Datenbanken

Dr. Jutta Göers,
Übungen: Petra Gremeyer
SS 1997

Inhalt:

1. Einführung
2. Architektur von Datenbanksystemen
3. Datenbankmodelle
4. Konzeptuelle Ebene: Datenbankentwurf, Datendefinition
5. Konzeptuelle Ebene: Relationaler DB-Entwurf
6. Externe Ebene: Anfragesprachen, Updatesprachen
7. Interne Ebene: Dateiorganisationsformen
8. DB-Anwendungsprogrammierung
9. Transaktionen, Datenschutz, Datensicherheit
10. Ausblick: OODBMS, ORDBMS, ...

1. Einführung

Gründe, die Vorlesung zu hören:

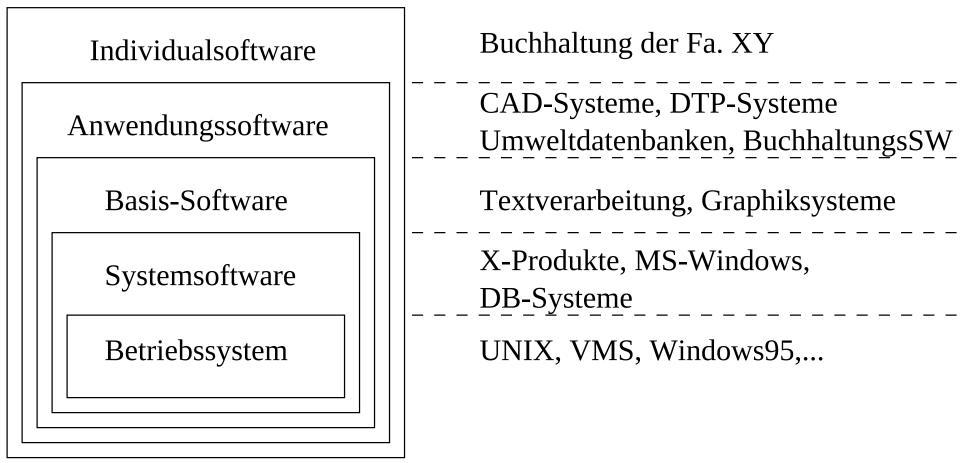
- gute Berufschancen für Informatiker mit DB-Kenntnissen
- 'Datenflut', Information als Wirtschaftsgut
- interessante Kombination von Theorie und Praxis

Datenbanken in der 'realen Welt':

- Reisebüro
- Einwohnermeldeamt, Verkehrssünderkartei
- Banken, Versicherungen
- Telefonauskunft
- Bibliotheken (Buchbestand, Entleiher)
- Supermarkt (Scannerkassen, Lagerbestand)
- Handel, Versandhandel (Kunden, Aufträge, Lager)
- Personalverwaltung
- CAD-/CIM-Daten
- Expertensysteme
- u.a.m.

Einführung

Einordnung in den Bereich EDV (Software)



Einführung

Vorteile von Datenbanksystemen:

- große Datenmengen effizient verarbeiten
- *eine* zentrale, genormte Datenhaltung für div. Anwendungssoftware (Buchhaltung, Auftragswesen, Lager, ..) -> keine Redundanz
- paralleles Arbeiten mehrerer Benutzer
- Anwendungen benutzbar, ohne daß Anwender
  - interne Darstellung der Daten
  - Speichermedium, auf dem Daten liegen

  kennen müssen -> Datenunabhängigkeit
- Anwender sehen nur Ausschnitt aller Daten -> Datenschutz
- weitere 'integrierte Funktionalität':
  - eingebaute Algorithmen für Suchen, Sortieren, Pufferverwaltung...
  - optimiert für Datenbestände die größer als verfügbarer HS sind
  - Überprüfung von Integritätsbedingungen
  - deskriptive Anfragesprachen
  - Anfragen an Meta-Daten
  - Datensicherungsmechanismen
  - Transaktionskonzept

Einführung

Nachteile von DB:

- teuer
- umfangreiche Dokumentation
- großer Festplatten- und HS-Bedarf
- nicht so effizient wie 'handoptimiertes' Programm
- Abhängigkeit vom DB-Hersteller

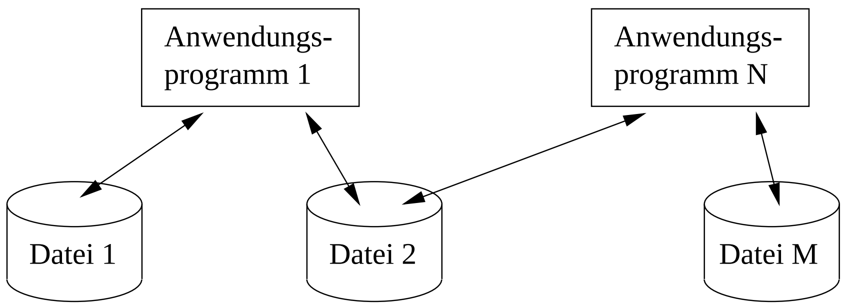
Vielleicht besser keine DB:

- Datenbestand sehr klein, einfach strukturiert, nur von einem Anwendungsprogramm benötigt
- keinerlei Erweiterungen (andere Anwendungen) auf gleichen Daten geplant
- kein Mehrbenutzerbetrieb
- Realzeit-Anwendung (extrem kurze Antwortzeit nötig)

Einführung

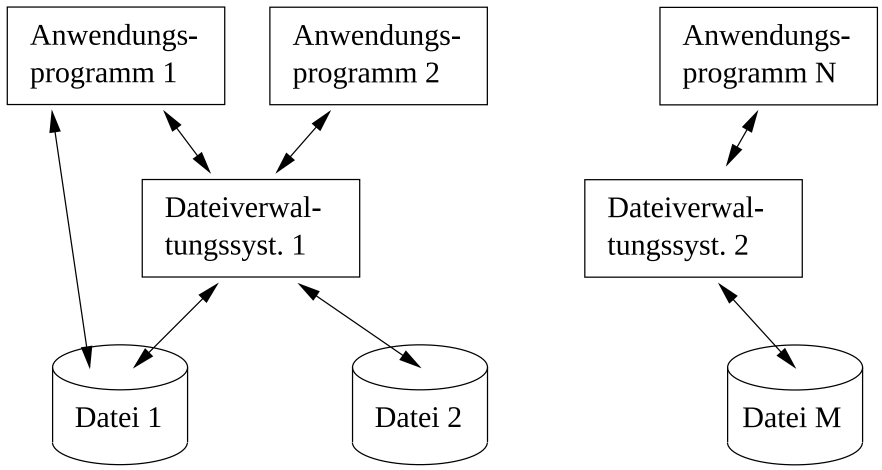
Historie:

1. Stufe: Dateisysteme; Daten in Dateien, anwendungsspezifische Datenorganisation, geräteabhängig, redundant, inkonsistent



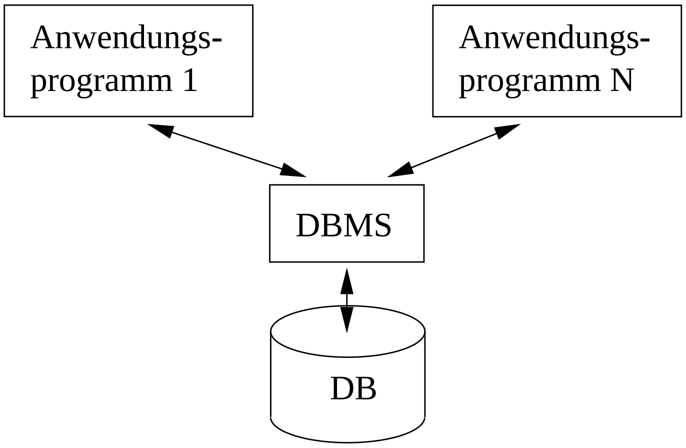
Einführung

2. Stufe: Dateiverwaltungssysteme; Geräteunabhängigkeit der Datenhaltung, zusätzl. Dienstprogramme z.B. zum Sortieren, redundant, inkonsistent



Einführung

3. Stufe: Datenbanksysteme; geräteunabhängig, redundanzfrei, konsistent



dabei
Ende 60er Jahre: DB-Systeme basierend auf hierarchischem Modell oder Netzwerkmodell
70er/80er: Relationale DB-Systeme
80er/90er: Wissensbanken
90er Jahre: Objektorientierte DB-Systeme
Mitte 90er Jahre: Objektrelationale DB-Systeme

Einführung

Begriffe:

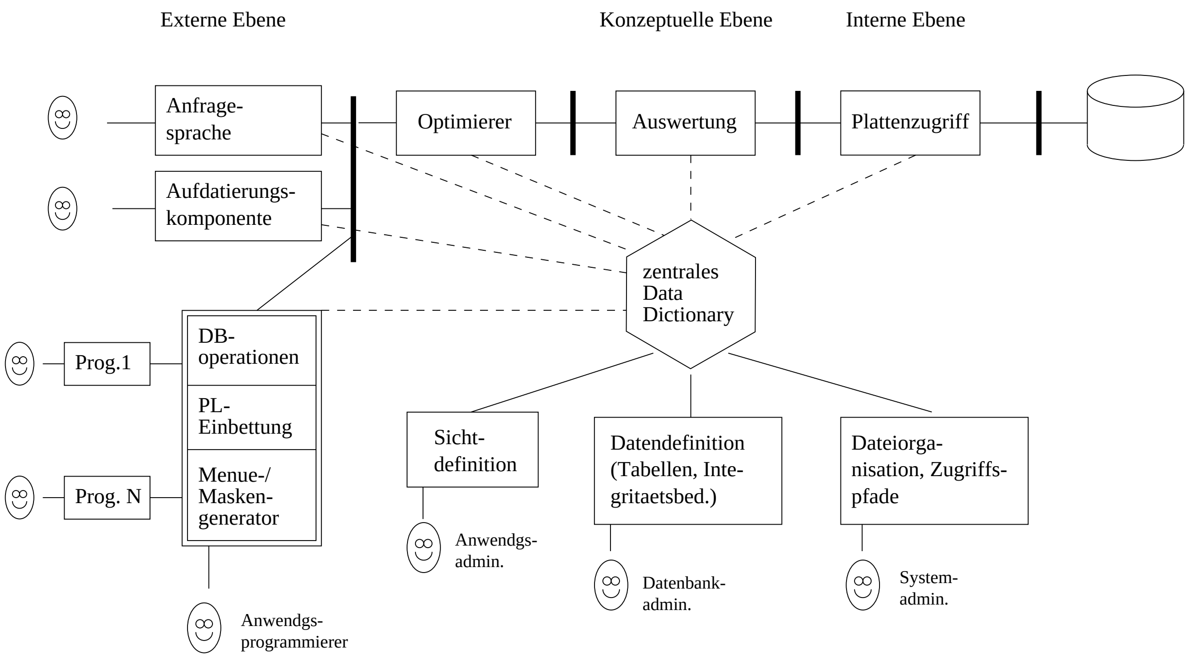
- DBMS: Datenbankmanagementsystem; Gesamtheit aller Module, die die Verwaltung einer DB übernehmen

- DBS: DBMS plus DB

- DB: Gesamtheit der (Anwendungs-)Daten

Einführung

Komponenten eines DBS:



Einführung

Weiterer Aufbau der Vorlesung:

- genauere Betrachtung der Architektur von DB-Systemen
- konzeptuelle Ebene: Datendefinition (hierarchisches, Netzwerk-, relationales u. oo Modell)
- externe Ebene: Anfragesprachen, Updates
- interne Ebene: Datenorganisation, Speicherverfahren
- externe Ebene: Einbettung in Programmiersprachen
- weitere Aufgaben wie Transaktionskonzepte, Sichten,...

Einführung

Literatur:

- J.D. Ullman: Principles of Database- and Knowledge-Base Systems; Computer Science Press, 1988

- G. Vossen: Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme; 2. Aufl. Addison-Wesley

- A. Heuer, G. Saake: Datenbanken - Konzepte und Sprachen; Intern. Thomson Publishing, 1995

- C.J. Date: An Introduction to Database Systems; 6. Aufl., Addison-Wesley, 1995

- R. Elmasri, S.B. Navathe: Fundamentals of Database Systems; 2. Aufl., Benjamin/Cummings, 1994

- diverse weitere DB-Bücher, auch Spezialbücher zu einzelnen Themen

Einführung

Vorlesungsbegleitende Beispiele:

1. (Uni-)Bibliothek mit Bibliotheksangestellten, Entleihern (Studenten, Dozenten), Büchern und Zeitschriften als Ausleihobjekte, Entleihungen;

zu
Angestellten: PANr., Name (Vor-, Nach-), Adresse, Geburtsdatum, Vorgesetzter, Gehalt
Studenten: PANr., Name, Adresse, Geburtsdatum, Matrikelnr., Studienfach
Dozenten: PANr., Name, Adresse, Geburtsdatum, Institut, Institutsadresse
Bücher: ISBN, Titel, Verlag, Autor, Auflage, Jahr, Stichworte
Zeitschriften: ISSN, Bandnr., Jahr, Heftnr., Titel, Verlag, Herausgeber (evtl. noch Artikel pro Zeitschrift mit Titel, Autor, Sichworte, Seiten (von,bis)
Entleihung: Entleiher, Ausleihobjekt, Ausleihdatum, Bibl.Angestellter

Einführung

2. Warenhaus mit Artikeln, Kunden, (Kunden-)Aufträgen, Lieferanten, (Lieferanten-)Angeboten, Abteilungen, Angestellten;

zu
Artikeln: Artikelnr, EAN, Artikelbezeichnung
Kunden: Kundennr., Name, Adresse, Kontostand
Lieferanten: Lieferantennr., Name, Adresse
Angebot: EAN, Preis pro Stück
Auftrag: Auftragsnr., Datum, Liste von Artikeln mit Mengenangabe
Abteilung: Abtlgsname, Abtlgsnr., Liste aller geführten Artikel, Liste aller Angestellten, Abtlgsleiter
Angestellter: Name, Gehalt
Abtlgsleiter: Name, Gehalt

2. Architekturen von DB-Systemen

verschiedene Blickwinkel

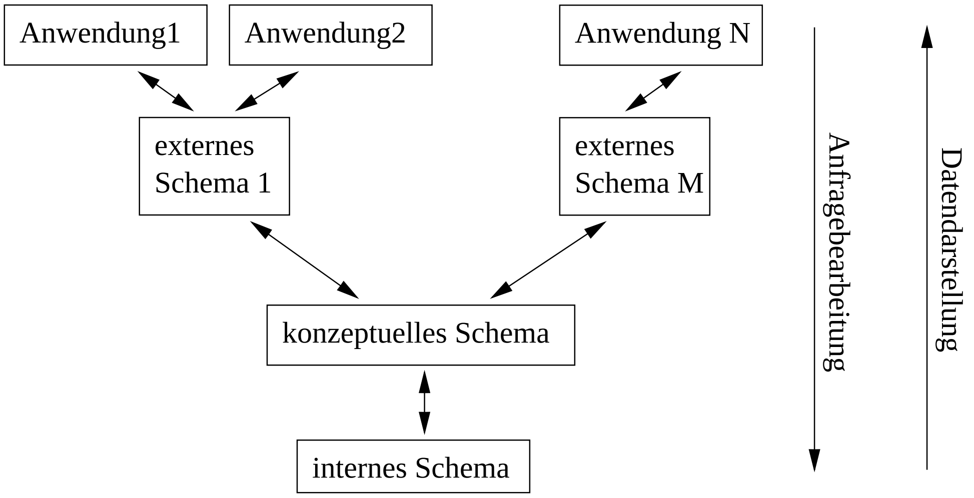
- Schemaarchitektur: beschreibt Zusammenhang zwischen konzeptuellem, internem, externem Schema und Anwendungsprogrammen

- Systemarchitektur: beschreibt Komponenten (Bausteine, Werkzeuge) und Schnittstellen dazwischen

- Anwendungsarchitektur: beschreibt Aufbau und Ablauf der Anwendungsentwicklung

2.1 Schemaarchitektur

- 3-Ebenen-Schema-Architektur
- von ANSI/X3/SPARC Study Group on Database Management Systems (70er Jahre)
-



- Anfragebearbeitung: Übersetzung Anfrage-/Änderungsoperationen -> Operationen auf internen Datenstrukturen
- Datendarstellung: umgekehrte Transformation von Anfrageergebnissen

Schemaarchitektur

Beispiel: Warendatenbank (Ausschnitt)

- konzeptuelle Sicht, hier als Relationen (in Tabellendarstellung):

Artikel

Art.Nr.	EAN	Artikelbezeichnung
1046	4-902580-306625	10 Disketten maxell
0731	8-000280-186625	100 FlipFrame Trans. 3M
2402	4-003969-120266	Heftklammern 24/6
1244	4-004182-031414	Disk.Label Zweckform
...	...	...

Bestand

Abtlgsnr.	Artikel
1	3001
1	3271
5	1046
5	2402
5	1244
...	...

Abteilung

Abtlgsnr.	Abtlgsname	Name
1	Damenoberbekl.	Lagerfeld
2	Herrenoberbekl.	Boss
5	Bürobedarf	Gates
...	...	...

- externe Sichten

  z.B.als flache Relation EAN, Abtlgsnr, Abtlgsleiter:

Sicht1

EAN	Abtlgsnr.	Leiter
4-902580-306625	5	Gates
4-003969-120266	5	Gates
4-837465-885923	2	Boss
...	...	...

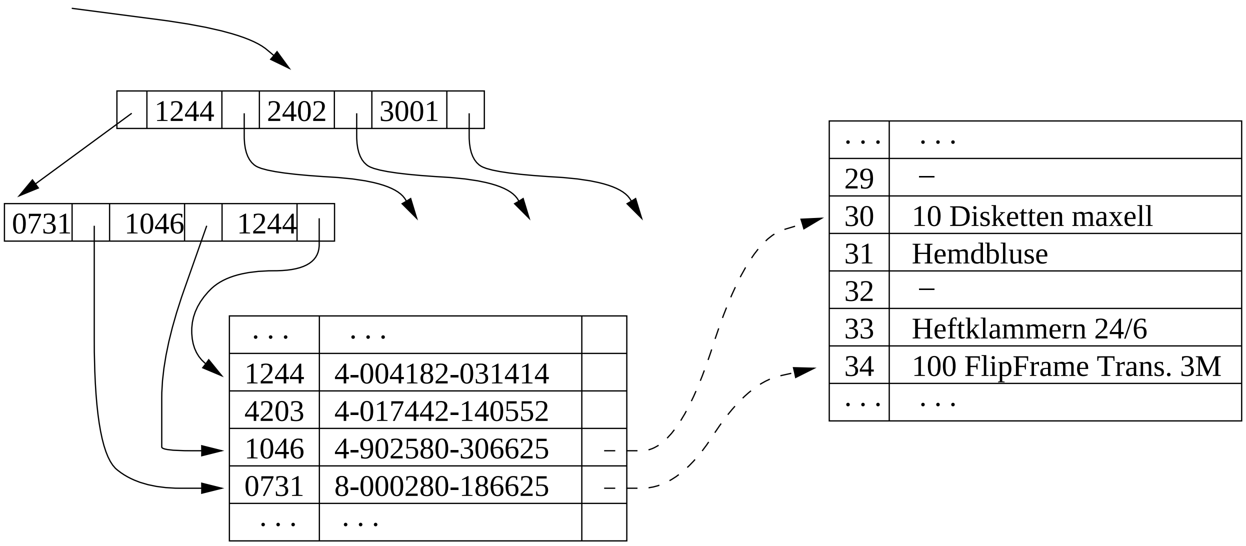
z.B. als Sicht in anderem Datenmodell Abtlgsname, Menge der Artikelbezeichnungen

Sicht2

Abtlgsname	Artikel
	{Artikel}
Damenoberbekl.	Wrangler Jeans
	Seidenbluse
	Hemdbluse
	...
Bürobedarf	10 Disketten maxell
	Heftklammern 24/6
	Disk.Label Zweckform
	...

- interne Darstellung

  z.B. mittels Bäumen, Hashtabellen, etc.



2.2 Systemarchitekturen

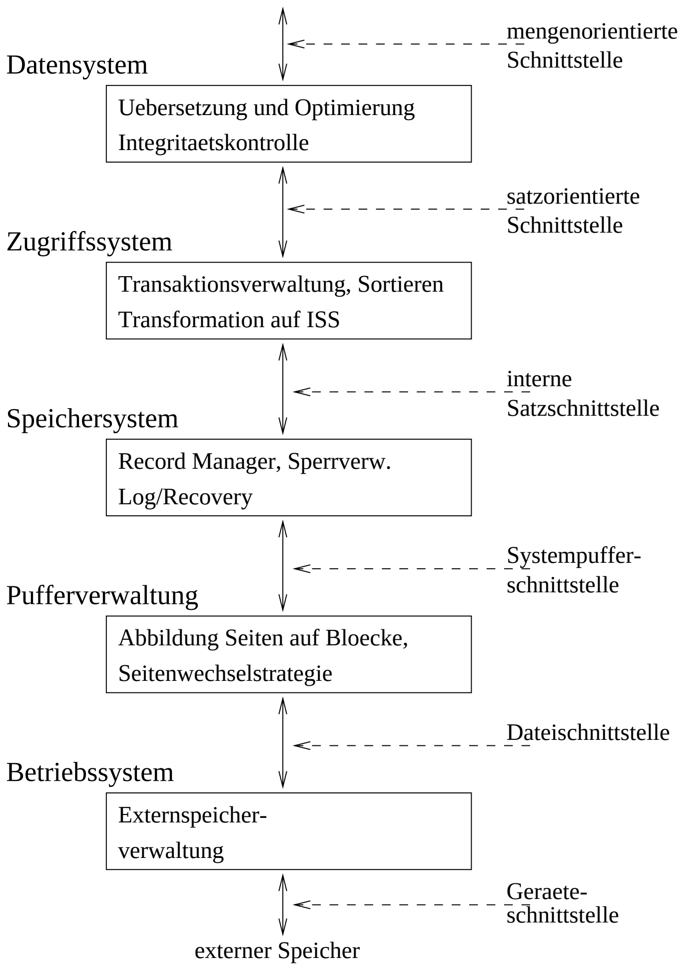
verschiedene Systemarchitekturen:

- 3-Ebenen-System-Architektur von ANSI-SPARC (s. Kap. 1, Abb. 1.5)
- Fünf-Schichten-Architektur
- Strawman-Architektur
- ...

Systemarchitekturen

Fünf-Schichten-Architektur

- nach Senko 1973, Härder 1987
- beschreibt Transformationskomponenten und Schnittstellen (von Modellebene bis Speicherzugriff)
-



- mengenorientierte Schnittstelle: deklarative DML auf Tabellen, Sichten, Zeilen

- satzorientierte Schnittstelle: navigierender Zugriff auf interne Darstellung der Relationen (Sätze, Dateien)

- interne Satzschnittstelle: (Manipulation von) Sätzen, Zugriffspfaden

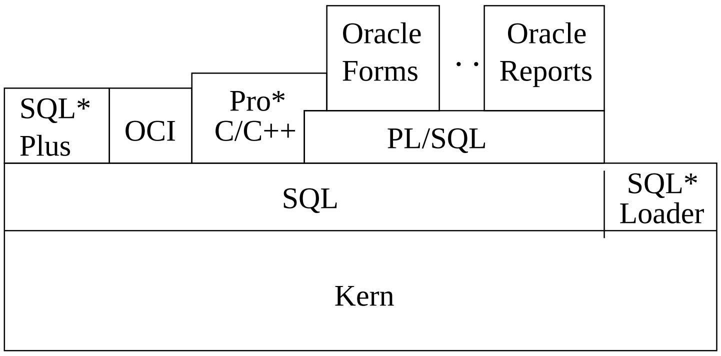
- Systempufferschnittstelle: interne Seiten, Seitenadressen, Freigeben u. Bereitstellen von Seiten

- Dateischnittstelle: Abbildung auf Dateizugriffe

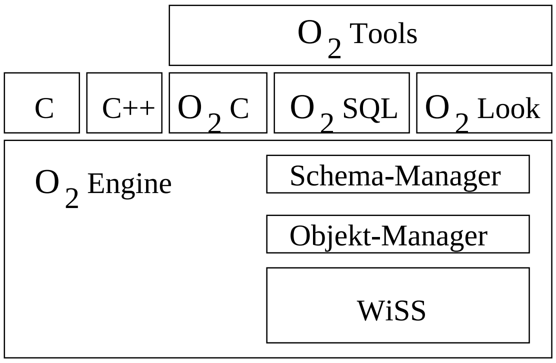
- Geräteschnittstelle: Spuren, Zylinder, Armbewegungen (gehört zum BS)

2.3 Anwendungsarchitektur

- beschreibt Benutzerkomponenten und -schnittstellen
- beschreibt Abarbeitung eines Anwendungsprogramms mit DB-Operationen
- Beispiel ORACLE:



- Beispiel O_2:



3. Datenbankmodelle

Datenmodelle allgemein

- Konzepte zur Erfassung der Informtionsstruktur einer Anwendung

- Informationsstruktur:

  1. statische Eigenschaften
     (a) Objekte
     (b) Beziehungen
  2. dynamische Eigenschaften
     (a) Operationen
     (b) Beziehungen
  3. Integritätsbedingungen
     (a) an Objekte
     (b) an Operationen

- Beispiele von Datenmodellen:

  – Programmiersprachen: Typsysteme
  – Graphiksysteme: Repräsentationmodelle
  – Expertensysteme: Wissensrepräsentationsmethoden
  – Datenbanksysteme: Datenbankmodelle

Datenbankmodelle

- enthalten/beschreiben 1.a, 1.b, 3.a; 2.a (nur implizite Standardoperationen)
- Modelle:

	Daten			Parallele: Algorithm.	
$\uparrow$abstrakter	ER-Modell			Flußdiagr.	
$\downarrow$konkreter	hierarch. Modell	Netzwerk-Modell	Relat. Modell	Modula-Progr.	C-Progr.

objektorientierte DBM; nur eingeschränkt aus ER-Modell ableitbar

3.1 Entity-Relationship-Modell

- nach P.P. Chen, 1976

- Grundkonzepte:

  – Entity: zu modellierende Informationseinheit, 'Gegenstand' der realen Welt; Bsp.: ein Professor, ein Student, eine Prüfung

  – Relationship: Beziehung zwischen Entities; Bsp.: *wird gehalten von* ist Beziehung zwischen Vorlesung und Professor

  – Attribut: Eigenschaft von Entities oder Beziehungen; Bsp.: Matrikelnr. bei Student, Name bei Professor

- weitere Konzepte:

  – Entity-Mengen: Zusammenfassung ähnlicher 'Gegenstände' der realen Welt; Bsp.: alle Professoren

  – Entity-Typ: Strukturbeschreibung einer Entity-Menge

  – Domäne: Wertebereich von Attributen; Bsp.: Note hat Domäne 1.0 ... 6.0, Basistypen wie `INTEGER`, `STRING`,...

