OLAP** (Online Analytical Processing) vs. OLTP (Online Transaction Processing)
  - Analyse betrieblicher Datenbestände
- häufiger Einsatz von **Data Warehouses**
  - Bereitstellung der Datenbestände eines Unternehmens für den Endbenutzer, so daß nicht nur ein vorgegebener Blickwinkel auf die Daten unterstützt wird
  - Datenbestand und aufbauende Werkzeuge für nahezu alle anfallenden Fragestellungen
  - Bsp.: Umsatzentwiclung nach Zeit, Produktklasse, Region, etc.
- **Data Mining**: Aufspüren von inhärenten Daten-/Informationsmustern aus großen dynamischen Datenbeständen
  - oft synonym: KDD (Knowledge and Data Discovery)

Data Warehouse-Umfeld



Zusammenfassung

- Datenverwaltung durch Dateisysteme unzureichend
- DBS-Charakteristika
  - Effiziente Verwaltung persistenter und strukturierter Daten
  - Datenstrukturierung und Operationen gemäß Datenmodell/DB-Sprache
  - Transaktionskonzept (ACID): Atomarität, Konsistenzerhaltung, kontrollierter Mehrbenutzerbetrieb (Isolation), Persistenz erfolgreicher Änderungen
  - zentrale (integrierte) Datenbank mit hohem Grad an Datenunabhängigkeit
- DB-Schnittstellen/-Sprachen
  - satzorientierte DB-Schnittstelle -> hierarchische / netzwerkartige DBS
  - hierarchische Modellierung führt zu hoher Redundanz
  - mengenorientierte DB-Schnittstelle -> relationale DBS
- 3-Ebenen-Architektur (ANSI/SPARC)
  - Externes Schema
  - Konzeptionelles Schema
  - Internes Schema

Zusammenfassung (2)

- Schichtenmodell eines DBVS
  - interne Schichten für Seiten, Sätze und Satzmengen mit zugeschnittenen Operationen
  - Querschnittsaufgaben: Transaktionsverwaltung und Metadaten
- Haupt-Einsatzformen von DBS in Unternehmen:
  - Transaktionssysteme (OLTP) / E-Business
  - Entscheidungsunterstützung (OLAP, Data Mining)
- zahlreiche weitere DBS-Einsatzfelder:
  - Geoinformationssysteme, Genomdatenbanken,
  - Multimedia-Archive, Wissensverwaltungssysteme,
  - Speicherung von XML-Dokumenten ...

2. Informationsmodellierung mit Entity-Relationship-Modell und UML

- Einführung Modellierung / Abstraktionskonzepte
- Entity-Relationship-Modell
  - Entity-Mengen
  - Attribute und Wertebereiche
  - Primärschlüssel
  - Relationship-Mengen
  - Klassifikation der Beziehungstypen (1:1, n:1, 1:n, n:m)
  - Kardinalitätsrestriktionen
  - Schwache Entity-Mengen
- Generalisierung / Spezialisierung
- Aggregation
- Modellierung mit UML (Klassendiagramme)

Lernziele Kapitel 2

- Kenntnis der Vorgehensweise beim DB-Entwurf
- Grundkonzepte des ER-Modells sowie von UML-Klassendiagrammen
- Kenntnis der Abstraktionskonzepte, insbesondere von Generalisierung und Aggregation
- Fähigkeit zur praktischen Anwendung der Konzepte
  - Erstellung von ER-Modellen und -Diagrammen bzw. UML-Modellen für gegebene Anwendungsszenarien
  - Festlegung der Primärschlüssel, Beziehungstypen, Kardinalitäten, Existenzabhängigkeiten etc.
  - Interpretation gegebener ER- bzw. UML-Modelle

Informations- und Datenmodellierung (DB-Entwurf)

Ziele:

- modellhafte Abbildung eines anwendungsorientierten Ausschnitts der realen Welt (Miniwelt)
- Entwurf der logischen und physischen DB-Struktur (DB-Entwurf)
- Nachbildung von Vorgängen durch Transaktionen

Nebenbedingungen:

- Vollständigkeit
- Korrektheit
- Minimalität
- Lesbarkeit
- Modifizierbarkeit

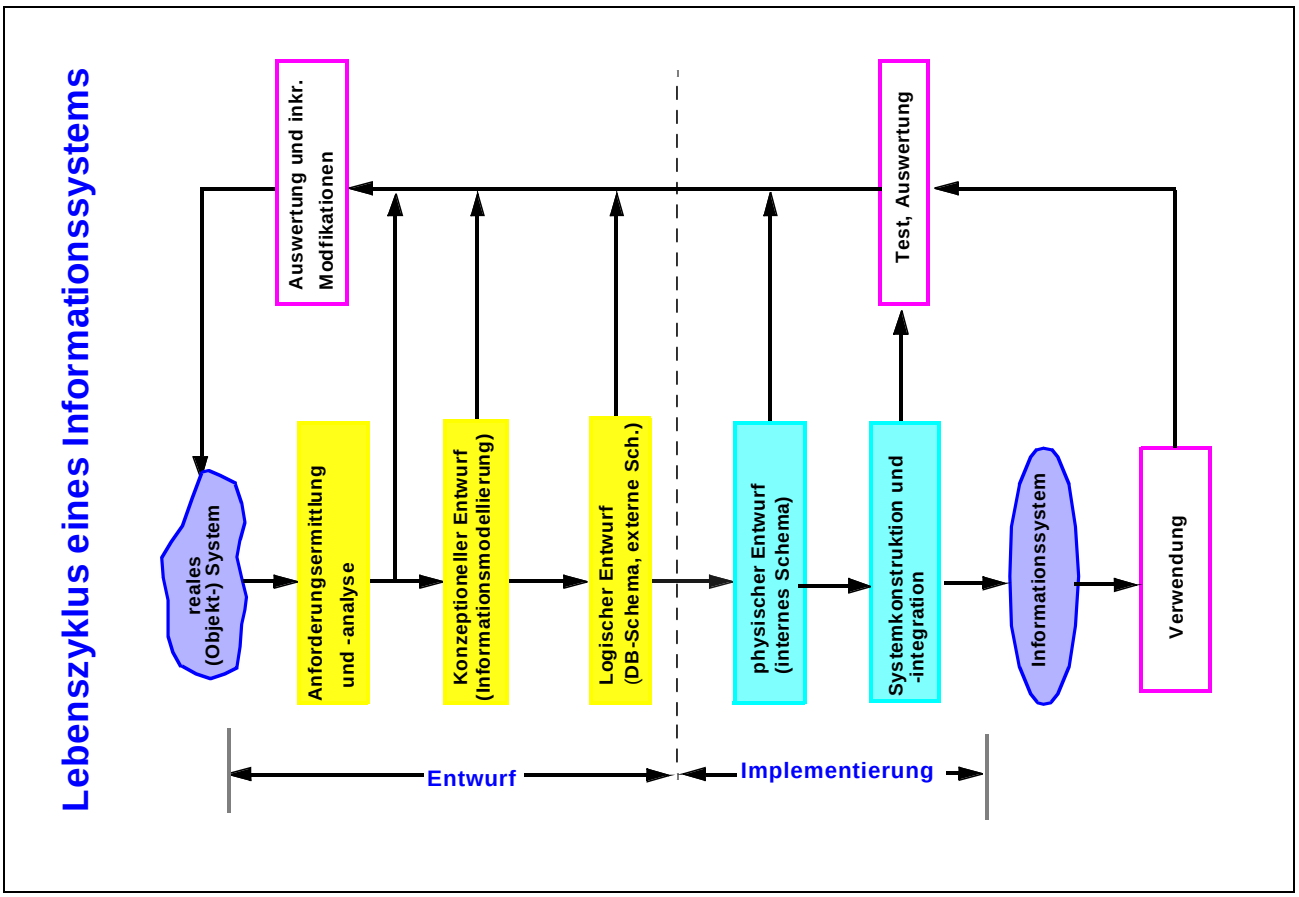
Schrittweise Ableitung: (Verschiedene Sichten)

1) Information in unserer Vorstellung

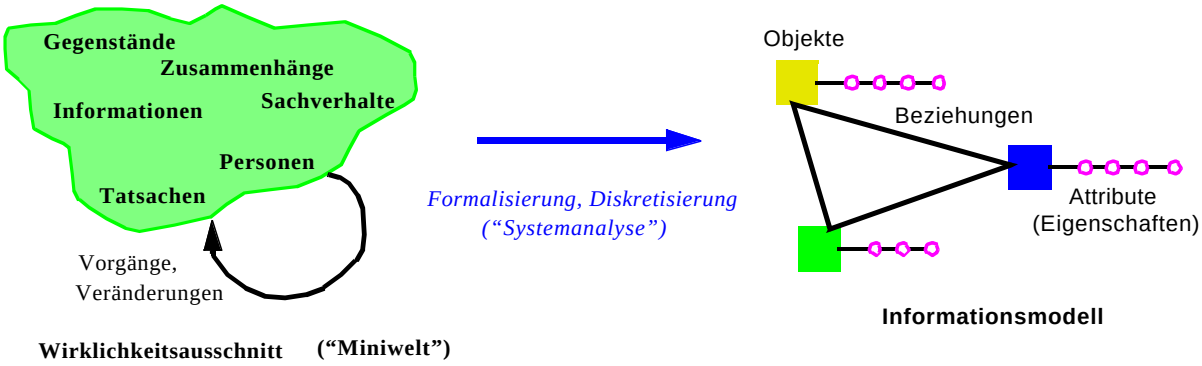
2) Informationsstruktur: Organisationsform der Information

3) Logische (zugriffspfadunabhängige) Datenstruktur (Was-Aspekt)

4) Physische Datenstruktur (Was- und Wie-Aspekt)



Informationsmodellierung



Wirklichkeitsausschnitt ("Miniwelt")

- Darstellungselemente + Regeln:
  - Objekte (Entities) und Beziehungen (Relationships)
  - Klassen von Objekten / Beziehungen
  - Eigenschaften (Attribute)
- Informationen über Objekte und Beziehungen nur wenn:
  - unterscheidbar und identifizierbar
  - relevant
  - selektiv beschreibbar

Abstraktionskonzepte

- Informations- und Datenmodelle basieren auf grundlegenden Abstraktionskonzepten der Klassifikation, Aggregation und Generalisierung (bzw. Spezialisierung)
- Klassifikation: faßt Objekte (Entities, Instanzen) mit gemeinsamen Eigenschaften zu einem neuen (Mengen-) Objekt (Entity-Menge, Klasse, Objekttyp) zusammen.

  Instanzen/Objekten einer Klasse unterliegen
  - gleicher Struktur (Attribute)
  - gleichen Operationen
  - gleichen Integritätsbedingungen
  - mathematisch: Mengenbildung
- Aggregation: Zusammenfassung potentiell unterschiedlicher Teilobjekte (Komponenten) zu neuem Objekt
  - mathematisch: Bildung von kartesischen Produkten
- Verallgemeinerung / Generalisierung: Teilmengenbeziehungen zwischen Elementen verschiedener Klassen
  - mathematisch: Bildung von Potenzmengen (bzw. Teilmengen)
  - wesentlich: Vererbung von Eigenschaften an Teilmengen

Entity-Relationship-Modell

- entwickelt von P. P. Chen (ACM Transactions on Database Systems 1976)
- Konzepte:
  - Entity-Mengen
  - Beziehungsmengen (Relationship-Mengen)
  - Attribute
  - Wertebereiche
  - Primärschlüssel
- unterstützt die Abstraktionskonzepte der Klassifikation und Aggregation
- graphische Darstellung durch Diagramme
- zahlreiche Erweiterungsvorschläge
- weite Verbreitung über konzeptionellen DB-Entwurf hinaus: Systemanalyse, Unternehmensmodellierung

Entity-Mengen

- *Entity* (Entität, Gegenstand): repräsentiert abtraktes oder physisches Objekt der realen Welt
- Gleichartige Entities (d.h. Entities mit gemeinsamen Eigenschaften) werden zu *Entity-Mengen* (Gegenstandstypen, Objekttypen) zusammengefaßt (Klassifikation)

  => Entities sind Elemente einer (homogenen) Menge: $e \in E$

  z.B. Personen, Projekte ...

  Bücher, Autoren ...

  Kunden, Vertreter, Wein, Behälter
- DB enthält endlich viele Entity-Mengen:

  $E_1, E_2, \ldots, E_n$; nicht notwendigerweise disjunkt

  z.B. E_1 ... Personen, E_2 ... Kunden: $E_2 \subseteq E_1$
- Symbol für Entity-Menge E:

Attribute und Wertebereiche

- Attribute und Attributwerte:
  - Eigenschaften von Entity-Mengen werden durch Attribute bestimmt
  - Eigenschaften einzelner Entities sind durch Attributwerte festgelegt
  - Nullwert: spezieller Attributwert, dessen Wert unbekannt oder nicht möglich ist (z.B. private Tel.Nr.)
- Jedem Attribut ist ein Wertebereich (Domain) zugeordnet, der festlegt, welche Attributwerte zulässig sind (Integritätsbedingung !)

  $E\ (A_1: D_1, A_2: D_2, \ldots A_n: D_n)$

  - Attribute ordnen damit jedem Entity einen Attributwert aus dem Domain zu, d.h., ein Attribut A entspricht einer mathematischen Funktion

    A : E --> dom (A)
- Attributsymbol in ER-Diagrammen:

Attributarten

- einfache vs. zusammengesetzte Attribute
  - Beispiele: NAME [Vorname: char (30), Nachname: char (30)]
    ANSCHRIFT [Strasse: char (30), Ort: char (30), PLZ: char (5)]
  - Domain für einfache Attribute: **dom (A) = W(A)**
  - Domain für zusammengesetztes Attribut A $[A_1, A_2, \ldots A_k]$:

    dom (A) = $W(A_1)$ x $W(A_2)$ x ... x $W(A_k)$
- einwertige (single-valued) vs. mehrwertige (multi-valued) Attribute
  - Beispiele: AUTOFARBE: {char (20)}
    KINDER: {[Name: char (30), Alter: int]}
  - Domain für mehrwertiges Attribut A: {W(A)}: **dom (A) =** $2^{W(A)}$
- Symbol für mehrwertige Attribute:

Primärschlüssel

- Schlüsselkandidat oder kurz Schlüssel (key)
  - einwertiges Attribut oder Attributkombination, die jedes Entity einer Entity-Menge eindeutig identifiziert
  - ggf. künstlich erzwungen (lfd. Nr.)
- Definition:

  $A = \{A_1, A_2, ..., A_m\}$ sei Menge der Attribute zu Entity-Menge E

  K $\subseteq$ **A** heißt **Schlüsselkandidat** von E $\Leftrightarrow$ **K** minimal;

  $$\forall\ e_i, e_j \in E \text{ mit } e_i \neq e_j \text{ ---> } \mathbf{K}(e_i) \neq \mathbf{K}(e_j)$$
- keine Nullwerte!
- Entity-Menge kann mehr als einen Schlüsselkandidaten haben
  - Beispiel: `Professor (Zi-Nr, Sekr-TelNr, Vorname, Name, PNR)`

  => Auswahl eines Primärschlüssels
- Primärschlüsselattribute werden durch Unterstreichung gekennzeichnet

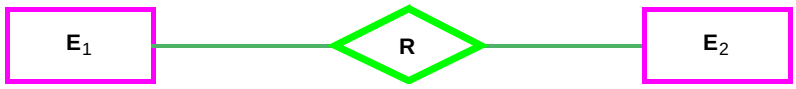
Relationships

- Relationship-Menge: Zusammenfassung von gleichartigen Beziehungen (Relationships) zwischen Entities, die jeweils gleichen Entity-Mengen angehören
- Beispiel: Beziehungen "Vorlesungsbesuch" zwischen "Student" und "Vorlesung"

- Relationship-Menge R entspricht einer math. Relation zwischen n Entity-Mengen E_i

  $R \subseteq E_1 \times E_2 \times ... \times E_n$, d.h. $R = \{r = [e_1, e_2, ..., e_n] \mid e_1 \in E_1, ..., e_n \in E_n\}$

  gewöhnlich: n=2 oder n=3
- Symbol:



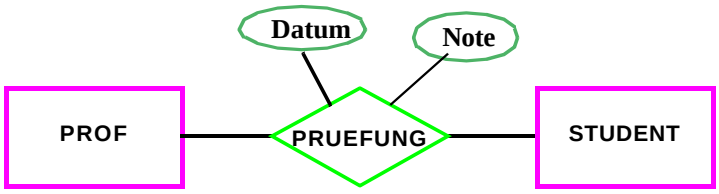
Relationships (2)

- Relationship-Mengen können <u>auch</u> Attribute besitzen

  $R \subseteq E_1 \times E_2 \times \ldots \times E_n \times dom(A_1) \times \ldots \times dom(A_m)$

  d.h. $R = \{r = [e_1, e_2, \ldots, e_n, a_1, a_2, \ldots a_m] \mid e_i \in E_i,\ a_j \in dom(A_j)\}$

- Beispiel



PROF (Pnr, Pname, Fach)

STUDENT (Matnr, Sname, Immdatum)

PRUEFUNG (PROF, STUDENT, Datum, Note)

- Entities e_i können durch ihre Primärschlüssel ersetzt werden

Relationships (3)

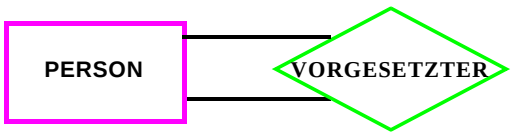
- keine Disjunktheit der beteiligten Entity-Mengen gefordert (*rekursive Beziehungen*)

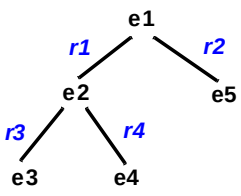
  VERHEIRATET $\subseteq$ PERSON x PERSON

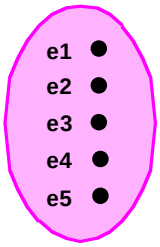
  VORGESETZTER $\subseteq$ PERSON x PERSON

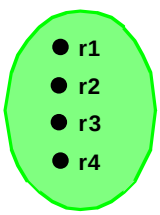
- Einführung von Rollennamen möglich (Reihenfolge !)

  VORGESETZTER (Chef: PERSON, Mitarbeiter: PERSON)



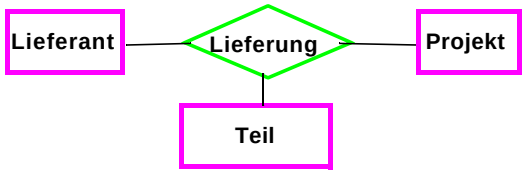




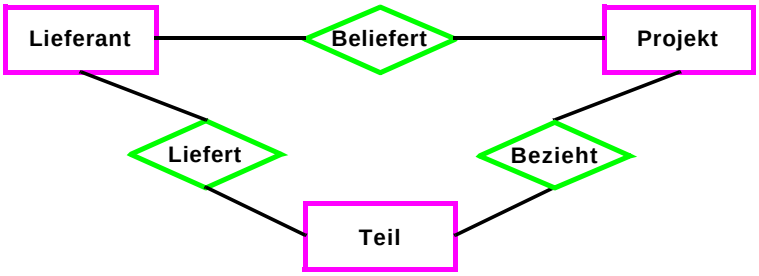


Relationships (4)

- Beispiel einer 3-stelligen Relationship-Menge



- nicht gleichwertig mit drei 2-stelligen (binären) Relationship-Mengen!



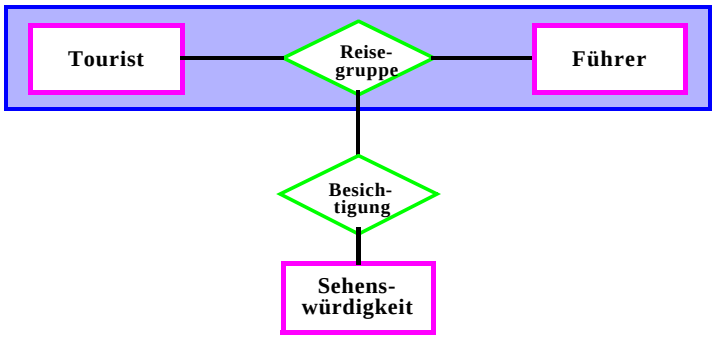
Beispiel:

L1 liefert T1,
L1 beliefert P1,
P1 bezieht T1
impliziert nicht notwendigerweise

Lieferung (L1, P1, T1)

Relationships (5)

- Beziehungen zwischen Relationship-Mengen werden im ERM nicht unterstützt (mangelnde Orthogonalität)
- Beispiel



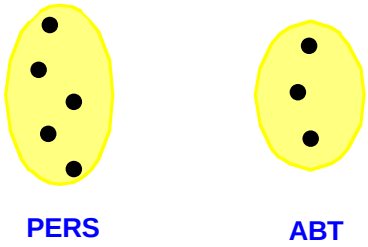
Kardinalität von Beziehungen

- Festlegung wichtiger struktureller Integritätsbedingungen
- Unterschiedliche Abbildungstypen für binäre Beziehung zwischen Entity-Mengen E_i und E_j
  - 1:1 eineindeutige Funktion (injektive Abbildung)
  - n:1 mathematische Funktion (funktionale Abbildung)
  - 1:n invers funktionale Abbildung
  - n:m mathematische Relation (komplexe Abbildung)
- Abbildungstypen implizieren nicht, daß für jedes $e \in E_i$ auch tatsächlich ein $e' \in E_j$ existiert!
  - n:1- sowie 1:1-Beziehungen repräsentieren somit i.a. nur partielle Funktionen
- Präzisierung der Kardinalitätsrestriktionen durch *Min-Max-Notation*

1:1-Beziehungen

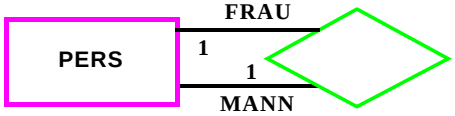
- 1:1-Beziehung zwischen unterschiedlichen Entity-Mengen

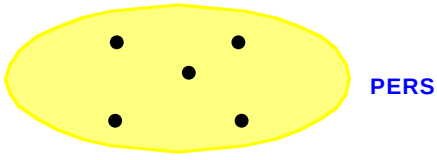
1:1 LEITET/WIRD-GELEITET: PERS <---> ABT



- rekursive 1:1-Beziehung

1:1 VERHEIRATET: PERS <---> PERS

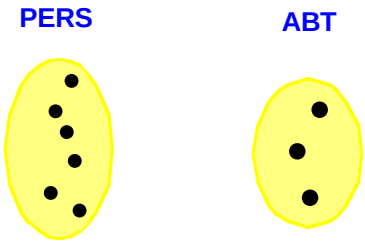




n:1-Beziehungen

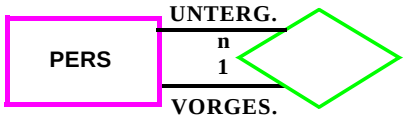
- n:1-Beziehung zwischen unterschiedlichen Entity-Mengen

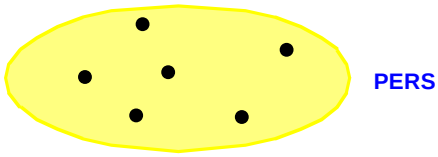
n:1 ARBEITET-FÜR/MITARBEITER: PERS ---> ABT