  – Schlüssel: Menge von Attributen, die ein Entity eindeutig identifizieren; Bsp.: {Matrikelnr.} ist Schlüssel für Studenten, aber kein Schlüssel für Prüfungen, {Matrikelnr.,Fach,Datum} Schlüssel für Prüfungen

Entity-Relationship-Modell

- weitere Konzepte:
  - is-a-Relationship: Spezialisierungs-/Generalisierungsbeziehung zwischen Entities, Attribute des allgem. Typs gelten auch für spezielleren; Bsp.: Professoren is-a Angestelle
  - funktionale/nichtfunktionale Relationships: Klassifizierung danach, wieviel Entities der einen Menge mit Entities der anderen in Beziehung stehen können:
    one-to-one: pro Entity existiert höchstens ein zugeordnetes in der anderen Entity-Menge (Bijektion); Bsp.: Professoren, Lehrstühle
    many-to-one: ein Entity der Menge M2 steht in Beziehung mit einer (evtl. leeren) Menge von Entities der Entity-Menge M1, jedes Entity aus M1 aber nur mit höchstens einem Entity aus M2 (Funktion); Bsp.: Vorlesungen, Professoren
    many-to-many: bel. viele Entities aus M1 stehen mit bel. vielen Entities aus M2 in Beziehung und umgekehr (Relation); Bsp.: Vorlesung, Studenten
  - mehrstellige Relationships: mehr als zwei Entity-Mengen stehen in Beziehung zueinander; Bsp.: Professor hält Vorlesung und empfiehlt dazu Bücher

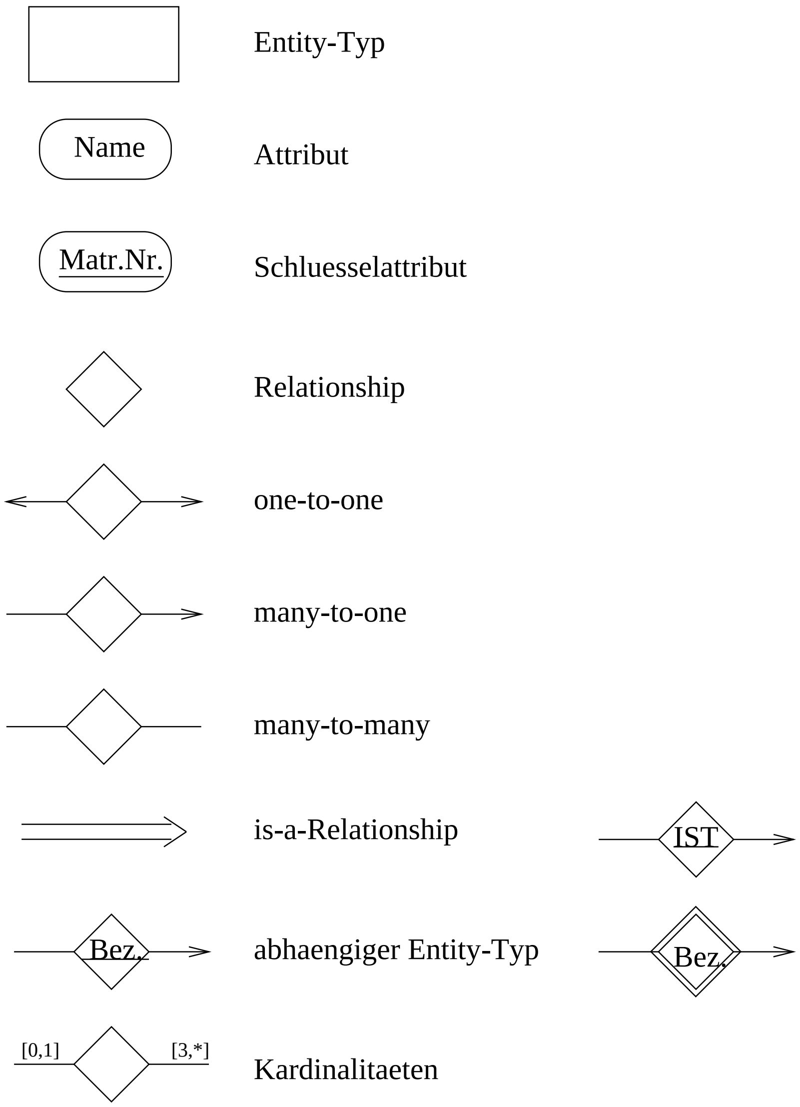
Entity-Relationship-Modell

- weitere Konzepte:
  - Kardinalitäten: genauere Angabe zu beteiligten Entities bei Beziehungen, Angabe jeweils eines minimalen und maximalen Wertes, * steht für unbegrenzte Anzahl; Bsp.: Studenten hören keine/eine/bel. viele Vorlesungen, jede Vorlesung wird von 3 oder mehr Studenten gehört (*hört(Studenten[0,*], Vorlesung[3,*])*)

    Quicky: Welche Kardinalitäten gehören zu one-to-one-Beziehung, many-to-one-Beziehung?
  - abhängige Entity-Typen (weak entity): Instanzen eines Entity-Typs hängen von Existenz eines anderen Entity-Typs ab, funktionale Beziehung zwischen den Entity-Mengen stellt Schlüssel des abhängigen Typs; Bsp.: Entity-Typ BuchExemplar (mit Nummer, Ausleiher, Rückgabe) über Beziehung *gehört zu* abhängig von Entity-Typ Buch (mit ISBN, Titel,..)

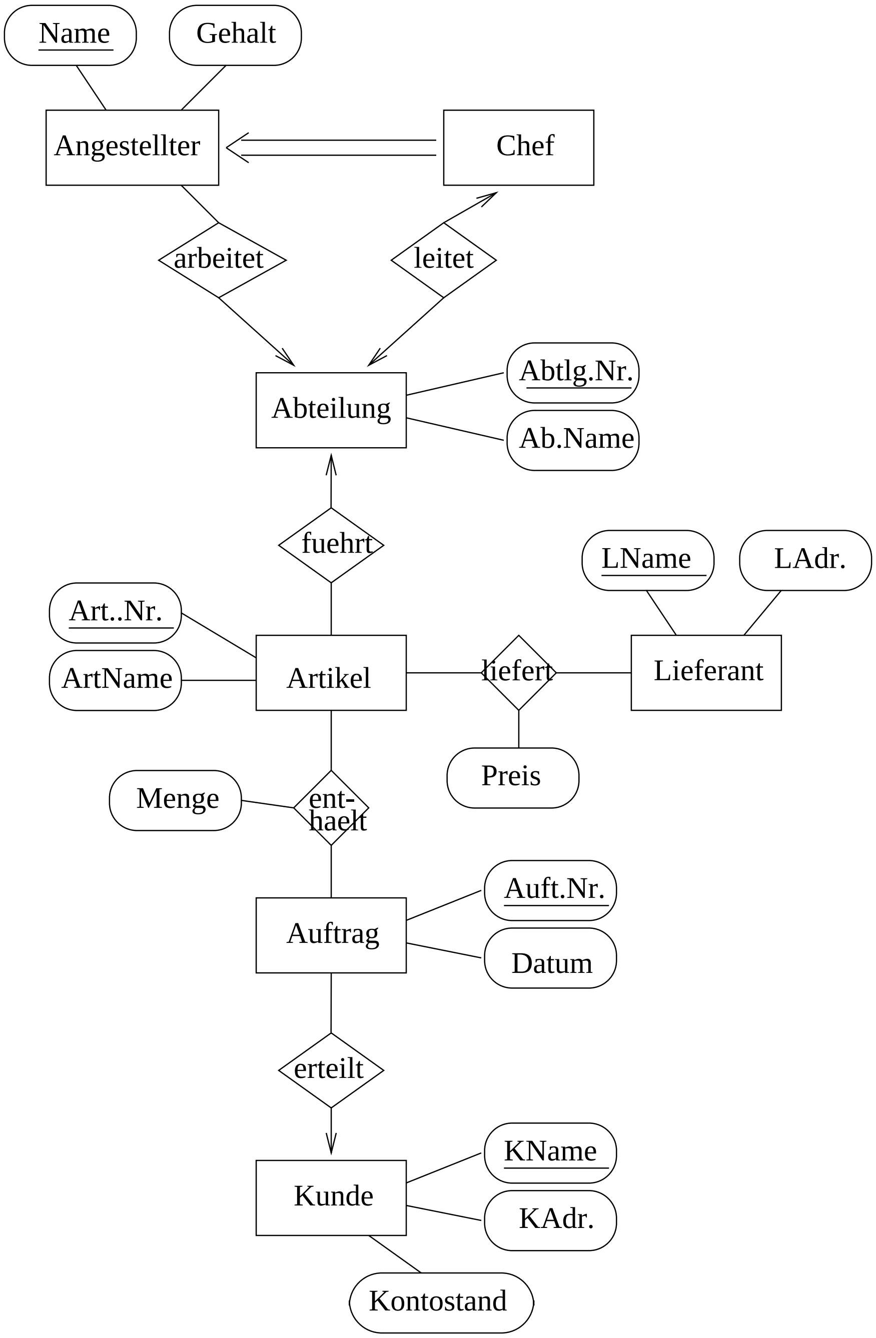
Entity-Relationship-Modell

- graphische Notation:


Entity-Typ
Name
Attribut
Matr.Nr.
Schluesselattribut
Relationship
one-to-one
many-to-one
many-to-many
is-a-Relationship
IST
Bez.
abhaengiger Entity-Typ
Bez.
[0,1]
[3,*]
Kardinalitaeten

Entity-Relationship-Modell

- Beispiel: Warenhaus



Entity-Relationship-Modell

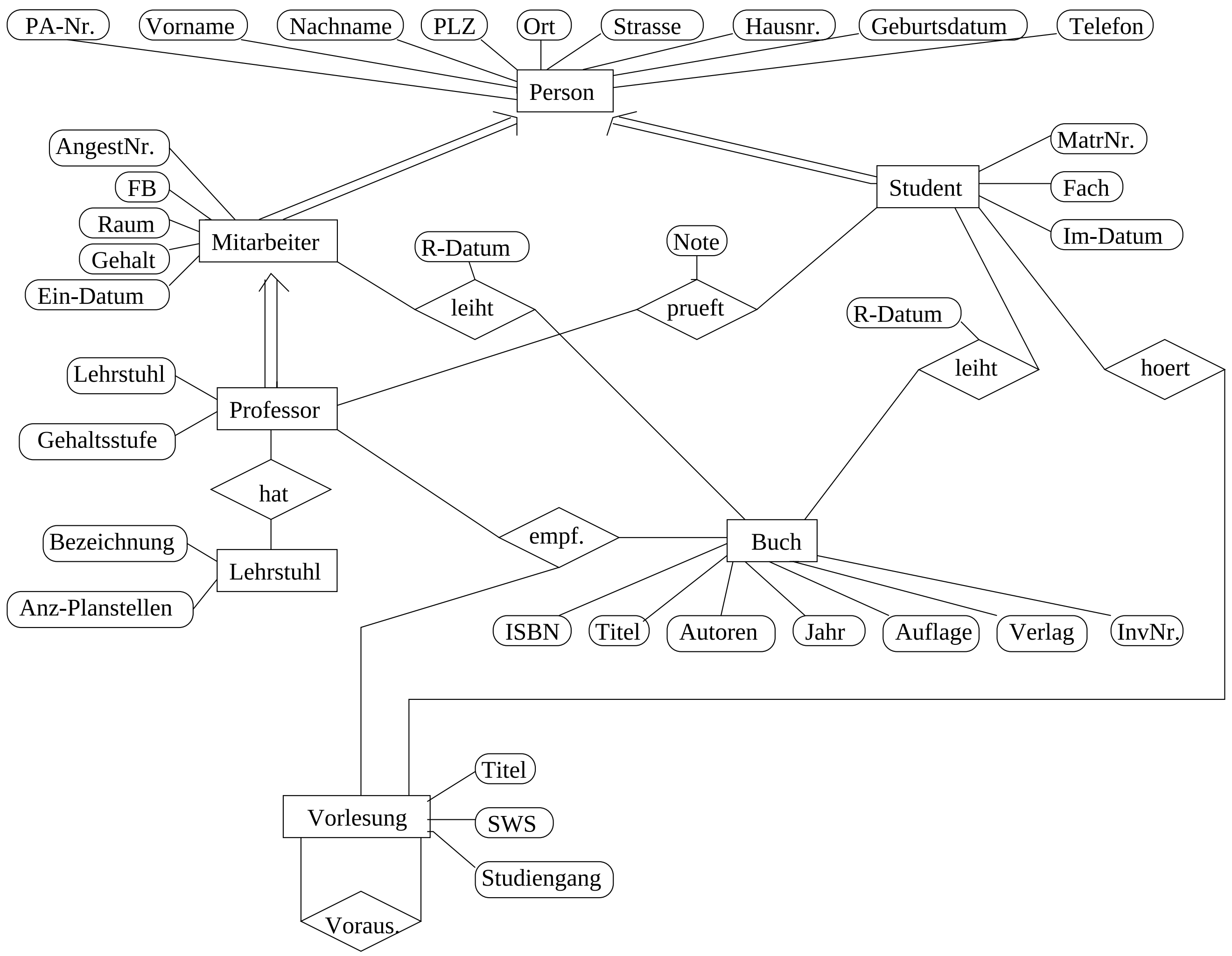
Quicky: Modellierung folgender Universitätsanwendung im ER-Modell?

- folgende Personen spielen eine Rolle: Mitarbeiter, Professoren, Studenten
- Personen haben als Eigenschaften eine PA-Nummer, Vor- und Nachname, Adresse bestehend aus PLZ, Ort, Straße und Hausnummer, ein Geburtsdatum und eine Reihe von Telefonnummern
- Mitarbeiter besitzen eine Angestelltennummer, einen Fachbereich, Raum, ein Einstellungsdatum und ein Gehalt
- Professoren sind Mitarbeiter, die einen Lehrstuhl innehaben und eine Gehaltsstufenangabe (C4, C3,..)
- Studenten haben eine Matrikelnr., ein Studienfach und ein Immatrikulationsdatum
- Lehrstühle kennzeichnen vertretene Fachrichtungen; notwendige Informationen dazu sind die Lehrstuhlbezeichnung und die Anzahl an Planstellen
- Vorlesungen haben einen Titel, eine Wochenstundenzahl, einen zugeordneten Studiengang und andere Vorlesungen als Voraussetzungen
- Bücher haben eine ISBN, einen Titel, eine Menge von Autoren, ein Erscheinungsjahr, eine Auflagennr., eine Verlagsangabe und eine Inventarnummer
- sowohl Mitarbeiter als auch Studenten dürfen Bücher ausleihen; zu jeder Ausleihe gibt es ein Rückgabedatum
- Professoren empfehlen bestimmte Bücher in ihren Vorlesungen

- einige Professoren sind Inhaber von Lehrstühlen
- Professoren lesen in bestimmten Semestern bestimmte Vorlesungen
- Studenten hören in bestimmten Semestern einige Vorlesungen
- einige Vorlesungen haben andere als Voraussetzung
- Professoren prüfen ihre Studenten in bestimmten Vorlesungen; für jede Prüfung wird dabei eine Note vergeben

Entity-Relationship-Modell

Musterlösung:

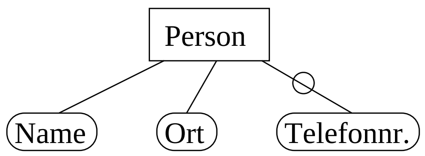


3.2. Erweiterungen

- es ex. div. erweiterte ER-Modelle (EER-Modelle)

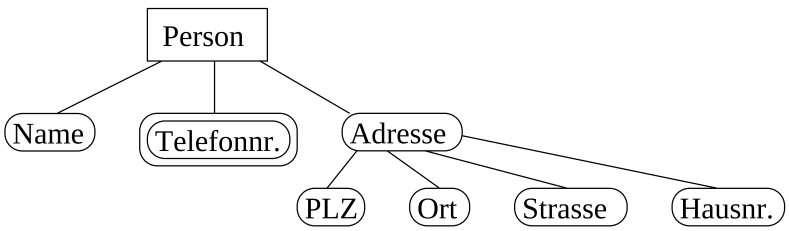
- mögliche Erweiterungen:

  – Optionalität von Attributen/Beziehungen: Angabe, ob Entity an Beziehung teilnehmen *muß* bzw. Attribut einen definierten Wert annehmen *muß*; Schreibweise (z.B.) `1c:m`, ohne `c`: Entity muß in Beziehung auftauchen; graphisch (z.B.) mittels transparentem Kreis an Verbindungslinie:



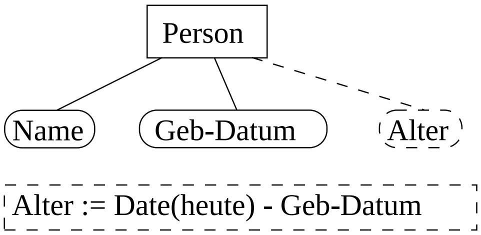
  – neben Standardtypen (`integer, string,...`) für Domänen auch benutzerdefinierte Typen, z.B. `date, time, image,..`

  – Typkonstruktoren bei Attributen, z.B. `TUPLE, LIST, SET, BAG`

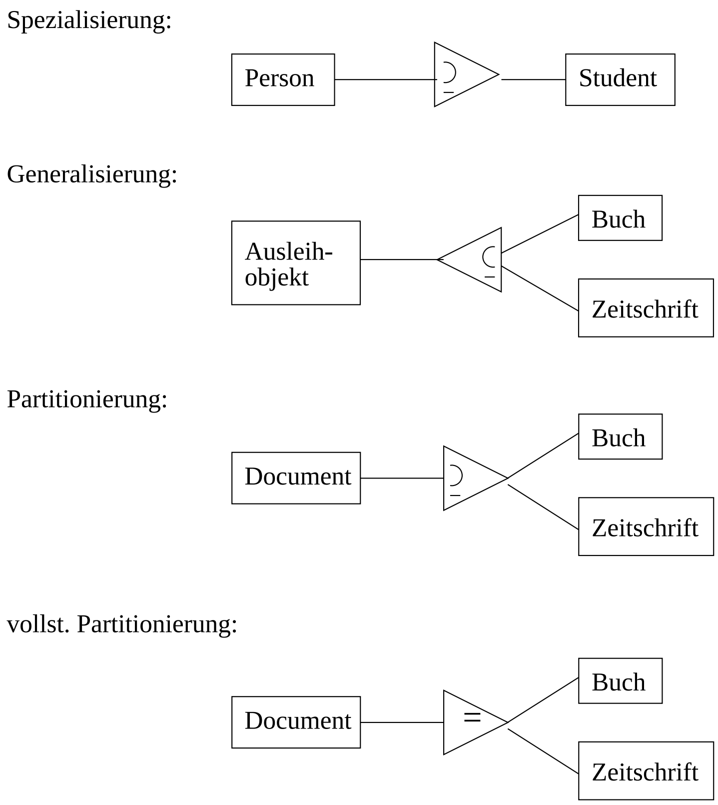


Erweiterungen

- abgeleitete (berechnete; derived) Attribute: Wert wird nicht abgespeichert, sondern mittels Anfrage berechnet



- is-a-Beziehung: Unterscheidung nach Spezialisierung, Generalisierung, Partitionierung
  Spezialisierung: speziellerer Kontext
  Generalisierung: Vereinigungsmenge mehrerer (spezieller) Entity-Mengen zu allgemeinerem Kontext
  Partitionierung: Entity-Typ wird in mehrere *disjunkte* Entity-Typen spezialisiert
  vollständige Partitionierung (Überdeckung): Vereinigung der disjunkten, speziellen Entity-Mengen ergibt genau Instanzen der allgemeineren Entity-Menge



- Assoziationen (Mengenbildung) und Aggregationen (Tupelbildung) zwischen Entities:

  Tupelbildung: Entity setzt sich aus anderem Entities unterschiedlicher Typen zusammen; Bsp.: Auto besteht aus Karosserie, Motor, Kabelbaum, etc.

  Mengenbildung: Entity wird durch Menge von Entities eines anderen Typs gebildet; Bsp.: Menge von Flächen bilden einen Würfel

  zu beachten: Existenzabhängigkeiten

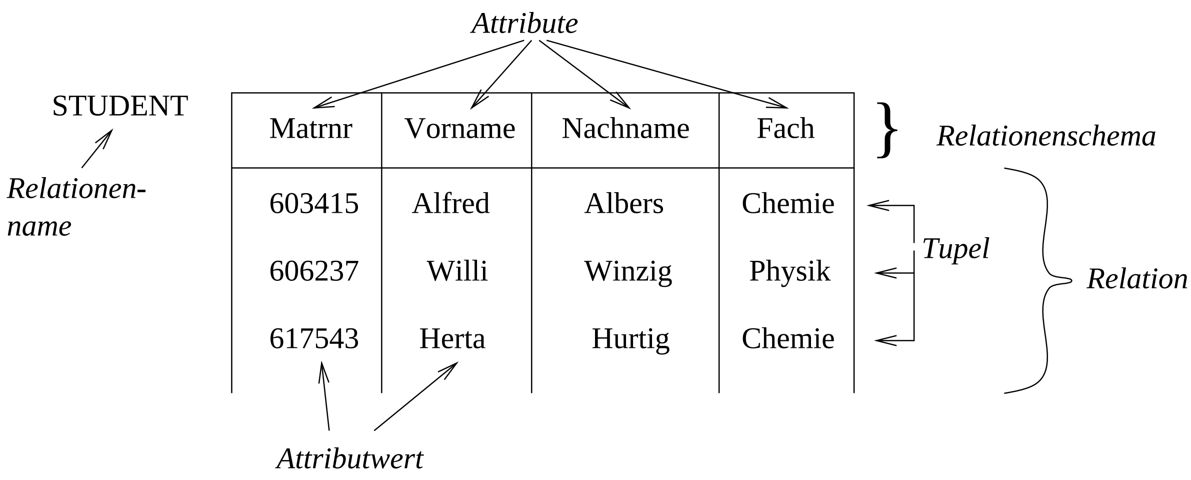
Erweiterungen

- Beziehungen höheren Typs (selten!): Spez./Gen. bei Beziehungen; Bsp.: Semesterausleihe, Kurzausleihe sind Spezialisierungen von Ausleihe

Quicky: was wäre bei Universitätsanwendung in EER-Modell anders?

3.3 Das Relationenmodell

- von Codd, 1970
- anschaulich:



mehrere Relationenschemata ergeben *Datenbank-schema*
mehrere Relationen ergeben *Datenbank*

Das Relationenmodell

formal:

- Eine endliche Menge *U* heißt *Universum*, ein A $\in$ *U* heißt *Attribut*. Sei $D = \{D_1,..,D_m\}$ eine Menge endlicher, nicht-leerer Mengen D_i; jedes D_i wird *Wertebereich* (*Domäne*) genannt.

  Es existiert total definierte Funktion *dom*: $U \to D$, *dom*(A) heißt *Wertebereich von A*, w $\in$ *dom*(A) *Attributwert* für A.

- R $\subseteq$ *U* heißt *Relationenschema*. Eine *Relation* r über R=$\{A_1,..,A_n\}$ (kurz: r(R)) ist eine endliche Menge von Abbildungen t: $R \to \bigcup_{i=1}^{m} D_i$

  t werden *Tupel* genannt, t(A) ist Restriktion von t auf A $\in$ R mit $t(A) := t\,|_A \in$ *dom*(A). Für X $\subseteq$ R heißt $t(X) =: t\,|_X$ *X-Wert von t*.

- Die Menge aller Relationenschemata $S := \{R_1,..,R_p\}$ heißt *Datenbankschema*. Eine *Datenbank* d über $S = \{R_1,..,R_p\}$ (kurz: d(S)) ist eine Menge von Relatonen $d = \{r_1,..,r_p\}$, wobei $r_i(R_i)$ für alle i $\in$ $\{1,..,p\}$ gilt. Ein r $\in$ d heißt *Basisrelation*.

Das Relationenmodell

Beispiel:

S = { ARTIKEL, ABTEILUNG, FÜHRT, AUFTRAG, ENTHÄLT }
ARTIKEL = { Artnr, EAN, Artname}
ABTEILUNG = { Abtlgsnr, Abtlgsname}
FÜHRT = { Abtlgsnr, Artnr}
AUFTRAG = { Auftragsnr, Datum}
ENTHÄLT = { Auftrnr, Artnr, Menge}

dom	Attribut	Domäne
	Artnr	INTEGER
	Artname	STRING(30)
	Abtlgsnr	INTEGER
	Abtlgsname	STRING(40)
	Auftragsnr	INTEGER
	Datum	DATE
	Menge	INTEGER
	EAN	STRING(15)

t_1(Artnr) = 1046,
t_1(EAN) = '4-902588-306625',
t_1(Artname) = '10 Disketten maxell'

t_1 = < 1046,'4-902588-306625','10 Disketten maxell'>
r(ARTIKEL) = {t_1,t_2,t_3,...}

Das Relationenmodell

Integritätsbedingungen:

- Eine *identifizierende Attributmenge* für eine Relation r(R) ist eine Menge $K:=\{B_1,..,B_k\} \subseteq R$ mit $\forall\ t_1,t_2 \in r\ [\ t_1 \neq t_2 \Rightarrow \exists\ B \in K\!: t_1(B) \neq t_2(B)]$
Ein *Schlüssel* ist eine bzgl. $\subseteq$ minimale, identifizierende Attributmenge. Jedes Attribut eines Schlüssels heißt *Primattribut*. Ein *Primärschlüssel* ist (genau) ein ausgezeichneter Schlüssel einer Relation r(R).

  Beispiel:

  { Artnr, Artname, EAN} (automatisch) identifizierende Attributmenge von `ARTIKEL`
{ Artnr} Schlüssel für `ARTIKEL`
{ EAN} Schlüssel für `ARTIKEL`
etwa: { Artnr} Primärschlüssel für `ARTIKEL`

Das Relationenmodell

- Ein *Fremdschlüssel* ist eine Attributmenge einer Relation, die in einer anderen Relation Primärschlüssel ist.
  Eine Relation r *genügt der Fremdschlüsslbedingung* X, wenn X Fremdschlüssel ist und r $|_X \subseteq$ r' $|_X$, wobei X für r' Primärschlüssel ist.