- rekursive n:1-Beziehung

n : 1 UNTERGEB./VORGESETZTER_VON: PERS ---> PERS

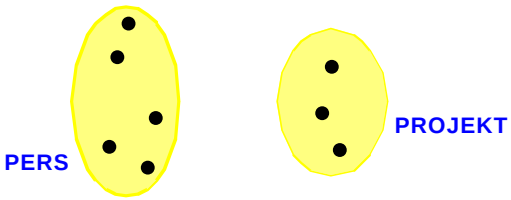




n:m-Beziehungen

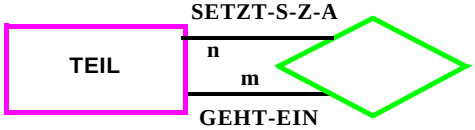
- n:m-Beziehung zwischen unterschiedlichen Entity-Mengen

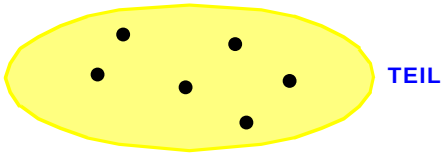
n:m ... ARBEITET_FÜR/MITARBEIT: PERS --- PROJEKT



- rekursive n:m-Beziehung

n : m SETZT_SICH_ZUSAMMEN_AUS/GEHT_EIN_IN : TEIL --- TEIL

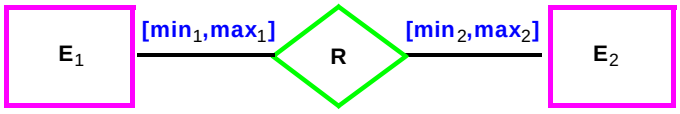




Kardinalitätsrestriktionen: Min-Max-Notation

- bisher: grobe strukturelle Festlegung der Beziehungen
  - z.B.: 1:1 bedeutet “höchstens eins zu höchstens eins”
  - nur zweistellige Beziehungen klassifizierbar
- Verfeinerung der Semantik eines Beziehungstyps durch Min-Max-Kardinalitätsrestriktionen
- Definition: sei $R \subseteq E_1 \times E_2 \times ... \times E_n$

  Kardinalitätsrestriktion $kard(R,E_i) = [min,max]$ bedeutet, daß jedes Element aus E_i in wenigstens min und höchstens max Ausprägungen von R enthalten sein muß
  (mit 0 <= min <= max, max >= 1)
- Diagrammdarstellung:



e_1 nimmt an $[min_1,max_1]$ Beziehungen von Typ R teil

e_2 nimmt an $[min_2,max_2]$ Beziehungen von Typ R teil

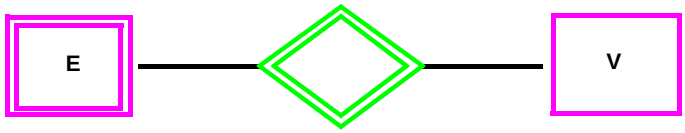
- Min-Max-Notation erlaubt Unterscheidung, ob Teilnahme eines Entities an einer Beziehung *optional (Mindestkardinalität 0)* oder *obligatorisch (Mindestkardinalität >= 1)* ist

Min-Max-Kardinalitätsrestriktionen: Beispiele

R	E_1	E_2	klass. Beziehungstyp	kard (R, E1)	kard (R, E2)
Abt.Leitung	ABT	PERS			
Proj.Mitarbeit	PERS	PROJEKT			
Parteimitglied	PARTEI	PERS			
Verheiratet	FRAU	MANN			
V.teilnahme	VORL	STUDENT			
Belegung	PERS	ZIMMER			
Eltern	PAARE	KIND			

Schwache Entity-Mengen (weak entities)

- Entity-Menge mit Existenzabhängigkeit zu anderer Entity-Menge
  - kein eigener Schlüsselkandidat, sondern Identifikation über Beziehung zur übergeordneten Entity-Menge
- Bsp.: Entity-Menge *Raum (Nummer, Größe)* abhängig von *Gebäude*
- Konsequenzen
  - i.a. n:1 bzw. 1:1-Beziehung zwischen schwacher Entity-Menge und Vater-Entity-Menge
  - jedes schwache Entity muß in Relationship-Menge mit Vater-Entity-Menge vertreten sein (obligatorische Beziehungsteilnahme, minimale Kardinalität 1)
  - Primärschlüssel ist zumindest teilweise von Vater-Entity-Menge abgeleitet
- ER-Symbole:

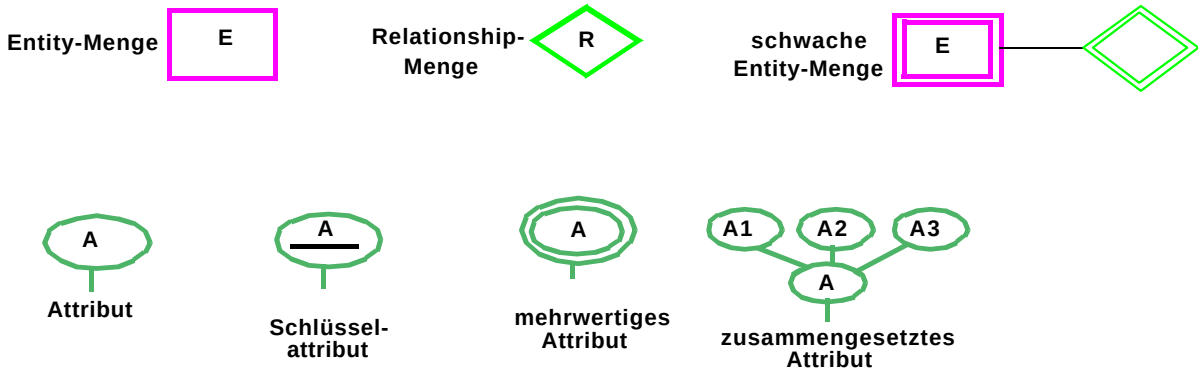


- Beispiel:

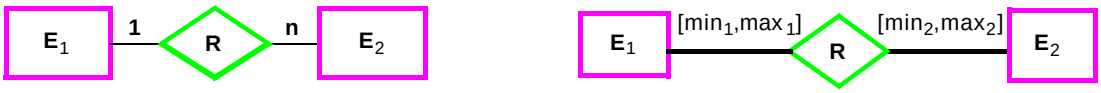
Definition von Entity- und Relationship-Mengen

- Entity-Typ = Deklaration einer Entity-Menge umfaßt
  - eindeutigen Namen E
  - Festlegung aller Attribute A mit Wertebereichen / Domains
  - Angabe des Primärschlüssels
- Relationship-Typ = Deklaration einer Relationship-Menge umfaßt
  - eindeutigen Namen R
  - Grad der Beziehung n
  - Festlegung der beteiligten Entity-Typen E_1 ... E_n
  - ggf. Festlegung von Rollennamen
  - Festlegung des Abbildungs/Beziehungstyps
  - Festlegung der Kardinalitätsrestriktionen
  - Festlegung von Relationship-Attributen A mit Wertebereichen

Überblick über ER-Diagrammsymbole



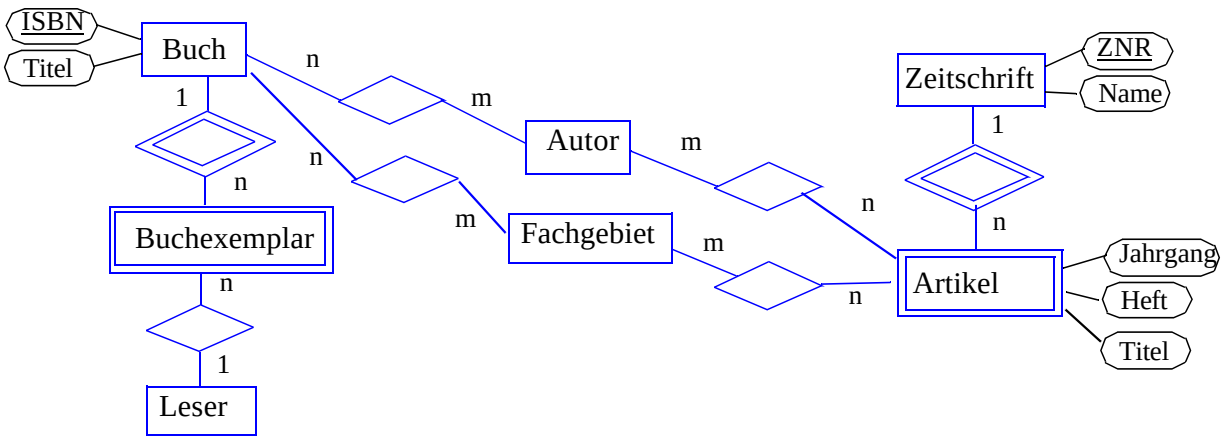
Kardinalitätsangaben



ERM: Anwendungsbeispiel

Eine Bibliothek besteht aus Büchern und Zeitschriften.

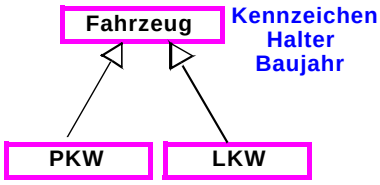
- Jedes Buch kann ggf. mehrere Autoren haben und ist eindeutig durch seine ISBN gekennzeichnet. Die Bibliothek besitzt teilweise mehrere Exemplare eines Buches.
- Zeitschriften dagegen sind jeweils nur einmal vorhanden. Sie erscheinen in einzelnen Heften und werden jahrgangsweise gebunden.
- Die in Zeitschriften publizierten Artikel sind ebenso wie Bücher einem oder mehreren Fachgebieten (z.B. Betriebssysteme, Datenbanksysteme, Programmiersprachen) zugeordnet.
- Ausgeliehen werden können nur Bücher (keine Zeitschriften).



Generalisierung / Spezialisierung

- Is-A-Beziehung zwischen Entity-Mengen führt zu Hierarchie von Super- und Subklassen

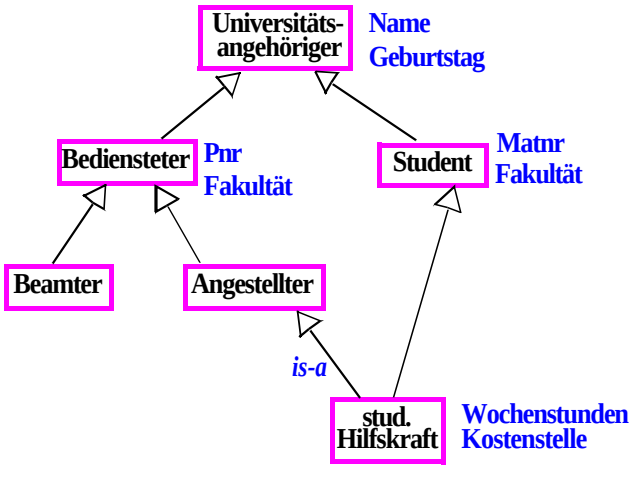
  - E_1 is-a E_2 bedeutet, daß jedes Entity e aus E_1 auch ein Entity aus E_2 ist, jedoch mit zusätzlichen strukturellen Eigenschaften
  - *Substitutionsprinzip*: alle Instanzen einer Subklasse sind auch Instanzen der Superklasse
- Generalisierung: Bottom-Up-Vorgehensweise
  - Bildung allgemeinerer Superklassen aus zugrundeliegenden Subklassen
  - Übernahme gemeinsamer Eigenschaften und Unterdrückung spezifischer Unterschiede
  - rekursive Anwendbarkeit => Generalisierungshierarchie



- Spezialisierung: Top-Down-Vorgehensweise
  - zuerst werden die allgemeineren Objekte (Superklassen), dann die spezielleren (Subklassen) beschrieben
- Vererbung von Eigenschaften (Attribute, Integritätsbedingungen, Methoden ...) der Superklasse an alle Subklassen

Generalisierung / Spezialisierung (2)

- Vorteile des Vererbungsprinzips
  - Wiederverwendbarkeit
  - Erweiterbarkeit
  - keine Wiederholung von Beschreibungsinformation, abgekürzte Beschreibung
  - Fehlervermeidung
- keine reine Hierarchien, sondern Netzwerke (n:m)
  - eine Klasse kann Subklasse mehrerer Superklassen sein
  - ein Objekt kann gleichzeitig Instanz verschiedener Klassen sein
  - Zyklen nicht erlaubt/sinnvoll (A is-a B, B is-a A)



- führt u.a. zum Problem der Mehrfach-Vererbung (multiple inheritance)
  - Namenskonflikte möglich
  - benutzergesteuerte Auflösung, z.B. durch Umbenennung

Spezialisierung: Definitionen

- *Klasse*: Menge von Entities (Entity-Mengen, Superklassen, Subklassen)
- *Subklasse*: Klasse S, deren Entities eine Teilmenge einer Superklasse G sind (is-a-Beziehung), d.h.

  $S \subseteq G$

  d.h. jedes Element (Ausprägung) von S ist auch Element von G.
- *Spezialisierung*: $Z = \{S_1, S_2, \ldots S_n\}$

  Menge von Subklassen S_i mit derselben Superklasse G
- Zusätzliche Integritätsbedingungen: Vollständigkeit (Überdeckung) und Disjunktheit von Spezialisierungen

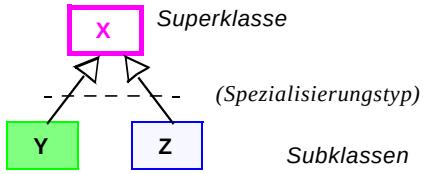
  Z heißt **vollständig (complete)**, falls gilt: $G = \cup\ S_i$ (i = 1..n)

  andernfalls **partiell (incomplete)**.

  Z ist **disjunkt (disjoint)**, falls $S_i \cap S_j = \{\}$ für $i \neq j$

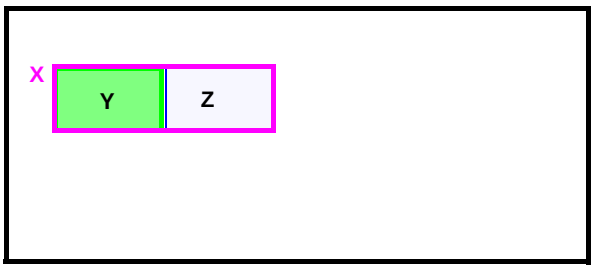
  andernfalls **überlappend (overlapping)**.

Arten von Spezialisierungen

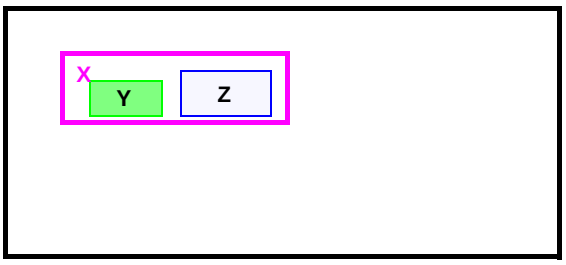


- disjunkte Spezialisierungen (Partitionierung)

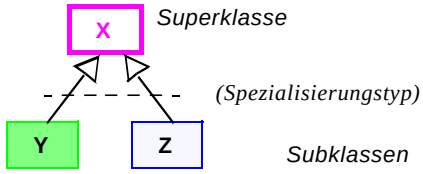
vollständig, disjunkt (*complete, disjoint*)



partiell, disjunkt (*incomplete, disjoint*)

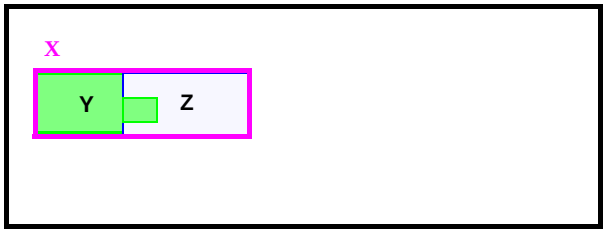


Arten von Spezialisierungen (2)

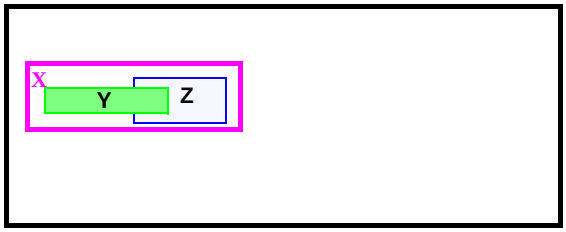


- überlappende Spezialisierungen

vollständig, überlappend (*complete, overlapping*)

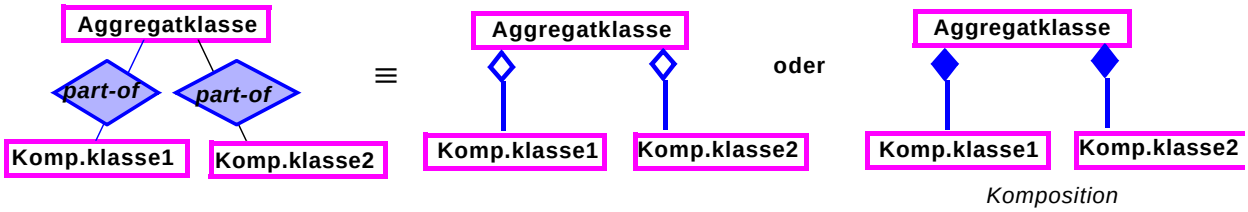


partiell, überlappend (*incomplete, overlapping*)



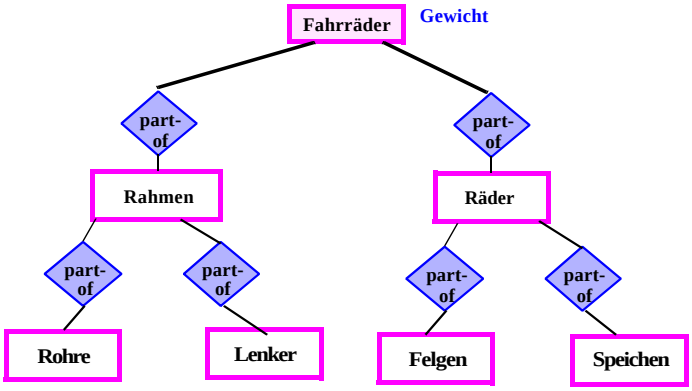
Aggregation

- Objekte werden als Zusammensetzung von anderen Objekten angesehen
  - eine Kombination von einfachen (atomaren, d.h. nicht weiter zerlegbaren) Objekten (Element, **Teil**) wird betrachtet als zusammengesetztes Objekt (**Aggregat**objekt)
  - Einfache Formen der Aggregation:
    - zusammengesetzte Attribute
    - Entity-Typ als Aggregation verschiedener Attribute
  - Erweiterung auf Beziehung zwischen Entity-Typen / Klassen
- Zwischen Komponentenobjekten besteht Part-of-Beziehung (Teil-von-Beziehung)
  - Elemente einer Subkomponente sind auch Elemente aller Superkomponenten dieser Subkomponente
  - *Referenzsemantik* ermöglicht, daß ein Objekt gleichzeitig Elemente verschiedener Komponenten bzw. Subkomponente von mehreren Superkomponenten sein -> Netzwerke, (n:m) !
  - *Wertesemantik* (**Komposition**): Teil-Objekt gehört genau zu einem Aggregat-Objekt; Existenzabhängigkeit!



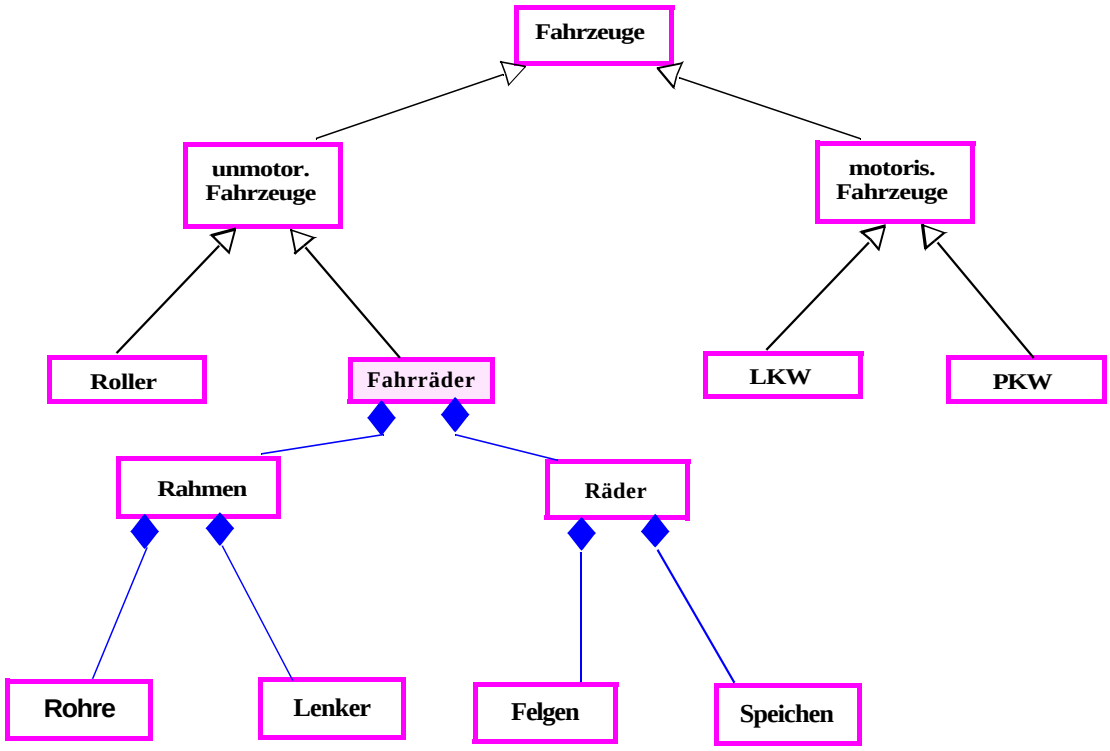
Komposition

Aggregation (2)



- Unterstützung komplex-strukturierter Objekte
- heterogene Komponenten möglich
- keine Vererbung !

Kombination von Generalisierung und Aggregation



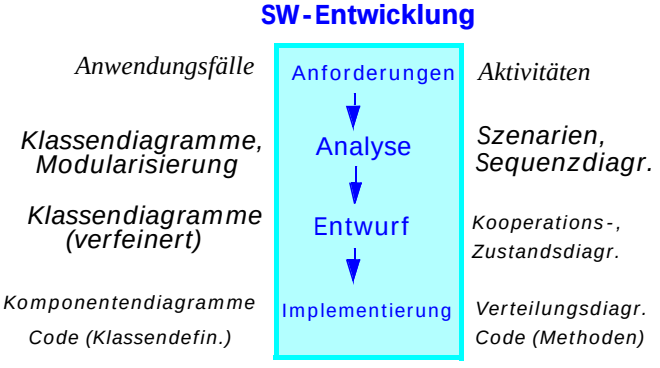
Unified Modeling Language (UML)

- standardisierte graphische Notation/Sprache zur Beschreibung objektorientierter Software-Entwicklung
- Kombination unterschiedlicher Modelle bzw. Notationen, u.a.
  - Booch
  - Rumbaugh (OMT)
  - Jacobson (Use Cases)

- Standardisierung durch Herstellervereinigung OMG (Object Management Group)
  - Nov. 1997: UML 1.1
  - Nov. 1999: UML 1.3
  - Mai 2001: UML 1.4
  - UML 2.0 in Vorbereitung
- Infos:
  - Web: *http://www.uml.org*
  - Buch-Tipp:
    M. Hitz, K. Kappel: *UML@Work*, dpunkt-Verlag, 2. Auflage 2003

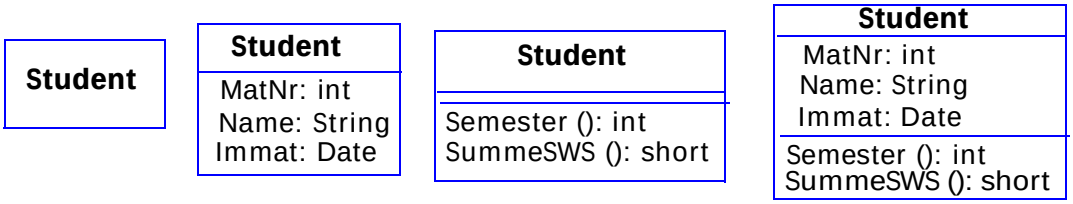
UML-Bestandteile

- UML umfasst
  - Modellelemente: Klassen, Interfaces, Komponenten, Anwendungsfälle (use cases) ...
  - Beziehungen (relationships): Assoziationen, Generalisierung, Abhängigkeiten (dependencies) ...
  - Diagramme: Klassendiagramme, Use Case-Diagramme, Interaktionsdiagramme, Paketdiagramme, Zustandsdiagramme (state chart diagrams), Aktivitätsdiagramme (activity diagrams), Verteilungsdiagramme (deployment diagrams) ...
- Generelle Sichtweisen
  - Klassen- vs. Instanzsicht (Objekt ist Instanz einer Klasse bzw. Klasse klassifiziert Instanzen)
  - Spezifikations- vs. Implementierungssicht (Interface spezifiziert Klasse, Klasse realisiert Interface)
  - Analyse- vs. Entwurf- vs. Laufzeit-Sichten
- (Statische) Strukturdiagramme
  - Festlegung von Elementen (Klassen, Interfaces, ...), deren interne Struktur sowie von Beziehungen
  - Klassendiagramme vs. Objektdiagramme

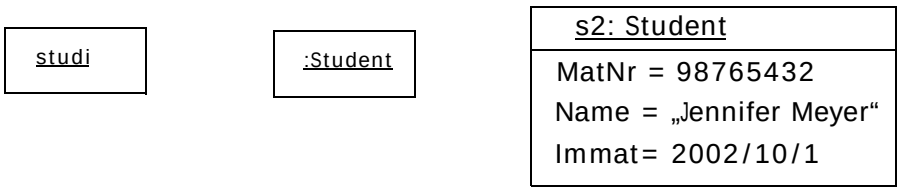


Darstellung von Klassen und Objekten

- Klassensymbol: Angabe von Klassenname, Attribute (optional), Methoden (optional)
  - i.a. werden nur relevante Details gezeigt

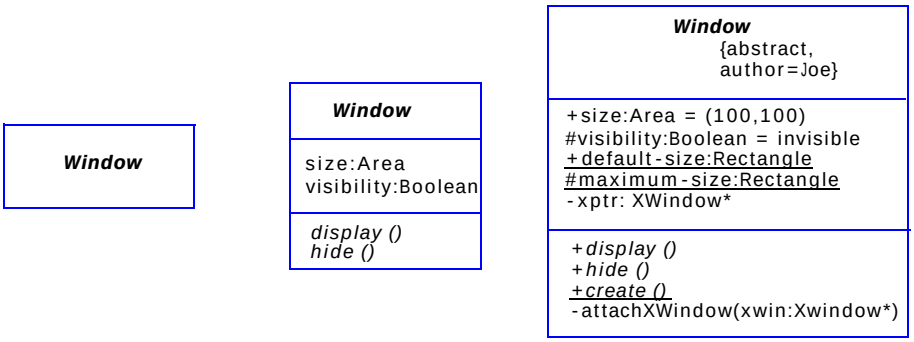


- analoge Darstellung von Klasseninstanzen (Objekten)
  - keine Methodenangabe



Darstellung von Klassen (2)

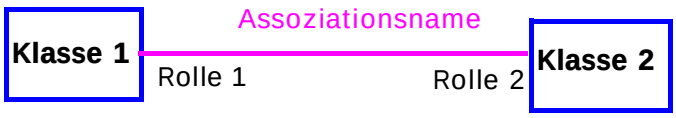
- Detaildarstellung auf Implementierungsebene
  - Attributspezifikation: *Sichtbarkeit Name: Typ = Default-Wert { Eigenschaften }*
  - Operationen: *Sichtbarkeit Name (Parameterliste) : Rückgabeausdruck { Eigenschaften }*
  - Deklaration der Sichtbarkeit : öffentlich / public (+), geschützt / protected (#), privat (-)
  - unterstrichen: Klassen-Attribute / -Operationen



- Darstellung von Bedingungen (Constraints) innerhalb geschweifter Klammern **{ }**
  - Formulierung der Bedingungen in beliebiger Sprache möglich
  - OCL (Object Contraint Language) Teil der UML

Assoziationen

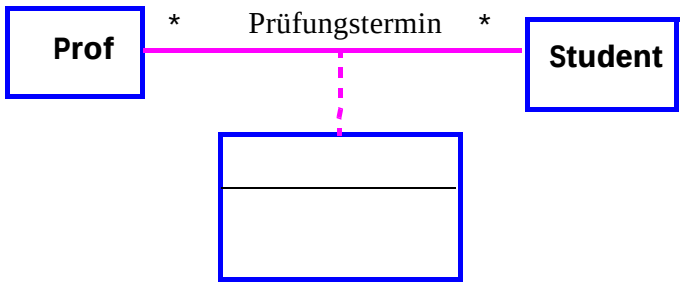
- Repräsentation von Beziehungen (relationships)
- optional: Festlegung eines Assoziationsnamens, seiner Leserichtung (►bzw.◄), von Rollennamen, Sichtbarkeit von Rollen (+, -, #) sowie Kardinalitätsrestriktionen



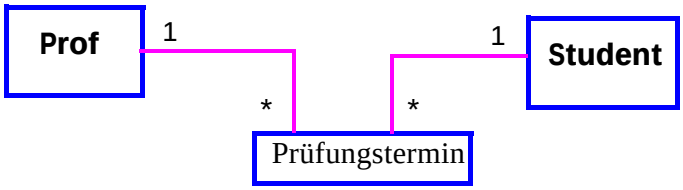
- Kardinalitätsrestriktionen (multiplicity)
  - Definition: „multiplicity specifies the *number of target instances* that may be associated with a single source instance *across* the given association“ (betrifft also „gegenüberliegende“ Klasse)
  - kein Default-Wert (fehlende Angabe bedeutet “unspezifiziert”)
  - x..y — mindestens x, maximal y Objekte nehmen an der Beziehung teil
  - * — “viele”
  - 0..* — optionale Teilnahme an der Beziehung
  - 1..*
  - 0..1
  - 1 — genau 1

Assoziationen (2)

- **Assoziations-Klassen**
  - notwendig für Beziehungen mit eigenen Attributen
  - gestrichelte Linie
  - Name der A.-Klasse entspricht dem der Assoziation

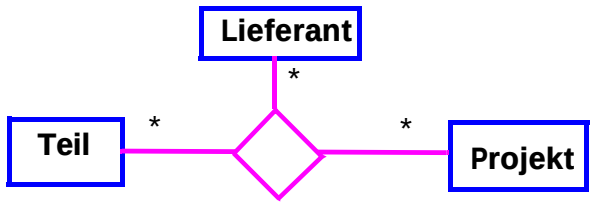


- Verwendung einer Assoziationsklasse unterscheidet sich i.a. von Nutzung einer „normalen“ Klasse



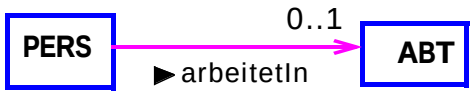
Assoziationen (3)

- 3-stellige Beziehung

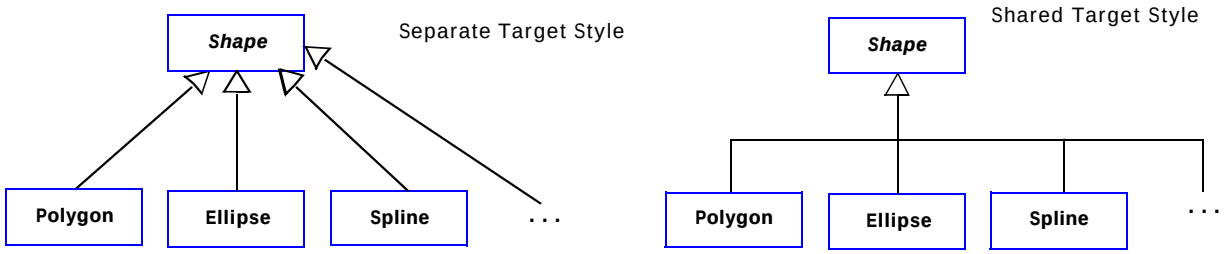


  - Multiplizitätsangabe einer Klasse regelt bei n-stelligen Beziehungen die Anzahl möglicher Instanzen (Objekte) zu einer *fixen Kombination* von je einem Objekt der übrigen n-1 Assoziationsenden

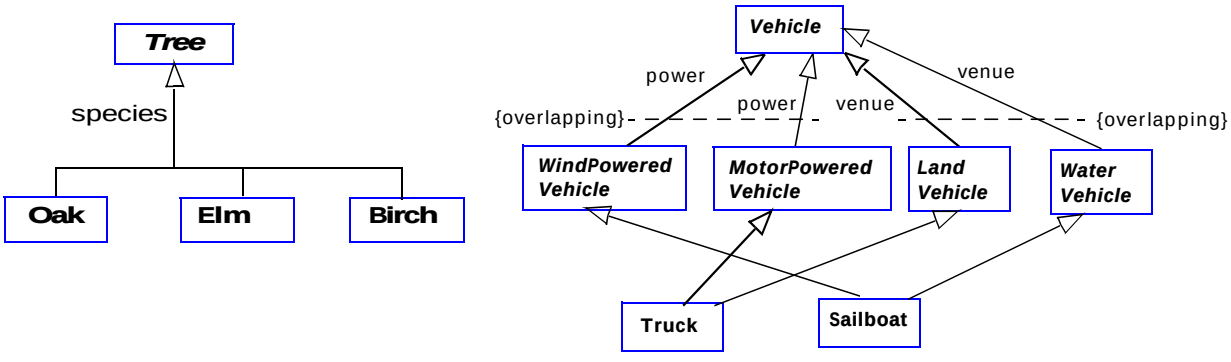
- gerichtete Assoziation
  - Einschränkung der **Navigierbarkeit:** keine direkte Navigationsmöglichkeit in umgekehrter Richtung (einfachere Implementierung)
  - auf konzeptioneller Ebene nicht notwendigerweise festzulegen



UML-Darstellung Generalisierung

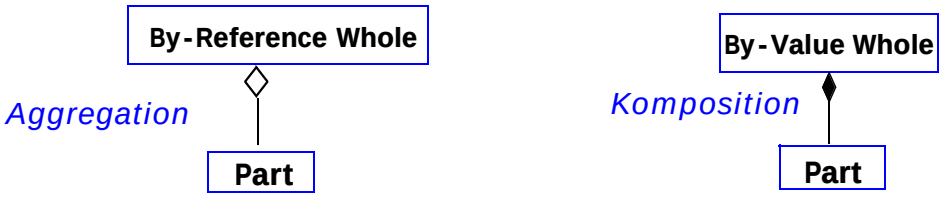


- Angabe von Diskriminatoren sowie Spezialierungsart (overlapping/disjoint, incomplete/complete)

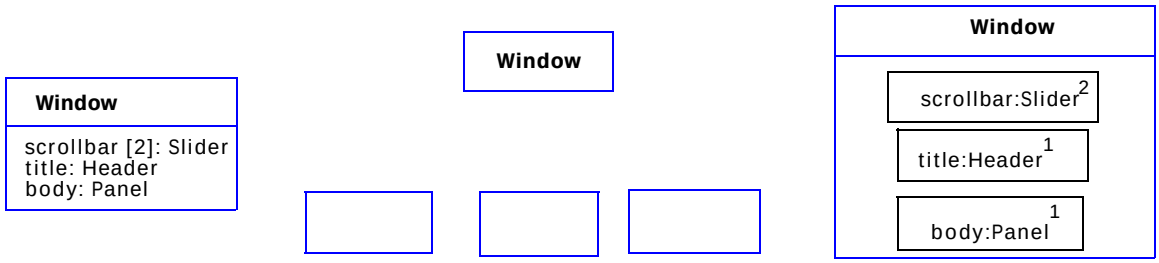


Aggregation (UML)

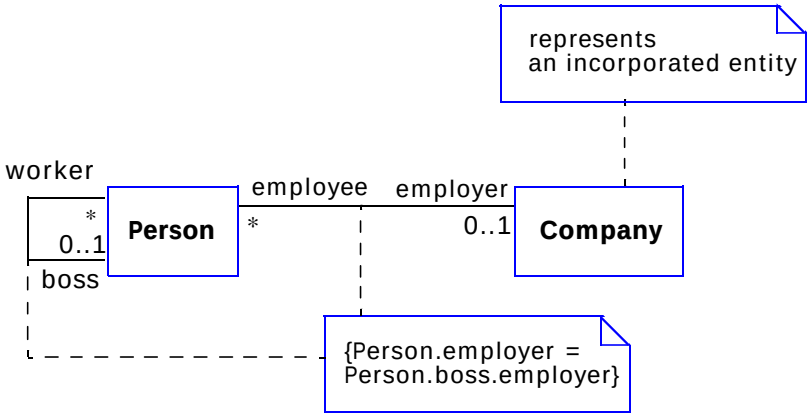
- **Aggregation**: spezielle Assoziation zwischen 2 Klassen, in der eine Klasse eine andere vollständig (whole) enthält

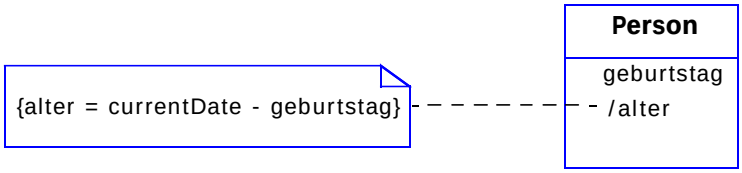


- **Komposition**: Aggregation, bei der eine Klasse ein Attribut einer anderen ist
  - Teil-Objekt gehört genau zu einem Aggregat-Objekt
  - Existenzabhängigkeit
- Unterschiedliche Darstellungsarten zur Komposition

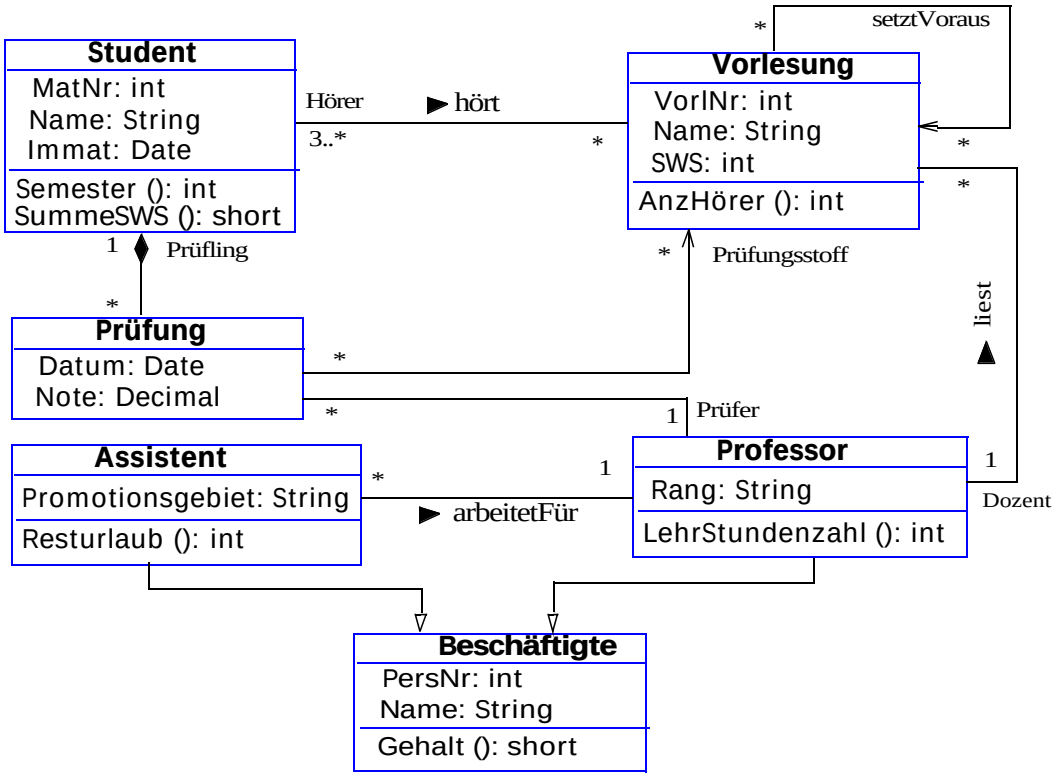


Kommentare / abgeleitete Attribute





Beispiel UML-Klassendiagramm



Zusammenfassung

- DB-Entwurf umfaßt
  - Informationsanalyse
  - konzeptioneller Entwurf (-> Informationsmodell)
  - logischer Entwurf (-> logisches DB-Schema)
  - physischer Entwurf (-> physisches DB-Schema)
- Informationsmodellierung mit dem ER-Modell
  - Entity-Mengen und Relationship-Mengen
  - Attribute, Wertebereiche, Primärschlüssel
  - Beziehungstypen (1:1, n:1, n:m) und Kardinalitätsrestriktionen
  - Diagrammdarstellung
- UML-Klassendiagramme: Unterschiede zu ER-Modell
  - standardisiert
  - Spezifikation von Verhalten (Methoden), nicht nur strukturelle Aspekte
  - Unterstützung der Abstraktionskonzepte der Generalisierung / Spezialisierung, Aggregation / Komposition
- keine festen Regeln zur eigentlichen Informationsmodellierung (i.a. mehrere Modellierungsmöglichkeiten einer bestimmten Miniwelt)

3. Grundlagen des Relationalen Datenmodells

Grundkonzepte

Relationale Invarianten

- Primärschlüsselbedingung
- Fremdschlüsselbedingung (referentielle Integrität)
- Wartung der referentiellen Integrität

Abbildung ERM → RM

Nachbildung der Generalisierung und Aggregation im RM

Relationenalgebra

- Mengenoperationen
- relationale Operatoren: Selektion, Projektion, Join

Kapitel 4: Die Standard-Anfragesprache SQL

Kapitel 5: Logischer DB-Entwurf (Normalformenlehre)

Kapitel 6: Datenmanipulation, -definition, und -kontrolle

Kapitel 7: DB-Anwendungsprogrammierung (evtl. in DBS2)

Lernziele

Grundbegriffe des Relationenmodells

Relationale Invarianten, insbesondere Vorkehrungen zur Wahrung der referentiellen Integrität

Abbildung von ER-Diagrammen in Relationenschema (und umgekehrt)

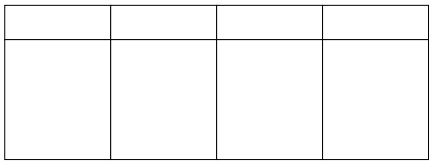
Operationen der Relationenalgebra: Definition und praktische Anwendung

Relationenmodell - Übersicht

Entwicklungsetappen

- Vorschlag von E.F. Codd (Communications of the ACM 1970)
- 1975: Prototypen: System R (IBM Research), Ingres (Berkeley Univ.)
- seit 1980: kommerzielle relationale DBS

Datenstruktur: Relation (Tabelle)

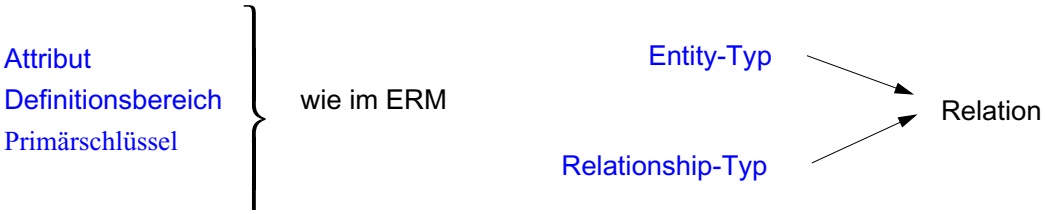


- einzige Datenstruktur (neben atomaren Werten)
- alle Informationen ausschließlich durch Werte dargestellt
- Integritätsbedingungen auf/zwischen Relationen: *relationale Invarianten*

Operatoren auf (mehreren) Relationen

- Vereinigung, Differenz
- Kartesisches Produkt
- Projektion
- Selektion
- zusätzlich: Änderungsoperationen (Einfügen, Löschen, Ändern)

Relationenmodell - Grundkonzepte



Def.: normalisierte Relation

$$R(A_1, A_2, \ldots, A_n) \subseteq W(A_1) \times W(A_2) \times \ldots \times W(A_n)$$

$W(A_1) \rightarrow D_i$, $W(A_2) \rightarrow D_j$, $W(A_n) \rightarrow D_k$

- Relation = Untermenge des kartesischen Produktes der Attributwertebereiche
- nur einfache Attibute (atomare Werte) !

Darstellungsmöglichkeit für R: n-spaltige Tabelle (*Grad* der Relation: n)

Relation ist eine Menge: Garantie der Eindeutigkeit der Zeilen/Tupel

-> Primärschlüssel (ggf. mehrere Schlüsselkandidaten)

Normalisierte Relationen in Tabellendarstellung

FAK

FNR	FNAME	...
WI	Wirtschaftswiss.	...
MI	Math./Informatik	...

STUDENT

MATNR	SNAME	FNR	STUDBEG
123 766	Coy	MI	1. 10. 00
654 711	Abel	WI	1. 10. 01
196 481	Maier	MI	1. 4. 00
226 302	Schulz	MI	1. 10. 00

Grundregeln:

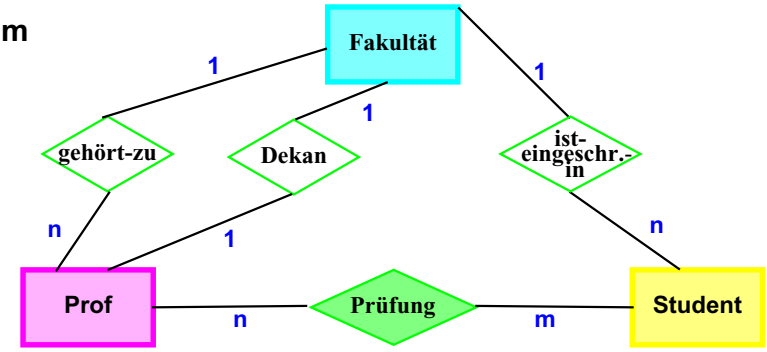
- Jede Zeile (Tupel) ist eindeutig und beschreibt ein Objekt (Entity) der Miniwelt
- Die Ordnung der Zeilen ist ohne Bedeutung; durch ihre Reihenfolge wird keine für den Benutzer relevante Information ausgedrückt
- Die Ordnung der Spalten ist ohne Bedeutung, da sie einen eindeutigen Namen (Attributnamen) tragen
- Jeder Datenwert innerhalb einer Relation ist ein atomares Datenelement
- Alle für den Benutzer bedeutungsvollen Informationen sind ausschließlich durch Datenwerte ausgedrückt

Darstellung "relationenübergreifender" Information durch Fremdschlüssel (foreign key)

- Attribut, das in Bezug auf den Primärschlüssel einer anderen (oder derselben) Relation definiert ist (gleicher Definitionsbereich)
- Beziehungen werden durch Fremdschlüssel und zugehörigen Primärschlüssel dargestellt!