  Beispiel:

  {Artnr} ist Fremdschlüssel in `FÜHRT` und in `ENTHÄLT`, da Artnr Primärschlüssel in `ARTIKEL` ist.

- Übersicht:

Begriff	Bedeutung
Attribut	Spalte einer Tabelle
Wertebereich/ Domäne	mögl. Werte eines Attributs
Attributwert	Element eines Wertebereichs
Relationenschema	Menge von Attributen (Tabellenkopf)
Relation	Menge von Zeilen einer Tabelle (Tabellenrumpf)
Tupel	Zeile einer Tabelle
DB-Schema	Menge von Relationenschemata
Datenbank	Menge von Basisrelationen
Schlüssel	minimale Menge von Attributen, deren Werte ein Tupel eindeutig identifizieren
Primärschlüssel	beim DB-Entwurf ausgezeichneter Schlüssel
Fremdschlüssel	Attributmenge, die in anderer Relation Primärschlüssel ist
Fremdschlüssel-bedingung	die Attributwerte eines Fremdschlüssels müssen in anderer Relation als Werte des Primärschlüssels auftauchen

Das Relationenmodell

Operationen:

- im Modell implizit enthaltene Operationen (= *generische* Operationen)

- Relationenalgebra umfaßt Selektion (σ), Projektion (π), natürlicher Verbund ($\bowtie$), Mengenoperationen ($\cup, \cap, -$), Umbenennung (β)

- Selektion wählt einige Tupel aus Relation aus, Tupel müssen Selektionsbedingung erfüllen

  Bsp.:
  $\sigma_{Abtlgsnr=5}(\ \mathrm{r}(\texttt{FÜHRT}))$ oder $\sigma_{Artnr=EAN}(\ \mathrm{r}(\texttt{ARTIKEL}))$

- Projektion wählt bestimmte Spalten einer Tabelle aus

  Bsp.: $\pi_{Artnr,Menge}(\mathrm{r}(\texttt{ENTHÄLT}))$

- natürlicher Verbund (Join) ist Verknüpfung zweier Relationen über allen gemeinsamen Attributen, jeweils Tupel mit gleichen Attributwerten werden zu neuen Tupeln verbunden

- Bsp.:
  a) r($\texttt{ARTIKEL}$) $\bowtie$ r($\texttt{ENTHÄLT}$) liefert

Artnr	EAN	Artname	Auftragsnr	Menge

  b) $\pi_{Artname,Menge}$(r($\texttt{ARTIKEL}$) $\bowtie$ r($\texttt{ENTHÄLT}$)) könnte dann folgendes Ergebnis liefern:

Artname	Menge
10 Disketten maxell	23
Hemdbluse	12
Heftklammern 24/6	45
Hemdbluse	4
...	...

  -> Operationen der Relationenalgebra können bel. kombiniert werden

- Mengenoperationen bilden die Vereinigung, den Durchschnitt bzw. die Differenz von zwei Relationen, die das gleiche Relationenschema besitzen müssen

- Umbenennung benennt Attribute um; ist sinnvoll, wenn z.B. Attribut beim Join nicht zur Verknüpfung genommen werden soll, obwohl es mit Attribut(name) in anderer Relation übereinstimmt

  Bsp.: $\beta_{EAN \leftarrow Artnr}$ (r($\texttt{ENTHÄLT}$))

Das Relationenmodell

Relationenalgebra formal:

Ω sei Menge an Operationen, Datenbank d ist Erzeugermenge, $[d]_\Omega$ ist Erzeugnis unter den Operationen aus Ω, $([d]_\Omega,\Omega)$ die Relationenalgebra

- Projektion $\pi_X(r) := \{t(X) \mid t \in r\}$

- Konstantenselektion mit $\theta \in \{<,\leq,=,>, \geq, \neq\}$, x X-Wert, $X \subseteq R$
  $\sigma_{X\theta x}(r) := \{t \mid t \in r \wedge t(X)\theta x\}$

- Attributselektion mit $X,Y \in R$
  $\sigma_{X\theta Y}(r) := \{t \mid t \in r \wedge t(X)\ \theta\ t(Y)\}$

- Selektion ist entweder Konstanten- oder Attributselektion

- natürlicher Verbund von $r_1(R_1),r_2(R_2)$
  $r_1 \bowtie r_2 := \{t \mid t \in (R_1 \cup R_2) \wedge [\ \forall i \in \{1,2\}\ \exists\ t_i \in r_i\colon t_i=t(R_i)]\}$

  wenn $R_1=R_2$: mengentheoretischer Durchschnitt
  wenn $R_1 \cap R_2 = \{\}$: kartesisches Produkt

- Mengenoperationen
  $r_1 \cup r_2 := \{t \mid t \in r_1 \vee t \in r_2\ \}$
  $r_1 \cap r_2 := \{t \mid t \in r_1 \wedge t \in r_2\ \}$
  $r_1 - r_2 := \{t \mid t \in r_1 \vee t \notin r_2\ \}$

- Umbenennung
  $\beta_{B\leftarrow A}(r) := \{t' \mid \exists\ t \in r\colon t'(R-\{A\}) = t(R-\{A\}) \wedge t'(B) = t(A)\}$ mit $B \notin R-\{A\}$, $R'=(R-\{A\})\cup\{B\}$, *dom*(A)=*dom*(B)

3.4 Das Netzwerkmodell

- 1971 vom Normungsausschuß CODASYL-DBTG festgelegt ('CODASYL-Datenbankmodell')

- $\approx$ ER-Modell mit ausschließlich binären many-to-one Relationships

- Relationships attributlos

- Vorteile: leichtere Graphendarstellung, günstigere Implementierung

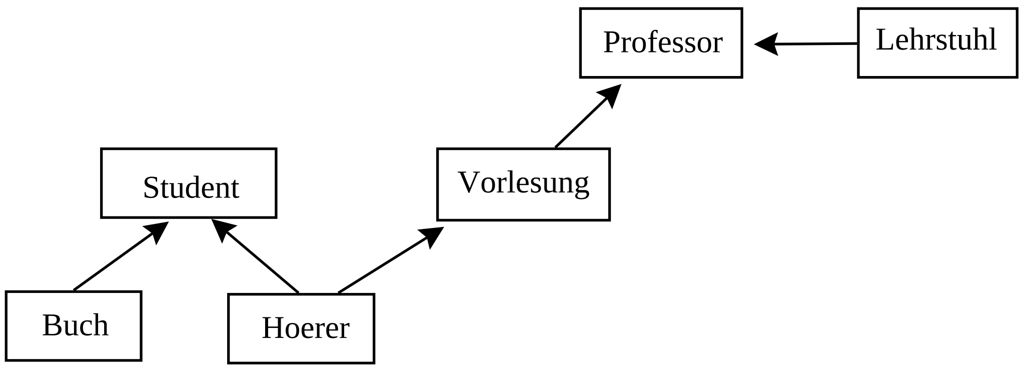
- Bezeichnungen:

Netzwerkmodell	ER-Modell	Relationenmodell
logical record	ein Entity	Tupel
record type	Entity-Typ	Relationenschema
field (Feld)	Attribut	Attribut
link (Set-Typ)	binäre many-to-one Relationship	(dargestellt durch Relation)

Das Netzwerkmodell

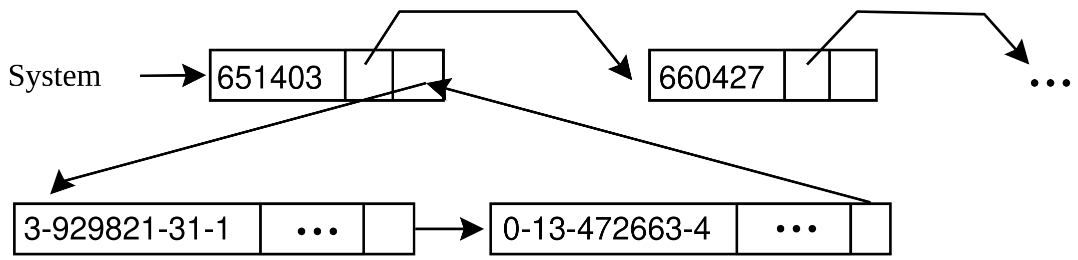
- Netzwerk-Schema: gerichteter Graph mit Knotenmenge = Menge der *record types*, Kantenmenge = Menge der *links*

- gerichtete Kante (E_1,E_2): E_1 steht in 1:n-Beziehung zu E_2, Pfeilende zeigt auf E_1 als 1-Seite:



Das Netzwerkmodell

- *Set-Ausprägung*: zu einem *Owner* gibt es bel. viele (auch 0) *Member*
  Durchlaufen einer Menge von *Members*: *Owner* als *System*, alle *Member* miteinander verkettet, von *Owner* Verweis auf ersten *Member*, von letztem *Member* Verweis auf *Owner* (Ring)

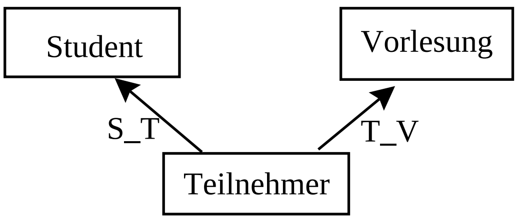


Das Netzwerkmodell

- nicht binäre bzw. nicht many-to-one-Beziehungen simulieren durch künstliche *logical record types* T
  T hat nur ein Attribut, wird per *link* mit *record types* der beteiligten Entities verbuden

  Bsp.: Student, Vorlesung: many-to-many

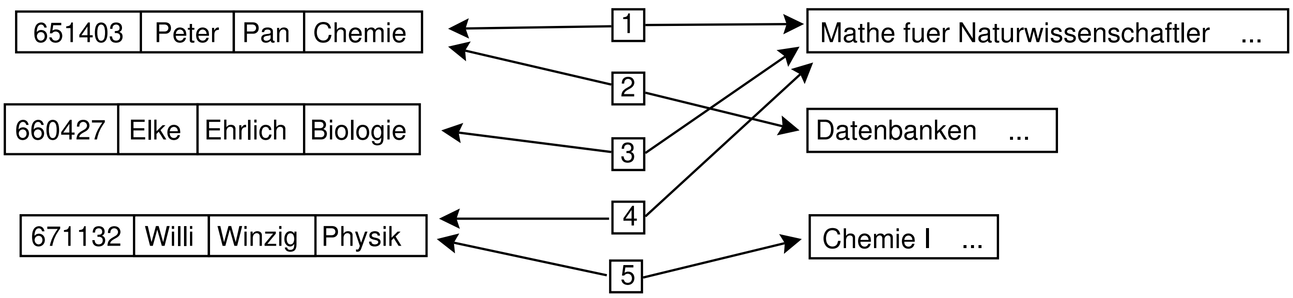
  Modell:



Teilnehmer hat nur ein Attribut, z.B. T_Nr.

Das Netzwerkmodell

Instanzen:



Quicky: wie sieht die Set-Ausprägung (als Ring) aus?

Das Netzwerkmodell

Typische Operationen auf Netzwerken:

- Selektion (in *logical record types*): Suche unter den Records eines Typs bestimmte heraus
  Bsp.: Suche alle Studenten die 'Physik' studieren

- Projektion: Filtere ein oder mehrere Felder heraus
  Bsp.: Gib die Matrikelnr. und das Fach aller Studenten aus

- Navigation: Finde mit Hilfe der *links* das nächste *Member* eines *Owner* oder finde den *Owner* zum momentanen *Member*
  Bsp.: Suche alle Studenten (Nachname, Vorname) der Vorlesung 'Datenbanken'
  Abarbeitung:

  - finde im Record-Typ Vorlesung einen Record mit Titel='Datenbanken'

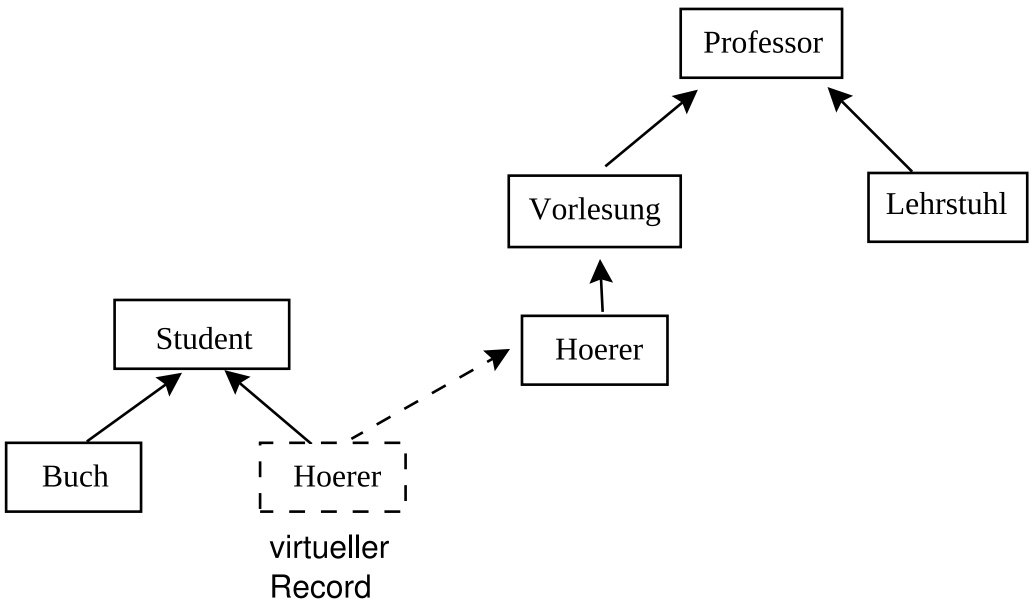
  - durchlaufe alle seine *Member*

  - für jeden *Member* finde dessen *Owner* im Record Student

  - gib bei jedem dieser *Owner* die Felder Vorname und Nachname aus

3.5 Das Hierarchische Modell

- 1969 von IBM (mit dem System IMS)
- eine Hierarchie ist ein Netzwerk, das ein Wald ist, Links zeigen dabei vom Nachfolger zum Vorgänger
- Hierarchie kann (ebefalls) keine allgemeinen Beziehungen darstellen, Abhilfe mittels *virtual records* (Zeiger)
- Entity-Typen heißen *Segmente*, Attribute heißen *Felder*
- Beispiel



Das Hierarchische Modell

- bei Operationen:
  Navigation nur von oben nach unten
  nicht machbar: Wer liest Vorlesung 'Chemie I'
  machbar: Suche alle Vorlesungen von Professor 'Schreiner' oder
  Wer hört die Vorlesung 'Java Programmierung' von Professor 'Schreiner'?
  Abarbeitung:
  - Einstieg in Baum mit Wurzel Professor
  - suche unter Professoren den mit Nachnamen 'Schreiner'
  - suche unter seinen Vorlesungssöhnen den mit Titel 'Java Programmierung'
  - falls gefunden: durchlaufe die Söhne der Vorlesung und gib sie aus

3.6 Erweiterte relationale und semantische Modelle

- nachteilig am relationalen Modell: nur flache Strukturen erlaubt -> hierarchische müssen flach realisiert werden

- $\Rightarrow$ Erweiterungen:
  - NF^2-Modell
  - PNF-Modell
  - eNF^2-Modell
  - semantische Datenmodelle

Erweiterte relationale und semantische Modelle

NF^2-Modell:

- Non First Normal Form

- erlaubt geschachtelte Relationen: Attribute können selbst wieder Relationen sein

- formal:
  R=$\{A_1,..,A_n\}$ mit $dom(A_i) \in D$ oder $A_i = \{A_{i1},..,A_{im}\}$

- Bsp.:

PERSON

PANr.	Name		Adresse				Hobbies
	Vor-	Nach-	PLZ	Ort	Strasse	Hnr.	Hobby
7135	Willi	Winzig	4907	Os.	Am Tie	19	Witze Wasser-ball
1763	Herta	Hurtig	3101	HH	Ölweg	7	Handar-beiten Handball Hunde
...	...	...	...	...	...	...	...

- zusätzliche Operationen notwendig
  - Nestung ν (Schachtelung)
  - Entnestung μ (Entschachtelung)

Erweiterte relationale und semantische Modelle

PNF-Modell:

- Partitioned Normal Form

- geschachtelte Relationen, haben auf jeder Schachtelungstiefe einen flachen Schlüssel

- können immer entschachtelt durch äquivalente 1NF-Relationen dargestellt werden

- Bsp.:

A	D	
	B	C
1	2	3
	4	2
2	1	1
	4	1
3	1	1

  A ist flacher Schlüssel auf oberer Ebene

Erweiterte relationale und semantische Modelle

eNF^2-Modell:

- nicht nur *set of tuple of* (Relationales Modell) bzw. bel. geschachtelt *set of* und *tuple of* (NF^2)
- dazu noch *list of, bag of, array of*
- alle Typkonstruktoren in bel. Kombination

semantische Datenmodelle:

- neben Typkonstruktoren weitere Abstraktionskonzepte wie Generalisierung, Spezialisierung, abgeleitete Attribute, Trigger, dynamische Integritätsbedingungen
- EER-Modelle zählen (oft) zu semantischen Datenmodellen

3.7 Objektorientierte Datenbankmodelle

- es ex. verschiedene Ansätze
- im Folgenden
  - theoretisch fundiertes Modell von Beeri
  - das Modell von O_2

Objektorientierte Datenbankmodelle

Referenz-Modell von Beeri (1989):

Gliederung in Strukturteil, Operationenteil, höhere Konstrukte

Strukturteil:

- Typen und Typkonstruktoren
- Objektidentität
- Klassen
- Beziehungen
- Strukturvererbung (Klassen- und Typhierarchie)
- Integritätsbedingungen

Operationenteil:

- SQL-Erweiterungen
- kalkülartige Sprachen
- deduktive Systeme

mit

- relationale Operationen
- objekterzeugende Operationen
- objekterhaltende Operationen

Höhere Konstrukte:

- Metaklassen
- Funktionen
- Methoden
- Vererbung und Overriding von Methoden
- Einkapselung

Objektorientierte Datenbankmodelle

Strukturteil:

- Typen und Typkonstruktoren
  - Standarddatentypen wie INTEGER, STRING,..
  - Typkonstruktoren wie SET OF und TUPLE OF: kompliziertere Typen
- Objektidentität
  - vom System vergeben
  - eindeutig
  - unveränderbar
  - für den Benutzer unsichtbar
- Klassen
  - beschreiben Objekte mit ähnlichen Eigenschaften
  - Typ, Objektvorrat und Objektbehälter
  - Methoden
- Komponenten-Beziehungen bei Klassen (z.B. Verlage Komponente von Büchern)
- Is-a-Beziehungen
  - Klassenhierarchie
    Objektmenge der Unterklasse ist Teilmenge der Objektmenge der Oberklasse (Studenten sind eine Teilmenge von Personen)
  - Typhierarchie
    Typ der Unterklasse hat mehr Eigenschaften als Typ der Oberklasse (Studenten haben zusätzlich noch Matrikelnr. und Fach)
- Integritätsbedingungen: die meisten (Schlüssel, Fremdschlüssel, teilweise Kardinalitäten,) inhärent in anderen Konzepten enthalten; weitere Bedingungen: z.B. Kardinalitäten an Mengen, dynamische Integritätsbedingungen)

Objektorientierte Datenbankmodelle

Operationenteil:

- Operationen
  - mindestens die Möglichkeiten wie in Relationenalgebra/SQL
  - relationale Semantik: man extrahiert Werte aus Zuständen von Objekten; liefert Relationen
  - objekterzeugende Semantik: man erzeugt neue Objekte als Anfrageergebnis mit Zuständen, die von vorhandenen Objekten extrahiert wurden;
    Ergebnis ist dynamisch erzeugte Klasse
  - objekterhaltende Semantik: man erhält eine Auswahl der in der DB vorkommenden Objekte mit neuen Zuständen; Ergebnis ist dynamisch erzeugte Ober-/Unterklasse
- Relationenmodell: generische Anfragen und Updates auf flachen Relationen

Objektorientierte Datenbankmodelle

Höhere Konzepte:

z.B. Methoden

- objekt- oder klassenspezifische Operationen
- werden wie Eigenschaften von Ober- zu Unterklasse vererbt
- Implementierung einer Methode kann bei Vererbung verändert werden (Overriding)
- System wählt selbständig zur Laufzeit passende Implementierung (dynamisches Binden)

Objektorientierte Datenbankmodelle

O_2:

Typen, Typkonstruktoren

- atomare: integer, char, boolean, string,...
- komplexe: tuple, set, list
- Klassen
- alles beliebig geschachtelt

Klassen

- mit Extension definierbar
- Objekt kann mehrere Rollen spielen

Klassenhierarchie

- Typhierarchie: Subtypen, in ihren Komponenten spezieller bzw. mehr Komponenten
- Klassenhierarchie: Typ der Sohnklasse ist Subtyp von Typ der Vaterklasse
- Mehrfachvererbung, bei Namenskonflikten: teilweise Umbenennung, teilweise Zurückweisung der Mehrfachvererbung

Generische Operationen

- O_2QUERY vollständige relationale Semantik
- eingeschränkte objekterhaltende Semantik
- keine dynamischen Typen

4. Datenbankentwurf, Datendefinition

Entwurfsaufgabe

1. Anwendungsdaten sollen (effizient) aus gespeicherten Daten abgeleitet werden können

2. nur vernünftige Anwendungsdaten sollen gespeichert werden können

3. Anwendungsdaten sollen möglichst nicht redundant dargestellt werden

Grob: drei Entwurfsschritte

- konzeptuelle Modellierung der Datenbank

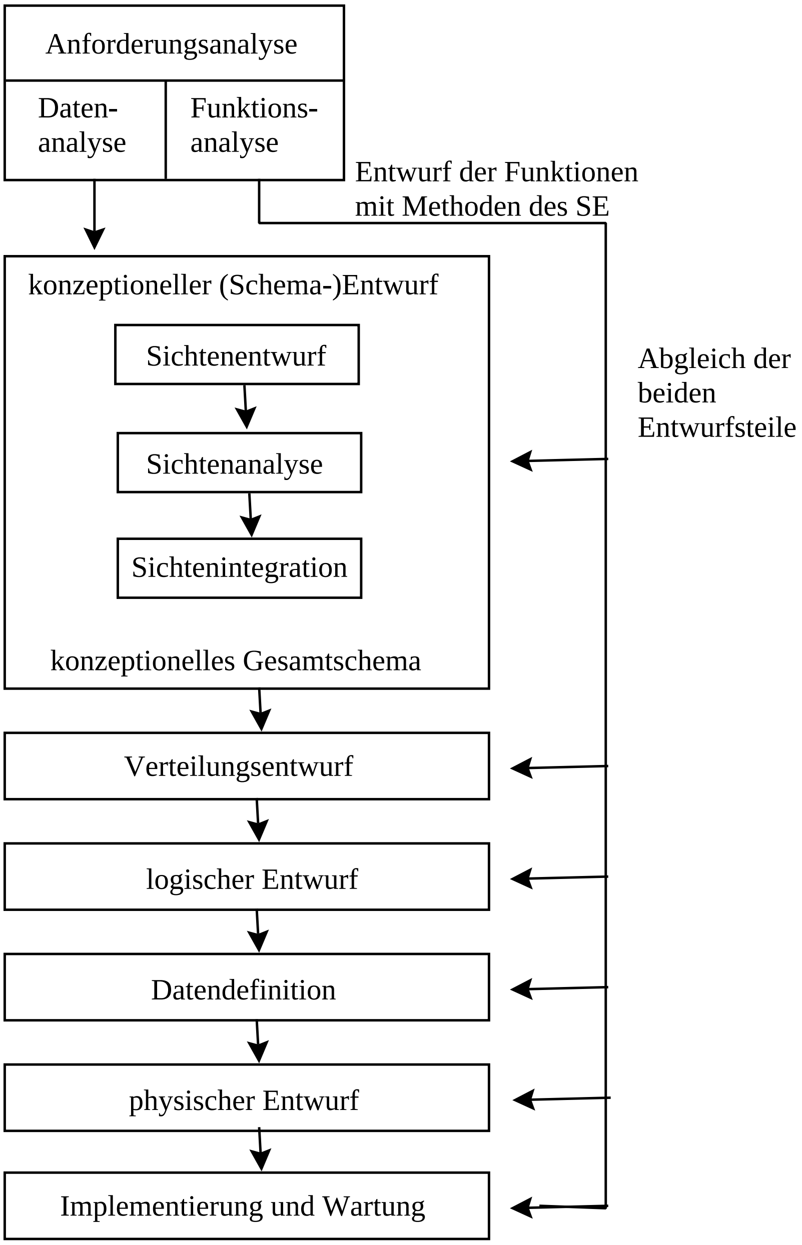
- Abbildung des konzeptuellen Modells auf kommerzielles DB-Modell

- Definition der DB in DDL

Genauer: Phasenmodell

Datenbankentwurf, Datendefinition

Phasenmodell



Datenbankentwurf, Datendefinition

Konzeptueller Entwurf, Sichtintegration

- erste 'formale' Beschreibung in semantischem Datenbankmodell (z.B. ER-Modell)
- Vorgehensweise:
  - Modellierung von Sichten in Fachabteilungen
  - Analyse der Sichten in Bezug auf
    - Namenskonflikte
    - Typ- und Wertebereichskonflikte
    - Bedingungskonflikte
  - Integration der Sichten in ein Gesamtschema
- Ergebnis:
  - Entity-Typen, Relationship-Typen
  - Attribute und Domänen
  - Integritätsbedingungen
  - Beschreibung des dynamischen Verhaltens

Datenbankentwurf, Datendefinition

Abbildung des ER-Modells auf andere DB-Modelle

- unterschiedliche Vorgehensweisen
  - manuelle Transformation der ER-Konzepte in Konzepte des konkreten DB-Modells nach Faustregeln
  - automatische Transformation der ER-Konzepte
- Ziele
  - Darstellung <u>aller Informationen</u> des ER-Diagramms im resultierenden DB-Schema
  - Erhaltung der Informationskapazität (im resultierenden DB-Schema genauso viele Instanzen darstellbar wie in ER-Diagramm - nicht mehr und nicht weniger!)

4.1 Abbildung ER-Modell → Relationenmodell

- Entity-Typ → Relationenschema mit allen Attributen des Entity-Typs, Schlüssel wird übernommen

- Relationship-Typ → Relationenschema mit allen Attributen des Relationship-Typs, zusätzlich alle Primärschlüssel der beteiligten Entity-Typen

  1:n : Primärschlüssel der n-Seite wird Schlüssel
  1:1 : beide Primärschlüssel werden je ein Schlüssel
  n:m : beide Primärschlüssel zusammen werden Schlüssel

- Is-a-Relationship → Relationenschema mit allen Attributen des spezielleren Entity-Typs, zusätzlich den Primärschlüssel des allgemeineren Entity-Typs

- bei *muß*-Beziehungen ([1,*])

  1:n : Entity-Relationenschema der n-Seite wird in Relationenschema der Relationship integriert
  1:1 : beide Entity-Relationenschemata verschmelzen

Abbildung ER-Modell → Relationenmodell

Beispiel: ER-Diagramm Warenhaus im Relationenmodell

Angestellter = { $\underline{\text{Name}}$, Gehalt}

Abteilung = { $\underline{\text{Abtlgsnr}}$, Abtlgsname}

Artikel = { $\underline{\text{Artnr}}$, Artname}

Lieferant = { $\underline{\text{LName}}$, LAdresse}

Auftrag = { $\underline{\text{Auftragsnr}}$, Datum}

Kunde = { $\underline{\text{KName}}$, KAdresse, Kontostand}

Chef = { $\underline{\text{Name}}$}

leitet = { $\underline{\text{Name}}$, $\underline{\text{Abtlgsnr}}$}

arbeitet = { $\underline{\text{Name}}$, Abtlgsnr}

führt = { $\underline{\text{Artnr}}$, Abtlgsnr}

liefert = { $\underline{\text{LName, Artnr}}$, Preis}

enthält = { $\underline{\text{Artnr, Auftragsnr}}$, Menge}

bestellt = { $\underline{\text{Auftragsnr}}$, KName}

Abbildung ER-Modell $\rightarrow$ Relationenmodell

Problemfälle ER-Modell $\rightarrow$ Relationenmodell

- gleichnamige, gleichbedeutende Relationships (leiht in Uni-Anwendung)
- Relationship eines Entity-Typs mit sich selbst (Voraus in Uni-Anwendung)
- mehrwertige Beziehungen (empfiehlt in Uni-Anwendung)

nicht abbildbar

- komplexe Attribute wie Stichwort, Autoren
- Generalisierung (Disjunktheitsbedingung)

Abbildung ER-Modell → Relationenmodell

Datendefinition im Relationenmodell:
mittels SQL-DDL (Teil von SQL; Normen: SQL-89, SQL-92, SQL3)

SQL als DDL:

- externe Ebene: `create view, drop view`
- konzeptuelle Ebene: `create table, alter table, drop table`
- interne Ebene: `create index, alter index, drop index`

SQL-89 mit IEF:
+ Definition von Schlüsseln
+ Definition von Fremdschlüsseln

SQL-92:
+ `default`-Klausel
+ `create domain`-Anweisung für benutzerdefinierte Wertebereiche
+ `check`-Klausel

Bsp.: `domain` Fachgebiet `char(20) default` 'Informatik'

```
create table Vorlesung
( Titel char(30) not null, primary key,
SWS integer check(SWS > 0);
Studiengang Fachgebiet )
```

Abbildung ER-Modell → Relationenmodell

Beispiel: Datendefinition Warenhaus mittels SQL-DDL von Oracle

```
create table artikel (
artnr number(4) constraint art_pk primary key,
artname varchar2(40) );


create table kunde (
kname varchar2(20) constraint kunde_pk primary key,
kadresse varchar2(30),
kontostand number(6,2) );

create table auftrag (
auftragnr number(6) constraint auf_pk primary key,
datum date );

create table arbeitet (
angname varchar2(40),
abtlgnr number(2),
constraint arbeitet_fk1 foreign key (angname) references ange-
stellter(angname),
constraint arbeitet_fk2 foreign key (abtlgnr)
references abteilung(abtlgnr) );

alter table chef
add constraint chef_primk primary key (angname);
```

4.2 Abbildung ER → Netzwerkmodell

- Entity-Typ → logical record type mit allen Attributen als fields

- Relationship-Typ → link (Set-Typ) im Fall von binärer 1:n Relationship ohne Attributen, andernfalls z.B. Ausweg über künstl. record type ('Kett-Record-Typ')

  1:n (binär, ohne Attribute) : link
  1:n mit Attributen : Attribute der Beziehung in Record-Typ der Member-Seite aufnehmen
  n:m : über Kett-Record-Typ (mit allen Attributen)
  1:1 : wie 1:n Relationship, Zusatzbedingung im Programm überwachen

- Is-a-Relationship → wie 1:n Relationship, zwei Zusatzbedingungen (1:1, gleiches Entity) im Programm überwachen

Abbildung ER → Netzwerkmodell

Beispiel: Warenhaus in Netzwerkmodell

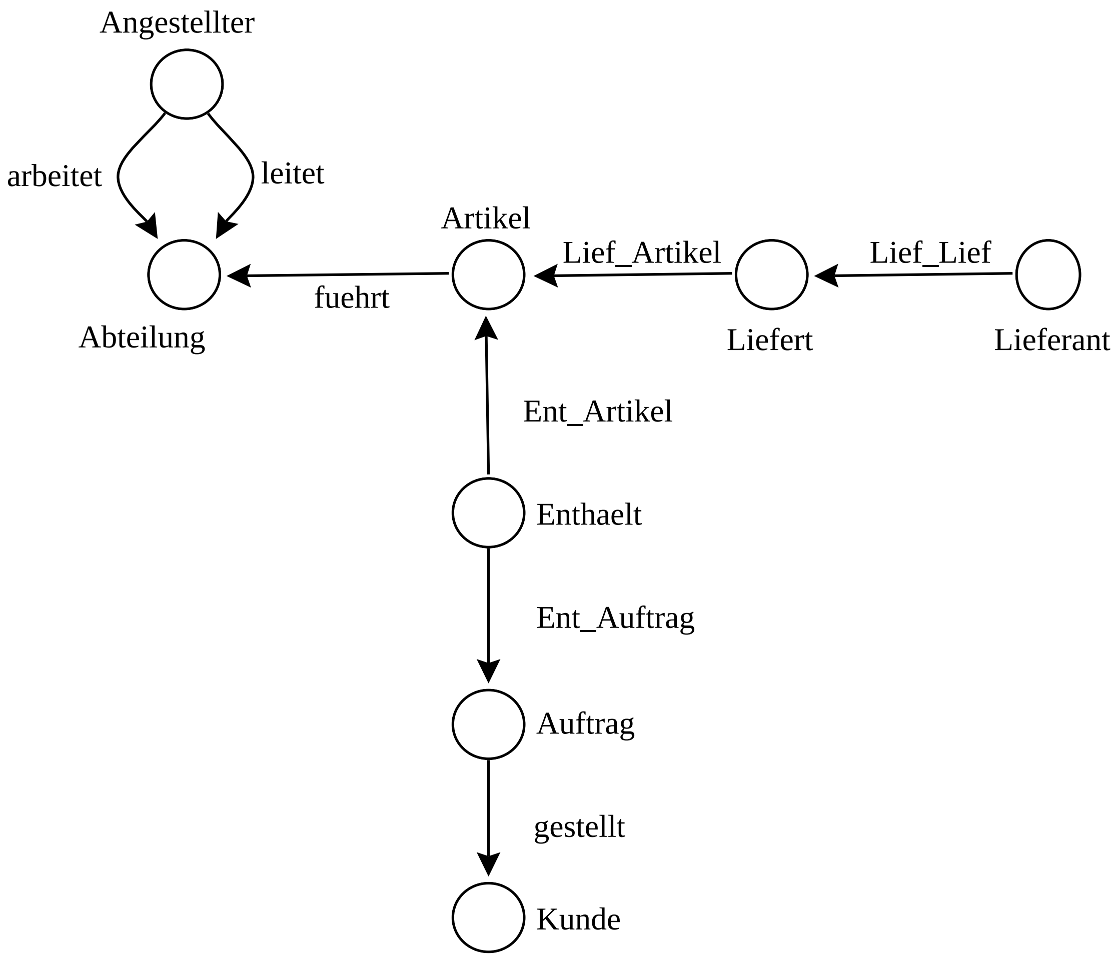


Abbildung ER $\rightarrow$ Netzwerkmodell

Datendefinition im Netzwerkmodell:
z.B. in DBTG DDL

Record-Typen allgemein:

```
RECORD r
   1 Feldname1 Typ1
   1 Feldname2 Typ2
   1 Feldname3
        2   Feldname31 Typ31
        2   Feldname32 Typ32
   1 Feldname4 Typ4
   KEY   Feldname_i
```

Set-Typen allgemein:

```
DBTG SET s
        OWNER IS o
        MEMBER IS m
```

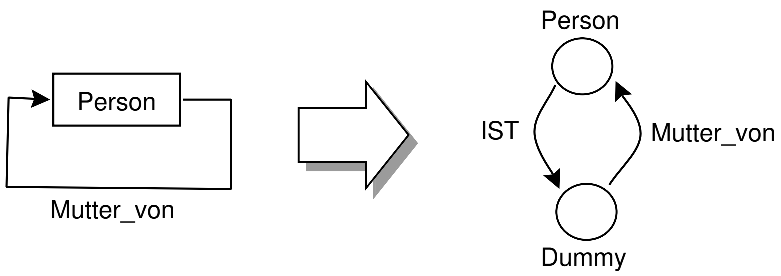
Abbildung ER → Netzwerkmodell

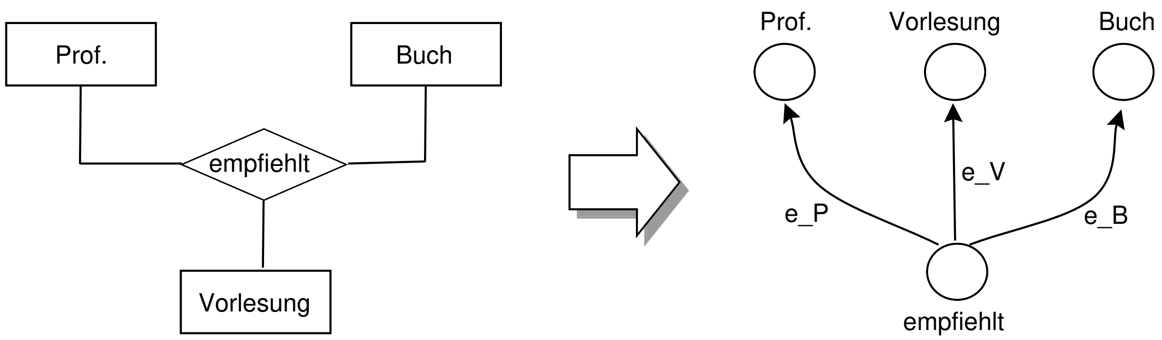
Datendefinition DBTG DDL Warenhaus:

```
RECORD Angestellter
   1 Name CHAR(40)
   1 Gehalt REAL
   KEY   Name

RECORD Abteilung
   1 Abtlgsnr INTEGER
   1 Abtlgsname CHAR(25)
   KEY    Abtlgsnr

...

RECORD Liefert
   1 Preis REAL

RECORD Enthaelt
   1 Menge INTEGER

DBTG SET arbeitet
        OWNER IS Abteilung
        MEMBER IS Angestellter

DBTG SET Leitet
        OWNER IS Abteilung
        MEMBER IS Angestellter

DBTG SET Lief_Artikel
        OWNER IS Artikel
        MEMBER IS Lieferant

DBTG SET Lief_Lief
        OWNER IS Lieferant
        MEMBER IS Liefert

...
```

Abbildung ER → Netzwerkmodell

Problemfälle ER → Netzwerkmodell:





4.3 Abbildung ER → Hierarchisches Modell

- da Hierarchisches Modell Einschränkung des Netzwerk-Modells: Hierarchisches Schema aus Netzwerk ableiten:
- aus Netzwerkschema Menge von Bäumen erstellen
- Merkregel: Links zeigen vom Nachfolger zum Vorgänger
- Ansatzpunkt: Ecken ohne auslaufende Kanten werden Wurzel von Bäumen
- Ecken mit mehr als einer auslaufenden Kante: Auftrennen mit virtual record types
- gerade n:m Beziehungen nachträglich durch wechselseitige virtual record types nachbessern

Beispiel Warenhaus: Wurzeln werdenAbteilung, Lieferant, Kunde
Hierarchisches Schema:

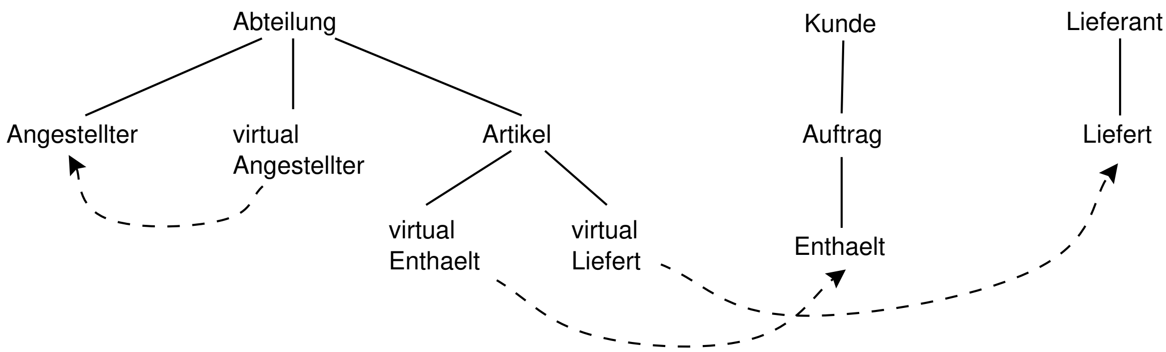


Abbildung ER → Hierarchisches Modell

Datendefinition im Hierarchischen Modell:
z.B. in IMS-DDL (hier nur angelehnt an IMS-DDL)

Baum mit Record-Typen allgemein:

```
TREE Baumname
RECORD r1 Liste_von_Record-Informationen
RECORD r2 Liste_von_Record-Informationen
...
```

wobei Record-Informationen sind

- Feldangabe (wie bei DBTG-DDL)
- Position im Baum:
  - ROOT
  - PARENT = Name_des_Vorfahren
- virtuelle Felder mit
  VIRTUAL Record-Typ IN Name_des_Baumes

Abbildung ER → Hierarchisches Modell

Datendefinition IMS-DDL Warenhaus:

```
TREE Abteilungsbaum
        RECORD Abteilung ROOT
                1 Abtlgsnr INTEGER
                1 Abtlgsname CHAR(25)
        RECORD Angestellter PARENT=Abteilung
                1  Name CHAR(40)
                1 Gehalt REAL
        RECORD Chef PARENT=Abteilung
                VIRTUAL Angestellter IN Abteilungsbaum
        RECORD Artikel PARENT = Abteilung
                ...
        ...
        RECORD Virt_Enthaelt PARENT=Artikel
                VIRTUAL Enthaelt IN Kundenbaum
        RECORD Virt_Lief PARENT=Artikel
                VIRTUAL Liefert IN Lieferantenbaum
...
```

5. Relationaler Datenbank-Entwurf

- zum Verfeinern des logischen Entwurfs
- Problem: Wann ist der logische Entwurf gut?
- Ziel: Vermeidung von Redundanzen durch Aufspaltung von Relationenschemata
  - ohne semantische Information zu verlieren (Abhängigkeitstreue)
  - ohne Möglichkeit zur Rekonstruktion zu verlieren (Verbundtreue)
- Hilfsmittel: Normalformen

5.1 Normalformen

- verhindern Redundanz
- sichern guten DB-Schema-Entwurf zu
- **1. Normalform** (1NF):
  - alle Attributwerte sind atomar
  - Bsp.:

LName	LAdresse	Artikelnrs
Meier	Bochum	3,7,19
Müller	Essen	3,4
...	...	...

    nicht erlaubt, da Artikelnrs strukturiert
  - Bsp. in erster Normalform:

LName	LAdresse	Artikelnrs
Meier	Bochum	3
Meier	Bochum	7
Meier	Bochum	19
Müller	Essen	3
Müller	Essen	4
...	...	...

Normalformen

- **2. Normalform** (2NF):
  - Relation R ist in 1 NF und jedes Nicht-Primattribut von R hängt voll von jedem Schlüssel ab
  - = keine partiellen Abhängigkeiten zwischen Schlüssel und weiteren Attributen
  - Definition: *Funktionale Abhängigkeit* (FD) zwischen Attributmenge X und Attributmenge Y (X→Y): falls Tupel in den Spalten X übereinstimmen, so stimmen sie auch in den Spalten Y überein
  - Bsp.:

Auftr.nr	Datum	KName	LName	Artikel	Menge
4711	4.6.96	Schulze	Meier	3	10
4711	4.6.96	Schulze	Meier	7	2
4712	4.6.96	Schnell	Müller	4	6
4713	7.6.96	Schmidt	Mauser	3	1
4713	7.6.96	Schmidt	Mauser	122	12

    FDs bspw. Auftr.nr → Datum,KName,LName
    Auftr.nr,Artikel → Datum,KName, Menge
    Auftr.nr,Datum → Auftr.nr.,Datum,LName

  - steht rechts von → ganzes Relationenschema und ist linke Seite minimal: linke Seite Schlüssel
  - partielle Abhängigkeit: ein Attribut hängt schon funktional von einem TEIL des Schlüssels ab
  - im Bsp.: Schlüssel ist Auftr.nr,Artikel; Datum hängt aber nur von Auftr.nr ab

Normalformen

- – Erreichen der 2NF durch Elimination der rechten Seite der part. Abhängigkeit, linke Seite wird kopiert
  – im Bsp.:

Auftr.nr.	Artikel	Menge
4711	3	10
4711	7	2
4712	4	6
4713	3	1
4713	122	12

Auftr.nr.	Datum	KName	LName
4711	4.6.96	Schluze	Meier
4712	4.6.96	Schnell	Müller
4713	7.6.96	Schmidt	Mauser

FDs formal

Definition 1: Funktionale Anhängigkeit (FD)
Sei R $\subseteq$ U, r(R), X,Y $\in$ R; R genügt der funktionalen Abhängigkeit X$\rightarrow$Y :$\Leftrightarrow$
$\forall$ X-Werte x gilt: $|\ \pi_y(\sigma_{X=x}(r))\ |\leq 1 \leftrightarrow$
$\forall t_1, t_2 \in r : t_1(X) = t_2(X) \Rightarrow t_1(Y) = t_2(Y)$

Bemerkung: eine Menge von FDs F kann als Menge von lokalen Integritätsbedingungen aufgefaßt werden

Beispiel:

r:

A	B	C
a1	b1	c1
a2	b1	c1
a3	b2	c1
a4	b2	c1

r genügt den FDs A$\rightarrow$B, B$\rightarrow$C, also auch A$\rightarrow$C, aber nicht C$\rightarrow$A oder C$\rightarrow$B

Definition 2: Die Hülle von F ist F$^+$:= $\{$f$|$ F impliziert f (F$\models$f)$\}$

Bemerkung: $\models$ kann über Ableitungsregeln simuliert werden; folgende Ableitungsregeln sind gültig, vollständig und unabhängig:

1. Reflexivität: $\{\}$ $\Rightarrow$ r genügt X$\rightarrow$X

2. Akkumulation: r genügt $\{$X$\rightarrow$YZ, Z$\rightarrow$VW$\}$ $\Rightarrow$ r genügt x$\rightarrow$YZV

3. Projektivität: r genügt X$\rightarrow$YZ $\Rightarrow$ r genügt X$\rightarrow$Y

Beispiel: $\{A \rightarrow B, B \rightarrow C\} \models A \rightarrow BC \models A \rightarrow C$

Definition 3: Die Hülle einer Attributmenge X bzgl. F ist $X^*_F = \{A \mid X \rightarrow A \in F^+\}$

Membership-Problem: '$X \rightarrow Y \in F^+$?' lösen durch '$Y \subseteq X^*_F$?'

Vorgehen:

1. bestimme X, setze $X^* := X$
2. gibt es FD $f_1 := X_1 \rightarrow Y_1 \in F$ mit $X_1 \subseteq X^*$?
3. Ja: dann $X^* := X^* \cup Y_1$
4. führe 2.,3. solange aus, bis X^* stabil; ist $Y \subseteq X^*$, dann ist $x \rightarrow Y \in F^+$

Beispiel: $A \rightarrow C \in \{A \rightarrow B, B \rightarrow C\}$?
1. $A^* := A$
2.,3. für $A \rightarrow B$ gilt: $A \in A^*$, also $A^* := A \cup B = AB$
für $B \rightarrow C$ gilt: $B \in AB$, also $A^* := AB \cup C = ABC$
4. $C \in ABC$, also $A \rightarrow C \in \{A \rightarrow B, B \rightarrow C\}$

Definition 4: Sei R Relationenschema; eine FD $X \rightarrow R$ heißt Schlüsselabhängigkeit, X ist dann Schlüssel von R

Definition 5: Seien F,G FD-Mengen über U; F heißt äquivalent zu G (F Überdeckung von G), kurz $F \equiv G$, $:\Leftrightarrow F^+ = G^+ \Leftrightarrow$
$F \models \{g \mid g \in G\} \wedge G \models \{f \mid f \in F\} \Leftrightarrow$
$\forall\, g \in G: g \in F^+ \wedge \forall\, f \in F: f \in G^+$

Normalformen

- **3. Normalform** (3NF):
  - Relation R ist in 1NF und kein Nicht-Primattribut hängt transitiv von einem Schlüssel ab
  - = keine transitiven Abhängigkeiten zwischen Schlüssel und weiteren Attributen
  - Bsp.:

<u>Auftr.nr</u>	LName	LAdresse
4711	Meier	Bochum
4712	Müller	Essen
4713	Mauser	Essen
4714	Meier	Bochum

    FDs: Auftr.nr $\rightarrow$ LName, LName $\rightarrow$ LAdresse

  - Transitive Abhängigkeit: Attributmenge X hängt vom Schlüssel ab, ein anderes Attribut A hängt von X ab
  - im Bsp.: Schlüssel ist Auftr.nr., LName hängt von Auftr.nr ab, aber LAdresse hängt allein von LName ab
  - Erreichen der 3NF durch Elimination von A, X wird kopiert
  - im Bsp.:

Auftr.nr	LName
4711	Meier
4712	Müller
4713	Mauser
4714	Meier

LName	LAdresse
Meier	Bochum
Müller	Essen
Mauser	Essen

Normalformen

- **4. Normalform** (4NF):
  - Keine zwei mehrwertigen Abhängigkeiten zwischen Attributen
  - Definition: mehrwertige Abhängigkeit (MD) zwischen Attributmenge X und Attributmenge Y ($X \rightarrow\rightarrow Y$), falls einem Wert in den Spalten X mehrere Werte in den Spalten Y zugeordnet werden
  - Bsp.:

Name	Kind	Hobby
Willy Winzig	Hugo	Wandern
Willy Winzig	Egon	Wandern
Willy Winzig	Hugo	Wasserball
Willy Winzig	Egon	Wasserball
Willy Winzig	Hugo	Wirtshäuser
Willy Winzig	Egon	Wirtshäuser

mehrwertige Abhängigkeiten: Name $\rightarrow\rightarrow$Kind, Name$\rightarrow\rightarrow$Hobby

  - Erreichen der 4NF durch Elimination der rechten Seite einer der beiden mehrwertigen Abhängigkeiten, linke Seite wird kopiert
  - im Bsp.:

Name	Kind
Willi Winzig	Hugo
Willi Winzig	Egon

Name	Hobby
Willi Winzig	Wandern
Willi Winzig	Wasserball
Willi Winzig	Wirtshäuser

5.2 Datenbankschemaeigenschaften

DB-Entwurfsproblem:

- aus gegebenem U und FD-Menge F soll lokal erweitertes DB-Schema $S = \{R_1,\ldots,R_p\}$ mit R_i=(R_i,K_i) erzeugt werden, wobei K_i Menge von Schlüsseln ist, denen die Basisrelationen r(R_i) genügen

- es sollen nur noch (unvermeidbare) Redundanzen bei Schlüsseln auftreten

- alle FDs sollen durch Schlüssel dargestellt werden können

- urspr. Relation soll immer durch Anfrageoperationen wiederhergestellt werden können

- man produziere dabei nicht unnötig viele Tabellen

Datenbankschemaeigenschaften

Datenbankschemaeigenschaften:

- **N1**: Abhängigkeitstreue

  S charakterisiert vollständig F $\Leftrightarrow$ F $\equiv \{K \rightarrow R \mid \mathcal{R}=(R,\mathcal{K}) \in S,\ K \in \mathcal{K}\}$, d.h. Hülle der Schlüsselabhängigkeiten ist gleich Hülle von F

- **N2**: Normalformen
  - man fordert meist 3NF für guten Entwurf
  - es ex. Boyce-Codd-NF (BCNF; R ist in 1NF und kein Attribut hängt transitiv von einem Schlüssel ab)
  - BCNF $\Rightarrow$ 3NF $\Rightarrow$ 2NF, aber BCNF und N1 i.A. unvereinbar

- **N3**: Minimalität
  - zur Vermeidung globaler Redundanzen
  - $\nexists$ S': |S'|<|S|, S' erfüllt N1, N2, N3, N4

Datenbankschemaeigenschaften

- **N4**: Verbundtreue
  - für die korrekte Informationswiedergewinnung
  - Forderung: die Dekomposition von U in $R_1,..,R_p$ ist bzgl. F $\pi \bowtie$-treu (verbundtreu)
  - Definition: die Dekomposition einer Attributmenge X in $X_1,..,X_p$ mit $X_1 \cup \ldots \cup X_p = X$ heißt $\pi \bowtie$-treu bzgl. einer FD-Menge F über X :$\Leftrightarrow$ $\pi_{X_1}(r) \bowtie \ldots \bowtie \pi_{X_p}(r) = r$
  - Bsp.:

    a) Dekomposition von ABC in AB, BC ist nicht $\pi \bowtie$-treu bzgl. $\{A \rightarrow B,\ C \rightarrow B\}$
    Quicky: warum?

    b) Dekomposition von ABC in AB, BC ist $\pi \bowtie$-treu bzgl. $\{A \rightarrow B,\ B \rightarrow C\}$

- formale Algorithmen zum Erreichen von N1 bis N4:
  - Dekomposition: liefert N2, N4
  - Synthese: liefert N1 bis N4

Noch ein paar formale Definitionen:

Primattribut:
Im Folgenden heiße ein Attribut A *Primattribut* von R, wenn A Attribut irgendeines Schlüssels von R ist.