Anwendungsbeispiel

ER-Diagramm



Relationales Schema

FAK

FNR	FNAME	DEKAN

STUDENT

MATNR	SNAME	FNR	STUDBEG

PROF

PNR	PNAME	FNR	FACHGEB

PRUEFUNG

PNR	MATNR	FACH	DATUM	NOTE

Relationale Invarianten

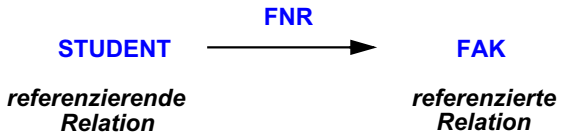
inhärente Integritätsbedingungen des Relationenmodells (Modellbedingungen)

1. Primärschlüsselbedingung (Entity-Integrität)
   - Eindeutigkeit des Primärschlüssels
   - keine Nullwerte!
2. Fremdschlüsselbedingung (referentielle Integrität):
   - zugehöriger Primärschlüssel muß existieren
   - d.h. zu jedem Wert (ungleich Null) eines Fremdschlüsselattributs einer Relation R2 muß ein gleicher Wert des Primärschlüssels in irgendeinem Tupel von Relation R1 vorhanden sein

DDL-Spezifikation in SQL:

```
CREATE TABLE FAK  (FNR CHAR(4), ...,PRIMARY KEY (FNR)
CREATE TABLE STUDENT  (MATNR CHAR (9), ... , FNR CHAR(4), ...,
      PRIMARY KEY (MATNR), FOREIGN KEY (FNR) REFERENCES FAK
```

Graphische Notation:



Relationale Invarianten (2)

Fremdschlüssel und zugehöriger Primärschlüssel tragen wichtige interrelationale (manchmal auch intrarelationale) Informationen

- sie sind auf dem gleichen Wertebereich definiert
- sie gestatten die Verknüpfung von Relationen mit Hilfe von Relationenoperationen

Fremdschlüssel

- können Nullwerte aufweisen, wenn sie nicht Teil eines Primärschlüssels sind.
- ein Fremdschlüssel ist zusammengesetzt, wenn der zugehörige Primärschlüssel zusammengesetzt ist. *(FS-Wert = "Null" bedeutet dann, daß alle Komponenten auf "Null" gesetzt sind)*

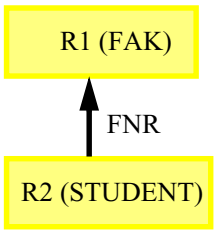
eine Relation kann mehrere Fremdschlüssel besitzen, die die gleiche oder verschiedene Relationen referenzieren

Zyklen sind möglich ***(geschlossener referentieller Pfad)***

eine Relation kann zugleich referenzierende und referenzierte Relation sein ***("self-referencing table")***.

Wartung der referentiellen Integrität

Gefährdung bei INSERT, UPDATE, DELETE



Fall 0: INSERT auf R1, DELETE auf R2

Fall 1: INSERT bzw. UPDATE auf FS der referenzierenden (abhängigen) Relation R2: Ablehnung falls kein zugehöriger PS-Wert in referenzierter Relation R1 besteht

Fall 2: DELETE auf referenzierter Relation R1 bzw. UPDATE auf PS von R1. Unterschiedliche Folgeaktionen auf referenzierender Relation R2 möglich, um referentielle Integrität zu wahren

Wartung der referentiellen Integrität (2)

SQL92-Standard erlaubt Spezifikation der referentiellen Aktionen für jeden Fremdschlüssel (FS)

Sind Nullwerte verboten?

- **NOT NULL**

Löschregel für Zielrelation (referenzierte Relation R1):

ON DELETE {NO ACTION | CASCADE | SET NULL | SET DEFAULT }

Änderungsregel für Ziel-Primärschlüssel (Primärschlüssel oder Schlüsselkandidat):

ON UPDATE {NO ACTION | CASCADE | SET NULL | SET DEFAULT}

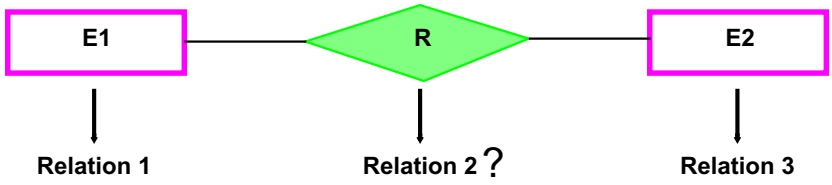
Dabei bedeuten:

- **NO ACTION:** Operation wird nur zugelassen, wenn keine zugehörigen Sätze (Fremdschlüsselwerte) vorhanden sind. Es sind folglich keine referentiellen Aktionen auszuführen
- **CASCADE**: Operation "kaskadiert" zu allen zugehörigen Sätzen
- **SET NULL**: Fremdschlüssel wird in zugehörigen Sätzen zu “Null” gesetzt
- **SET DEFAULT:** Fremdschlüssel wird auf einen benutzerdefinierten Default-Wert gesetzt

Anwendungsbeispiel

```
CREATE TABLE   TEIL (TNR, BEZEICHNUNG, . . . ),                PRIMARY KEY (TNR)

CREATE TABLE   LIEFERANT (LNR, LNAME, ORT, . . . )             PRIMARY KEY (LNR)

CREATE TABLE   LIEFERUNG (TNR, LNR, MENGE, DATUM)   PRIMARY KEY (TNR, LNR),
               FOREIGN KEY (TNR)      REFERENCES TEIL,  NOT NULL,
                                      ON DELETE OF TEIL NO ACTION
                                      ON UPDATE OF TEIL.TNR CASCADE,
               FOREIGN KEY (LNR)      REFERENCES LIEFERANT, NOT NULL,
                                      ON DELETE OF LIEFERANT NO ACTION,
                                      ON UPDATE OF LIEFERANT.LNR CASCADE
```

Abbildung ERM -> RM



Kriterien

- Informationserhaltung
- Minimierung der Redundanz
- Minimierung des Verknüpfungsaufwandes

aber auch:

- Natürlichkeit der Abbildung
- keine Vermischung von Objekten
- Verständlichkeit

Regeln:

- Jeder Entity-Typ *muß* als eigenständige Relation (Tabelle) mit einem eindeutigen Primärschlüssel definiert werden.
- Relationship-Typen *können* als eigene Relationen definiert werden, wobei die Primärschlüssel der zugehörigen Entity-Typen als Fremdschlüssel zu verwenden sind.

2 Entity-Typen mit n:1 - Verknüpfung

1.) Verwendung von drei Relationen

ABT (ANR, ANAME, ...)

PERS (PNR, PNAME, ...)

ABT-ZUGEH (PNR, ANR,)

2.) Besser: Verwendung von zwei Relationen

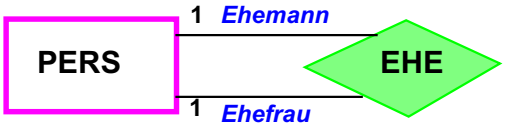
ABT (ANR, ANAME, ...)

PERS (PNR, PNAME, ..., ANR)

Regel: n:1-Beziehungen lassen sich ohne eigene Relation darstellen.

- Hierzu wird in der Relation, der maximal 1 Tupel der anderen Relation zugeordnet ist, der Primärschlüssel der referenzierten Relation als Fremdschlüssel verwendet

1 Entity-Typ mit 1:1 - Verknüpfung



1.) Verwendung von zwei Relationen

PERS (PNR, PNAME, ...)

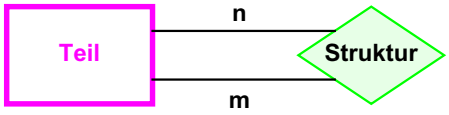
EHE (PNR , GATTE, ...)

2.) Verwendung von einer Relation

PERS (PNR, PNAME, ..., GATTE)

Unterscheidung zu n:1 ?

1 Entity-Typ mit m:n - Verknüpfung

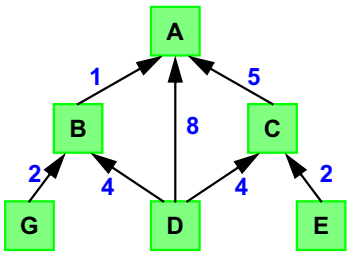


Stückliste

Darstellungsmöglichkeit im RM:

TEIL (TNR, BEZ, MAT, BESTAND)
STRUKTUR (OTNR, UTNR, ANZAHL)

Gozinto-Graph



TEIL

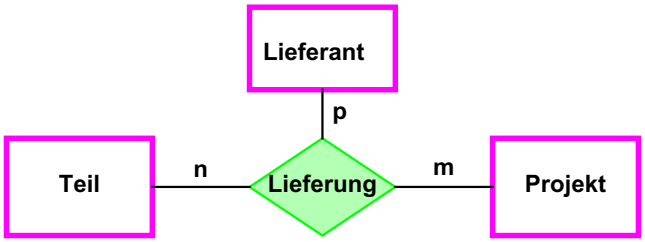
TNR	BEZ	MAT	BESTAND
A	Getriebe	-	10
B	Gehäuse	Alu	0
C	Welle	Stahl	100
D	Schraube	Stahl	200
E	Kugellager	Stahl	50
F	Scheibe	Blei	0
G	Schraube	Chrom	100

STRUKTUR

OTNR	UTNR	ANZAHL
A	B	1
A	C	5
A	D	8
B	D	4
B	G	2
C	D	4
C	E	2

Regel: Ein n:m-Relationship-Typ muß durch eine eigene Relation dargestellt werden. Die Primärschlüssel der zugehörigen Entity-Typen treten als Fremdschlüssel auf.

3 Entity-Typen mit m:n - Verknüpfung



LIEFERANT (LNR, LNAME, LORT, ...)

PROJEKT (PRONR, PRONAME, PORT, ...)

TEIL (TNR, TBEZ, GEWICHT, ...)

LIEFERUNG (LNR, PRONR, TNR, ANZAHL, DATUM)

Abbildung mehrwertiger Attribute bzw. schwacher Entity-Typen

Entity-Typ

```
PERS   (PNR, NAME,    {Lieblingsessen}, {Kinder (Vorname, Alter)})

         P1, Müller,  {Schnitzel, Rollmops},  -
         P2, Schulz,  {Pizza},              {(Nadine, 5), (Philip, 2)}
```

Darstellungsmöglichkeit im RM

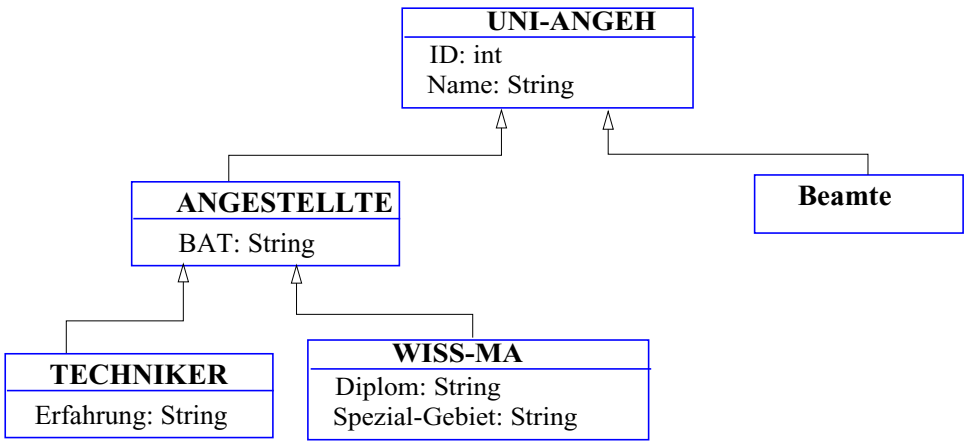
```
PERS        (PNR,   NAME   ...)

L.ESSEN     (PNR,    GERICHT)

KINDER      (PNR,   VORNAME, ALTER)
```


Abbildung von Generalisierung und Aggregation im RM

RM sieht keine Unterstützung der Abstraktionskonzepte vor

- keine Maßnahmen zur Vererbung (von Struktur, Integritätsbedingungen, Operationen)
- "Simulation" der Generalisierung und Aggregation eingeschränkt möglich

Generalisierungsbeispiel:



Generalisierung - relationale Sicht

pro Klasse 1 Tabelle

Lösungsmöglichkeit 1: vertikale Partitionierung

- jede Instanz wird entsprechend der Klassenattribute in der IS-A-Hierarchie zerlegt und in Teilen in den zugehörigen Klassen (Relationen) gespeichert.
- nur das ID-Attribut wird dupliziert

UNI-ANGEH

ID	Name
007	Garfield
123	Donald
333	Daisy
765	Grouch
111	Ernie

ANGESTELLTE

ID	BAT
007	Ib
123	IVa
333	VII
765	IIa

WISS-MA

ID	Diplom	SPEZ-GEB
007	Informatik	ERM
765	Mathe	OO

TECHNIKER

ID	Erfahrung
123	Sun

Eigenschaften

- geringfügig erhöhte Speicherungskosten, aber hohe Aufsuch- und Aktualisierungkosten
- Integritätsbedingungen: $TECHNIKER.ID \subseteq ANGESTELLTE.ID$, usw.
- Instanzenzugriff erfordert implizite oder explizite Verbundoperationen
- Beispiel: Finde alle TECHNIKER-Daten $TECHNIKER \underset{ID=ID}{\bowtie} ANGESTELLTE \underset{ID=ID}{\bowtie} UNI\text{-}ANGEH$

Generalisierung - relationale Sicht (2)

Lösungsmöglichkeit 2: horizontale Partitionierung

- Jede Instanz ist genau einmal und vollständig in ihrer "Hausklasse" gespeichert.
- keinerlei Redundanz

UNI-ANGEH

ID	Name
111	Ernie

ANGESTELLTE

ID	NAME	BAT
333	Daisy	VII

WISS-MA

ID	Diplom	SPEZ-GEB	NAME	BAT
007	Informatik	ERM	Garfield	IIa
765	Mathe	OO	Grouch	IIa

TECHNIKER

ID	Erfahrung	NAME	BAT
123	SUN	Donald	IVa

Eigenschaften

- Niedrige Speicherungskosten und keine Änderungsanomalien
- Eindeutigkeit von ID zwischen Relationen aufwendiger zu überwachen
- Retrieval kann rekursives Suchen in Unterklassen erfordern.
- Explizite Rekonstruktion durch Relationenoperationen (π, $\cup$)

=> Beispiel: Finde alle ANGESTELLTE:
$\pi_{ID, NAME, BAT}(TECHNIKER) \cup \pi_{ID, NAME, BAT}(WISS\text{-}MA) \cup ANGESTELLTE$

Generalisierung - relationale Sicht (3)

Lösungsmöglichkeit 3: volle Redundanz

- Eine Instanz wird wiederholt in jeder Klasse, zu der sie gehört, gespeichert.
- Sie besitzt dabei die Werte der Attribute, die sie geerbt hat, zusammen mit den Werten der Attribute der Klasse

UNI-ANGEH

ID	Name
007	Garfield
123	Donald
333	Daisy
765	Grouch
111	Ernie

ANGESTELLTE

ID	NAME	BAT
007	Garfield	Ib
123	Donald	IVa
333	Daisy	VII
765	Grouch	IIa

WISS-MA

ID	NAME	BAT	Diplom	SPEZ-GEB
007	Garfield	IIa	Informatik	ERM
765	Grouch	IIa	Mathe	OO

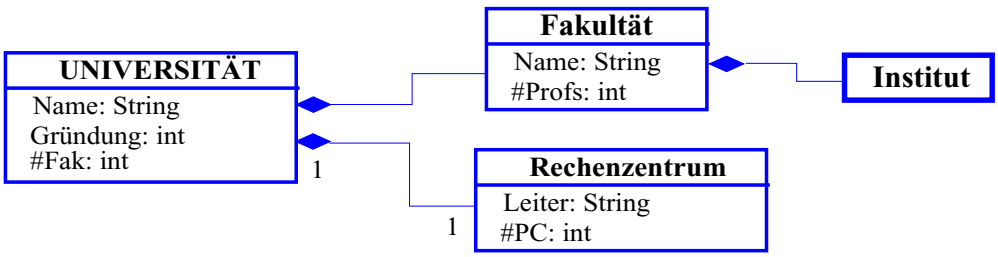
TECHNIKER

ID	NAME	BAT	Erfahrung
123	Donald	IVa	Sun

Eigenschaften

- Sehr hoher Speicherplatzbedarf und Auftreten von Änderungsanomalien.
- Sehr einfaches Retrieval, da nur die Zielklasse (z. B. ANGESTELLTE) aufgesucht werden muß

Aggregation - relationale Sicht



Universität

ID	Name	Gründung	#Fak
UL	Univ. Leipzig	1409	14
TUD	TU Dresden	1828	14

Institut

IID	FID	Name
1234	123	Neutestamentliche Wissenschaft
1235	123	Alttestamentliche Wissenschaft
1236	123	Praktische Theologie
1322	132	Informatik

Fakultät

FID	Uni	Name	#Profs
123	UL	Theologie	14
132	UL	Mathematik/Informatik	28

Komplexe Objekte erfordern Zerlegung über mehrere Tabellen

Sprachen für das Relationenmodell

Datenmodell = Datenobjekte + Operatoren

Unterstützung verschiedener Benutzerklassen:

- Anwendungsprogrammierer
- DBA
- anspruchsvolle Laien
- parametrische Benutzer
- gelegentliche Benutzer

im RM wird vereinheitlichte Sprache angestrebt für:

- Anfragen (Queries) im 'Stand-Alone'-Modus
- Datenmanipulation und Anfragen eingebettet in eine Wirtssprache
- Datendefinition
- Zugriffs- und Integritätskontrolle

Verschiedene Grundtypen:

- Formale Ansätze: Relationenalgebra und Relationenkalkül
- Abbildungsorientierte Sprachen (z.B. SQL)
- Graphik-orientierte Sprachen (z.B. Query-by-Example)

Relationenalgebra

Algebra: ein System, das aus einer nichtleeren Menge und einer Familie von Operationen besteht

Relationen sind Mengen

Operationen auf Relationen arbeiten auf einer oder mehreren Relationen als Eingabe und erzeugen eine Relation als Ausgabe (Abgeschlossenheitseigenschaft)
=> mengenorientierte Operationen

Operationen:

***Klassische Mengenoperationen*:**

- Vereinigung
- Differenz
- kartesisches Produkt
- Durchschnitt (ableitbar)

***Relationenoperationen*:**

- Restriktion (Selektion)
- Projektion
- Verbund (Join) (ableitbar)
- Division (ableitbar)

Selektion (Restriktion)

Auswahl von Zeilen einer Relation über Prädikate, abgekürzt σ_P

P = log. Formel (ohne Quantoren !) zusammengestellt aus:

a) Operanden — Konstanten / Attributnamen

b) Vergleichsoperatoren $\theta \in \{<, =, >, \leq, \neq, \geq\}$

c) logische Operatoren: $\vee$, $\wedge$, $\neg$

$$\sigma_P(R) = \{ t \mid t \in R \wedge P(t)\}$$

Eigenschaften: grad $(\sigma_P(R))$ = grad (R); card $(\sigma_P(R))$ <= card (R)

Beispiele:

$$\sigma_{ANR='K55' \wedge GEHALT > 50000}\ (PERS)$$

$$\sigma_{GEHALT < PROVISION}\ (PERS)$$

Projektion

Auswahl der Spalten (Attribute) $A_1, A_2, \ldots, A_k$ aus einer Relation R (Grad n >= k)

$$\pi_{A_1, A_2, \ldots, A_k}(R) = \{ p \mid \exists t \in R : p = < t[A_1], \ldots, t[A_k] >\}$$

Eigenschaften:

- Wichtig: Duplikate werden entfernt ! (Mengeneigenschaft)
- grad $(\pi_A(R))$ <= grad (R);
- card $(\pi_A(R))$ <= card (R)

Beispiel:

$$\pi_{NAME, GEHALT}\ (PERS)$$

Relationenalgebra: Beispiel-DB

ABT

ANR	ANAME	ORT
K51	Planung	Leipzig
K53	Einkauf	Frankfurt
K55	Vertrieb	Frankfurt

PERS

PNR	Name	Alter	Gehalt	ANR	MNR
406	Abel	47	50700	K55	123
123	Schulz	32	43500	K51	-
829	Müller	36	40200	K53	406
574	Schmid	28	36000	K55	123

Finde alle Angestellten ausAbteilung K55, die mehr als 40.000 verdienen

Finde alle Abteilungsorte

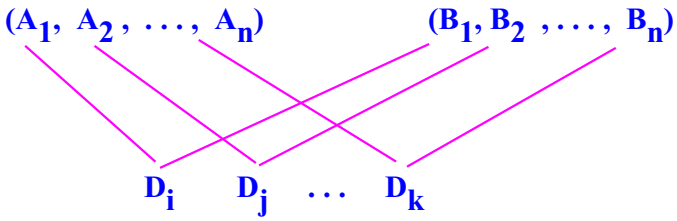
Finde den Abteilungsnamen von Abteilung K53

Finde alle Angestellten (PNR, ALTER, ANAME), die in einer Abteilung in Frankfurt arbeiten und älter als 30 sind.

Klassische Mengenoperationen

Voraussetzung: *Vereinigungsverträglichkeit* der beteiligten Relationen:

Gleicher Grad - Gleiche Bereiche: => $W(A_i) = W(B_i)$: $i = 1, n$



Vereinigung (UNION): $R \cup S = \{ t \mid t \in R \vee t \in S \}$

- card (R ∪ S) <= card (R) + card (S)

Differenz: $R - S = \{ t \mid t \in R \wedge t \notin S \}$

- card (R – S) <= card (R)

Durchschnitt (INTERSECTION): $R \cap S =$
$R - (R-S) = \{ t \mid t \in R \wedge t \in S \}$

- card (R ∩ S) <= min (card (R), card (S))

(Erweitertes) Kartesisches Produkt

R (Grad r) und S (Grad s) beliebig

$$R \times S = \{ k \mid \exists x \in R, y \in S : k = x \mid y \}$$

! $k = x \mid y = < x_1, \ldots, x_r, y_1, \ldots, y_s >$

nicht $<<x_1, \ldots, x_r>, <y_1, \ldots, y_s>>$ wie übliches kart. Produkt

- grad (R x S) = grad (R) + grad (S); card (R x S) = card (R) x card (S)

Beispiel

R

A	B	C
a	γ	1
d	α	2
b	β	3

S

D	E	F
b	γ	3
d	α	2

R x S

Verbund (Join, Θ-Join)

grob:

- Kartesisches Produkt zwischen zwei Relationen R (Grad r) und S (Grad s).
- eingeschränkt durch Θ-Bedingungen zwischen Attribut A von R und Attribut B von S.

Θ-Verbund zwischen R und S:

$$R \underset{A\Theta B}{\bowtie} S = \sigma_{A\Theta B}(R \times S)$$

mit $\Theta \in \{<, =, >, \leq, \neq, \geq\}$
(arithm. Vergleichsoperator)

Bemerkungen:

- ***Gleichverbund (Equijoin):*** θ = '=' :
- Ein Gleichverbund zwischen R und S heißt *verlustfrei*, wenn alle Tupel von R und S am Verbund teilnehmen. Die inverse Operation Projektion erzeugt dann wieder R und S (**lossless join**).

Natürlicher Verbund (Natural Join)

grob: Gleichverbund über alle gleichen Attribute und Projektion über die verschiedenen Attribute

Natürlicher Verbund zwischen R und S:

gegeben: $R(A_1, A_2, \ldots, A_{r-j+1}, \ldots, A_r)$
$S(B_1, B_2, \ldots, B_j, \ldots, B_s)$

o.B.d.A.:(sonst. Umsortierung:

$B_1 = A_{r-j+1}$
$B_2 = A_{r-j+2}$
$\vdots$
$B_j = A_r$

$$R \bowtie S = \pi_{A_1, \ldots, A_r, B_{j+1}, \ldots, B_s} \sigma_{(R.A_{r-j+1} = S.B_1) \wedge \ldots \wedge (R.A_r = S.B_j)} (R \times S)$$

$\bowtie$ = Zeichen für Natural Join $\Rightarrow$ Θ = '='

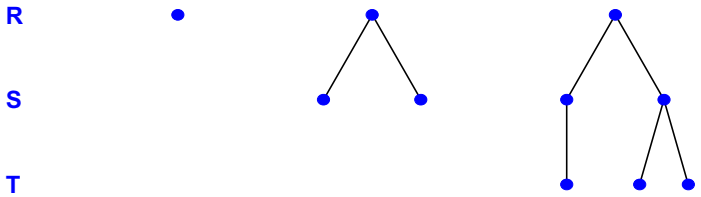
Bemerkung: Attribute sind durch Übereinstimmungsbedingung gegeben

Äußerer Verbund (Outer Join)

Ziel:Verlustfreier Verbund soll erzwungen werden

Bisher:$R \bowtie S \bowtie T$ liefert nur "vollständige Objekte"

- Bei $R \bowtie S \bowtie T$ sollen auch Teilobjekte als Ergebnis geliefert werden (z.B. komplexe Objekte).



- Trick: Einfügen einer speziellen Leerzeile zur künstlichen Erzeugung von Verbundpartnern

Def.: Seien A die Verbundattribute, $\{\equiv\}$ der undefinierte Wert und

$R' := R \cup ((\pi_A(S) - \pi_A(R)) \times \{\equiv\} \times \ldots \times \{\equiv\})$ $\quad S' := S \cup ((\pi_A(R) - \pi_A(S)) \times \{\equiv\} \times \ldots \times \{\equiv\})$

Äußerer Gleichverbund

$R \overset{R.A = S.A}{\supset\!\bowtie\!\subset} S := R' \overset{R'.A = S'.A}{\bowtie} S'$

Äußerer natürlicher Verbund

$R \supset\!\bowtie\!\subset S := R' \bowtie S'$

zweimalige Anwendung des äußeren Gleichverbundes $R \supset\!\bowtie\!\subset S \supset\!\bowtie\!\subset T$ liefert gewünschtes Ergebnis

Outer Join (2)

Linker äußerer Gleichverbund

- Bei bei dieser Operation bleibt die linke Argumentrelation verlustfrei, d.h. bei Bedarf wird ein Tupel durch Nullwerte “nach rechts” aufgefüllt.

Linker äußerer Gleichverbund: $R \overset{R.A = S.A}{\supset\bowtie} S := R \overset{R.A=S'.A}{\bowtie} S'$

Rechter äußerer Gleichverbund

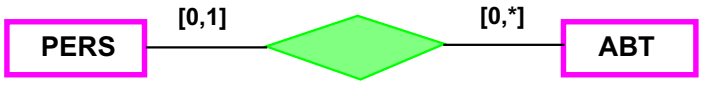
- Dabei bleibt analog die rechte Argumentrelation verlustfrei; fehlende Partnertupel werden durch Auffüllen mit Nullwerten “nach links” ergänzt

Rechter äußerer Gleichverbund: $R \overset{R.A = S.A}{\bowtie\subset} S := R' \overset{R'.A = S.A}{\bowtie} S$

Zusammenfassung

- Die Anwendung des äußeren Gleichverbundes liefert die maximale Information bezüglich der Operationsfolge. Selbst isolierte Tupel werden zu einem Pfad expandiert.
- Der Einsatz des Gleichverbundes mit $R \bowtie S \bowtie T$ bringt das Minimum an Information; nur vollständig definierte Pfade werden ins Ergebnis übernommen.
- Mit dem linken (rechten) äußeren Gleichverbund werden nur Pfade zurückgeliefert, die am “linken (rechten) Rand” definiert sind.

Outer Join - Beispiel



PERS

PNR	ANR ...
P1	A1
P2	A1
P3	A2
P4	-
P5	-

ABT

ANR	ANAME ...
A1	A
A2	B
A3	C

PERS $\bowtie$ ABT

PNR	ANR	ANAME ...

PERS $\supset\bowtie$ ABT

PNR	ANR	ANAME ...

PERS $\bowtie\subset$ ABT

PNR	ANR	ANAME ...

PERS $\supset\bowtie\subset$ ABT

PNR	ANR	ANAME ...

Division

Beantwortung von Fragen, bei denen eine "ganze Relation" zur Qualifikation herangezogen wird

Simulation des Allquantors => ein Tupel aus R steht mit allen Tupeln aus S in einer bestimmten Beziehung

Definition

Sei R vom Grad r und S vom Grad s, $r > s$ und $s \neq 0$.

t sei (r-s)-Tupel, u sei s-Tupel; S-Attribute $\subset$ R-Attribute

Dann gilt: $R \div S = \{ t \mid \forall u \in S : tu \in R \}$

Zusammenhang zwischen Division und kartesischem Produkt:

$$(R \times S) \div S = R$$

Beispiel

LPT

LNR	PNR	TNR
L1	P1	T1
L1	P2	T1
L2	P1	T1
L2	P1	T2
L2	P2	T1

÷

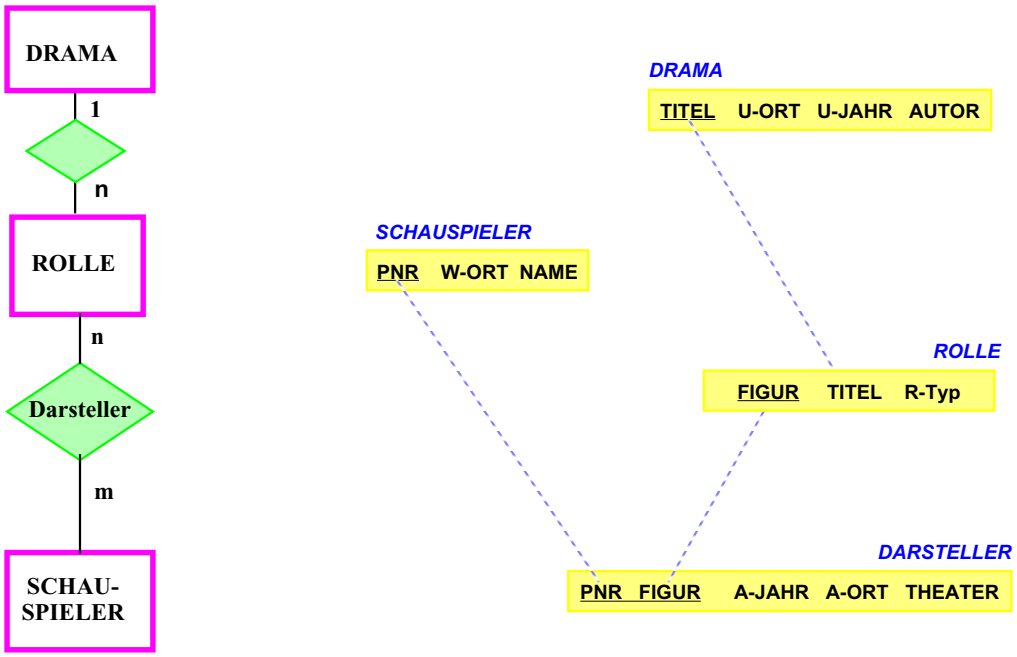
PNR	TNR
P1	T1
P1	T2
P2	T1

PT

Welche Lieferanten beliefern alle Projekte ?

Welche Lieferanten liefern alle Teile ?

Beispiel-DB: BÜHNE



Beispielanfragen

Welche Darsteller (PNR) haben im Schauspielhaus gespielt?

Finde alle Schauspieler (NAME, W-ORT), die einmal im 'Faust' mitgespielt haben.

Finde die Schauspieler (PNR), die nie gespielt haben

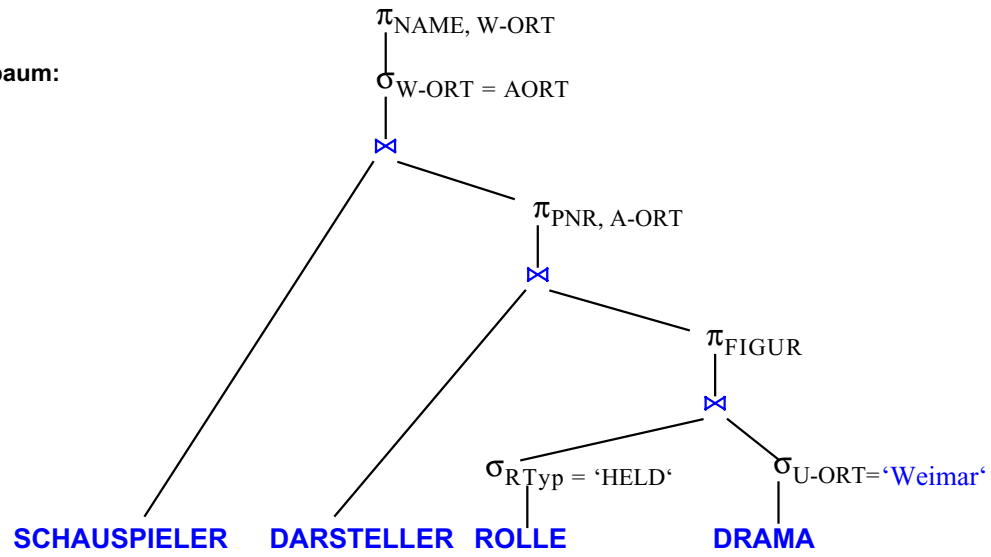
Finde alle Schauspieler (NAME), die alle Rollen gespielt haben.

Beispielanfragen (2)

Finde alle Schauspieler (NAME, W-ORT), die bei in Weimar uraufgeführten Dramen an ihrem Wohnort als 'Held' mitgespielt haben.

$$Q = \pi_{\text{NAME, W-ORT}}(\sigma_{\text{W-ORT = AORT}}(\text{SCHAUSPIELER}) \bowtie (\pi_{\text{PNR, AORT}}(\text{DARSTELLER} \bowtie \pi_{\text{FIGUR}}(\sigma_{\text{RTyp = 'HELD'}}(\text{ROLLE})) \bowtie \sigma_{\text{U-ORT = 'WEIMAR'}}(\text{DRAMA}))))$$

Operatorbaum:



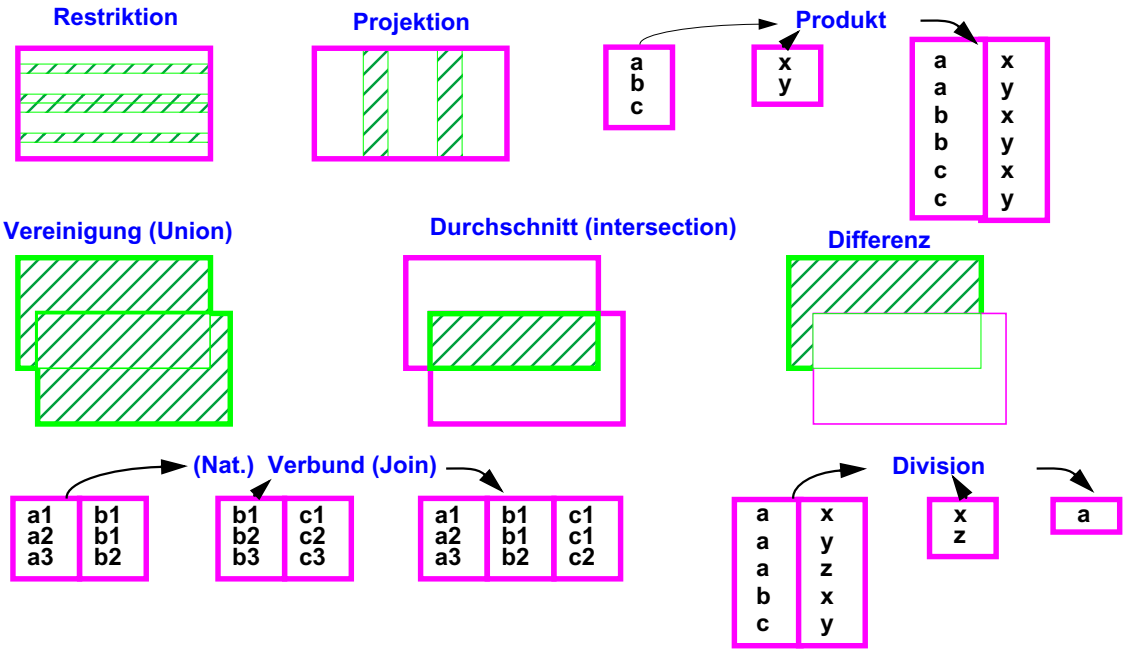
Zusammenfassung Relationenalgebra

saubere mathematische Definition

mengenorientierte Opertationen

keine Änderungsoperationen!

für Laien nicht leicht verständlich



4. SQL-Einführung

- Grundlagen: Anfragekonzept, Einsatzbereiche, Befehlsübersicht
- mengenorientierte Anfragen deskriptiver Art (SELECT)
  - einfache Selektions-, Projektions- und Join-Anfragen
  - geschachtelte Anfragen, Aggregatfunktionen, Gruppenanfragen
  - Suchbedingungen: Vergleichs-, LIKE-, BETWEEN-, IN-Prädikate, Nullwertbehandlung, quantifizierte Prädikate (ALL/ANY, EXISTS)
  - mengentheoretische Operationen: UNION, INTERSECT, EXCEPT
  - Join-Ausdrücke
  - Verallgemeinerte Verwendung von Sub-Queries
  - Vergleich mit der Relationenalgebra

Kap. 6: Datendefinition und -manipulation in SQL

- Datendefinition (DDL): Datentypen, Domains
- Erzeugen/Ändern/Löschen von Tabellen, Sichtkonzept
- Änderungsoperationen (INSERT, DELETE, UPDATE)
- Integritätsbedingungen

Kap. 7: Kopplung mit einer Wirtssprache

Entwicklung von SQL

- seit 1975 viele Entwürfe für relationale Anfragesprachen
  - SEQUEL: Structured English Query Language (System R)
  - QUEL (Ingres), . . .
- Sprachentwicklung von SQL
  - Entwicklung einer vereinheitlichten DB-Sprache für alle Aufgaben der DB-Verwaltung
  - leichter Zugang durch verschiedene "Sprachebenen"
  - einfache Anfragemöglichkeiten für den gelegentlichen Benutzer, mächtige Sprachkonstrukte für den besser ausgebildeten Benutzer, spezielle Sprachkonstrukte für den DBA
- Standardisierung von SQL durch ANSI und ISO
  - erster ISO-Standard 1987
  - verschiedene Addenda (1989)
  - 1992: Verabschiedung von „SQL2“ bzw. SQL-92 (Entry, Intermediate, Full Level)
  - 1999/2003: SQL:1999 („SQL3“) und SQL:2003 („SQL4“) mit objektorientierten Erweiterungen etc. (-> objekt-relationale DBS)
- SQL "intergalactic data speak"

Möglichkeiten der Anfrage in SQL

- SQL: strukturierte Sprache, die auf englischen Schlüsselwörtern basiert
  - Zielgruppe umfaßt auch Nicht-Programmierer
  - Auswahlvermögen äquivalent der Relationenalgebra (relational vollständig)
- Grundbaustein

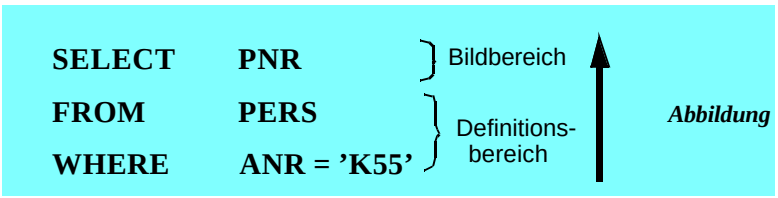


Abbildung: Ein bekanntes Attribut oder eine Menge von Attributen wird durch Angabe von Bedingungen (WHERE) mit Hilfe einer Relation (FROM) in ein gewünschtes Attribut oder eine Menge von Attributen abgebildet

- Allgemeines Format

  <Spezifikation der Operation>
  <Liste der referenzierten Tabellen>
  [WHERE Boolescher Prädikatsausdruck]

Erweiterungen zu einer vollständigen DB-Sprache

- Möglichkeiten der Datenmanipulation
  - Einfügen, Löschen und Ändern von individuellen Tupeln und von Mengen von Tupeln
  - Zuweisung von ganzen Relationen
- Möglichkeiten der Datendefinition
  - Definition von Wertebereichen, Attributen und Relationen
  - Definition von verschiedenen Sichten auf Relationen
- Möglichkeiten der Datenkontrolle
  - Spezifikation von Bedingungen zur Zugriffskontrolle
  - Spezifikation von Zusicherungen (assertions) zur semantischen Integritätskontrolle
- Kopplung mit einer Wirtssprache
  - deskriptive Auswahl von Mengen von Tupeln
  - sukzessive Bereitstellung einzelner Tupeln

	Retrieval	Manipulation	Daten-definition	Daten-kontrolle
Stand-Alone DB-Sprache	**SQL RA**	**SQL**	**SQL**	**SQL**
Eingebettete DB-Sprache	**SQL**	**SQL**	**SQL**	**SQL**

Befehlsübersicht (Auswahl)

Datenmanipulation (DML):

SELECT
INSERT
UPDATE
DELETE
Aggregatfunktionen: COUNT, SUM, AVG, MAX, MIN

Datendefinition (DDL):

CREATE SCHEMA
CREATE DOMAIN
CREATE TABLE
CREATE VIEW
ALTER TABLE
DROP SCHEMA
DROP DOMAIN
DROP TABLE
DROP VIEW

Datenkontrolle:

Constraints-Definitionen bei CREATE TABLE
CREATE ASSERTION
DROP ASSERTION
GRANT
REVOKE
COMMIT
ROLLBACK

Eingebettetes SQL:

DECLARE CURSOR
FETCH
OPEN CURSOR
CLOSE CURSOR
SET CONSTRAINTS
SET TRANSACTION
CREATE TEMPORARY TABLE

Anfragemöglichkeiten in SQL

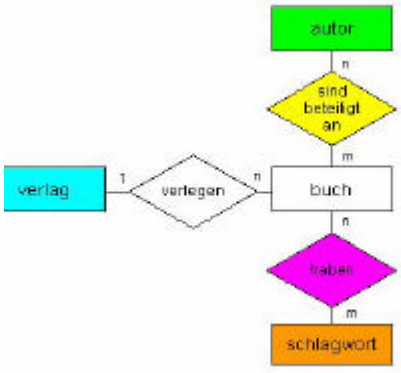
```
select-expression ::=
    SELECT [ALL | DISTINCT] select-item-list
    FROM table-ref-commalist
    [WHERE cond-exp]
    [GROUP BY column-ref-commalist]
    [HAVING cond-exp]
    [ORDER BY order-item-commalist ]

 select-item ::= derived-column | [range-variable.] *
 derived-column ::= scalar-exp [AS column]
 order-item ::= column [ ASC | DESC ]
```

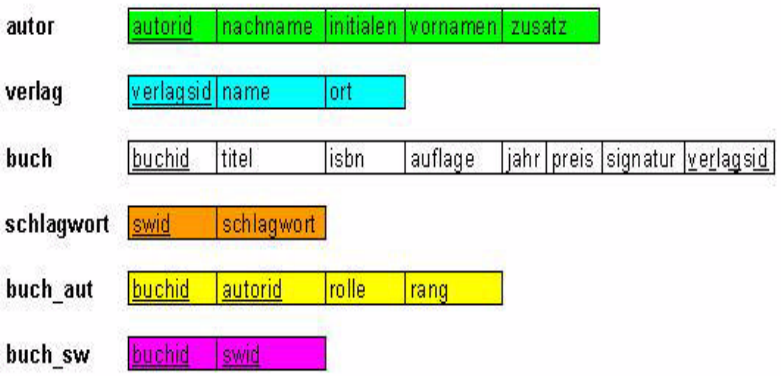
- Mit *SELECT* * werden alle Attribute der spezifizierten Relation(en) ausgegeben
- FROM-Klausel spezifiziert die Objekte (Relationen, Sichten), die verarbeitet werden sollen
- *WHERE*-Klausel kann eine Sammlung von Prädikaten enthalten, die mit *NOT*, *AND* und *OR* verknüpft sein können
- dabei sind Vergleichsprädikate der Form $A_i\ \theta\ a_i$ bzw. $A_i\ \theta\ A_j$ möglich ($\theta \in \{ =, <>, <, \leq, >, \geq \}$)

SQL-Trainer

- *http://sql-trainer.uni-leipzig.de*
- entstanden im Rahmen des Projekts „Bildungsportal Sachsen“ durch studentische Hilfskräfte
- Inhalt
  - „freies Üben“ auf einer SQL-Datenbank (nur SELECT-Anweisungen)
  - „aktives“ SQL-Tutorial
  - geplant: Übungsblätter mit Online-Bearbeitung
- Realisierung auf Basis von Postgres
- Gast-Login mit Zugriff auf 1 Datenbank (user: gast, PW: gast)

Test-Datenbank





- *buch_aut.rolle* kann sein: Herausgebers (H), Verfasser (V), Übersetzer (U), Mitarbeiter (M)
- *buch_aut.rang:* Position des Autors in der Autorenliste (z.B. 1 für Erstautor)
- *autor.zusatz:* Namenszusatz wie „von“ oder „van“
- *buch.signatur* entspricht der Signatur in der IfI-Bibliothek (Stand 1998)

- Mengengerüst (> 18.000 Sätze)
  - "buch" : 4873 Datensätze, "verlag" : 831 Datensätze
  - "autor" : 5045 Datensätze, "buch_aut" : 5860 Datensätze
  - "schlagwort" : 843 Datensätze, "buch_sw" : 789 Datensätze

Einfache Selektionen und Projektionen

Q1: Welche (Berliner) Verlage gibt es?

Q2: Welche Bücher erschienen vor 1980 in einer Neuauflage?