Transitive Abhängigkeit:
R sei Relationenschema, $X \subseteq R$, F FD-Menge über R; ein $A \in R$ heißt *transitiv abhängig* von X bzgl. F genau dann, wenn es ein $Y \subseteq R$ gibt mit
$X \to Y$, $Y \not\to X$, $Y \to A$, $A \notin XY$.

Mehrdimensionale Abhängigkeit (MD, MVD):
R sei Relationenschema, r(R) Relation, $X,Y \subseteq R$, $Z := R - (X \cup Y)$; r genügt der MD $X \to\to Y$ genau dann, wenn
$\forall$ t1, t2 $\in$ r: [(t1 $\neq$ t2 und t1(X) = t2(X)) $\Rightarrow$
$\exists$ t3 $\in$ r: t3(X) = t1(X) und t3(Y) = t1(Y) und t3(Z) = t2(Z)]

6. Externe Ebene: Anfragesprachen, Updatesprachen

- bisher: Relationenschemata mit Basisrelationen
- jetzt:
  - Änderungen in Basisrelationen (Updates)
  - Ableiten neuer Relationenschemata (Anfrage, Sicht, Snapshot)
- Anfrage: Folge von Operationen, die aus Basisrelationen eine Ergebnisrelation berechnet; Ergebnis kann angezeigt werden, kann Grundlage weiterer Anfragen sein
- Sicht: Folge von Anfrage-Operationen, die unter einem (Sicht-)Namen langfristig abgespeichert wird; kann später unter diesem Namen erneut aufgerufen werden, berechnet Ergebnis (Sichtrelation) jedes Mal neu;
  in Oracle:
  CREATE VIEW viewname AS sfw-Anfrage;
- Snapshot: Ergebnisrelation einer Anfrage, die unter einem Snapshot-Namen abgespeichert wird; wird nie ein zweites Mal berechnet;
  in Oracle: CREATE SNAPSHOT snapshotname AS sfw-Anfrage;

6.1 Kriterien für Anfragesprachen

- Abgeschlossenheit: Ergebnis ist wieder Relation, kann wie Basisrelation als Eingabe für weitere Anfragen dienen

- Adäquatheit: Alle Konstrukte des zugrunde liegenden Datenmodells werden unterstützt

- Orthogonalität: Sprachkonstrukte sind in ähnlichen Situationen auch ähnlich anwendbar

- Optimierbarkeit: Sprache besteht aus wenigen, formal definierten Operationen, für die es Optimierungsregeln gibt

- Effizienz: Jede Grundoperation ist effizient ausführbar (im Relationenmodell Komplexität $\leq O(n^2)$ bei n Tupeln in Relation)

- Mengenorientiertheit: Jede Operation soll auf Mengen von Daten gleichzeitig arbeiten können, nicht nur navigierend auf einzelnen Elementen

- Sicherheit: Keine (syntaktisch korrekte) Anfrage darf ein unendliches Ergebnis liefern oder unendlich lange ausgewertet werden

- Eingeschränktheit: Anfragesprache darf keine komplette Programmiersprache sein

- Vollständigkeit: Anfragesprache muß mindestens die Anfragen ausdrücken können, die eine Standardsprache stellen kann (im Relationenmodell: Relationenalgebra oder Relationenkalkül)

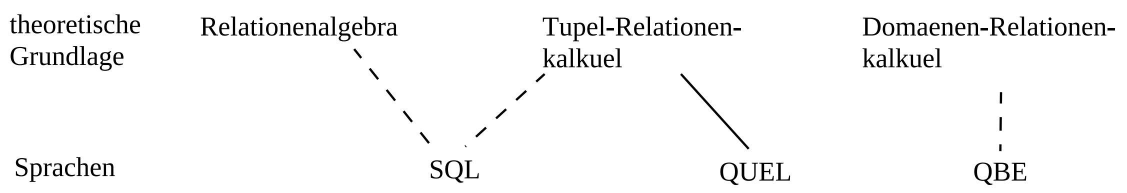
- Deskriptivität: Benutzer soll formulieren, was er haben will, nicht, wie er an das kommt, was er haben will

- Ad-hoc-Formulierung: Benutzer soll Anfrage formulieren können, ohne vollständiges Programm schreiben zu müssen

Anfragesprachen

- im Relationenmodell: Anfragen durch Operationen einer Algebra, eines Kalküls oder einer Anfragesprache stellen

- Entwicklung:



- spätere Techniken: regelbasierte Anfragesprachen (deskriptiv, erlauben Rekursion)

- im Netzwerk-Modell, hierarchischen Modell: keine Anfragesprachen, nur navigierende Operationen

- in objektorientierten Datenbankmodellen: Erweiterungen relat. Anfragesprachen, z.B. OQL

6.2 Relationenalgebra

- entwickelt von E.F. Codd
- Standardversion: 6 Operatoren
- **Projektion** π, Ausblenden von Spalten
  - formal: (R Relation, X $\subseteq$ R, r(R))
    π_X(r) := { t(X) | t $\in$ r }
  - Ergebnis: Menge neuer Tupel über Schema X
  - Beispiel: $\pi_{LAdresse}$(r(Lieferant)) =

LAdresse
...

  - Optimierungsregel : $\pi_X(\pi_Y(\pi_Z(r))) = \pi_X(r)$
    falls $X \subseteq Y \subseteq Z$

Relationenalgebra

- **Konstantenselektion** σ, Heraussuchen von Zeilen
  - formal: c sei X-Wert, $\theta \in \{ <, \leq, =, \geq, >, \neq \}$
    $\sigma_{X\theta c}$(r) := { t | t ∈ r ∧ t(X) θc }
  - Ergebnis: Menge von Tupeln die der Selektionsbedingung genügen
  - Beispiel: $\sigma_{Art.nr=4711}$(r(Liefert)) =

LName	Art.nr.	Preis
...		

    $\sigma_{Preis<40.0}(\pi_{Art.nr.,Preis}$(r(Liefert))) =

Art.nr.	Preis
...	

  - Optimierungsregeln
    * $\sigma_{A\theta a}(\sigma_{B\theta b}(r)) = \sigma_{B\theta b}(\sigma_{A\theta a}(r))$
    * x ⊆ R, A ∈ x: $\sigma_{A\theta a}(\pi_x (r)) =$
      $\pi_x(\sigma_{A\theta a} (r))$
    * x,y ⊆ R, y ⊈ x, A ∈ x ∪ y:
      $\pi_x(\sigma_{A\theta a} (r)) = \pi_x(\sigma_{A\theta a} (\pi_{xy}(r)))$

Relationenalgebra

- **Attributselektion** σ, wie Konstantenselektion, nur mit Vergleich zweier Attribute
  - formal: $\sigma_{A\theta B}$(r) := { t | t $\in$ r $\wedge$ t(A) θ t(B) }
  - Beispiel: $\sigma_{Abtlgsnr=Art.nr}$(r(Führt))
  - Optimierungsregeln: analog zur Konstantenselektion

Relationenalgebra

- **Verbund** (Join) $\bowtie$, Verknüpfen von Tabellen
  - formal: $r_1(R_1)$, $r_2(R_2)$
    $r_1 \bowtie r_2 := \{ t \mid t$ Tupel über $R_1 \cup R_2 \wedge [\forall i \in \{1,2\} \exists t_i \in r_i\colon t_i = t(R_i)]\}$
  - alle Tupel aus r_1, r_2 werden über gleichbenannte Spalten zu einem neuen Tupel verbunden, wenn die Werte in diesen Spalten übereinstimmen
  - Beispiel: (r(Abteilung) $\bowtie$ r(Führt)) $\bowtie$ r(Artikel)
  - Quicky: r(Abteilung) $\bowtie$ r(Artikel) = ??
  - $R_1 \cap R_2 = \{\} \Rightarrow r_1 \bowtie r_2 = r_1 \times r_2$ (kartesisches Produkt)
  - $\pi_{R_1}(r_1 \bowtie r_2) \subseteq r_1$, i.a. nicht =
  - 'dangling Tupel': Tupel, die beim $\bowtie$ mit keinem anderen verbindbar sind, daher herausfallen
  - vollständiger Verbund: jedes Tupel ist mit anderen Tupeln verbindbar; hier gilt dann
    $\pi_{R_1}(r_1 \bowtie r_2) = r_1$
  - $\pi_{R_1}(r) \bowtie \pi_{R_2}(r) \supseteq r$; i.a. nicht =
    Quicky: man konstruiere solche r, R, R_1, R_2
    gilt doch =: Verbund heißt $\pi \bowtie$-treu (verbundtreu)

Relationenalgebra

– Optimierungsregeln:

  – $r_1 \bowtie r_2 = r_2 \bowtie r_1$

  – $(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (\ r_2 \bowtie r_3)$

  – Quicky:
    a) wann gilt $\sigma_F(r_1 \bowtie r_2) = \sigma_{F_1}(r_1) \bowtie \sigma_{F_2}(r_2)$?

    b) wann gilt $\pi_X(r_1 \bowtie r_2) = \pi_{X_1}(r_1) \bowtie \pi_{X_2}(r_2)$?

  – falls F nur Selektionsbedingung auf r_1:
    $\sigma_F(r_1 \bowtie r_2) = \sigma_F(r_1) \bowtie r_2$

  – falls $F = F_1 \wedge F_2$, F_1 Selektionsbedingung auf r_1, F_2 Selektionsbedingung auf r_1 und r_2:
    $\sigma_F(r_1 \bowtie r_2) = \sigma_{F_2}(\sigma_{F_1}(r_1) \bowtie r_2)$

Relationenalgebra

- **Umbenennung**: β, Spalten (Attributnamen) werden umbenannt
  - formal: $\beta_{B \leftarrow A}$(r) := { t' | ∃ t ∈ r: t'(R-{A})=t(R-{A}) ∧ t'(B)=t(A) }
  - β ändert nichts an Relation
  - durch β kann $\bowtie$ ermöglicht werden, wo bisher $\times$, und umgekehrt
  - Beispiel:
    r(Kunde) $\bowtie$ $\beta_{KAdresse \leftarrow LAdresse}$(r(Lieferant))
  - Quicky: formuliere: 'Finde alle Lieferanten, die mindestens 2 verschiedene Artikel liefern.'

Relationenalgebra

- **Mengenoperationen Vereinigung, Durchschnitt, Differenz**
  - formal:
    $r_1 \cup r_2 := \{ t \mid t \in r_1 \vee t \in r_2 \}$
    $r_1 \cap r_2 := \{ t \mid t \in r_1 \wedge t \in r_2 \}$
    $r_1 - r_2 := \{ t \mid t \in r_1 \wedge t \notin r_2 \}$
  - $\cap$ eigentlich überflüssig, da $r_1 \cap r_2 = r_1 - (r_1 - r_2)$
  - Beispiel:
    a) Welcher Lieferant liefert keinen Artikel:
    $\pi_{LName}(r(\text{Lieferant})) - \pi_{LName}(r(\text{Liefert}))$

    b) Gib alle Geschäftspartner (Kunden und Lieferanten) des Warenhauses aus:
    $\pi_{Lname}(r(\text{Lieferant})) \cup$
    $\beta_{LName \leftarrow KName}(\pi_{KName}(r(\text{Kunde})))$
  - Optimierungsregeln: $\circ \in \{ \cup, \cap , -\}$

    * $\sigma_{A='a'}(r_1 \circ r_2) = \sigma_{A='a'}(r_1) \circ \sigma_{A='a'}(r_2)$
    * $\pi_x(r_1 \cup r_2) = \pi_x(r_1) \cup \pi_x(r_2)$

Minimale Algebra besteht aus $\sigma, \pi, \bowtie, \beta, \cup$ und $-$

6.3 Relationenkalküle

- mit Variablen über Tupeln: Tupel-Relationenkalkül (TRK)

- mit Variablen über Attributwerten: Domänen-Relationenkalkül (DRK)

6.4 Update-Operationen

- bisher: Anfragen ohne Änderungen an Basisrelationen
- jetzt: Updates zur Veränderung der Basisrelationen
- Update-Operationen:
  - Einfügen neuer Tupel in Relation
  - Löschen von Tupeln aus Relation
  - Ändern von Tupeln in Relation
- jeweils als
  - Eintupel-Operation (Erfasse neuen Artikel)
  - Mehrtupel-Operationen (Erhöhe den Preis aller Artikel um 2,5%)
- in DBS mit graphischer Benutzeroberfläche: Update-Operationen meist menü- bzw. formulargesteuert
- im folgenden: Update-Operationen in SQL

Update-Operationen

in SQL:

- INSERT: Einfügen eines oder mehrerer Tupel in eine Basisrelation oder Sicht
- UPDATE: Ändern eines oder mehrerer Tupel in einer Basisrelation oder Sicht
- DELETE: Löschen eines oder mehrerer Tupel aus einer Basisrelation oder Sicht

Update-Operationen

INSERT:

- allgemein (Oracle):
  INSERT
  INTO basisrelation [(attribut$_1$,...,attribut$_n$)]
  VALUES (konstante$_1$,...,konstante$_n$);

- INSERT INTO kunde
  VALUES ('Winzig','Osnabrueck', 120.00);
  Werte müssen gleiche Reihenfolge haben wie zugehörige Attribute in CREATE TABLE-Anweisung

- INSERT INTO kunde (KName,Kontostand)
  VALUES ('Hurtig', 20.50);
  Wert von KAdresse wird auf NULL gesetzt

- INSERT INTO fuehrt
  VALUES (9999, 99);
  in ANSI-SQL: keine Integritätsbedingungen -> Nonsens möglich!
  in SQL2:FrEmdschlüssel als Integritätsbedingung in CREATE TABLE gesetzt:
  ERROR at line 1: ORA-02291: integrity constraint (JUTTA.FUEHRT_FK2) violated - parent key not found

Update-Operationen

DELETE:

- allgemein (Oracle):
  DELETE
  FROM basisrelation
  [WHERE bedingung];

- Löschen eines Tupels:
  DELETE FROM kunde WHERE KName = 'Schnell';

- Löschen mehrerer Tupel:
  DELETE FROM kunde WHERE KAdresse = 'Osnabrueck';

- Löschen der gesamten Relation:
  DELETE FROM kunde;

- Verletzen von Integritätsbedingungen:
  DELETE FROM Artikel WHERE Artnr = 19;

Update-Operationen

UPDATE:

- allgemein (Oracle):
  UPDATE basisrelation
  SET attribut$_1$ = ausdruck$_1$,
  ...
  attribut$_n$ = ausdruck$_n$
  [WHERE bedingung];

- Wirkung: in allen Tupel von basisrelation, die bedingung erfüllen, werden die Attributwerte wie angegeben ersetzt

- Beispiel:

  Angestellter:

Angname	Gehalt
Meyer	3000
Schulz	3500
Bond	7200
Schulze	4400

- UPDATE angestellter SET gehalt = gehalt+1000
  WHERE gehalt < 5000;
  Ergebnis: neue Basisrelation

Angname	Gehalt
Meyer	4000
Schulz	4500
Bond	7200
Schulze	5400

- als Eintupel-Operation: Angabe eines Schlüsselwertes hinter WHERE:
  UPDATE angestellter SET gehalt = 8000
  WHERE angname='Bond';

- Änderung der gesamten Relation:
  UPDATE angestellter SET gehalt = 3000;

Update-Operationen

Probleme:

- nicht sehr komfortabel für Anwender
- folgenschwere Fehlermöglichkeiten: Ändern, Löschen einer ganzen Relation als einfachste Möglichkeit

6.5 Relationale Datenbank-Anfragesprachen

- 6.5.1: SQL-89
- 6.5.2: SQL-92
- 6.5.3: SQL3
- 6.5.4: QUEL
- 6.5.5: QBE
- 6.5.6: Weitere Anfragesprachen

Relationale Datenbank-Anfragesprachen

Historie

- SQUARE (1975)
- SEQUEL (1975, IBM Research Labs San Jose)
- SEQUEL2 (1976, IBM Research Labs San Jose)
- SQL (1982, IBM)
- ANSI-SQL (1986)
- ISO-SQL (1989, drei Sprachen Level 1, Level 2, +IEF)
- div. Firmenstandards (IBM SQL, OS/2 SQL, X/Open UNIX SQL, ...)
- SQL2 (SQL-92)
- SQL3 (in Bearbeitung)

SQL-Normen

ANSI/ISO Level 1

- keine Nullwerte
- kein !=, kein EXISTS
- kein UNION
- ...

ANSI/ISO Level 2

- wie bisher vorgestellt

ANSI/ISO Level 2 + IEF

- CHECK-Klausel: WHERE-Klausel als Integritätsbedingungen
- Primärschlüssel, Fremdschlüssel

SQL2 (SQL-92)

- neue Datentypen (INTERVAL, DATE, TIME) (in Oracle: nur DATE)
- ALTER TABLE flexibler (auch Spalten löschbar, nicht in Oracle)
- allgemeine Integritätsbedingungen
- andere String-Operationen

- Namen für abgeleitete Spalten
- Join als eigener Operator (nicht in Oracle)
- neben UNION auch INTERSECT und EXCEPT
- Abgeschlossenheit!!

SQL3 (4 Ausbaustufen)

6.5.1 SQL-89

SQL-Anfrage-Teil (SQL-IQL)

- (fast) vollständig
- nicht ganz abgeschlossen
- kann einige Zusatzfunktionen (Arithmetik, Aggregatfunktionen,...)

Überblick: SFW-Block

- SELECT
  - Projektionsliste
  - arithmetische Operationen und Aggregatfunktionen
- FROM
  - zu verwendende Relationen
  - evtl. Umbenennungen (durch Tupelvariable oder 'alias')
- WHERE
  - Selektionsbedingungen
  - Verbundbedingungen
  - geschachtelte Anfragen (wieder ein SFW-Block)

- GROUP BY
  - Gruppierung für Aggregatfunktionen
- HAVING
  - Selektionsbedingung an Gruppen

SQL-89

Syntax allgemein:

SELECT *
FROM relationenliste
[WHERE bedingungen]
[GROUP BY attributliste [HAVING bedingungen]]

Beispiele:

1. SELECT * FROM Artikel;
   gibt gesamte Relation Artikel aus (z.B. 20 Einträge)

2. SELECT * FROM Artikel, fuehrt;
   berechnet kartesisches Produkt (20 * 2 =40 Tupel) statt Verbund (2 Tupel)

3. SELECT * FROM Angestellter eins, Angestellter zwei;
   berechnet kartesisches Produkt von Angestellter mit sich selbst, liefert bei 4 Angestellten 16 Tupel, 4 Spalten (eins.angname, eins.gehalt, zwei.angname, zwei.gehalt)

SQL-89

SELECT

- Syntax: SELECT [DISTINCT] attribute, arithm. ausdrücke, aggr.-funktionen FROM....;

- SELECT abtlgnr FROM fuehrt; ergibt

abtlgnr
5
5

  keine Projektion: doppelte werden nicht eliminiert (einfacher zu implementieren, wird manchmal bei Aggregatfunktionen benötigt)

- SELECT DISTINCT abtlgnr FROM fuehrt; ergibt genau Projektion aus Relationenalgebra

- SELECT Artnr, Preis /1.65 FROM liefert; ergibt Preise der Artikel in Dollar

- SELECT avg(Preis) FROM liefert WHERE Artnr=2; ergibt Durchschnitts-Lieferantenpreis für Eier

avg(preis)
1.335

SQL-89

WHERE

- Syntax: SELECT...FROM...WHERE bedingung
- bedingung kann sein:
  - Konstantenselektion (*): attribut $\{=,!=,<,>,<=,>=\}$ konstante
  - Attributselektion (*): attribut1 $\{=,!=,<,>,<=,>=\}$ attribut2
  - Verbundbedingung: relation1.attribut = relation2.attribut

    ```
    SELECT Artikel.ArtName FROM Artikel, liefert
    WHERE Artikel.Artnr = liefert.Artnr;
    ```

    gibt alle lieferbaren Artikel (deren Bezeichnung) aus
  - Bereichsselektion (*): attribut BETWEEN konstante1 AND konstante2
  - Ungewißheitsselektion (*): attribut LIKE spezialkonstante
    spezialkonstante kann auch '%' beinhalten, steht für kein oder beliebig viele Zeichen
    SELECT * FROM Artikel WHERE ArtName LIKE '%milch%';
    ergibt

Artnr	ArtName
1	1 L frische Vollmilch
5	0.5 L frische Vollmilch

- Nullselektion (*): attribut IS NULL
- Quantifizierte Bedingungen (*): ALL, ANY, SOME, EXISTS
- Verknüpfungen von Bedingungen (*): NOT, OR, AND

(*): in Oracle entsprechend

SQL-89

Variationen bei Verbunden

- bisher: $\bowtie$ (natürlicher Verbund)
- in SQL-89: Verbundbedingung mit = (Gleichverbund)
- auch möglich: allgemeinerer θ-Verbund durch Verbundbedingung mit θ
- ab SQL-92: auch äußere Verbunde möglich (outer joins)
  - Outer Join: Dangling Tupel beider Operanden werden zusätzlich in das Verbundergebnis übernommen und mit Nullwerten aufgefüllt
  - Left (Right) Outer Join: Nur die Dangling Tupel des linken (rechten) Operanden werden zusätzlich in das Ergebnis übernommen und mit Nullwerten aufgefüllt
- Beispiele (in Oracle):

  Right Outer Join:
  SELECT Artikel.Artnr, LName, Preis
  FROM Artikel, liefert
  WHERE Artikel.Artnr (+) = liefert.Artnr;

Left Outer Join:
SELECT Artikel.Artnr, LName, Preis
FROM Artikel, liefert
WHERE Artikel.Artnr = liefert.Artnr (+);
ergibt

Artnr	LName	Preis
1	Suedmilch	.99
1	Kerry	.99
2	Suedmilch	1.29
2	Kerry	1.38
3	Harry	1.25
4	Bauer Doenges	1.73
5	Bauer Doenges	.64
6		
7		
...		
20		

SQL-89

Schachtelung

- Geschachtelte Bedingung: attribut IN (sfw-block)
- Beispiel:
  SELECT ArtName FROM Artikel
  WHERE Artnr IN (SELECT Artnr FROM liefert);
- verwirklicht ebenfalls Verbund mit anschließender Projektion
- Abarbeitung:
  - zunächst bestimmt System das Ergebnis der inneren SELECT-Anweisung
  - Ergebnis wird hinter IN als Konstantenliste eingesetzt
  - Anfrage wird zu
    SELECT ArtName FROM Artikel
    WHERE Artnr IN (1,2,3,4,5);

SQL-89

- Verzahnung von innerer und äußerer Abfrage: innere Abfrage ist von äußerer abhängig

```
SELECT Preis FROM liefert
WHERE Artnr IN (SELECT Artnr FROM enthaelt,erteilt
          WHERE enthaelt.auftragnr=erteilt.auftragnr
          AND KName=LName);
```

```
SELECT LName FROM liefert X
WHERE X.Artnr IN
          (SELECT Y.Artnr FROM liefert Y
          WHERE Y.Preis=X.Preis
          AND Y.LName != X.LName);
```

ergibt

LName
Suedmilch
Kerry

SQL-89

Gruppierung

- GROUP BY: Gruppierung der Tabelle, Aggregatfunktionen wirken auf einzelnen Gruppen
- Syntax: SELECT...FROM...WHERE...GROUP BY attribute
- Beispiel:
  SELECT count(*),LName FROM liefert
  GROUP BY LName;
  ergibt

count(*)	LName
2	Bauer Doenges
1	Harry
2	Kerry
2	Suedmilch

- HAVING: Selektion von Gruppen
- Syntax: SELECT...FROM...WHERE...
  GROUP BY...HAVING bedingung
- Beispiel:
  SELECT count(*),LName FROM liefert
  GROUP BY LName HAVING count(*) >1;

  liefert nur die Tupel mit Kerry, Suedmilch, Bauer Doenges
- Quicky: was liefert

  SELECT Artnr FROM liefert
  GROUP BY Artnr HAVING avg(Preis) <
  (SELECT avg(Preis) FROM liefert);

SQL-89

Mengenoperation:

- nur UNION, keine Differenz, kein Durchschnitt
  Syntax: sfw-block1 UNION sfw-block2
- SFW-Blöcke müssen gleiche Anzahl von Attributen mit den gleichen Domänen haben
- UNION nur als 'äußerste' Operation erlaubt ⇒ SQL-89 nicht abgeschlossen

SQL-89

Quantoren, Mengenvergleiche:

Relationenschemata hier Lieferant={LName, LAdresse},
Artikel={Artnr, Artname, Verkaufspreis},
liefert={LName, Artnr, Lieferpreis, Anzahl}

- Existenzquantor

  ```
  SELECT LAdresse
  FROM Lieferant
  WHERE LName = ANY ( SELECT LName FROM liefert);
  ```

  gibt die Orte aus, aus denen Lieferanten kommen, die bereits Ware geliefert haben

- Allquantor

  ```
  SELECT Preis
  FROM liefert
  WHERE LName='Suedmilch'
  AND Preis >= ALL (SELECT Preis FROM liefert WHERE
  LNme='Suedmilch');
  ```

  gibt den höchsten Preis aus, den ein von 'Suedmilch' geliefertes Produkt hat

- Mengenvergleiche müssen simuliert werden

SQL-89

Aggregatfunktionen, arithm. Operationen:

- auch: Build-In-Funktionen
  - count: Anzahl der Werte einer Spalte
  - count(*): Anzahl der Tupel einer Relation
  - sum: Summe der Werte einer Spalte (auf num. Domänen)
  - avg: arithmetisches Mittel der Werte einer Spalte (auf num. Domänen)
  - max: größter Wert einer Spalte
  - min: kleinster Wert einer Spalte
- bis auf count(*): ALL (Voreinstellung) oder DISTINCT möglich
- Nullwerte werden vorher eliminiert (bis auf count(*))
- Beispiele:
  - Gib die Anzahl aller Kunden aus:
    SELECT count(*) FROM Kunde;
  - Gib die Anzahl der liefernden Lieferanten aus:
    SELECT count(DISTINCT LName) FROM liefert;

- Forts. Beispiele
  - Gib den durchschnittlichen Lieferpreis für Vollmilch aus:

    SELCT avg(Lieferpreis)
    FROM liefert, Artikel
    WHERE liefert.Artnr=Artikel.Artnr AND Artikel.ArtName
    LIKE '%Vollmilch%';

- arithmetische Operationen: + (unär, binär),- (unär, binär),*,/

- in Oracle: -/+ als Vorzeichen, *, /, +, -, || (Konkatenation von char-Strings), +, !=, >, <, >=, <=, [NOT] IN, ANY, SOME, ALL, [NOT] BETWEEN x AND y, EXISTS, [NOT] LIKE, IS [NOT] NULL, AND, OR, NOT,
  numerische Funktionen wie ABS, COS, SIN, EXP, SQRT, ROUND,...
  Char/String-Funktionen wie CHR, CONCAT, LOWER, UPPER, ASCII, LENGTH, ...
  Datums-Funktionen wie ADD_MONTH, NEXT_DAY, SYSDATE, TO_DATE,...
  Gruppierungen wie AVG, COUNT, MAX, MIN, SUM, STDDEV, VARIANCE
  sonstige wie GREATEST, LEAST, NVL, UID, USER,...

SQL-89

Verbunde mit derselben Relation

- auch: Self-Join

- eine Anwendung: Arbeiten auf Datenmengen (mit IN, s. vorne)

- andere Anwendung:'Zählen' ohne count

  ```
  SELECT DISTINCT X.LName
  FROM liefert X
  WHERE X.Artnr IN (SELECT Y.Artnr FROM liefert Y
  WHERE Y.LName != X.LName);
  ```

  ohne IN und Schachtelung formulierbar

  ```
  SELECT DISTINCT X.LName
  FROM liefert X, liefert Y
  WHERE X.Artnr=Y.Artnr
  AND X.Lname != Y.Lname;
  ```

SQL-89

Sortierung

- ORDER BY attributliste [ASC | DESC]
- letzte Klausel nach SELECT,FROM,WHERE,GROUP BY, HAVING, UNION
- Beispiel:

```
SELECT Artnr, LName
FROM liefert
ORDER BY LName ASC;
```

- Sortierung aufsteigend: ASC (Voreinstellung), absteigend: DESC
- Sortierung nach mehreren Attributen möglich

```
SELECT Artnr, LName
FROM liefert
ORDER BY LName DESC, Artnr DESC;
```

- Sortierung wird auf Ergebnis der SFW-Anfrage angewendet, daher nicht erlaubt:

```
SELECT Artnr
FROM liefert
ORDER BY LName ASC;
```

- Hat eine Spalte keinen Namen: Spaltennummer verwenden

```
SELECT Artnr, Preis/1.65
FROM liefert
ORDER BY 2;
```

6.5.2 SQL-92

Erweiterungen bei Verbunden:

- explizites kartesisches Produkt (in Oracle: nur auf 1. Art):

  SELECT * FROM liefert, Artikel;

  oder

  SELECT * FROM liefert CROSS JOIN Artikel;

- natürlicher Verbund:
  SELECT * FROM Artikel, enthaelt
  WHERE Artikel.Artnr = enthaelt.Artnr;

  oder

  SELECT * FROM Artikel JOIN enthaelt
  ON Artikel.Artnr = enthaelt.Artnr;

  oder

  SELECT * FROM Artikel JOIN enthaelt
  USING (Artnr);

  oder

  SELECT * FROM Artikel NATURAL JOIN enthaelt;

- statt INNER auch OUTER JOIN (LEFT, RIGHT)
(in Oracle mit (+)-Angabe rechts bzw. links, s. vorne)

- mit Tupelvariablen beim SELECT (nicht in Oracle):
SELECT Ergebnis.Artnr
FROM (Artikel NATURAL JOIN enthaelt);
AS Ergebnis

SQL-92

- UNION an beliebiger Stelle (in Oracle: zwischen 2 bel. SFW-Statements)

- SELECT-Ausdrücke auch hinter FROM (in Oracle)

- INTERSECT und EXCEPT als Mengenoperationen für Durchschnitt und Differenz (in Oracle INTERSECT, MINUS)

- Angabe der Attribute bei Mengenoperationen (CORRESPONDING BY) (in Oracle: nur über alias AS...)

- Benennungen:
  SELECT COUNT(*) AS Anzahl..... (in Oracle)

- dreiwertige Logik (IS TRUE, IS FALSE, IS UNKNOWN) (in Oracle: FALSE, TRUE , UNKNOWN)

- Darstellung von Intervallen, Anfragen an Intervalle (OVERLAPS) (in Oracle nur: [NOT] IN, [NOT] BETWEEN x AND y)

6.5.3 SQL3

- enthält auch objektorientierte Features wie (Value- u. Object-)ADTs, ADT-Hierarchien, Einkapselung, Polymorphismus,...

- daneben relationale Erweiterungen wie Tabellen-Hierarchien, weitere Join-Typen (USING PRIMARY KEY/FOREIGN KEY/CONSTRAINT), rekursive Anfragen, Multimengen, Mengen u. Listen als komplexere Datentypen, Aufzählungstypen, Trigger, ...

- Produkte vermutlich erst in gut 5 Jahren erhältlich

SQL3

Beispiel:

```
CREATE OBJECT TYPE Person_adt
WITH OID NOT VISIBLE
(
PUBLIC Ausweisnr INTEGER PRIMARY KEY
PUBLIC Name VARCHAR2(30)
PRIVATE Gebdatum DATE
PUBLIC Alter INTEGER VIRTUAL
GET WITH Ermittle_Alter CHECK(VALUE.Alter > 0)
PUBLIC Vater Person_adt

PUBLIC ACTOR FUNCTION Ermittle_Alter (:p Person_adt)
RETURNS INTEGER;
RETURN (CURRENT_DATE - :p.Gebdatum)

PUBLIC ACTOR FUNKTION Einkommen (:p Person_adt)
RETURNS INTEGER;
RETURN 0

.....
);
```

```
CREATE OBJECT TYPE Student_adt
WITH OID NOT VISIBLE UNDER Person_adt
(
PUBLIC Matrnr INTEGER
PUBLIC Fach VARCHAR2(30)
PUBLIC Bafoeg INTEGER

PUBLIC ACTOR FUNKTION Einkommen (:p Student_adt)
RETURNS INTEGER;
RETURN :p.Bafoeg
);
```

```
CREATE TABLE Personen (
Person Person_adt INSTANCE
);
```

```
INSERT INTO Personen( Matrnr(Person), Name(Person), Geb-
datum(Person))
VALUES Person_adt(007,'James Bond', 05-Feb-45);
```

6.5.4 QUEL

- QUEry Language, von INGRES
- ist vollständige Datenbanksprache (IQL + DDL + VDL + DML + SSL), im folgenden nur IQL
- basiert auf Tupelrelationenkalkül:
  - alle Tupelvariablen sind ∃-quantifiziert, an Relationen gebunden
  - Bindung durch
RANGE OF tupelvariable IS relationenname
  - Einschränkung einer Tupelvariablen t auf ein Attribut A: t.A
- Artikel={Artnr, EAN, Artname}
liefert={LName, Artnr, Preis}
Lieferant={LName, LAdresse}
mit den Bindungen
RANGE OF A IS Artikel
RANGE OF A1 IS Artikel
RANGE OF L IS liefert
RANGE OF La IS Lieferant
- Syntax einer QUEL-Anfrage

  RETRIEVE (liste_von_tupelvariablen)
WHERE bedingung

QUEL

Beispielanfragen:

- Was ist der Name des Artikels mit der Nummer 4711?
  RETRIEVE (A.Artname)
  WHERE A.Artnr = 4711

- Welche Namen und Preise haben Artikel, die von Osnabrücker Lieferanten geliefert werden?
  RETRIEVE (A.Artname, L.Preis)
  WHERE A.Artnr = L. Artnr
  AND L.LName = La.LName
  AND La.LAdresse = 'Osnabrueck'

- mit Umbenennung: Gibt es unterschiedliche Artikel mit dem gleichen Artikelnamen?
  RETRIEVE (T1 = A.Artnr, T2 = A1.Artnr)
  WHERE A.Artname = A1.Artname
  AND A.Artnr <> A1.Artnr

- Gib alle Attribute des Artikels 'Leitz-Ordner A4' aus!
  RETRIEVE A.ALL
  WHERE A.Artname = 'Leitz-Ordner A4'

QUEL

- arithmetr. Ausdrücke (wie in SQL) in RETRIEVE und in Bedingung erlaubt

- Was kostet durchschnittlich der Artikel 0815?
  RETRIEVE (Durchschnitt = AVG(L.Preis
  WHERE L.Artnr = 0815))

- Gruppierung mittels BY: Wieviele Lieferanten gibt es pro Artikel?
  RETRIEVE (L. Artnr, Anzahl = COUNT (L.LName
  BY L.Artnr))

6.5.5 QBE

- Query-By-Example, IBM-eigene Konkurrenz zu SQL

- ist ebenfalls IQL + DDL + DML + VDL + SSL, hier nur IQL

- Idee: Anfrage formulieren durch Ausfüllen einer Tabelle mit Variablen und Konstanten, basiert auf Domänenrelationenkalkül

- Variablen sind ∃-quantifiziert, Name mit _ eingeleitet, stellen beispielhafte Werte aus der Domäne dar

- Konstanten: ohne _

- Konstante oder Variable mit Präfix `P.` werden ausgegeben

- Variablen, die nur einmal in Anfrage auftauchen, können durch blanks ersetzt werden

Beispielanfragen:

- SELECT Artname FROM Artikel; entspricht:

Artikel

Artnr	EAN	Artname
		P._otto

- SELECT EAN FROM Artikel WHERE Artnr = 4711; entspricht:

Artikel

Artnr	EAN	Artname
4711	P._ean	

- Verbund durch

Artikel

Artnr	EAN	Artname
_999		

liefert

LName	Artnr	Preis
	_999	

QBE

- Welche Artikel(Nr.) werden von Lieferanten aus Osnabrück oder Münster geliefert?

liefert

LName	Artnr	Preis
_meyer	P.	
_mueller	P.	

Lieferant

LName	LAdresse
_meyer	Osnabrueck
_mueller	Muenster

- Welche Artikel(Nr.) werden von Lieferanten aus Osnabrück und Münster geliefert?

liefert

LName	Artnr	Preis
_meyer	P._999	
_mueller	P._999	

Lieferant

LName	LAdresse
_meyer	Osnabrueck
_mueller	Muenster

- Gib alle Artikel (Nr.) aus, deren Artikelname 'Hemd' enthält

Artikel

Artnr	EAN	Artname
P.		_xy Hemd_xz

- Gib Namen und Preise der Artikel aus, die Meyer liefert

Artikel

Artnr	EAN	Artname
_999		_nam

liefert

LName	Artnr	Preis
'Meyer'	_999	_pr

Ergebnis

	Name	Preis
P.	_nam	_pr

QBE

- kompliziertere Bedingungen: mittels Condition-Box
  Welche Artikelnr. haben die gleiche Bezeichnung (Artname)?

Artikel

Artnr	EAN	Artname
_999		_nam
_998		_nam

Conditions
_999 < _998

Ergebnis P.

Artnr1	Artnr2
_999	_998

- Sortiere die liefert-Relation aufsteigend nach LName und dann aufsteigend nach Artnr

liefert

LName	Artnr	Preis
AO1	AO2	

AO: Ascending Order, DO: Descending Order

- Was ist der durchschnittliche Preis vom Artikel Nr. 4711?

liefert

LName	Artnr	Preis
	4711	P.AVG.ALL._pr

- Was ist der durchschnittliche Preis pro Artikelnr.?

liefert

LName	Artnr	Preis
	P.G.	P.AVG.ALL.

6.5.6 Weitere Anfragesprachen

1. Universalrelationen-Anfragesprachen:

   - Idee: Formulierung der Anfragen ohne explizite Verbunde
   - vorzustellen: Universalrelation mit allen in DB vorhandenen Attributen
   - noch nötige Operationen: i.w. Projektion, Selektion
   - Bsp.: DB-Schema mit Relationen Studenten, Prüfer, Prüfungen

     statt

     ```
     SELECT SName
     FROM Studenten, Pruefer, Pruefungen
     WHERE Studenten.Matrnr = Pruefungen.Matrnr
     AND Pruefungen.Pruefernr = Pruefer.Pruefernr
     AND Pruefer.PName = 'Schreiner';
     ```

     nur kurz

     ```
     SELECT SName
     WHERE PName = 'Schreiner';
     ```

   - zugehörige Relationen und Verbunde muß System selber bestimmen
   - Voraussetzungen: Attributnamen global eindeutig, Schema azyklisch

Weitere Anfragesprachen

2. Regelbasierte Anfragesprachen:

- Idee: wie regelbasierte Programmiersprachen (z.B. PROLOG)
- Grundlagen (hier von DATALOG):
  - Prädikate (Atome): beschreiben Relation oder eine Selektion auf einer Relation (abgeleitete Relation); daneben: Vergleichsprädikate der Form X = Y, X < Y, X < 3, ...
  - Literal: Atom oder negiertes Atom
  - Klausel: Disjunktion von Literalen
  - Horn-Klausel: Klausel mit höchstens einem nicht-negativen Literal;
    - nur ein nicht-negatives Literal: Fakt
    - kein nicht-negatives Literal: Integritätsbedingung
    - ein nicht-negatives und ein oder mehrere negative Literale: Regel
    - Beachte: $\neg P_1 \vee ... \vee \neg P_n \vee P \Longleftrightarrow$
$P_1 \wedge ... \wedge P_n \Rightarrow P$
geschrieben als:
P :- P_1, ..., P_n.
- DATALOG-Programm: Menge von Horn-Klauseln
- Variablen die nur im Rumpf vorkommen sind $\exists$-quantifiziert, alle anderen $\forall$-quantifiziert

- Beispiel (Basisrelation: Eltern(Sohn_Tocher, Elternteil)):

  Eltern(Kain,Adam).
  Eltern(Kain,Eva).
  Eltern(Abel,Adam).
  Eltern(Abel,Eva).
  Eltern(Muth,Kain).
  Eltern(Joseph,Kain).
  Eltern(Maria,Muth).
  Eltern(Magda,Muth).
  Eltern(Magdalena, Joseph).
  Eltern(Jacob,Joseph).
  Geschwister(X,Y) :- Eltern(X,Z), Eltern(Y,Z), X $\neq$ Y.
  Cousin(X,Y) :- Eltern(X,A), Eltern(Y,B), Geschwister(A,B).
  Cousin(X,Y) :-Eltern(X,A), Eltern(Y,B), Cousin(A,B).
  Verwandt(X,Y) :- Geschwister(X,Y).
  Verwandt(X,Y) :- Verwandt(X,Z), Eltern(Y,Z).
  Verwandt(X,Y) :- Verwandt(Z,Y), Eltern(X,Z).

- rekursive Regel: Regel kann nicht mehr 'auf einmal' ausgewertet werden ->
  Start der Auswertung: alle abgeleiteten Relationen sind leer, Ausführung aller Regeln 1x (parallel);
  danach: alle Regeln wiederholt auf allen bisher abgeleiteten Relationen ausführen;
  Ende: wenn Fixpunkt erreicht ist (abgeleitete Relationen sich nicht mehr verändern).

7. Interne Ebene

- zentrale Aufgabe: Abbildung der Strukturen der konzeptuellen Ebene in Strukturen der internen Ebene und umgekehrt

konzeptuelle Ebene		interne Ebene
Relationen	$\longrightarrow$	Dateien (Files)
Tupel	$\longrightarrow$	Sätze (Records)
Attributwerte	$\longrightarrow$	(Satz-)Felder

- Abbildung
  - i.a. nicht bijektiv
  - Einflußfaktoren: Organisationsformen, Adressierung, Codierung

- Datensatz: Folge von Datenfeldern mit bestimmtem Typ und Länge; Sätze müssen in Blöcke eingepaßt werden (blocken)

- Blocken abhängig von
  - variablen/festen Feldlängen $\Rightarrow$ variable/feste Satzlängen
  - Blockungstechniken (Nicht-Spannsätze, Spannsätze)

Interne Ebene

- Dateioperationen:
  - Einfügen eines Datensatzes
  - Löschen eines Datensatzes
  - Modifizieren eines Datensatzes
  - Suchen und Finden eines Satzes ('Lookup')

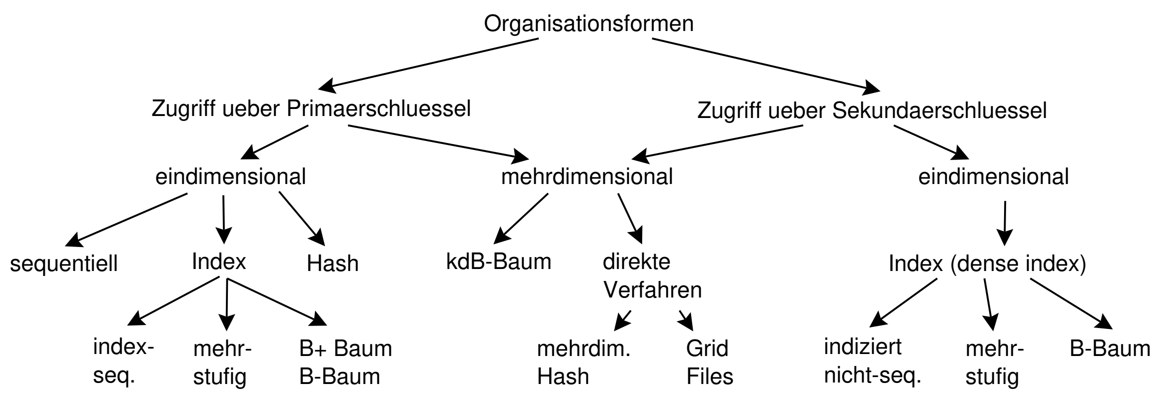
- Zugriff auf Datensätze
  - Standard: Datensätze in Abhängigkeit vom Primärschlüsselwert in einer Datei geordnet/gehasht gespeichert
  - manchmal wünschenswert: schneller Zugriff über andere Attribut(mengen) $\rightarrow$ Sekundärschlüssel (muß kein echter Schlüssel sein!)

Interne Ebene

Einordnungskriterien für Zugriffsverfahren:

- Primärschlüssel/Sekundärschlüssel
- eindimensional/mehrdimensional
- statisch/dynamisch

Verfahren:



Interne Ebene

Grundlagen:

- Sekundärspeicher: billig, langsam; aufgebaut aus Zylindern, Spuren, Blöcke (128 Bytes - 4 KB); phys. Datei: Folge von Blöcken

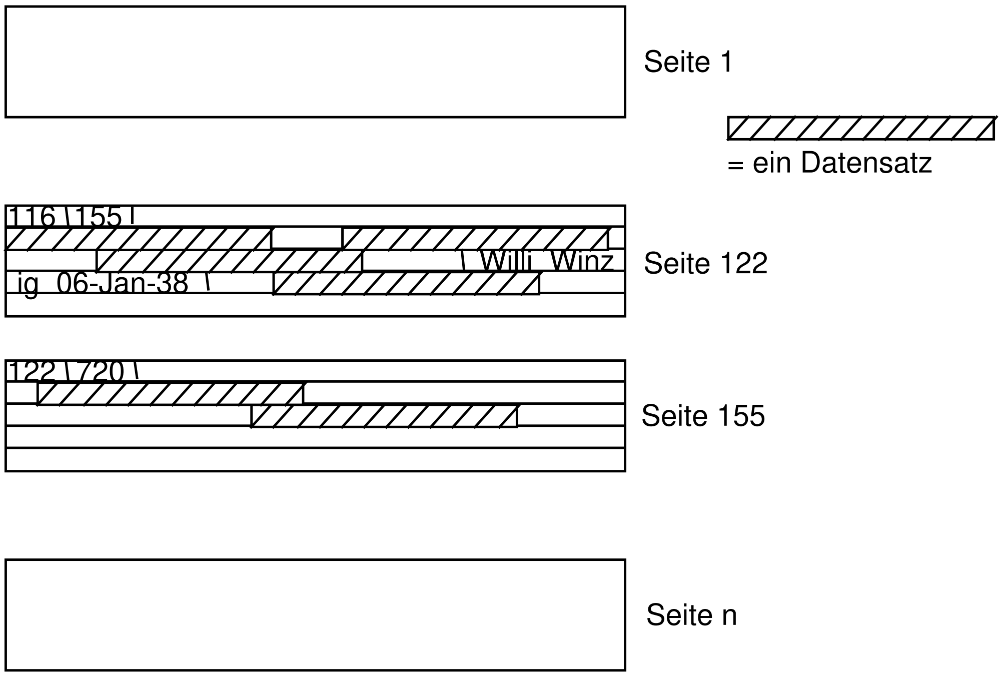
- Block: Folge von Sätzen (bei Nicht-Spannsätzen)

- Haupspeicher: schnell, teuer; bestehend aus Seiten

- Block wird im DBS Seite (im HS) zugeordnet: in der Regel 1Seite $\cong$ 1 Block, aber auch 1 Seite $\cong$ 2,4,.. Blöcke

- Seite besteht aus
  - Header (Infos über Vorgänger-/Nachfolgeseite, Nummer, Art der Sätze, freier Platz,..)
  - Datensätze
  - unbelegte Bytes

- Modell des Speichers:



Adresse des Willi-Winzig-Datensatzes etwa 122.107, mit 122 Seitenadresse = Blockadresse, 107 Offset in Bytes vom Seitenanfang

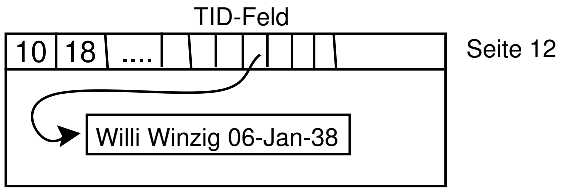
Interne Ebene

Zeiger auf Seiten:

- absolute Seitenadresse
- Offset bzgl. eines virtuellen Speichers

Zeiger auf Sätze:

- Zeiger auf Seite + Offset
- Zeiger auf Seite (Satz frei beweglich auf Seite)
- Zeiger auf Seite + Offset im TID-Feld (Teil des Headers);
  Bsp.: Adresse des Datensatzes 12.4

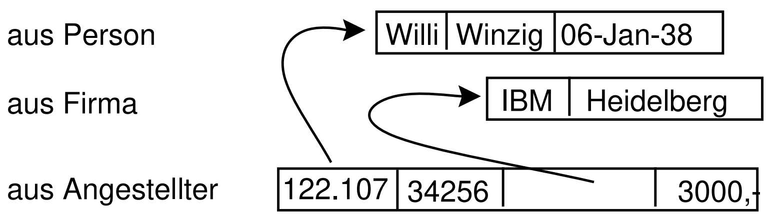


Interne Ebene

Satztypen:

- pinned records: fixierte Sätze ; Sätze sind an ihre Position gebunden, da auf sie mit einem direkten Zeiger verwiesen wird; Problem beim Löschen: dangling pointer;

  Bsp.:



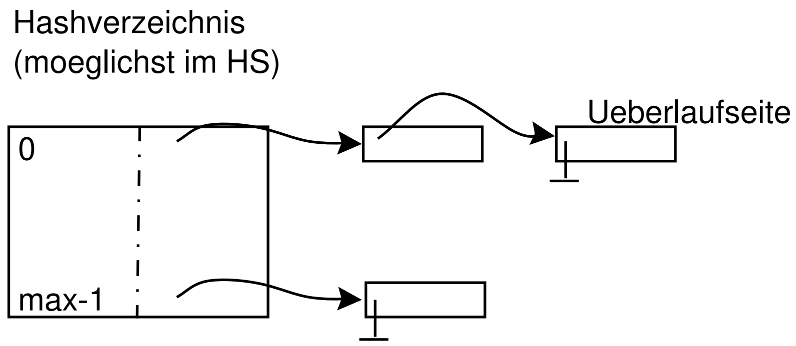
- unpinned records: unfixierte Sätze

- Sätze fester Länge: alle Datenfelder haben feste Länge; möglicher Aufbau aus Informationsfeld (Format, Länge,...), Freiraum, relevanten Daten

- Sätze variabler Länge: 2 Formen
  - jedes Datenfeld variabler Länge beginnt mit Längenzeiger; evtl. am Satzanfang Längenangabe für gesamten Satz
  - am Satzanfang ein Zeigerfeld für alle variabel langen Datenfelder

7. 1 Interne Ebene: Hashorganisation (eindimensional, statisch)

- exakte Definition, Varianten, usw. s. Vorlesung Informatik A
- Prinzip: Hashfunktion, bildet Menge der Primärschlüsselwerte ab auf Menge von Bucketadressen (1 Bucket = 1 oder mehrere Seiten, i.F. 1 Seite)
- Implementierung:



Hashorganisation (eindimensional, statisch)

Operationen:

- Lookup: 'w Primärschlüsselwert, finde Satz mit diesem Wert'
  Hashwert von w berechnen, zugeordneten Zeiger im Hashverzeichnis holen, über Zeiger erste Seite holen, Sätze auf Seite durchsuchen, falls nicht gefunden Nachfolgeseite holen und durchsuchen,..., falls keine Nachfolgeseite da: Satz nicht gefunden.