Ausgabebearbeitung

- Sortierte Ausgabe (ORDER BY-Klausel)

Q3: Q2, jedoch sortiert nach Jahr (absteigend), Titel (aufsteigend)

```
SELECT
FROM
WHERE
```

- ohne ORDER-BY ist die Reihenfolge der Ausgabe durch das DBS bestimmt (Optimierung der Auswertung)
- statt Attributnamen können in der ORDER BY-Klausel auch relative Positionen der Attribute aus der Select-Klausel verwendet werden

- Duplikateliminierung
  - Default-mäßig werden Duplikate in den Attributwerten der Ausgabeliste nicht eliminiert (ALL)
  - *DISTINCT* erzwingt Duplikateliminierung

Q4: Welche Verlagsorte gibt es?

Ausgabebearbeitung (2)

- Benennung von Ergebnis-Spalten

```
SELECT titel AS Buchtitel, (preis/2)  AS Preis_in_Euro
FROM buch
WHERE waehrung = 'DM'
ORDER BY 2 DESC
```

- Umbenennung von Attributen (AS)
- Vergabe von Attributnamen für Texte und Ausdrücke

- Umbenennung von Tabellen (FROM-Klausel)
  - Einführung sogenannter Alias-Namen bzw. Korrelationsnamen
  - Schlüsselwort AS optional
  - Alias-Name überdeckt ursprünglichen Namen

```
SELECT B.titel
FROM buch AS B
WHERE B.preis > 300
```


Join-Anfragen

Q5: Welche Buchtitel wurden von Berliner Verlagen herausgebracht?

```
SELECT
FROM
WHERE
```

- Angabe der beteiligten Relationen in FROM-Klausel
- WHERE-Klausel enthält Join-Bedingung sowie weitere Selektionsbedingungen
- analoge Vorgehensweise für Equi-Join und allgemeinen Theta-Join
- Formulierung der Join-Bedingung erfordert bei gleichnamigen Attributen Hinzunahme der Relationennamen oder von Alias-Namen (Korrelationsnamen)

Q6: Welche Bücher sind von Autor „Rahm“ vorhanden?

```
SELECT
FROM
WHERE
```

Join-Anfragen (2)

- Hierarchische Beziehung auf einer Relation (PERS)

Beispielschema:

```
PERS (PNR int, NAME, BERUF, GEHALT, ..., MNR int, ANR int,
       PRIMARY KEY (PNR), FOREIGN KEY (MNR) REFERENCES PERS)
```

Q7: Finde die Angestellten, die mehr als ihre (direkten) Manager verdienen (Ausgabe: NAME, GEHALT, NAME des Managers)

```
SELECT
FROM
WHERE
```

- Verwendung von Korrelationsnamen obligatorisch!

Join-Ausdrücke

- unterschiedliche Join-Arten können direkt spezifiziert werden

```
join-table-exp ::= table-ref [NATURAL] [join-type] JOIN table-ref
                        [ON cond-exp | USING (column-commalist) ]
                 | table ref CROSS JOIN table-ref | (join-table-exp)
table-ref ::= table | (table-exp) | join-table-exp
join type ::= INNER | { LEFT | RIGHT | FULL } [OUTER] | UNION
```

- Beispiel:

```
SELECT  *
FROM  buch B, verlag V
WHERE B.verlagsid =  V.verlagsid
```

```
buch NATURAL JOIN  verlag

buch JOIN  verlag USING (verlagsid)

buch B JOIN verlag V
       ON B.verlagsid =  V.verlagsid
```

Q6: Welche Bücher sind von Autor „Rahm“ vorhanden?

Join-Ausdrücke (2)

- Outer Joins: LEFT JOIN, RIGHT JOIN, FULL JOIN

```
schlagwort LEFT OUTER JOIN buch_sw USING (swid)
```

- Kartesisches Produkt:

```
A  CROSS JOIN B  <=> SELECT * FROM A, B
```


Join-Ausdrücke (3)

- Outer Union: UNION JOIN
  - Vereinigung von nur teilweise vereinigungsverträglichen Relationen)
- Beispiel:

STUDENT

Matnr	Name	Fak	Imm
123	Schmidt	MI	X
234	Müller	WiWi	Y

PERS

Pnr	Name	Fak	Wochenstunden
543	Schulz	WiWi	40
897	Weber	MI	20

```
STUDENT  UNION JOIN  PERS
```

Matnr	Name	Fak	Imm	Pnr	Wochenstunden
123	Schmidt	MI	X	-	-
234	Müller	WiWi	Y	-	-
-	Schulz	WiWi	-	543	40
-	Weber	MI	-	897	20

Geschachtelte Anfragen (Sub-Queries)

- Die Menge, die zur Qualifikation herangezogen wird, kann Ergebnis einer geschachtelten Abbildung sein

Q5: Welche Buchtitel wurden von Berliner Verlagen herausgebracht?

```
SELECT
FROM
WHERE
```

- innere und äußere Relationen können identisch sein
- eine geschachtelte Abbildung kann beliebig tief sein
- Join-Berechnung mit Sub-Queries
  - teilweise prozedurale Anfrageformulierung
  - weniger elegant als symmetrische Notation
  - schränkt Optimierungsmöglichkeiten des DBS ein

Sub-Queries (2)

- Einfache Sub-Queries
  - 1-malige Auswertung der Sub-Query
  - Ergebnismenge der Sub-Query (Zwischenrelation) dient als Eingabe der äußeren Anfrage
- Korrelierte Sub-Queries (verzahnt geschachtelte Anfragen)
  - Sub-Query bezieht sich auf eine äußere Relation
  - Sub-Query-Ausführung erfolgt für jedes Tupel der äußeren Relation
  - Verwendung von Korrelationsnamen i.a. erforderlich

Welche Buchtitel wurden von Berliner Verlagen veröffentlicht?

```
SELECT B.titel
FROM buch B
WHERE 'Berlin' IN
 (SELECT V.ort
  FROM verlag V
  WHERE V.verlagsid = B.verlagsid)
```

```
SELECT B.titel
FROM buch B
WHERE B.verlagsid IN
(SELECT V.verlagsid
FROM verlag V
WHERE V.ort = 'Berlin')
```

- besser: Join-Berechnung ohne Sub-Queries

Benutzung von Aggregat (Built-in)-Funktionen

```
aggregate-function-ref ::=COUNT(*) | {AVG | MAX | MIN | SUM | COUNT}
                                   ([ALL | DISTINCT] scalar-exp)
```

- Standard-Funktionen: AVG, SUM, COUNT, MIN, MAX
  - Elimination von Duplikaten : DISTINCT
  - keine Elimination : ALL (Defaultwert)
  - Typverträglichkeit erforderlich

Q8: Bestimme das Durchschnittsgehalt der Angestellen, die mehr als ihre Manager verdienen.

```
SELECT
FROM
WHERE
```

- Auswertung
  - Built-in-Funktion (AVG) wird angewendet auf einstellige Ergebnisliste (GEHALT)
  - keine Eliminierung von Duplikaten
  - Verwendung von arithmetischen Ausdrücken ist möglich: AVG (GEHALT/12)

Aggregatfunktionen (2)

- keine Aggregatfunktionen in WHERE-Klausel
- keine geschachtelte Nutzung von Funktionsreferenzen!

Q9: Wie viele Verlage gibt es?

```
SELECT
FROM
```

Q10: An wievielen Orten gibt es Verlage?

```
SELECT
FROM
```

Q11: An welchen Orten gibt es mehr als drei Verlage?

```
SELECT
FROM
WHERE
```

Q12: Welches Buch (Titel, Jahr) ist am ältesten?

```
SELECT
FROM
```

Aggregatfunktionen (3)

- Beispielschema:

```
PERS (PNR, NAME, BERUF, GEHALT, ..., MNR, ANR)
       PRIMARY KEY (PNR), FOREIGN KEY (MNR) REFERENCES PERS
```

Q13a: Wieviele Angestellte haben einen Vorgesetzten ?

```
SELECT
FROM
```

Q13b: Wieviele Angestellte haben keinen Vorgesetzten ?

```
SELECT
FROM
```

Q13c: Wieviele Angestellte sind Vorgesetzte ?

```
SELECT
FROM
```


Partitionierung einer Relation in Gruppen

```
SELECT ... FROM ... [WHERE ...]
[ GROUP BY column-ref-commalist ]
```

- Gruppenbildung auf Relationen: GROUP-BY-Klausel
  - Tupel mit übereinstimmenden Werten für Gruppierungsattribut(e) bilden je eine Gruppe
  - ausgegeben werden können neben Gruppierungsattributen nur Konstanten sowie das Ergebnis von Aggregatfunktionen (-> 1 Satz pro Gruppe)
  - die Aggregatfunktion wird jeweils auf die Tupeln einer Gruppe angewendet

Q14: Liste alle Abteilungen und das Durchschnitts- sowie Spitzengehalt ihrer Angestellten auf.

```
SELECT
FROM
```

Q15: Liste alle Abteilungen (ANR und ANAME) sowie die Anzahl der beschäftigten Angestellten auf

```
SELECT
FROM
WHERE
```

Auswahl von Gruppen (HAVING-Klausel)

```
SELECT ... FROM ... [WHERE ...]
[ GROUP BY column-ref-commalist ]
[ HAVING cond-exp ]
```

- Fragen werden in den folgenden Reihenfolge bearbeitet:
  1. Tupeln werden ausgewählt durch die WHERE-Klausel.
  2. Gruppen werden gebildet durch die GROUP-BY-Klausel.
  3. Gruppen werden ausgewählt, wenn sie die HAVING-Klausel erfüllen

Q16: Für welche Abteilungen zwischen K50 und K60 ist das Durchschnittsalter kleiner als 30?

```
SELECT
FROM
WHERE
```

Q17: Bestimme die Namen und Gehaltssummen der Abteilungen mit mehr als 5 Mitarbeitern

```
SELECT
FROM
```


Suchbedingungen

- Sammlung von Prädikaten
  - Verknüpfung mit AND, OR, NOT
  - Auswertungsreihenfolge ggf. durch Klammern
- nicht-quantifizierte Prädikate:
  - Vergleichsprädikate
  - LIKE-, BETWEEN-, IN-Prädikate
  - Test auf Nullwert
  - UNIQUE-Prädikat: Test auf Eindeutigkeit
  - MATCH-Prädikat: Tupelvergleiche
  - OVERLAPS-Prädikat: Test auf zeitliches Überlappen von DATETIME-Werten
- quantifizierte Prädikate
  - ALL
  - ANY
  - EXISTS

Vergleichsprädikate

comparison-cond ::= row-constructor *q* row-constructor

row-constructor ::= scalar-exp | (scalar-exp-commalist) | (table-exp)

- Skalarer Ausdruck (scalar-exp): Attribut, Konstante bzw. Ausdrücke, die einfachen Wert liefern
- Tabellen-Ausdruck (table-exp) darf hier höchstens 1 Tupel als Ergebnis liefern (Kardinalität 1, Row Sub-Query)
- Vergleiche zwischen Tupel-Konstruktoren (row constructor) mit mehreren Attributen
  - (a1, a2, ... an) = (b1, b2, ...bn) ⇔ a 1 = b1 AND a2 = b2 ... AND an = bn
  - (a1, a2, ... an) < (b1, b2, ...bn) ⇔ (a1 < b1) OR ((a1 = b1) AND (a2 < b2)) OR (...)

```
SELECT

FROM

WHERE
```


LIKE-Prädikate

char-string-exp [NOT] LIKE char-string-exp [ESCAPE char-string-exp]

- Unterstützung der Suche nach Datenstrings, von denen nur Teile bekannt sind (pattern matching).
- LIKE-Prädikat vergleicht einen Datenwert mit einem "Muster" bzw. einer "Maske".
- Aufbau einer Maske mit Hilfe zweier spezieller Symbole

  % bedeutet "null oder mehr beliebige Zeichen"

  _ bedeutet "genau ein beliebiges Zeichen"

  - Das LIKE-Prädikat ist TRUE, wenn der entsprechende Datenwert der aufgebauten Maske mit zulässigen Substitutionen von Zeichen für "%" und " _" entspricht
  - Suche nach "%" und "_" durch Voranstellen eines Escape-Zeichens möglich.

LIKE-Klausel	*wird erfüllt von*
NAME LIKE '%SCHMI%'	
ANR LIKE '_7%'	
NAME NOT LIKE '%-%'	
STRING LIKE '%_%' ESCAPE '\'	

BETWEEN-Prädikate

row-constr[NOT] BETWEENrow-constr AND row-constr

- Semantik

  y BETWEEN x AND z ⇔ x ≤ y AND y ≤ z

  y NOT BETWEEN x AND z ⇔ NOT (y BETWEEN x AND z)

- Beispiel

```
SELECT NAME
FROM   PERS
WHERE  GEHALT BETWEEN 50000 AND 80000
```


IN-Prädikate

```
row-constr[NOT] IN (table-exp) |
scalar-exp [NOT]   IN  (scalar-exp-commalist)
```

- Ein Prädikat in einer *WHERE*-Klausel kann ein Attribut auf Zugehörigkeit zu einer Menge testen:
  - *explizite Mengendefinition:* A_i IN (a_1, a_j, a_k)
  - *implizite Mengendefinition:* A_i IN (SELECT . . .)
- Semantik

  x IN (a, b, . . ., z) ⇔ x = a OR x = b . . . OR x = z

  x NOT IN erg ⇔ NOT (x IN erg)

Q18: Finde die Autoren mit Nachname Maier, Meier oder Müller

```
SELECT *
FROM   autor
WHERE
```

Q19: Finde die Schlagworte, die nicht verwendet wurden

```
SELECT *
FROM   schlagwort
WHERE
```

NULL-Werte

- für jedes Attribut kann Zulässigkeit von Nullwerten festgelegt werden
- unterschiedliche Bedeutungen:
  - Datenwert ist momentan nicht bekannt
  - Attributwert existiert nicht für ein Tupel
- Behandlung von Nullwerten
  - Das Ergebnis einer arithmetischen Operation (+, -, *, /) mit einem NULL-Wert ist ein NULL-Wert
  - Tupel mit NULL-Werten im Verbundattribut nehmen am Verbund nicht teil
  - Auswertung eines NULL-Wertes in einem Vergleichsprädikat mit irgendeinem Wert ist UNKNOWN (?)
- Bei Auswertung von Booleschen Ausdrükken wird 3-wertige Logik eingesetzt
  - Das Ergebnis ? bei der Auswertung einer WHERE-Klausel wird wie FALSE behandelt.

NOT	
T	F
F	T
?	?

AND	T	F	?
T	T	F	?
F	F	F	F
?	?	F	?

OR	T	F	?
T	T	T	T
F	T	F	?
?	T	?	?

- spezielles Prädikat zum Test auf NULL-Werte:

```
row-constr IS [NOT]   NULL
```

```
SELECT  PNR, PNAME
FROM    PERS
WHERE   GEHALT  IS  NULL
```


NULL-Werte: Problemfälle

- 3-wertige Logik führt zu unerwarteten Ergebnissen

  Bsp.: PERS (Alter <= 50) vereinigt mit PERS (Alter > 50)

  ergibt nicht notwendigerweise Gesamtrelation PERS

- Nullwerte werden bei SUM, AVG, MIN, MAX nicht berücksichtigt, während COUNT(*) alle Tupel (inkl. Null-Tupel, Duplikate) zählt.

```
SELECT  AVG  (GEHALT)  FROM  PERS    →  Ergebnis  X

SELECT  SUM  (GEHALT)  FROM  PERS    →  Ergebnis  Y

SELECT  COUNT (*)      FROM  PERS    →  Ergebnis  Z
```

Es gilt nicht notwendigerweise, daß X = Y / Z (-> Verfälschung statistischer Berechnungen)

Quantifizierte Prädikate

- All-or-Any-Prädikat

```
row-constr θ  { ALL | ANY  | SOME}    (table-exp)
```

θ **ALL:** Prädikat wird zu "true" ausgewertet, wenn der θ-Vergleich für alle Ergebniswerte von table-exp "true" ist.

θ **ANY**/ θ **SOME:** analog, wenn der θ-Vergleich für einen Ergebniswert "true" ist.

Q20: Finde die Manager, die mehr verdienen als alle ihre Angestellten

```
SELECT
FROM
WHERE
```

```
Q22b: Finde die Manager, die weniger als einer ihrer Angestellten verdienen
```


Existenztests

```
[ NOT]  EXISTS  (table-exp)
```

- das Prädikat wird zu "false" ausgewertet, wenn table-exp auf die leere Menge führt, sonst "true"
- Im EXISTS-Kontext darf table-exp mit (SELECT * ...) spezifiziert werden (Normalfall)
- Semantik

x θ ANY (SELECT y FROM T WHERE p) ⇔ EXISTS (SELECT * FROM T WHERE (p) AND (x θ T.y))

x θ ALL (SELECT y FROM T WHERE p) ⇔
NOT EXISTS (SELECT * FROM T WHERE (p) AND NOT (x θ T.y))

Q21: Finde die Manager, die mehr verdienen als alle ihre Angestellten.

```
SELECT
FROM
WHERE
```

Existenztests (2)

Q22: Finde die Schlagworte, die für mindestens ein (... kein) Buch vergeben wurden

```
SELECT S.*
FROM   schlagwort S
WHERE
```

Q23: Finde die Bücher, die alle Schlagworte des Buchs mit der ID 3545 abdecken

(andere Formulierung: Finde die Bücher, zu denen kein Schlagwort "existiert", das nicht auch für Buch 3545 existiert).

```
SELECT B.titel
FROM   buch B
WHERE
```


Skalare Funktionen (Auswahl)

- CASE

```
SELECT MATNR,  PUNKTE,
       CASE WHEN PUNKTE > 90 THEN 'Sehr gut'
            WHEN PUNKTE > 75 THEN 'Gut'
            WHEN PUNKTE > 60 THEN  'O.K.'
            ELSE  'SCHLECHT'     END  AS NOTE
FROM ...
```

  - fehlender ELSE-Zweig: NULL-Wert für sonstige Fälle

- String-Funktionen
  - || (String-Konkatenation), CHAR_LENGTH. BIT_LENGTH
  - SUBSTRING *Bsp.:* SUBSTRING (NAME FROM 1 FOR 20)
  - TRANSLATE, CONVERT
  - POSITION, LOWER, UPPER
  - TRIM *Bsp.:* TRIM (TRAILING ' ' FROM NAME)

- Verschiedene Funktionen
  - USER, CURRENT_USER, SESSION_USER, SYSTEM_USER
  - CURRENT_TIME, CURRENT_DATE, CURRENT_TIMESTAMP
  - EXTRACT (Herausziehen von YEAR, MONTH, ... aus Datum)
  - CAST (Typkonversionen) *Bsp.:* CAST ('2002-04-24' AS DATE)
  - NULLIF, COALESCE, ...

Einsatz mengentheoretischer Operatoren

- Vereinigung (UNION), Durchschnitts- (INTERSECT) und Differenzbildung (EXCEPT) von Relationen bzw. Query-Ergebnissen

```
table-exp ::= nonjoin-table-exp | join-table-exp
nonjoin-table-exp ::= nonjoin-table-term | table-exp {UNION | EXCEPT}
          [ALL][CORRESPONDING  [BY (column-commalist)]] table-term
nonjoin-table-term ::= nonjoin-table-primary | table-term INTERSECT
          [ALL][CORRESPONDING  [BY (column-commalist)]] table-primary
table-term ::= nonjoin-table-term | join-table-exp
table-primary ::= nonjoin-table-primary | join-table-primary
nonjoin-table-primary ::= simple-table | select-exp |(nonjoin-table-exp)
```

- vor Ausführung werden Duplikate aus den Operanden entfernt, außer wenn ALL spezifiziert ist.
- für die Operanden wird Vereinigungsverträglichkeit gefordert
- Abschwächung:
  - *CORRESPONDING BY (A1, A2, ...An)*: Operation ist auf Attribute Ai beschränkt, die in beiden Relationen vorkommen müssen (-> n-stelliges Ergebnis)
  - *CORRESPONDING*: Operation ist auf gemeinsame Attribute beschränkt

Q24: Welche Schlagworte wurden nie verwendet ? (Q19, Q22)

Weitergehende Verwendung von Sub-Queries (table expressions)

- 3 Arten von Sub-Queries
  - Table Sub-Queries (mengenwertige Ergebnisse)
  - Row Sub-Queries (Tupel-Ergebnis)
  - Skalare Sub-Queries (atomarer Wert; Kardinalität 1, Grad 1)
- Im SQL-Standard können Table-Sub-Queries überall stehen, wo ein Relationenname möglich ist, insbesondere in der FROM-Klausel.

```
SELECT  ANAME
FROM    (Select ANR, Sum (GEHALT) AS GSUMME FROM PERS GROUP BY ANR)
        AS GSUM JOIN ABT USING (ANR)
WHERE GSUMME > 100000
```

- Skalare Sub-Queries können auch in SELECT-Klausel stehen.

```
SELECT  P.PNAME,  (SELECT  A.ANAME  FROM  ABT  A
                   WHERE  A.ANR = P.ANR) AS  ABTEILUNG
FROM  PERS  P
WHERE  BERUF  = "Programmierer"
```

Relationenalgebra vs. SQL (Retrieval)

Relationenalgebra	SQL
$\pi_{A1, A2, \ldots Ak}(R)$	
$\sigma_{P}(R)$	
$R \times S$	
$R \underset{R.A\ \theta\ S.B}{\bowtie} S$	
$R \bowtie S$	
$R \overline{\bowtie} S$	
$R \boxtimes S$	
$R \cup S$	
$R \cap S$	
$R - S$	
$R \div S$	

Zusammenfassung

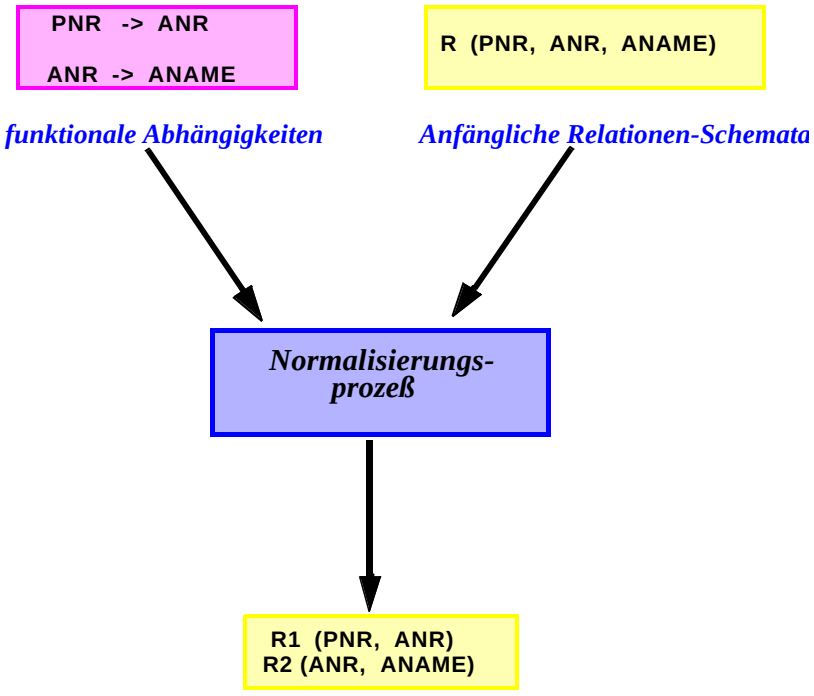
- SQL wesentlich mächtiger als Relationenalgebra
- Hauptanweisung: SELECT
  - Projektion, Selektion, Joins
  - Aggregatfunktionen
  - Gruppenbildung (Partitionierungen)
  - quantifizierte Anfragen
  - Unteranfragen (einfache und korrelierte Sub-Queries)
  - allgemeine Mengenoperationen UNION, INTERSECT, EXCEPT
- hohe Sprachredundanz
- SQL-Implementierungen weichen teilweise erheblich von Standard ab (Beschränkungen / Erweiterungen)

5. Logischer DB-Entwurf (Normalisierung)

- Ziel: Theoretische Grundlage für den Entwurf eines "guten" relationalen DB-Schemas (→ Entwurfstheorie, Normalisierungslehre)
- Güte:
  - leichte Handhabbarkeit, Übersichtlichkeit, ...
  - Entwurfstheorie präzisiert/formalisiert "Güte" zum Teil
- Was macht einen schlechten DB-Schema-Entwurf aus?
  - implizite Darstellung von Informationen
  - Redundanzen
  - Potentielle Inkonsistenz (Änderungsanomalien)
  - Einfügeanomalien
  - Löschanomalien ...

  oft hervorgerufen durch "Vermischung" von Entities, Zerlegung und wiederholte Speicherung von Entities, ...
- Normalisierung von Relationen hilft einen gegebenen Entwurf zu verbessern.

Normalisierung von Relationen (Bsp.)



Definitionen und Begriffe

- Konventionen

R, S	Relationenschemata (Relationenname, Attribute)	W, X, Y, Z,...	Mengen von Attributen
R, S	Relationen der Relationenschemata **R, S**	XY ≡ X ∪ Y	Mengen brauchen nicht disjunkt zu sein
A, B, C,...	einfache Attribute	a, b, c	Werte einfacher Attribute
A = {A_1,...,A_n}	Attributmenge eines Relationenschemas	x, y, z	Werte von X, Y, Z

- Def.: ***Funktionale Abhängigkeit (FA)***

  Die FA X → Y gilt (X bestimmt Y funktional), wenn für alle R von **R** gilt:
  zwei Tupel, deren Komponenten in X übereinstimmen, stimmen auch in Y überein.
  ∀u ∈ R ∀v ∈ R (u[X] = v[X]) ⇒ (u[Y] = v[Y])

  alternativ: Die Relation R erfüllt die FA X → Y, wenn für jeden X-Wert x der Ausdruck
  $\pi_Y(\sigma_{X=x}(R))$ höchstens ein Tupel hat.

Graphische Notation:

Definitionen und Begriffe (2)

- Eigenschaften:
  - triviale FA: X → X
  - falls X Schlüsselkandidat von **R**, dann gilt für alle Y aus **R**: X → Y
  - FA beschreiben semantische Integritätsbedingungen bezüglich der Attribute eines Relationen*schemas*, die jederzeit erfüllt sein müssen

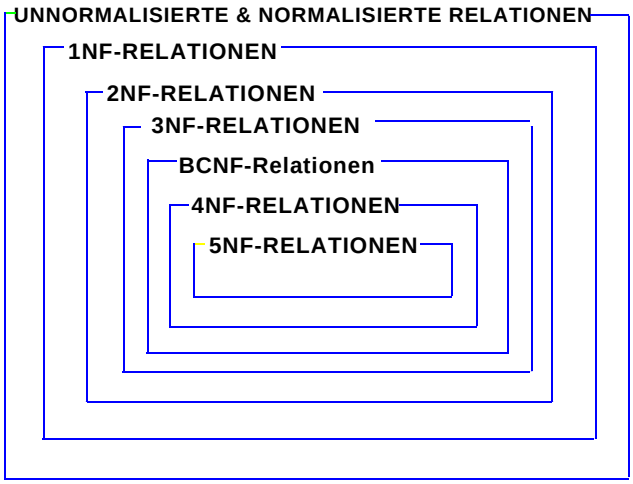
- Volle funktionale vs. partielle Abhängigkeit:

  Sei $A_1, A_2, ..., A_n \rightarrow B_1, B_2, ..., B_m$

  $B = \{B_1, B_2, ..., B_m\}$ ist ***voll funktional abhängig*** von $A = \{A_1, A_2, ..., A_n\}$, wenn B funktional abhängig von A ist, aber nicht funktional abhängig von einer echten Teilmenge von A ist.

  A → B ist eine ***partielle Abhängigkeit***, wenn ein Attribut A_i in A existiert, so daß
  $(A - \{A_i\}) \rightarrow B$ gilt.

Normalisierung von Relationen



- Zerlegung eines Relationenschemas R in höhere Normalformen
  - Beseitigung von Anomalien bei Änderungsoperationen
  - Erhaltung aller nicht-redundanter Funktionalabhängigkeiten von R ($\rightarrow$ sie bestimmen den Informationsgehalt von R)
  - fortgesetzte Anwendung der Projektion im Zerlegungsprozeß
  - Gewährleistung der Rekonstruktion von R durch verlustfreie Verbunde
  - bessere "Lesbarkeit" der aus R gewonnenen Relationen

Normalisierung von Relationen (2)

- Unnormalisierte Relation: Non-First Normal-Form ($NF^{2)}$

Prüfungsgeschehen

PNR	PNAME	FACH	STUDENT (MATNR, NAME, ...)
3678	Rahm	DBS	196481 Maier ... 123766 Coy ... 900550 Schmitt ...
1234	Gerber	TI	654711 Abel ... 123766 Coy ...

Anomalien, z.B.:

- **Insert** Student
- **Delete** Prof
- **Update** Student

  - enthält “Attribute”, die wiederum Relationen sind
  - Darstellung von komplexen Objekten (hierarchische Sichten, Clusterbildung)

- Nachteile:
  - Unsymmetrie (nur eine Richtung der Beziehung)
  - implizite Darstellung von Information
  - Redundanzen bei (n:m)-Beziehungen
  - Anomalien bei Aktualisierung

- Normalisierung:
  - “Herunterkopieren” von Werten führt hohen Grad an Redundanz ein $\rightarrow$ Zerlegung von Relationen
  - aber: Erhaltung ihres Informationsgehaltes

Überführung in 1NF

- Unnormalisierte Relation

  Prüfungsgeschehen **(PNR, PNAME, FACH, STUDENT)**

  (MATNR, NAME, GEB, ADR, FNR, FNAME, DEKAN, PDAT, NOTE)

  STUDENT = Wiederholungsgruppe mit 9 einfachen Attributen (untergeordnete Relation)

- Normalisierung (=> 1NF):
  1. Starte mit der übergeordneten Relation (Vaterrelation).
  2. Nimm ihren Primärschlüssel und erweitere jede unmittelbar untergeordnete Relation damit zu einer selbständigen Relation.
  3. Streiche alle nicht-einfachen Attribute (untergeordnete Relationen) aus der Vaterrelation.
  4. Wiederhole diesen Prozeß ggf. rekursiv.

- Regeln:
  - Nicht-einfache Attribute bilden neue Relationen.
  - Primärschlüssel der übergeordneten wird an untergeordnete Relation angehängt ('copy down the key')

- Relationenschema in 1NF

  PRÜFER (PNR, PNAME, FACH)

  PRÜFUNG (PNR, MATNR, NAME, GEB, ADR, FNR, FNAME, DEKAN, PDAT, NOTE)

Überführung in 2NF

- 1NF verursacht immer noch viele Änderungsanomalien, da verschiedene Entity-Mengen in einer Relation gespeichert werden können bzw. aufgrund von Redundanz innerhalb einer Relation (Bsp.: PRÜFUNG)

- 2NF vermeidet einige der Anomalien dadurch, indem nicht voll funktional (partiell) abhängige Attribute eliminiert werden.

  ⇒ Separierung verschiedener Entity-Mengen in eigene Relationen

- Definitionen:

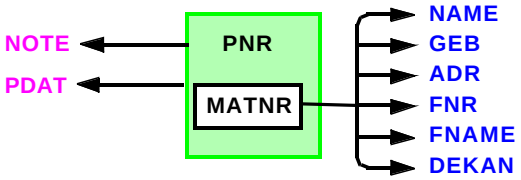
  Ein ***Primärattribut*** (Schlüsselattribut) eines Relationenschemas ist ein Attribut, das zu mindestens einem Schlüsselkandidaten des Schemas gehört.

  Ein Relationenschema **R** ist in ***2NF***, wenn es in 1NF ist und jedes Nicht-Primärattribut von **R** voll funktional von jedem Schlüsselkandidaten in **R** abhängt.

- Überführung in 2NF:
  1. Bestimme funktionale Abhängigkeiten zwischen Nicht-Primärattributen und Schlüsselkandidaten
  2. Eliminiere partiell abhängige Attribute und fasse sie in eigener Relation zusammen (unter Hinzunahme der zugehörigen Primärattribute).

Überführung in 2NF (2)

- Voll funktionale Abhängigkeiten in PRÜFUNG



- Relationenschema in 2NF

Prüfung'

PNR	MATNR	PDAT	NOTE
1234	123 766	22.10.00	4
1234	654 711	14.02.01	3
3678	196 481	21.09.01	2
3678	123 766	02.03.01	4
8223	226 302	12.07.01	1

Prüfer

PNR	PNAME	FACH
1234	Gerber	TI
3678	Rahm	DBS
8223	Ehrenberg	WI

Student'

MATNR	NAME	GEB	ADR	FNR	FNAME	DEKAN
123 766	Coy	05.05.79	XX	F11	Wirtschaftswissenschaften	A
654 711	Abel	21.11.78	XY	F19	Mathematik/Informatik	B
196 481	Maier	01.01.80	YX	F19	Mathematik/Informatik	B
226 302	Schulz	31.07.79	YY	F11	Wirtschaftswissenschaften	A

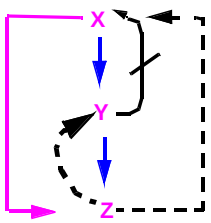
Überführung in 3NF

- Änderungsanomalien in 2NF sind immer noch möglich aufgrund von transitiven Abhängigkeiten.
  - Beispiel: Vermischung von Fakultäts- und Studentendaten in Student'
- Definitionen:

  Eine Attributmenge Z von Relationenschema **R** ist ***transitiv abhängig*** von einer Attributmenge X in **R**, wenn gilt:

  - X und Z sind disjunkt
  - es existiert eine Attributmenge Y in R, so daß gilt:

$$X \rightarrow Y,\ Y \rightarrow Z,\ Y \not\rightarrow X,\ Z \notin Y$$



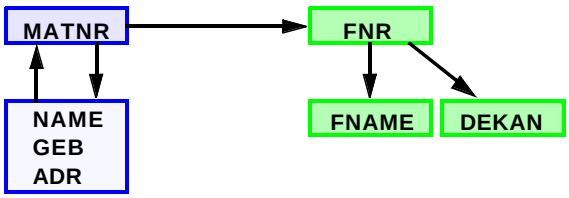
$Z \rightarrow Y$ zulässig

strikte Transitivität: $Z \not\rightarrow Y$

Ein Relationenschema **R** befindet sich in ***3NF,*** wenn es sich in 2NF befindet und jedes Nicht-Primärattribut von **R** von keinem Schlüsselkandidaten von **R** transitiv abhängig ist.

Überführung in 3NF (2)

- Funktionale Abhängigkeiten in STUDENT'



- Relationenschema in 3NF

Prüfung'

PNR	MATNR	PDAT	NOTE
1234	123 766	22.10.00	4
1234	654 711	14.02.01	3
3678	196 481	21.09.01	2
3678	123 766	02.03.01	4
8223	226 302	12.07.01	1

Fakultät

FNR	FNAME	DEKAN
F11	Wirtschaftswissenschaften	A
F12	Medizin	C
F19	Mathematik/Informatik	B

Prüfer

PNR	PNAME	FACH
1234	Gerber	TI
3678	Rahm	DBS
8223	Ehrenberg	WI

Student''

MATNR	NAME	GEB	ADR	FNR
123 766	Coy	05.05.79	XX	F11
654 711	Abel	21.11.78	XY	F19
196 481	Maier	01.01.80	YX	F19
226 302	Schulz	31.07.79	YY	F11

Boyce/Codd-Normalform (BCNF)

- Definition der 3NF hat gewisse Schwächen bei Relationen mit mehreren, *sich überlappenden* Schlüsselkandidaten
- Beispiel:

```
PRÜFUNG (PNR, MATNR, FACH, NOTE)
   PRIMARY KEY (PNR,MATNR),
   UNIQUE (MATNR,FACH)
```

  - es bestehe eine (1:1)-Beziehung zwischen PNR und FACH
  - einziges Nicht-Primärattribut: NOTE
    ⇒ PRÜFUNG ist in 3NF
  - jedoch Änderungsanomalien, z.B. bei FACH

PNR	MATNR	Fach	NOTE
45	1234	Betriebssysteme	1
45	4711	Betriebssysteme	3
45	5678	Betriebssysteme	2
56	1234	Technische Informatik	4

- Ziel: Beseitigung der Anomalien für Primärattribute
- Definition: Ein Attribut (oder eine Gruppe von Attributen), von dem andere voll funktional abhängen, heißt ***Determinant***.
- Welches sind die Determinanten in PRÜFUNG?

Boyce/Codd-Normalform (2)

- Definition:

  Ein Relationenschema **R** ist in ***BCNF***, wenn jeder Determinant ein Schlüsselkandidat von **R** ist.

- Formale Definition:

  Ein Relationenschema ist in ***BCNF***, falls gilt: Wenn eine Sammlung von Attributen Y (voll funktional) abhängt von einer disjunkten Sammlung von Attributen X, dann hängt jede andere Sammlung von Attributen Z auch von X (voll funktional) ab.

  D. h. für alle X, Y, Z mit X und Y disjunkt gilt: $X \rightarrow Y$ impliziert $X \rightarrow Z$

- Zerlegung von Prüfung

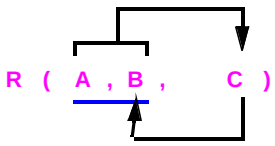
  PRÜF (PNR, MATNR, NOTE) oder PRÜF2 (MATNR, FACH, NOTE)

  FBEZ (PNR, FACH) FBEZ (PNR, FACH)

- Beide Zerlegungen führen auf BCNF-Relationen
  - Änderungsanomalie ist verschwunden
  - alle funktionalen Abhängigkeiten sind erhalten

Boyce/Codd-Normalform (3)

- Sind BCNF-Zerlegungen immer sinnvoll?



ist in 3NF, weil B Primärattribut ist !

- Beispiel:
  STUDENT, FACH $\rightarrow$ PRÜFER,
  PRÜFER $\rightarrow$ FACH
  - Jeder Prüfer prüft nur ein Fach (aber ein Fach kann von mehreren geprüft werden)
  - Jeder Student legt in einem bestimmten Fach nur eine Prüfung ab

SFP

STUDENT	FACH	PRÜFER
Sloppy	DBS	Bachmann
Hazy	DBS	Rahm
Sloppy	TI	Gerber

- Wie sieht die BCNF-Zerlegung aus?

- Neue Probleme: STUDENT, FACH -> PRÜFER ist nun "extern" (Konsistenzprüfung?)

- BCNF hier zu streng, um bei der Zerlegung alle funktionalen Abhängigkeiten zu bewahren (key breaking dependency)

Probleme der Normalisierung

- Weitestgehende Zerlegung nicht immer sinnvoll
- Beispiel:

  Relation PERS (PNR, PLZ, ORT)

  mit FA PLZ $\rightarrow$ ORT

- Normalisierung verlangt Zerlegung in

  PERS' (PNR, PLZ)

  R2 (PLZ, ORT)

  - Änderungshäufigkeit?
  - Aufsuchen der Adresse (Verbundoperation) !
  - Sind ORT oder PLZ in diesem Kontext eigenständige Entities? (als Kandidaten für eigene Relation in 3NF)

  => besser PERS in 2NF!

Zusammenfassung

- Normalisierung von Relationen
  - arbeitet auf existierenden Datenstrukturen
  - Ziel: guter DB-Entwurf, so daß durch einen Satztyp (Relation) nur ein Objekttyp beschrieben wird
  - Eliminierung von Änderungsanomalien
  - wachsender Informationsgehalt mit zunehmender Normalisierung
- Funktionale Abhängigkeit: n:1-Beziehung zwischen zwei Attributmengen einer Relation
  - Festlegung aller FA unterstützt präzises Denken beim Entwurf
  - erlaubt Integritätskontrollen durch das DBS
- schrittweise Normalisierung:
  - 1NF: normalisierte Relationen (einfache Attribute)
  - 2NF: keine partiellen (funktionalen) Abhängigkeiten
  - 3NF: keine transitiven Abhängigkeiten (jedes Nicht-Primärattribut ist direkt von jedem SK abhängig)
  - BCNF: jeder Determinant ist Schlüsselkandidat
  - 3NF meist ausreichend
- Überarbeitung des DB-Schemas: Stabilitätsgesichtspunkte/Änderungshäufigkeiten können schwächere Normalformen erzwingen

6. Datenmanipulation / -definition / -kontrolle in SQL

- Datenmanipulation: INSERT, UPDATE, DELETE von Tupeln
- Datendefinition
  - Datentypen, Domains
  - Erzeugen von Tabellen
  - Ändern/Löschen von Tabellen
- Sichtkonzept (Views)
- Integritätsbedingungen und Trigger
  - Klassifikation von Integritätsbedingungen
  - Integritätsregeln / Trigger
  - Einsatzformen von Triggern
- Zugriffskontrolle/Autorisierung: GRANT, REVOKE
- Zugriff auf Metadaten

Einfügen von Tupeln (INSERT)

```
INSERT INTO table   [ (column-commalist) ]
     { VALUES row-constr-commalist |  table-exp |  DEFAULT VALUES }
```

- Satzweises Einfügen (direkte Angabe der Attributwerte)

```
Bsp.: Füge den Schauspieler Garfield mit der PNR 4711 ein
```

  - alle nicht angesprochenen Attribute erhalten Nullwerte
  - falls alle Werte in der richtigen Reihenfolge versorgt werden, kann Attributliste entfallen
  - Integritätsbedingungen müssen erfüllt werden
- mengenorientiertes Einfügen: einzufügende Tupeln werden aus einer anderen Relation mit Hilfe einer SELECT-Anweisung ausgewählt

```
Bsp.: Füge die Schauspieler aus L in die Relation TEMP ein
```

  - (leere) Relation TEMP mit kompatiblen Attributen sei vorhanden
  - die spezifizierte Tupelmenge wird ausgewählt und in die Zielrelation kopiert
  - Die eingefügten Tupel sind unabhängig von denen, von denen sie abgeleitet/kopiert wurden.

Ändern von Tupeln (UPDATE)

```
searched-update ::= UPDATE table SET update-assignment-commalist
                        [WHERE cond-exp]
update-assignment ::= column = {scalar-exp | DEFAULT | NULL }
```

Gib den Schauspielern, die am Schauspielhaus spielen, eine Gehaltserhöhung von 2% (Attribute GEHALT und THEATER seien in SCHAUSPIELER).

Erhöhe das Gehalt der Schauspieler des Schauspielhauses um 2%, das der Schauspieler der Oper um 2.5%.

Löschen von Tupeln (DELETE)

```
searched-delete ::= DELETE FROM table [WHERE cond-exp]
```

- Der Aufbau der WHERE-Klausel entspricht dem in der SELECT-Anweisung.

Lösche den Schauspieler mit der PNR 4711

Lösche alle Schauspieler, die nie gespielt haben.

Datendefinition in SQL

- SQL-Umgebung (Environment) besteht aus:
  - Katalogen
  - Benutzern (authorization identifiers)
- ein Katalog enthält:
  - für jede Datenbank ein Schema
  - ein INFORMATION_SCHEMA (Metadaten über alle Schemata)
    => dreiteilige Objektnamen: <catalog>.<schema>.<object>
- Schema-Definition

```
CREATE SCHEMA  [schema] AUTHORIZATION user
      [DEFAULT CHARACTER SET char-set]
      [schema-element-list]
```

  - jedes Schema ist einem Benutzer (user) zugeordnet, z.B. DBA
  - Schema erhält Benutzernamen, falls keine explizite Namensangabe erfolgt
  - Definition aller Definitionsbereiche, Basisrelationen, Sichten (Views), Integritätsbedingungen und Zugriffsrechte
- Beispiel: `CREATE SCHEMA FLUG-DB AUTHORIZATION LH_DBA1`

Datentypen

- String-Datentypen

```
CHARACTER [ ( length ) ]                    (Abkürzung: CHAR)
CHARACTER VARYING  [ ( length ) ]           (Abkürzung: VARCHAR)
NATIONAL CHARACTER [ ( length ) ]           (Abkürzung: NCHAR)
NCHAR VARYING  [ ( length ) ]
BIT [ ( length ) ]
BIT VARYING  [ ( length ) ]
```

- Numerische Datentypen

```
NUMERIC [ ( precision [ , scale] ) ]
DECIMAL [ ( precision [ , scale ] ) ]      (Abkürzung: DEC)
INTEGER                                    (Abkürzung: INT)
SMALLINT
FLOAT [ ( precision ) ]
REAL
DOUBLE PRECISION
```

- Datums-/Zeitangaben (Datetimes)

```
DATE
TIME
TIMESTAMP
TIME WITH TIME ZONE
TIMESTAMP WITH TIME ZONE
INTERVAL             (* Datums- und Zeitintervalle *)
```

Definitionsbereiche (Domains)

- Festlegung zulässiger Werte durch Domain-Konzept

```
CREATE DOMAIN  domain [AS] data-type
    [DEFAULT  { literal |  niladic-function-ref | NULL} ]
    [ [CONSTRAINT constraint] CHECK (cond-exp) [deferrability]]
```

- optionale Angabe von Default-Werten
- Wertebereichseingrenzung durch benamte CHECK-Constraint möglich
- Beispiele:

```
CREATE DOMAIN  ABTNR  AS  CHAR (6)
CREATE DOMAIN  ALTER AS INT DEFAULT NULL CHECK(VALUE=NULL OR  VALUE > 18)
```

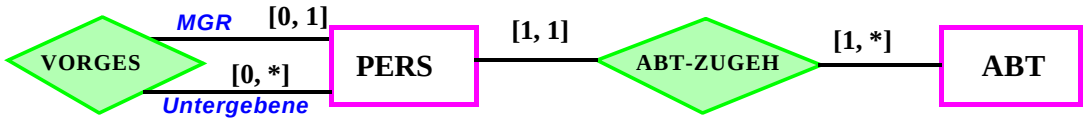
- Beschränkungen
  - keine echten benutzerdefinierten Datentypen
  - keine strenge Typprüfung
  - Domains können nur bzgl. Standard-Datentypen (nicht über andere Domains) definiert werden

Erzeugung von Basisrelationen

```
CREATE [ [GLOBAL | LOCAL] TEMPORARY] TABLE base-table
       (base-table-element-commalist)
      [ON COMMIT {DELETE | PRESERVE} ROWS]

base-table-element::= column-def  | base-table-constraint-def
```

- permanente und temporäre Relationen
- zwei Typen von temporären Relationen:
  - LOCAL: Lebensdauer auf erzeugende Transaktion begrenzt
  - GLOBAL: Lebensdauer auf "Session" eines Benutzers begrenzt; Inhalt kann beim Commit zurückgesetzt werden
- Bei der Attributdefinition (column definition) werden folgende Angaben spezifiziert:
  - Attributname sowie Datentyp bzw. Domain
  - Eindeutigkeit (UNIQUE bzw. PRIMARY KEY), FOREIGN-KEY-Klausel
  - Verbot von Nullwerten (NOT NULL), CHECK-Bedingung, Default-Werte
- Integritätsbedingungen, die mehrere Attribute der Relation betreffen, können als eigene "table constraint" definiert werden

CREATE TABLE: Beispiel




```
CREATE   TABLE   PERS
    (PNR        INT       PRIMARY KEY,
    BERUF       VARCHAR (50),
    PNAME       VARCHAR (50)NOT NULL,
    PALTER      ALTER,    (* siehe Domain-Definition *)
    MGR         INT       REFERENCES  PERS,
    ANR         ABTNR                    (*  Domain-Definition *)
    GEHALT      DEC (7)   DEFAULT  0  CHECK (VALUE < 120000)
    FOREIGN  KEY (ANR)   REFERENCES  ABT )

CREATE   TABLE   ABT
    (ANR        ABTNR     PRIMARY KEY,
     ANAME      VARCHAR (50)NOT NULL  )
```


Dynamische Änderung einer Relation

- Bei Relationen können dynamisch (während ihrer Lebenszeit) Schemaänderungen durchgeführt werden.
  - Hinzufügen, Ändern und Löschen von Attributen
  - Hinzufügen und Löschen von Constraints

```
ALTER  TABLE  base-table
  {   ADD [COLUMN] column-def
    |  ALTER [COLUMN] column {SET default-def  | DROP DEFAULT}
    |  DROP [COLUMN] column {RESTRICT | CASCADE}
    |  ADD  base-table-constraint-def
    |  DROP  CONSTRAINT constraint {RESTRICT | CASCADE}}
```

- Beispiele

```
ALTER TABLE PERS ADD  SVNR INT UNIQUE
ALTER TABLE PERS DROP COLUMN SVNR RESTRICT
```

  - Wenn das Attribut das einzige der Relation ist, wird die Operation zurückgewiesen
  - RESTRICT führt zur Rückweisung der Operation, wenn das Attribut (Column) in einer Sicht oder einer Integritätsbedingung (Check) referenziert wird
  - CASCADE dagegen erzwingt die Folgelöschung aller Sichten und Check-Klauseln, die von dem Attribut abhängen.