- Modifizieren: falls modifizierte Attribute Teil des Primärschlüssels: unveränderten Satz löschen, modifizierten Satz neu einfügen;
  falls nicht: Lookup ausführen, falls Satz gefunden: Attributwerte ändern

- Einfügen: Lookup durchführen,
  falls Satz gefunden: Fehler, da Primärschlüsselwert schon vorhanden,
  falls Satz nicht gefunden: Hashwert von w berechnen, Zeiger holen, in (Überlauf-)Seiten bis zum letzten Satz suchen, Satz dahinter eintragen

- Löschen: Lookup durchführen,
  falls Satz gefunden: Bit im Belegungsvektor auf 0 setzen (Vorsicht bei pinned records)

Zeitkomplexität:
wenn Hashverzeichnis nicht im HS: jede Operation benötigt einen Seitenzugriff für relevanten Teil des Hashverzeichnisses;
für Lookup: maximal $1 +$ Anzahl Überlaufseiten Seitenzugriffe

Hashorganisation (eindimensional, statisch)

Probleme:

- Anpassung der Hashfunktion an wachsende oder schrumpfende Datenmengen, wenn max verändert wird: komplette Reorganisation notwendig (alle Datensätze neu hashen)

- Leistungsfähigkeit abhängig von Wahl der Hashfunktion, bei schlechter Wahl evtl. fast alle Sätze auf gleicher Seite (+Überlaufseiten)

- Bestimmte Anfragen umständlich zu beantworten, z.B.
  'Gib alle Sätze sortiert aus' oder
  'Gib alle Milch-Artikel aus' (Primärschl. ist Artnr; Bereichsanfrage)

7.2 Interne Ebene: Hashorganisation (eindimensional, dynamisch)

- gegen erstes Problem bei statischem Hashen: Einsatz dynamischer Hashfunktionen

- Lineares Hashen (Litwin, 1980):

  - h_i: dom(Primärschlüssel) $\rightarrow$ {0,..., 2^i*N} Folge von Hashfunktionen, N Anfangsgröße des Hashverzeichnisses

  - Heuristik: es gilt
    $h_{i+1}(w) = h_i(w)$ für etwa die Hälfte aller $w \in$ dom(prim.)
    $h_{i+1}(w) = h_i(w) + 2^i*N$ für die andere Hälfte
    mit $h_i(w) = w \bmod 2^i*N$
    für ein w sind höchstens zwei Hashfunktionen zuständig, Entscheidung zwischen diesen beiden durch Parameter

    * sp: Splitpointer, gibt an, welche Seite als nächstes geteilt wird, initial. mit 0

    * lv: Level, gibt an, welche Hashfunktion benutzt wird, initial. mit 0

  - Lookup: geg.: Primärschlüsselwert w
    gesucht: Hashnummer s im Hashverzeichnis:

```
IF (sp==0) s = h_lv(w);
           ELSE { s = h_lv-1(w);
                  IF (s < sp) s = h_lv(w);
                  }
```

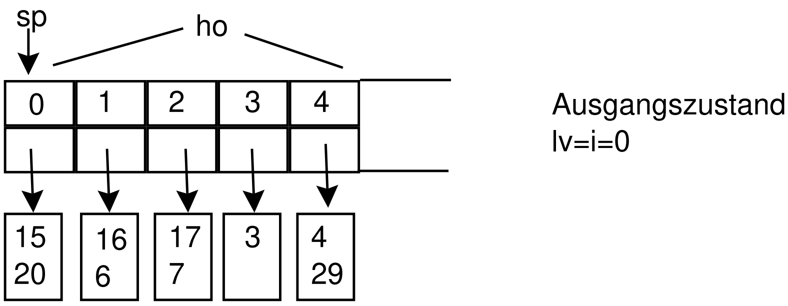
Hashorganisation (eindimensional, dynamisch)

- Einfügen, Löschen: wie Lookup,
  - falls Füllfaktor (Anzahl abzuspeichender Tupel/durchschn. Anzahl mögl. Sätze auf Seite) höher als vorgegebener Maximalwert automatisch (lokale) Reorganisation: Sätze, auf die sp zeigt, werden gemäß h_{lv} neu verteilt (d.h. etwa die Hälfte der Sätze wird auf Seiten unter Hashnr. 2^{lv-1}*N+sp geschoben),
    sp = sp+1
    IF (sp==2^{lv-1}*N) {lv = lv+1; sp = 0;}

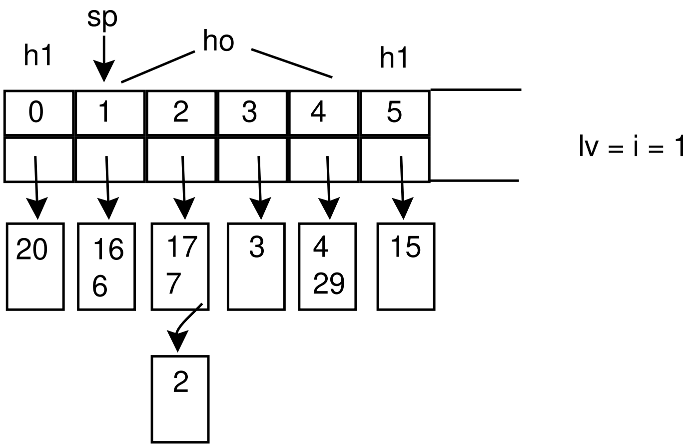
    Ausnahme: erstes Splitten: lv = lv+1, sp=sp+1
  - falls Füllfaktor niedriger als vorgegebener Minimalwert, wird autom. lokal reorganisiert: zwei Hashnummern und ihre Seiten wieder zusammenlegen,
    sp = sp -1
    evtl. lv = lv -1

Hashorganisation (eindimensional, dynamisch)

Beispiel: $h_i(w)$ = w mod 2^i*5 (N=5)



Annahme: beim Einfuegen von 2 wird Fuellfaktor ueberschritten



$h_0(w)$ = w mod 2^0*5
$h_1(w)$ = w mod 2^1*5

Quicky: was passiert bei Suche nach 15, 20, 3, 2?

Nachteil des lin. Hasing: evtl. werden Seiten gespittet, die gar nicht voll sind

7.3 Interne Ebene:Indexsequentielle Dateiorganisation

- allgemein: mind. zweistufiger Baum, wobei Blattebene Hauptdatei mit eigentl. Datensätzen enthält, jede andere Baumstufe Indexdateien

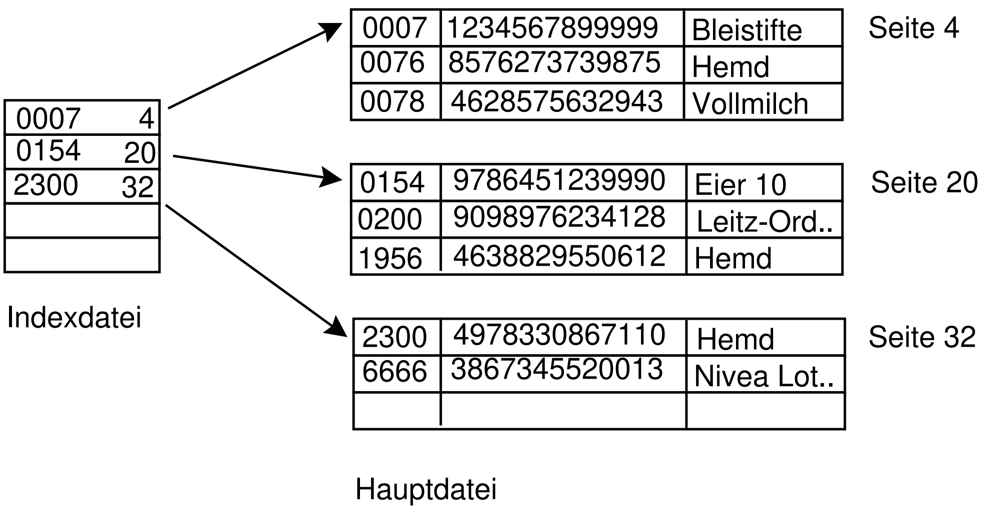
- einstufiger Index: 1 Hauptdatei + 1 Indexdatei

- n-stufiger Index: 1 Hauptdatei + n Indexdateien

- Aufbau Hauptdatei: Datensätze sequentiell nach Primärschlüssel sortiert

- Aufbau Indexdatei: Datensätze der Form (Primärschlüssel, Seitennummer),
  zu jeder Seite in Hauptdatei ex. nur genau ein Datensatz in Indexdatei (enthält Primärschl.wert des 1. Satzes der Seite) → Name *sparse index* (dünner Index),
  Datensätze in Indexdatei bzgl. Primärschl.wert sortiert

- Beispiel: Artikel-Relation, Artnr als Primärschl.


0007 4
0154 20
2300 32
Indexdatei
0007 1234567899999 Bleistifte
0076 8576273739875 Hemd
0078 4628575632943 Vollmilch
Seite 4
0154 9786451239990 Eier 10
0200 9098976234128 Leitz-Ord..
1956 4638829550612 Hemd
Seite 20
2300 4978330867110 Hemd
6666 3867345520013 Nivea Lot..
Seite 32
Hauptdatei

Indexsequentielle Dateiorganisation

- Lookup: gesucht Datensatz zum Primärschlüsselwert w
  Indexdatei sequentiell durchlaufen bis Satz (v1, s) gefunden mit $v1 \leq w$ und
  (v1, s) ist letzter Satz der Indexdatei, oder
  der nächste Satz (v2, s') im Index hat $v2 > w$
  dann sequentielle Suche in Seite s der Hauptdatei

- Einfügen: Lookup anwenden,
  falls möglich, Satz sortiert in gefundener Seite speichern
  sonst: neue Seite holen (von Freiseitenverwaltung), Sätze der 'zu vollen Seite' gleichmäßig auf alte und neue Seite verteilen, Index anpassen

- Löschen: Lookup anwenden,
  Satz in gefundener Seite löschen; falls erster Satz Index anpassen, nachfolgende Sätze nach vorne schieben (Vorsicht bei pinned records)
  falls Seite nach Löschen leer: Index anpassen, Seite an Freiseitenverwaltung

- Suchverfahren:
  - lineares Suchen in Indexdatei, lineares Suchen in Seite der Hauptdatei
  - binäres Suchen der richtigen Indexseite, lineares Suchen in gefundener Indexseite, lineares Suchen in Seite der Hauptdatei

7.4 Interne Ebene: Indiziert-nichtseq. Zugriff

- keine eigene Dateiorganisationsform
- zur Unterstützung von Sekundärschlüsseln
- wie in 7.3 einstufig und mehrstufig
- Aufbau der Hauptdatei: durch Dateiorganisationsform (z.B. indexseq.) vorgegeben, ohne Ordnung bzgl. Sekundärschlüsselwerten
- Aufbau der Indexdatei: zu jedem Satz in Hauptdatei ex. ein Satz in Indexdatei der Form (w, s) mit w Sekundärschlüsselwert des Satzes, s zugeordnete Seite;
  Sortierung innerhalb der Indexdatei: erst nach w, dann nach s
- da Sekundärschl.werte aller Sätze in Indexdatei stehen: *dense index* (dichter Index)
- Beispiel: Artikel-Relation mit Sekundärschl. Artname

Bleistifte	4
Eier 10	20
Hemd	4
Hemd	20
Hemd	32

Leitz-Ord..	20
Nivea Lot..	32
Vollmilch	4

Indexdatei

Seite 4

0007	1234567899999	Bleistifte
0076	8576273739875	Hemd
0078	4628575632943	Vollmilch

Seite 20

0154	9786451239990	Eier 10
0200	9098976234128	Leitz-Ord..
1956	4638829550612	Hemd

Seite 32

2300	4978330867110	Hemd
6666	3867345520013	Nivea Lot..

Hauptdatei

Indiziert-nichtseq. Zugriff

- Lookup: bei gegebenen Sekundärschlüsselwert w lineares oder binäres Suchen nach w in Indexdatei (w kann mehrfach auftreten),
  Seiten zu den ermittelten Seitenadressen komplett durchsuchen

- Einfügen: Lookup und Einfügen in Hauptdatei entspr. der Dateiorganisationsform,
  Indexdatei aufdatieren (Sortierung beachten, keine doppelten Einträge vornehmen!)

- Löschen: Lookup und Löschen entspr. der Dateiorganisationsform,
  Indexdatei aufdatieren (Vorsicht bei gleichen Sekundärschlüsselwerten auf einer Hauptdatei-Seite)

7.5 Interne Ebene: B- und B*-Bäume

- exakte Definition usw. s. Vorlesung Informatik A

- B-Baum: dynamischer Indexbaum, Knoten im Baum = Seiten die mehrere Indexeinträge enthalten, jeder Indexeintrag zeigt auf Seite in Hauptdatei

- Idee: vollständiges Ausgleichen (Wege zu allen Blättern gleichlang, pro Knoten gleichviele Daten) zu teuer → (R. Beyer, 1970): jede Seite außer der Wurzelseite enthält zwischen x und 2x Daten; falls dann n Daten in Hauptdatei reichen $\log_x(n)$ Seitenzugriffe

- 'Definition' von B-Bäumen der Ordnung x:
  - jede Seite enthält höchstens 2x Elemente
  - jede Seite außer der Wurzelseite enthält mindestens x Elemente
  - jede Seite
    - ist entweder Blattseite ohne Nachfolger
    - oder hat y+1 Nachfolger, falls y die Anzahl ihrer Elemente ist
  - alle Blattseiten liegen auf der gleichen Stufe

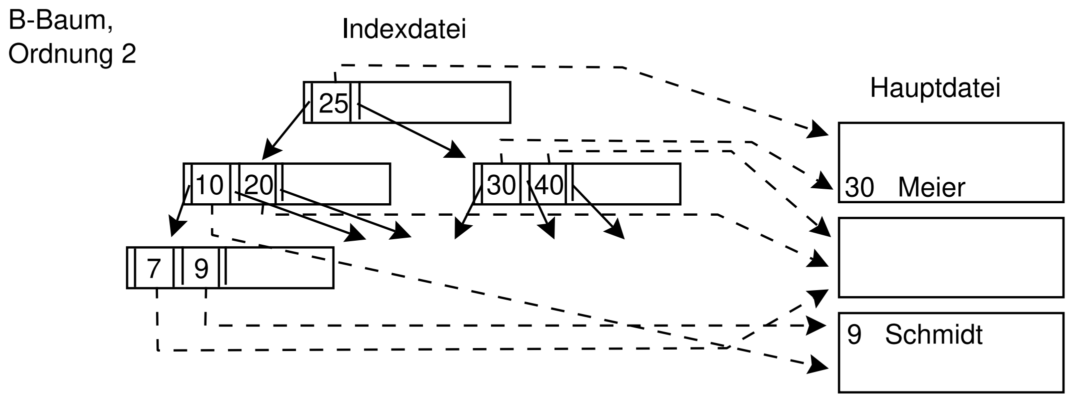
B- und B*-Bäume

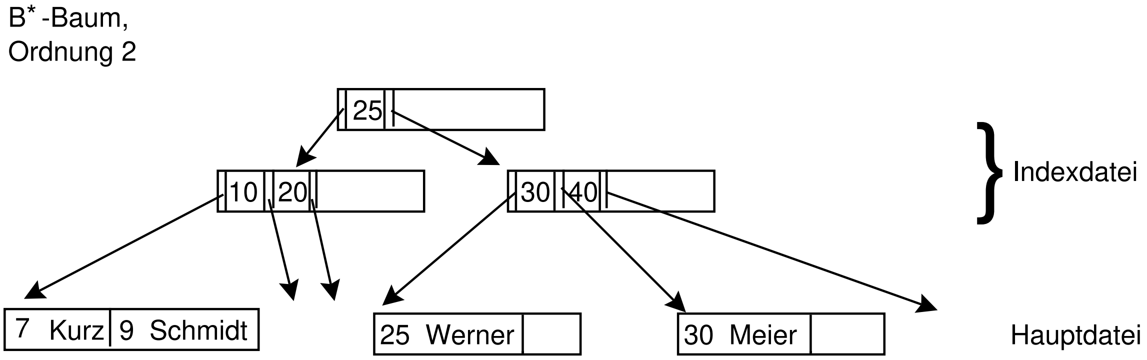
Unterschied B-Baum zu B*-Baum

- B-Baum: bei jedem Elementeintrag in Knoten Zeiger auf Seite in Hauptdatei

- B*-Baum: nur von Blattknoten gehen Zeigen auf Seiten in Hauptdatei

Beispiel:





B- und B*-Bäume

Operationen (hier am B-Baum)

- Lookup: Suche nach Primärschlüsselwert w in B-Baum
  Start bei Wurzelseite, suche zwischen w_1 und w_y Wert w,
  wenn gefunden: fertig,
  falls nicht gefunden: wenn $w_i < w < w_{i+1}$ setze Suche auf Seite p_i fort, wenn $w_y < w$ setze Suche auf Seite p_y fort, wenn $w < w_1$ setzte Suche auf Seite p_0 fort
  solange bis w gefunden oder Nullzeiger erreicht

- Einfügen: füge Primärschlüsselwert w ein
  Lookup ausführen, passende Seite ermitteln
  falls passende Seite $y < 2x$ Elemente hat: w einsortieren, fertig
  falls passende Seite 2x Elemente hat: neue Seite erzeugen, x Elemente auf alter Seite belassen, x Elemente auf neue Seite, 1 (mittleres Element) nach oben einfügen, evtl. Ausgleich rekursiv bis zur Wurzel fortsetzen

- Löschen: Löschen von Primärschlüsselwert w
  Lookup ausführen,
  w auf Blattseite gefunden: löschen, evtl. Unterlauf behandeln, wenn Blattseite < x Elemente hat
  w nicht auf Blattseite gefunden: löschen, durch lexikographisch nächstgrößeres (oder nächstkleinere) Element ersetzen, evtl. Unterlauf behandeln

Quicky: Nacheinander 20, 40, 10, 30, 15, 35, 7, 26, 18, 22, 5, 42, 13, 46, 27, 8, 32, 38, 24, 45, 25 in B-Baum der Ordnung 2 einfügen;
danach nacheinander 25, 45, 24, 38, 32, 8, 27, 46, 13, 42, 5, 22, 18, 26, 7, 35, 15 wieder löschen

7.6 Interne Ebene: KdB-Bäume

- mehrdimensionaler (k-dimensionaler) B-Baum

- kann Primärschlüssel und mehrere Sekundärschlüssel gleichzeitig verwalten

- div. Variationen (KDB-tree, Kd-tree,..)

- Idee: auf jeder Seite steckt Teilbaum, der nach mehreren Attributen hintereinander verzweigt

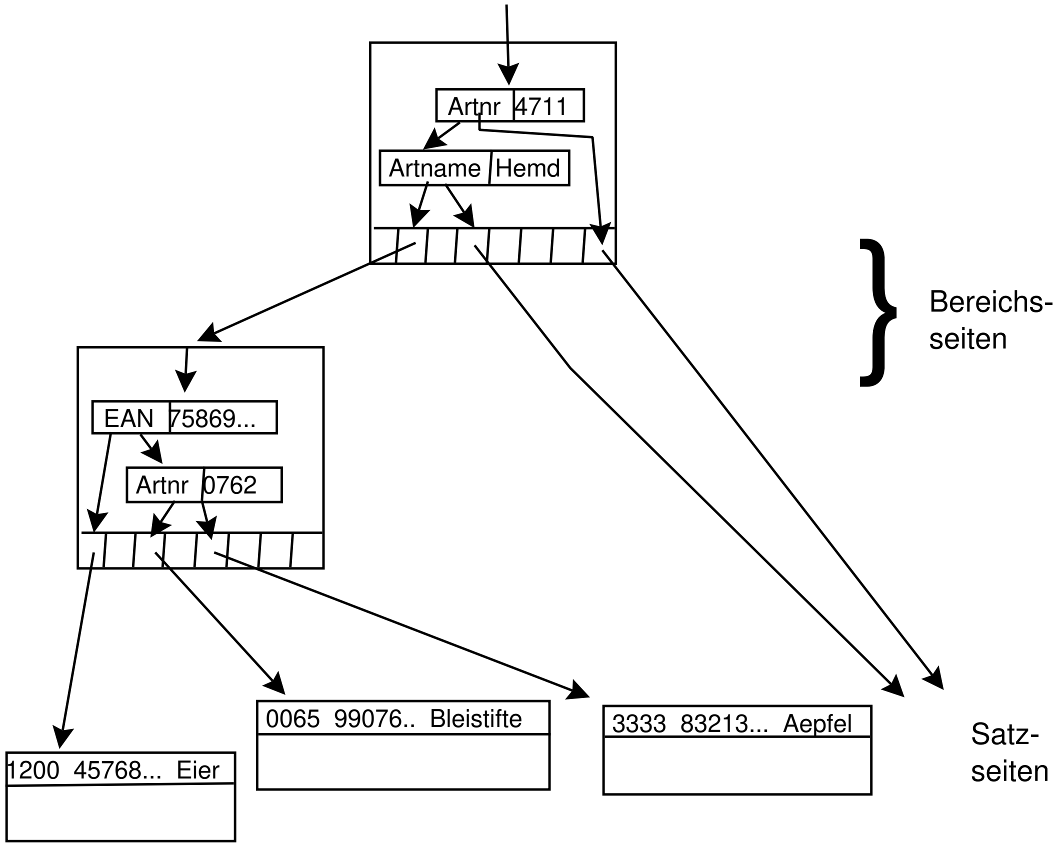
- KdB-Baum vom Typ (b,t):
  - innere Knoten (Bereichsseiten): enthalten einen Kd-Baum mit max. b internen Knoten
  - Blätter (Satzseiten): enthalten bis zu t Tupel der gespeicherten Relation

- Bereichsseite enthält: Anzahl der Schnitt- und Adreß-Elemente der Seite, Zeiger auf Wurzel des in der Seite enthaltenen Kd-Baums, Schnitt- und Adreß-Elemente

- Schnittelement enthält: Index (Nummer eines Attributs), Wert des Attributs, 2 Zeiger auf Nachfolgeknoten innerhalb des Kd-Baums

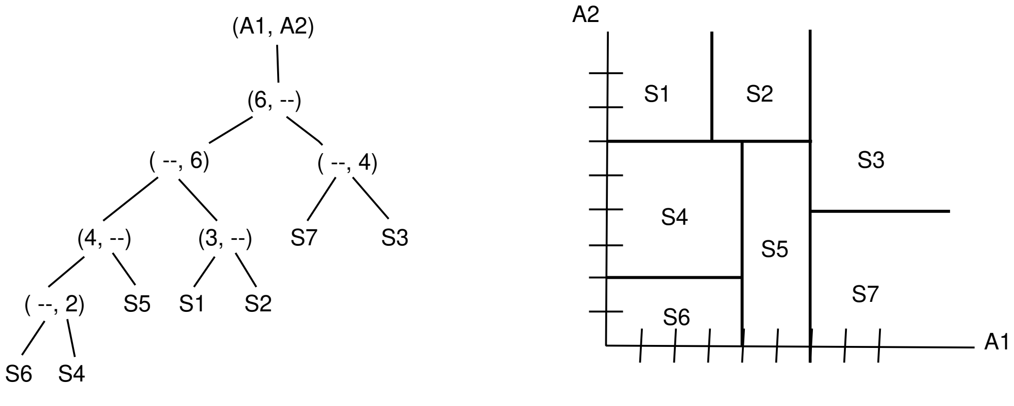
- Adreß-Element enthält: Adresse eines Nachfolgers der Bereisseite im KdB-Baum

- Satzseite enthält: Anzahl der Sätze der Seite, Datensätze


Artnr 4711
Artname Hemd
EAN 75869...
Artnr 0762
Bereichs-
seiten
1200 45768... Eier
0065 99076.. Bleistifte
3333 83213... Aepfel
Satz-
seiten

KdB-Bäume

- Operationen analog zum B-Baum, auch in O(log n) Seitenzugriffen
- es gilt: alle Tupel im linken Teilbaum eines Kd-Baum-Knotens sind bzgl. des dort enthaltenen (Trenn-)Attributs höchstens so groß wie der darin enthaltene (Trenn-)Wert; rechts analog
- Reihenfolge der Trennattribute zyklisch (s. Bsp.)
- 2d-Baum entspr. 2-dimensionaler Einteilung des Datenraums (Vergl. Gridfiles)



7.7 Interne Ebene: Gridfiles

- mehrdimensionale, dynamische Organisationsform
- von Hinrichs und Nievergelt, ETH Zürich, 1981
- wesentliche Zielsetzung: jeder Datensatz in 2 Zugriffen erreichbar ('Zweiplatten-Zugriffsgarantie')
- n-dimensionale Quader bilden Suchregion
- ähnliche Objekte (benachbarte Punkte im Datenraum) sollen auf gleicher Seite gespeichert werden
- symmetrische Behandlung aller Raum-Dimensionen (Mehrdimensionalität)
- dynamische Anpassung beim Löschen und Einfügen

Gridfiles

Prinzip der 2 Plattenzugriffe

- Voraussetzung: Datensatz gesucht, alle k Attributwerte vorgegeben (=*exact match*)

- 1) Abbildung des gesuchten k-Tupels auf die Intervalle der Skalen mittels einer bestimmten Funktion;
  für jeden Attributwert ex. Dimension und damit Skala, die sich aus Intervallen zusammensetzt; Intervalle werden durch Indizes repräsentiert → Kombination von Indizes entspr. der Intervalle ermitteln; da Skalen im Hauptspeicher: 1 Hauptspeicherzugriff

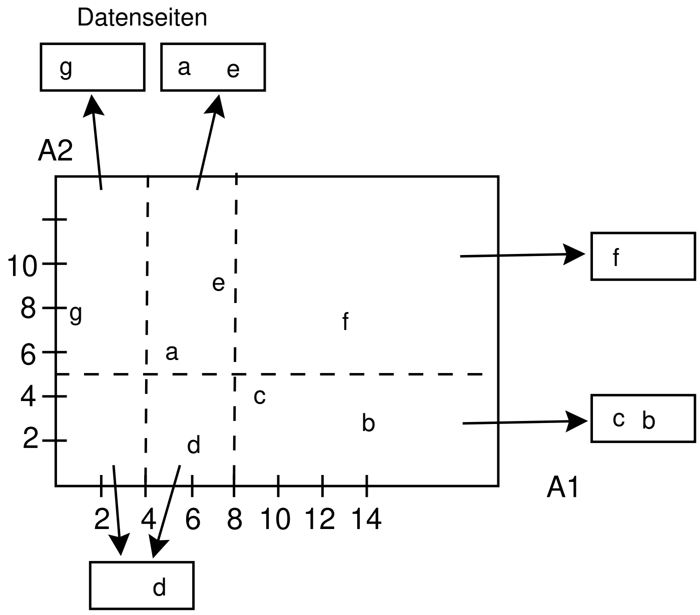
- 2) Über Indizes Zugriff auf Grid-Directory;
  Grid-Directory speichert Adressen der zugeordneten Datenseiten; ist selbst auf Sekundärspeicher gespeichert; Index verweist auf Eintrag im Grid-Directory ; 1 Plattenzugriff

- 3) Falls gesuchter Datensatz existiert, liegt er auf entspr. Datenseite, → Zugriff auf diese Datenseite; 1 Plattenzugriff

Gridfiles

Beispiel: k=2, Relation mit 2 Attributen A1, A2 vom Typ Integer
Skalen: bzgl. A1: 0/4/8, bzgl. A2: 0/5
Grid-Directory G: (mehrere Gitterzellen können auf gleicher Datenseite liegen, bilden eine Region)

	G[*,0]	G[*,1]
G[0,*]	adr1	adr2
G[1,*]	adr1	adr3
G[2,*]	adr4	adr5



Gridfiles

Operationen

- Lookup: Suche nach Datensatz (x,y)
  suche in Skala zu x nach letztem Eintrag der kleiner ist als x (z.B. Index i=3),
  suche in Skala zu y nach letztem Eintrag der kleiner ist als y (z.B. Index j=1),
  lade Teil des Grid-Directories in den HS, der G[i,j] enthält,
  lade Seite mit der Adresse unter G[i,j]