Löschen von Objekten

```
DROP { TABLE base-table | VIEW view  | DOMAIN domain | SCHEMA schema }
          {RESTRICT | CASCADE}
```

- Falls Objekte (Relationen, Sichten, ...) nicht mehr benötigt werden, können sie durch die DROP-Anweisung aus dem System entfernt werden
- Mit der CASCADE-Option können 'abhängige' Objekte (z.B. Sichten auf Relationen oder anderen Sichten) mitentfernt werden
- RESTRICT verhindert Löschen, wenn die zu löschende Relation noch durch Sichten oder Integritätsbedingungen referenziert wird
- Beispiele:

```
DROP   TABLE   PERS RESTRICT
```

PersConstraint sei definiert auf PERS:

```
1. ALTER TABLE PERS DROP CONSTRAINT  PersConstraint  CASCADE
2. DROP TABLE  PERS  RESTRICT
```


Sichtkonzept

- Sicht (View): mit Namen bezeichnete, aus Basisrelationen abgeleitete, virtuelle Relation (Anfrage)
- Korrespondenz zum externen Schema bei ANSI/SPARC (Benutzer sieht jedoch i.a. mehrere Views und Basisrelationen)

```
CREATE VIEW view [ (column-commalist ) ] AS table-exp
      [WITH [ CASCADED | LOCAL] CHECK OPTION]
```

- Beispiel: Sicht auf PERS, die alle Programmierer mit einem Gehalt unter 30000 umfaßt

```
CREATE  VIEW  ARME_PROGRAMMIERER (PNR, NAME, BERUF, GEHALT, ANR) AS
                SELECT PNR, NAME, BERUF, GEHALT, ANR FROM  PERS
                WHERE  BERUF = 'Programmierer' AND  GEHALT <  30 000
```

- Sicht kann wie eine Relation behandelt werden (Sichten auf Sichten sind möglich)
- Vorteile:
  - Erhöhung der Benutzerfreundlichkeit
  - erhöhte Datenunabhängigkeit
  - Datenschutz / Zugriffskontrolle

Sichtkonzept (2)

- Sichtsemantik
  - allgemeine Sichten werden nicht materialisiert, sondern als Anfrageergebnis interpretiert, das dynamisch beim Zugriff generiert wird
  - Sicht entspricht einem “dynamisches Fenster” auf zugrundeliegenden Basisrelationen
  - Sicht-Operationen müssen durch (interne) Query-Umformulierung auf Basisrelationen abgebildet werden
  - eingeschränkte Änderungen: aktualisierbare und nicht-aktualisierbare Sichten

- Sonderform: *Materialisierte Sichten*
  - physische Speicherung des Anfrageergebnisses
  - unterstützt schnelleren Lesezugriff
  - Notwendigkeit der Aktualisierung (automatisch durch das DBS)
  - erhöhter Speicherbedarf
  - kein Bestandteil von SQL92, jedoch in vielen DBS verfügbar (CREATE MATERIALIZED VIEW ...)

Sichtkonzept (3)

- Abbildung von Sicht-Operationen auf Basisrelationen
  - Sichten werden i.a. nicht explizit und permanent gespeichert, sondern Sicht-Operationen werden in äquivalente Operationen auf Basisrelationen umgesetzt
  - Umsetzung ist für Leseoperationen meist unproblematisch

```
SELECT  NAME, GEHALT
FROM  ARME_PROGRAMMIERER
WHERE ANR = "K55"
```

- Abbildungsprozeß auch über mehrere Stufen durchführbar

```
CREATE  VIEW  V  AS  SELECT  ... FROM  R  WHERE  P
CREATE  VIEW  W  AS  SELECT  ... FROM V  WHERE  Q

SELECT ... FROM  W  WHERE  C
```

- Einschränkungen der Abbildungsmächtigkeit
  - keine Schachtelung von Aggregatfunktionen und Gruppenbildung (GROUP-BY)
  - keine Aggregatfunktionen in WHERE-Klausel möglich

```
CREATE VIEW ABTINFO (ANR, GSUMME)ASSELECT AVG(GSUMME)FROM ABTINFO
   SELECT ANR,SUM(GEHALT)
   FROM PERS
   GROUP BY ANR
```

Sichtkonzept (4)

- Probleme für Änderungsoperationen auf Sichten
  - erfordern, daß zu jedem Tupel der Sicht zugrundeliegende Tupel der Basisrelationen eindeutig identifizierbar sind
  - Sichten auf einer Basisrelation sind nur aktualisierbar, wenn der Primärschlüssel in der Sicht enthalten ist.
  - Sichten, die über Aggregatfunktionen oder Gruppenbildung definiert sind, sind nicht aktualisierbar
  - Sichten über mehr als eine Relation sind im allgemeinen nicht aktualisierbar

```
CREATE VIEW READONLY (BERUF, GEHALT) AS
  SELECT BERUF, GEHALT  FROM PERS
```

- CHECK-Option:
  - Einfügungen und Änderungen müssen das die Sicht definierende Prädikat erfüllen. Sonst: Zurückweisung
  - nur auf aktualisierbaren Sichten definierbar
- Löschen von Sichten:

```
DROP VIEW  ARME_PROGRAMMIERER CASCADE
```


Datenkontrolle

- Zugriffskontrolle
  - Maßnahmen zur Datensicherheit und zum Datenschutz
  - Sichtkonzept
  - Vergabe und Kontrolle von Zugriffsrechten
- Integritätskontrolle
  - *Semantische Integritätskontrolle*
  - Einhaltung der *physischen Integrität* sowie der *Ablaufintegrität* (operationale Integrität)
- Transaktionskonzept (ACID-Eigenschaften)
  - Verarbeitungsklammer für die Einhaltung von semantischen Integritätsbedingungen
  - im SQL-Standard: COMMIT WORK, ROLLBACK WORK, Beginn einer Transaktion implizit
  - Verdeckung der Nebenläufigkeit (concurrency isolation)
  - Verdeckung von (erwarteten) Fehlerfällen (-> Logging und Recovery)

Art der Integrität	Transaktionseigenschaft	realisierende DBS-Komponente
Semantische Integrität	C (Consistency, Konsistenz)	Integritätskontrolle
Physische Integrität	D: Dauerhaftigkeit A: Atomarität	Logging, Recovery (+ korrekte Implementierung der DB-Operationen)
Ablaufintegrität	I (Isolation)	Synchronisation (z.B. Sperrverwaltung)

Semantische Integritätsbedingungen

- Ziele
  - nur 'sinnvolle' und 'zulässige' Änderungen der DB
  - möglichst hohe Übereinstimmung von DB-Inhalt und Miniwelt (Datenqualität)
- bei COMMIT müssen alle semantischen Integritätsbedingungen erfüllt sein (Transaktionskonsistenz)
- zentrale Überwachung von semantischen Integritätsbedingungen durch DBS ("system enforced integrity") anstelle durch Anwendungen
  - größere Sicherheit
  - vereinfachte Anwendungserstellung
  - leichtere Änderbarkeit von Integritätsbedingungen
  - Leistungsvorteile
  - Unterstützung von interaktiven sowie programmierten DB-Änderungen
- Integritätsbedingungen der Miniwelt sind explizit bekannt zu machen, um automatische Überwachung zu ermöglichen
  - zunächst schwache Unterstützung in SQL (z.B. NOT NULL, UNIQUE)
  - erst seit SQL92 umfangreiche Spezifikationsmöglichkeiten (relationale Invarianten, benutzerdefinierte Integritätsbedingungen)

Klassifikation von Integritätsbedingungen

- modellinhärente vs. anwendungsspezifische Bedingungen
  - modellinhärente Integritätsbedingungen, z.B. Relationale Invarianten (Primärschlüsselbedingung, referentielle Integrität) und Wertebereichsbeschränkung für Attribute
- Reichweite

Reichweite	Beispiele
Attribut	GEB-JAHR ist numerisch, 4-stellig
Satzausprägung	ABT.GEHALTSSUMME < ABT.JAHRESETAT
Satztyp	PNR ist eindeutig
mehrere Satztypen	ABT.GEHALTSSUMME ist Summe aller Angestelltengehälter

- Zeitpunkt der Überprüfbarkeit
  - unverzögert (sofort bei Änderungsoperation)
  - verzögert (am Transaktionsende)
- Art der Überprüfbarkeit
  - Zustandsbedingungen (statische Integritätsbedingungen)
  - dynamische Integritätsbedingungen: Übergangsbedingungen oder temporale Bedingungen
- Beispiele dynamischer Integritätsbedingungen:
  - Übergang von FAM-STAND von 'ledig' nach 'geschieden' ist unzulässig
  - Gehalt darf nicht kleiner werden
  - temporale Bedingung: Gehalt darf innerhalb von 3 Jahren nicht um mehr als 25% wachsen

Integritätsbedingungen in SQL92

- Eindeutigkeit von Attributwerten
  - UNIQUE bzw. PRIMARY KEY bei CREATE TABLE
  - Satztypbedingungen

```
CREATE  TABLE  PERS ...
    PNR INT UNIQUE
      (bzw. PRIMARY KEY)
```

- Fremdschlüsselbedingungen
  - FOREIGN-KEY-Klausel
  - Satztyp- bzw. satztypübergreifende Bedingung

- Wertebereichsbeschränkungen von Attributen
  - CREATE DOMAIN,
  - NOT NULL
  - DEFAULT
  - Attribut- und Satztyp-Bedingungen

Integritätsbedingungen in SQL92 (2)

- Allgemeine Integritätsbedingungen
  - CHECK-Constraints bei CREATE TABLE
  - allgemeine Assertions, z.B. für satztypübergreifende Bedingungen

CHECK-Constraints bei CREATE TABLE

```
CREATE TABLE PERS ....
   GEB-JAHR  INT
   CHECK (VALUE BETWEEN 1900 AND 2100)
CREATE TABLE ABT .....
    CHECK (GEHALTSSUMME <  JAHRESETAT)
```

Anweisung CREATE ASSERTION

```
CREATE ASSERTION A1
CHECK (NOT EXISTS
(SELECT * FROM ABT A
 WHERE GEHALTSSUMME <>
(SELECT SUM (P.GEHALT) FROM PERS P
   WHERE P.ANR = A.ANR)))
DEFERRED
```

- Festlegung des Überprüfungszeitpunktes:
  - IMMEDIATE: am Ende der Änderungsoperation (Default)
  - DEFERRED: am Transaktionsende (COMMIT)

- keine direkte Unterstützung für dynamische Integritätsbedingungen in SQL92
  -> Trigger (SQL:1999)

Integritätsregeln

- Standardreaktion auf verletzte Integritätsbedingung: ROLLBACK
- Integritätsregeln erlauben Spezifikation von Folgeaktionen, z.B. um Einhaltung von IB zu erreichen
  - SQL92: deklarative Festlegung referentieller Folgeaktionen (CASCADE, SET NULL, ...)
  - SQL99: Trigger
- Trigger: Festlegung von Folgeaktionen für Änderungsoperationen INSERT, UPDATE oder DELETE
- Beispiel: Wartung der referentiellen Integrität

deklarativ:

```
CREATE TABLE PERS
  ( PNR INT PRIMARY KEY,
    ANR  INT  FOREIGN KEY
      REFERENCES ABT
      ON DELETE CASCADE
            ... );
```

Trigger-Lösung:

```
CREATE TRIGGER MITARBEITERLÖSCHEN
AFTER DELETE ON ABT
REFERENCING OLD AS A
DELETE FROM PERS P
WHERE P.ANR = A.ANR;
```

- Trigger wesentlicher Mechanismus von ***aktiven Datenbanksystemen***

Trigger

- ausführbares, benanntes DB-Objekt, das implizit durch bestimmte Ereignisse (“triggering event”) aufgerufen werden kann
- Triggerspezifikation besteht aus
  - auslösendem Ereignis (Event)
  - Ausführungszeitpunkt
  - optionaler Zusatzbedingung
  - Aktion(en)

```
CREATE TRIGGER GEHALTSTEST
AFTER UPDATE OF GEHALT ON PERS
REFERENCING OLD AS AltesGehalt,
NEW AS NeuesGehalt
WHEN (NeuesGehalt < AltesGehalt)
ROLLBACK;
```

- zahlreiche Einsatzmöglichkeiten
  - Überwachung nahezu aller Integritätsbedingungen, inkl. dynamischer Integritätsbedingungen
  - Validierung von Eingabedaten
  - automatische Erzeugung von Werten für neu eingefügten Satz
  - Wartung replizierter Datenbestände
  - Protokollieren von Änderungsbefehlen (Audit Trail)
  - • • •

Trigger (2)

- SQL99-Syntax

```
CREATE TRIGGER <trigger name>
 { BEFORE | AFTER } { INSERT | DELETE
    UPDATE [OF <column list>] }
 ON <table name>
 [ ORDER <order value> ]
 [ REFERENCING <old or new alias list> ]
 [ FOR EACH { ROW | STATEMENT } ]
 [ WHEN  (  <search condition>  )  ]
      <triggered SQL statement>
```

```
<old or new alias> ::=
OLD  [ AS ]  <old values correlation name> |
NEW  [ AS ]  <new values correlation name> |
OLD_TABLE  [ AS ]  <old values table alias> |
NEW_TABLE  [ AS ]  <new values table alias>
```

- Merkmale
  - Trigger-Events: INSERT, DELETE, UPDATE
  - Zeitpunkt: BEFORE oder AFTER
  - mehrere Trigger pro Event/Zeitpunkt möglich (benutzerdefinierte Aktivierungsreihenfolge)
  - Bedingung: beliebiges SQL-Prädikat (z.B. mit komplexen Subqueries)
  - Aktion: beliebige SQL-Anweisung (z.B. auch neue prozedurale Anweisungen)
  - Trigger-Bedingung und -Aktion können sich sowohl auf alte als auch neue Tupelwerte der betroffenen Tupel beziehen
  - Ausführung von Trigger-Bedingung und -Aktion kann für jedes betroffene Tupel einzeln erfolgen (FOR EACH ROW) oder nur einmal für die auslösende Anweisung (FOR EACH STATEMENT)

Trigger (3)

- weiteres Beispiel: Wartung einer materialisierten Sicht ARME_PROGRAMMIERER

  CREATE TRIGGER

- Probleme von Triggern
  - teilweise prozedurale Semantik (Zeitpunkte, Verwendung alter/neuer Werte, Aktionsteil im Detail festzulegen)
  - Trigger i.a. beschränkt auf Änderungsoperationen einer Tabelle (UPDATE, INSERT, DELETE)
  - derzeit i.a. keine verzögerte Auswertung von Triggern
  - Gefahr zyklischer, nicht-terminierender Aktivierungen
  - Korrektheit des DB-/Trigger-Entwurfes (Regelabhängigkeiten, parallele Regelausführung, ...)
- Verallgemeinerung durch sogenannte ECA-Regeln (Event / Condition / Action)

Zugriffskontrolle (Autorisierung)

Subjekte: Benutzer, Terminals

Objekte: Programme (Anwendungs-, Dienstprogramme), DB-Objekte (Relationen, Sichten, Attribute)

Operationen: Lesen, Ändern, Ausführen, Erzeugen, etc
Weitergabe von Zugriffsrechten

Berechtigungsmatrix

Subjekte, Benutzer	Objekte				
	O_1	O_2	O_3	...	O_n
B_1	P_1, P_2		P_3		P_i
B_2		P_1	P_2, P_3		P_1
B_3		P_2, P_3	P_2		
...					
B_m	P_1, P_2	P_i	P_1		P_i, P_k

- Klassifikationskriterien
  - zentrale Vergabe (DBA) vs. dezentrale Vergabe (Eigentümer) von Zugriffsrechten
  - wertabhängige vs. wertunabhängige Objektfestlegung

Zugriffskontrolle in SQL

- Sicht-Konzept: wertabhängiger Zugriffsschutz (Untermengenbildung, Verknüpfung von Relationen, Verwendung von Aggregatfunktionen)
- Vergabe von Rechten auf Relationen bzw. Sichten

```
GRANT  {privileges-commalist | ALL PRIVILEGES}
       ON accessible-object TO grantee-commalist [WITH GRANT OPTION]
```

- Zugriffsrechte: SELECT, INSERT, UPDATE, DELETE, REFERENCES, USAGE
  - Erzeugung einer "abhängigen" Relation erfordert REFERENCES-Recht auf von Fremdschlüsseln referenzierten Relationen
  - USAGE erlaubt Nutzung spezieller Wertebereiche (character sets)
  - Attributeinschränkung bei INSERT, UPDATE und REFERENCES möglich
  - dynamische Weitergabe von Zugriffsrechten: WITH GRANT OPTION (dezentrale Autorisierung)
- Empfänger: Liste von Benutzern bzw. PUBLIC
- Beispiele:

```
GRANT  SELECT  ON  ABT  TO  PUBLIC
GRANT  INSERT, DELETE ON ABT  TO  Mueller, Weber  WITH  GRANT  OPTION
GRANT  UPDATE (GEHALT) ON PERS  TO  Schulz
GRANT  REFERENCES (PRONR) ON PROJEKT TO  PUBLIC
```

Zugriffskontrolle in SQL (2)

- Rücknahme von Zugriffsrechten:

```
REVOKE[GRANT OPTION FOR]  privileges-commalist
   ON  accessible-object FROM grantee-commalist {RESTRICT | CASCA-
```

Beispiel:

```
REVOKE SELECT ON ABT FROM Weber CASCADE
```

- ggf. fortgesetztes Zurücknehmen von Zugriffsrechten
- wünschenswerte Entzugssemantik: Der Entzug eines Rechtes ergibt einen Zustand der Zugriffsberechtigungen, als wenn das Recht nie erteilt worden wäre
- Probleme:
  - Rechteempfang aus verschiedenen Quellen
  - Zeitabhängigkeiten
- Führen der Abhängigkeiten in einem *Autorisierungsgraphen* erforderlich

Zugriff auf Metadaten

- jeder SQL-Katalog enthält ein INFORMATION_SCHEMA zur Beschreibung der Metadaten aller Datenbanken (Schemas) des Katalogs
- INFORMATION_SCHEMA enthält vordefinierte Sichten, auf die über normale SQL-Anweisungen (lesend) zugegriffen werden kann
- Folgende Sichten sind u.a. vorgesehen:

- DB-Objekte
  - SCHEMATA
  - DOMAINS
  - TABLES
  - VIEWS
  - COLUMNS
- Constraints
  - TABLE_CONSTRAINTS
  - REFERENTIAL_CONSTRAINTS
  - DOMAIN_CONSTRAINTS
  - CHECK_CONSTRAINTS
  - ASSERTIONS
- Abhängigkeiten
  - COLUMN_DOMAIN_USAGE
  - VIEW_TABLE_USAGE
  - VIEW_COLUMN_USAGE
  - CONSTRAINT_TABLE_USAG
  - CONSTRAINT_COLUMN_USAGE
- Zugriffsberechtigungen
  - TABLE_PRIVILEGES
  - COLUMN_PRIVILEGES
  - USAGE_PRIVILEGES

Zusammenfassung

- Datenmanipulation: INSERT, UPDATE, DELETE von Tupeln
- Datendefinition: CREATE / DROP TABLE, VIEW, DOMAIN, SCHEMA; ALTER TABLE
- Sicht-Konzept (Views)
  - Korrespondenz zum externen Schema bei ANSI/SPARC
  - Reduzierung von Komplexität, erhöhte Datenunabhängigkeit, Zugriffsschutz
  - Einschränkungen bezüglich Änderbarkeit
- Integritätsbedingungen
  - Klassifikation: modellinhärent/anwendungsspezifisch, statisch/dynamisch, unverzögert/verzögert, Reichweite
  - Unterstützung in SQL (CREATE TABLE, CREATE DOMAIN, ASSERTIONS, Trigger)
- Trigger
  - automatische Reaktion bei DB-Änderungen (-> "aktive DBS")
  - zahlreiche Anwendungsmöglichkeiten: Integritätskontrolle, materialisierte Sichten, ...
- dezentrales Autorisierungskonzept zur Zugriffskontrolle (GRANT, REVOKE)
- Standardisierte SQL-Sichten für Metadaten (INFORMATION SCHEMA)

LEHRVERANSTALTUNGEN WS 2003 / 2004

- Datenbanksysteme 2 (2 + 1 SWS)
  - Kernstudium Praktische Informatik
- Mehrrechner-DBS (2 SWS)
  - Kernstudium Praktische Informatik oder Schwerpunkt Praktische Informatik
- Datenbanken in der Bioinformatik (2 SWS)
  - Bioinformatik oder Praktische/Angewandte Informatik
- Relationales DB-PRAKTIKUM (4 SWS)
  - Schwerpunkt Praktische Informatik
  - Voraussetzung: DBS1-Schein
  - Anmeldung ab Ende Sep. / Anfang Oktober (2er-Gruppen)
- Problemseminar Peer-to-Peer Data Management
  - Schwerpunkte Praktische o. Angewandte Inf.
  - Vorbesprechung mit Themenvorstellung / -vergabe 1. Semesterwoche