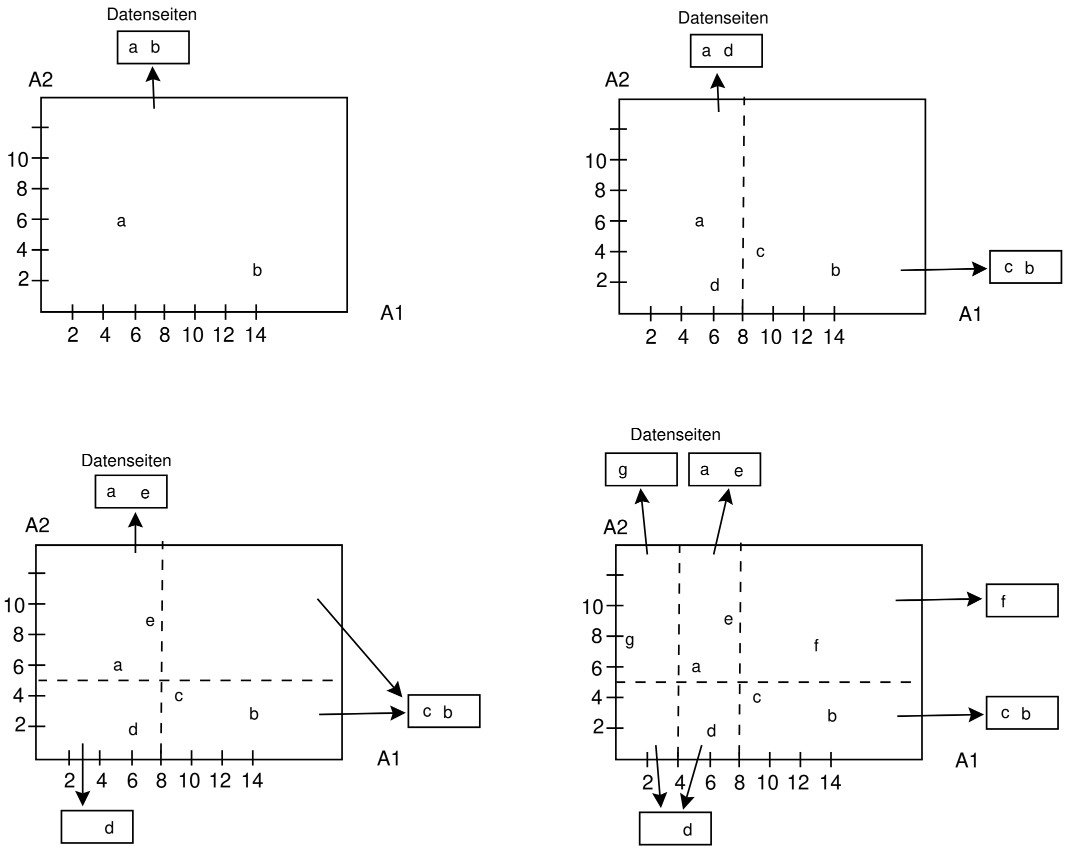
- Einfügen: Start mit Grid-Directory mit 1 Zelle (einer Region) und einer zugeordneten Datenseite,
  ist Seite, auf die Datensatz eingetragen werden soll, voll: Seite teilen,
  gehört nur eine Grid-Zelle zur Datenseite: Unterteilung eines Intervalls auf einer Skala in 2 Intervalle, Aufdatieren des Grid-Directories, Datensätze entspr. der Teilung verteilen, neuen Datensatz auf entspr. Seite speichern

- Löschen: Lookup durchführen,
  Datensatz von Seite löschen, ist Seite leer: Zusammenfügen zweier Regionen

- Modifikation: Datensatz löschen, geänderten Datensatz einfügen

Gridfiles

Beispiel: mehrfaches Einfügen von Datensäten, Annahme: max. 2 Datensätze pro Datenseite



7.8 Interne Ebene: Oracle

- unterstützt nur Index und Hashen

- Index:

  - CREATE [UNIQUE] INDEX indexname
    ON tabellenname (spalte1 [ASC | DESC] [,spalte2 [ASC | DESC],..])
    [optional weitere bestimmte Indexeigenschaften....];

  - ALTER INDEX indexname.....;
    nur zur Änderung der zusätzlichen Indexeigenschaften

  - DROP INDEX indexname;

- Hashing:

  - CREATE CLUSTER clustername
    (spalte1 typ1 [,spalte2 typ2,...])
    [optional weitere Clustereigenschaften]
    HASHKEYS anz_hashwerte [HASH IS hashfkt];

  - hashfkt: voreingestellte Hashfunktion oder eigene nutzbar, muß Wert vom Typ NUMBER liefern

  - anz_hashwerte keine Primzahl: wird von Oracle auf nächstgrößere Primzahl vergrößert

  - ALTER CLUSTER clustername...;
    nur zum Ändern der optionalen Clustereigenschaften

Interne Ebene: Oracle

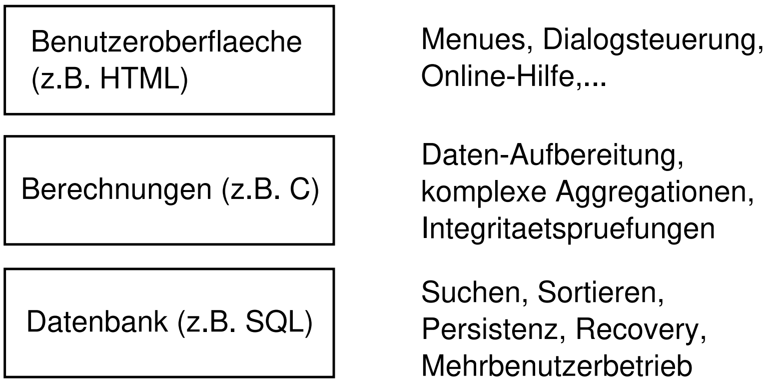
- Forts. Hashing:
  - CREATE TABLE tabellenname
    (...)
    CLUSTER clustername (spalte1 [,spalte2 ...]);

    bewirkt, daß tabellenname entspr. über angegebene Attribute gehasht wird (ins Hash-Cluster aufgenommen wird)
  - DROP CLUSTER clustername;

    löscht Hashcluster, dazu muß Cluster leer sein (alle Tabellen daraus gedropt sein)

8. DB-Anwendungsprogrammierung

- SQL reicht nicht aus
- bel. Programmiersprache reicht nicht aus
- => typische DB-Anwendung:



- Probleme:
  - Benutzer muß 3 Sprachen kennen
  - Reibungsverluste an den Schnittstellen ('Impedance Mismatch')

DB-Anwendungsprogrammierung

Lösungen:

- Erweiterung einer Programmiersprache um call-Schnittstelle, dazu passende Bibliothek mit Funktionen für DB-Zugriff

- Persistente (Datenbank-)Programmiersprachen, z.B.
  - Erweiterung von Pascal um Datentyp Relation
  - Erweiterung von C++, Smalltalk, ... um Persistenz

- Deduktive DBS: Integration von DB und z.B. Prolog

- Programmiersprachliche Erweiterung der DB-Srachen, z.B. von SQL

DB-Anwendungsprogrammierung

in Oracle:

- PL/SQL: Erweiterung von SQL um Kontrollstrukturen
- Embedded SQL: Einbettung von SQL-Anweisungen in C-Programme

PL/SQL

- prozedurale Erweiterung von SQL
- Oracle-Produkt, nicht genormt!
- nicht für Ein-/Ausgaben gedacht

Beispiel:

```
PROCEDURE ueberweisen( x NUMBER, y NUMBER, betrag NUMBER) IS
s konto.kontostand%TYPE; - - Deklarationsteil
BEGIN
SELECT kontostand INTO s FROM konto
WHERE kontonr = x FOR UPDATE OF kontostand;

IF s < betrag THEN
INSERT INTO abgelehnt VALUES (SYSDATE,x,y,betrag);
ELSE
s := s-betrag;
UPDATE konto
SET kontostand = s WHERE kontonr = x;
UPDATE konto
SET kontostand = kontostand+betrag WHERE kontonr = y;

INSERT INTO gebucht
VALUES (SYSDATE,x,y,betrag);
END IF;
COMMIT;
END;
```

PL/SQL

- block-orientiert
- Grundeinheiten: namenloser Block, Prozedur, Funktion
- Aufbau Block:

```
[ DECLARE ....]
BEGIN
..... - - Anweisungen
[ EXCEPTION ..... ]
END;
```

- Blöcke können in Anweisungen und Ausnahmebehandlungen geschachtelt auftreten

PL/SQL

Deklarationsteil:

- Variablen mit SQL-Datentyp (ohne Größenangabe!, nur eine pro Zeile):
  ```
  artikelnr NUMBER;
  gebdat DATE;
  ```

- Variablen mit vordefinierten Subtypen:
  ```
  preis REAL;
  m INT :=2;
  ```

- erlaubte SQL- und Subtypen: char, long, long raw, varchar, varchar2, date, number,
  string, character, boolean, float, int, integer, natural, decimal, dec, real, smallint,...

- Variablen vom Typ eines ex. Attributs bzw. einer ganzen Tabelle:
  ```
  angest angestellter%ROWTYPE;
  angnummer angestellter.angnr%TYPE;
  ```

- Konstanten:
  ```
  limit CONSTANT REAL := 500.00;
  ```

- Cursor-Variablen: Art Tupelzeiger, notwendig um alle Tupel einer (Ergebnis-)Relation abzulaufen
  ```
  CURSOR cname IS sfw-anfrage;
  ```

PL/SQL

Anweisungen: Zuweisungen, IFs, Schleifen, Anfragen, Updates, Prozedur- und Funktionsaufrufe, Cursor-Funktionen

- Zuweisungen (s. obiges Beispiel)

- IF:

```
IF a<b THEN.... - - Anweisungen
END IF;


IF a=c THEN .....
ELSIF ......
ELSE .....
END IF;
```

PL/SQL

Forts. Anweisungen:

- Schleifen:
  - LOOP:

    ```
    LOOP
    .....
    END LOOP;
    ```

    in Verbindung mit `EXIT;` bzw. `EXIT WHEN bed.;` zum Herausspringen aus der Schleife

  - ```
    WHILE a!=b LOOP
    .....
    END LOOP;
    ```

  - ```
    FOR zaehler IN unten..oben LOOP
    ....
    END LOOP;
    ```

    `zaehler` wird automatisch inkrementiert

  - GOTO:

    ```
    ....
    GOTO label1;
    ....
    <<label1>>
    ...
    ```

PL/SQL

Forts. Anweisungen:

- Anfragen:

```
SELECT attr1,... [ INTO var1 ]
FROM relation
WHERE bedingung [ FOR UPDATE OF attr1];
```

  FOR UPDATE-Angabe bewirkt Sperren der Tabelle, `INTO..`-Angabe bei nur einem Ergebnistupel, welches sofort einer Variablen zugeordnet wird (bei mehreren Ergebnistupeln: Cursor benutzen)

- Updates:

```
INSERT INTO relation VALUES (...);
UPDATE relation SET ..... WHERE bedingung;
DELETE FROM relation WHERE bedingung;
```

- Prozedur- und Funktionsaufrufe:
  procname(aktparam1,..);
  var1 := fktname(aktparam1,..);

PL/SQL

Forts. Anweisungen:

- Cursor-Funktionen:

```
DECLARE
CURSOR c1 IS SELECT * FROM artikel;
art artikel%ROWTYPE;
BEGIN
...
OPEN c1; - - jetzt erst wird die Anfrage ausgeführt
...
FETCH c1 INTO art; - - holt (nächstes) Ergebnistupel der Anfrage,
z.B. in LOOP
...
CLOSE c1; - - schließt Cursor, Anfrageergebnis nicht mehr
vorhanden
```

Erneutes `OPEN` nach `CLOSE` stellt Anfrage neu

PL/SQL

Forts. Anweisungen:

- Beispiel:

```
CREATE PROCEDURE auftr_numerieren IS
i NUMBER(6);
auftr auftrag%ROWTYPE;
CURSOR c IS SELECT * FROM auftrag FOR UPDATE; - - mit Locking
BEGIN
i := 1; OPEN c;
LOOP
FETCH c INTO auftr;
EXIT WHEN c%NOTFOUND;
UPDATE auftrag SET auftrnr = i WHERE CURRENT OF c;
i := i + 1;
END LOOP;
CLOSE c;
END;
```

- Cursor-Attribute: c%FOUND, c%NOTFOUND, c%ISOPEN, (jeweils TRUE oder FALSE), c%ROWCOUNT (Anzahl Tupel)

PL/SQL

Exceptions

- um bei Fehlerfällen eigene Routinen zur Ausnahmebehandlung aufzurufen

- Bsp: liefert `SELECT ...INTO ... FROM..WHERE` kein oder mehr als ein Tupel: Fehler `NO_DATA_FOUND` bzw. `TOO_MANY_ROWS` wird ausgelöst

- vordefinierte Behandlungsroutinen für Fehler: zumeist Fehlermeldung, Programmabbruch

- Definition eigener Ausnahmebehandlung für vordefinierte Fehler:

```
...
EXCEPTION
WHEN NO_DATA_FOUND THEN
...- - eigene Anweisungen
WHEN TOO_MANY_ROWS THEN
... - - eigene Anweisungen
WHEN OTHERS THEN
... - - eigene Anweisungen
```

- vordefinierte Fehler:

  NO_DATA_FOUND, TOO_MANY_ROWS, LOGIN_DENIED, NOT_LOGGED_ON, CURSOR_ALREADY_OPEN, INVALID_CURSOR, INVALID_NUMBER, DUP_VAL_ON_INDEX, ROWTYPE_MISMATCH, STORAGE_ERROR, TIMEOUT_ON_RESOURCE, VALUE_ERROR, PROGRAM_ERROR, ZERO_DIVIDE

PL/SQL

Forts. Exceptions

- 'eigene' Fehler: Fehler deklarieren, auslösen, behandeln

- deklarieren:

```
DECLARE
my_except EXCEPTION;
...
```

- auslösen:

```
...
IF (a=0) RAISE my_except;
...
```

- behandeln:

```
WHEN my_except THEN anweisungen;
```

PL/SQL

Aufbau Funktion:

- ```
  FUNCTION fktname [ (par1,...)] RETURN typ IS
  [lokale Deklarationen ]
  BEGIN
  ....- - Anweisungen, darunter mind. eine der Art:
  RETURN wert;
  [EXCEPTION......]
  END;
  ```

- Aufbau der Parameter par1,...:

  parametername [**IN** | **OUT** | **IN OUT**] **typ** [**:= expr** | **DEFAULT expr**]
  voreingestellt: `IN`, **`DEFAULT`**-Angabe nur bei `IN` erlaubt

- Beispiel:

  ```
  FUNCTION erhoehe_gehalt (prozent REAL DEFAULT 1.5)
  RETURN REAL - - hoechstes Gehalt soll zurueckgegeben werden..
  IS
  hoechstes REAL; -- lokale Deklaration
  BEGIN
  UPDATE angestellter
  SET gehalt = gehalt + (prozent*gehalt/100);
  COMMIT; - - Transaktion beenden, Änderung dauerhaft machen
  SELECT max(gehalt) INTO hoechstes
  FROM angestellter;
  RETURN hoechstes;
  END;
  ```
  ```

PL/SQL

Aufbau Prozedur:

- ```
  PROCEDURE pname [(par1,...)] IS
  [lokale Deklarationen ]
  BEGIN
  .... - -Anweisungen
  [EXCEPTION .....]
  END;
  ```
- in SQL*Plus möglich: Prozeduren mittels `CREATE PROCEDURE.....` definieren, mit / Definition abschließen;
  Prozedur mit `EXECUTE pname(aktpar1,....)` ausführen;
- Prozedur neu definieren: `CREATE OR REPLACE PROCEDURE pname....`
- Prozeduren, Funktion löschen mit `DROP PROCEDURE pname` bzw. `DROP FUNCTION fktname`
  ```

Embedded SQL

- SQL-Anweisungen mit dem Präfix `EXEC SQL`, eingefügt in C-Programm namens *dateiname.pc*

- Precompiler *proc* erzeugt daraus 'normales' C-Programm *dateiname.c*

- dieses mit herkömmlichem C-Compiler übersetzen *dateiname.o*

- beim Linken DB-Bibliotheken hinzubinden

- Embedded SQL ist standardisiert

Embedded SQL

- Übergabe von Daten an SQL: mit `:var` Wert der C-Variablen `var` übergeben

```
EXEC SQL INSERT INTO artikel VALUES (:anr, :bez);
```

- dazu: in SQL-Anweisungen auftretende C-Variablen dem Precompiler in Deklarationsblock bekannt machen:

```
EXEC SQL BEGIN DECLARE SECTION;
NUMBER anr;
VARCHAR2 bez;
int zaehler;
EXEC SQL END DECLARE SECTION;
```

- Typumwandlungen:
  - meist automatisch (z.B. zwischen `NUMBER` und `int`; z.T. geht Genauigkeit verloren (bei `float`))
  - Strings: aus `char feld[n]` wird SQL-Typ `CHAR(n)` (Zeichenkette fester Länge) -> Auffüllen mit Leerzeichen notwendig
  - Strings: `VARCHAR2 str[n]` wird vom Precompiler übersetzt in
    ```
    struct{
    unsigned short len;
    unsigned char arr[n];
    } str;
    ```
    wird str durch Oracle mit String belegt, ist str nicht '\0'-terminiert!
  - Strings: alternativ
    `char s[n]; EXEC SQL VAR s IS STRING(n);`

Embedded SQL

Fehlerbehandlung:

- nach jeder SQL-Anweisung sollte Fehlerabfrage durchgeführt werden
- DB schreibt Return-Code in Variable `SQLSTATE`; Variable muß selbst deklariert werden: `char SQLSTATE[6];`
- alternativ: Variable `long SQLCODE` deklarieren; Wert 0 bedeutet 'fehlerfreie Ausführung', Wert 100 bedeutet 'keine weiteren Daten', negative Werte bedeuten Fehler
- weitere Infos zu Fehlern enthält Struktur namens SQL-Communications-Area; Deklaration mit `EXEC SQL INCLUDE SQLCA;`
  Variable heißt dann `sqlca`; in `sqlca.sqlcode` steht Return-Code entsprechend `SQLCODE`,
  in `sqlca.sqlerrm.sqlerrmc` Text der Fehlermeldung (70 Zeichen lang),
  in `sqlca.sqlerrd[2]` Anzahl Tupel, die von letzter SQL-Anweisung betroffen waren

Embedded SQL

Forts. Fehlerbehandlung:

- mit `WHENEVER`-Anweisung automatische Fehlerabfrage nach jeder SQL-Anweisung möglich:

```
EXEC SQL WHENEVER SQLERROR GOTO label;
```

- anstelle von GOTO alternativ: `CONTINUE`
- anstelle von SQLERROR alternativ: `NOT FOUND`
- Beispiel:

```
EXEC SQL WHENEVER SQLERROR GOTO fehler;
...
fehler: printf("\nFehler! %70s\n",sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK RELEASE;
exit(1);
```

- mehrere `WHENEVER`-Statements erlaubt; `CONTINUE` hebt dabei frühere `GOTO`-Anweisung auf

Embedded SQL

Anmelden bei der Datenbank:

- `EXEC SQL CONNECT benutzer IDENTIFIED BY passwort;`

  oder

  `EXEC SQL CONNECT benutzer/passwort;`

- ```
  VARCHAR userid[10];
  VARCHAR passwd[9];
  ....
  strcpy( (char *) userid.arr, "jutta");
  userid.len = strlen( (char *) userid.arr);
  strcpy( (char *) passwd.arr, "geheim!");
  passwd.len = strlen( (char *) passwd.arr);
  EXEC SQL CONNECT :userid IDENTIFIED BY :passwd;
  ```

- Connect-Anweisung entspricht nicht SQL92-Standard
  ```

Embedded SQL

Abmelden bei der Datenbank:

- bei erfolgreichem Programm: `EXEC SQL COMMIT RELEASE;`
- Durch Commit werden Transaktionen dauerhaft gemacht; durch Release werden für Anwendung reservierte Resourcen freigegeben, Verbindung zur Datenbank beendet
- im Fehlerfall: `EXEC SQL ROLLBACK RELEASE;`
- alle noch nicht mittels Commit dauerhaft gemachten Änderungen werden zurückgenommen, durch Release wie oben Abmeldung bei der Datenbank
- in SQL92: `DISCONNECT`-Kommando

Embedded SQL

Arbeiten in C mit SQL-Anfrageergebnissen:

Bei genau einem Ergebnistupel: mit `INTO` arbeiten

Beispiel:

```
VARCHAR bez;
...
EXEC SQL SELECT artname INTO :bez FROM artikel
WHERE artnr = 15;
if(sqlca.sqlcode == 0) {
bez.arr[bez.len]='\0';
printf("Bezeichnung zu Artnr. 15 ist %s\n",bez.arr);
}
..}
```

Embedded SQL

Arbeiten in C mit SQL-Anfrageergebnissen:

- Bei mehreren Ergebnistupeln: nacheinander in Schleife auslesen, mit Cursor
- `EXEC SQL DECLARE c1 CURSOR FOR sfw-Anfrage;`
- `EXEC SQL DECLARE c CURSOR`
  `SELECT artnr FROM artikel WHERE artname = :aname;`
- dann Cursor öffnen: `EXEC SQL OPEN c;`
  jetzt wird Anfrage ausgeführt mit aktuellem Wert von `aname`
- mit Fetch nächstes Tupel holen: `EXEC SQL FETCH c INTO :artikelnr;`
- zuletzt Cursor schließen: `EXEC SQL CLOSE c;`
- bei `UPDATE` und `DELETE` zuletzt mit Fetch geholtes Tupel ansprechbar: `EXEC SQL DELETE FROM artikel WHERE CURRENT OF c;`
  dazu bei Cursor-Deklaration `FOR UPDATE` hinzufügen

Embedded SQL

Nullwerte:

- Wenn Anfrageergebnis Nullwert liefern kann: neben Variable noch Indikator-Variable angeben
- Indikatorvariable ist -1, wenn Nullwert geliefert wurde, sonst 0
- Syntax: `variable INDICATOR ind-variable`
- Schlüsselwort INDICATOR kann auch weggelassen werden, Indikator-Variable zuvor als `short` deklarieren
- Beispiel:

```
short v1_ind, v2_ind;
...
EXEC SQL FETCH c1 INTO :liefname, :artikelnr,
:preis INDICATOR :v1_ind;
...
EXEC SQL FETCH c2 INTO :artikelnr, :bez :v2_ind;
```

Embedded SQL

- innerhalb von Embedded SQL: PL/SQL-Blöcke

- dazu:
  `EXEC SQL EXECUTE ...- - PL/SQL-Block END-EXEC;`

- Precopiler mit Option `SQLCHECK=SEMANTICS` aufrufen

- Relationen, die in PL/SQL-Block benutzt werden, mit **`EXEC SQL DECLARE relname TABLE (attr1 Typ1,...);`** bekannt machen

- Beispiel:

```
#include<stdio.h>
#include<string.h>

EXEC SQL BEGIN DECLARE SECTION;
VARCHAR userid[9];
VARCHAR passwd[9];
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA;

int main(void)
{
char user[9];
char passwort[9];

printf("Ihr Login-Name?");
scanf("%s",user);
printf("\nIhr Passwort?");
scanf("%s",passwort);


EXEC SQL WHENEVER SQLERROR GOTO fehler;
EXEC SQL WHENEVER NOT FOUND GOTO ende;
```

```
strcpy( (char *) userid.arr, user);
userid.len = strlen( (char *) userid.arr);
strcpy( (char *) passwd.arr, passwort);
passwd.len = strlen( (char *) passwd.arr);
EXEC SQL CONNECT :userid IDENTIFIED BY :passwd;

printf("Alle Milch-Artikel: \n");
printf("ab jetzt in der Tabelle 'milchprod'\n");


EXEC SQL DECLARE artikel TABLE
(artnr number(4),
artname VARCHAR2(40)
);
EXEC SQL DECLARE milchprod TABLE (
nr number(4),
bez varchar2(40)
);
EXEC SQL EXECUTE
DECLARE
CURSOR c IS SELECT * FROM artikel WHERE artname LIKE '%milch%';
BEGIN
FOR rec IN c LOOP
INSERT INTO milchprod
VALUES (rec.artnr,rec.artname);
END LOOP;
END;
END-EXEC;

ende: EXEC SQL COMMIT RELEASE;
return 0;

fehler: printf("\nFehler! %70s\n",sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK RELEASE;
exit(1);
}
```

Embedded SQL, PL/SQL

Rekursion:

- durch rekursive PL/SQL-Funktion
- durch rekursives C-Programm mit Embedded SQL-Anweisungen
- durch 'erweitertes' Select-Statement mit `START WITH` und `CONNECT BY PRIOR`-Angaben
- Beispiel:

Embedded SQL, PL/SQL

```
#include<stdio.h>

EXEC SQL BEGIN DECLARE SECTION;
VARCHAR userid[9];
VARCHAR passwd[9];
VARCHAR ch[40];
VARCHAR un[40];
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE c CURSOR FOR
SELECT chef, angname INTO ch, un
FROM sn_ang_chef
START WITH chef='Bond'
CONNECT BY PRIOR angname=chef;

EXEC SQL INCLUDE SQLCA;

int main(void)
{...
printf("Alle Untergebenen: \n");
EXEC SQL OPEN c;
if(sqlca.sqlcode == 0) {
printf("Vorgesetzter: \t\t Untergebener:\n");
while(1){
EXEC SQL FETCH c INTO :ch, :un;
ch.arr[ch.len]='\0';
un.arr[un.len]='\0';
printf("%s\t\t %s\n",ch.arr,un.arr);
}
}
...
}
```

Embedded SQL

- bisher: statisch (SQL-Anweisung steht bei Precompiler-Lauf fest)
- auch möglich: dynamisch (zur Laufzeit wird Anfrage erst erzeugt)
- dazu
  1. SQL-Variable vom Typ statement vereinbaren:

     ```
     EXEC SQL DECLARE s1 STATEMENT;
     ```

  2. Statement-Variablen einen SQL-Anfragestring zuweisen:

     ```
     EXEC SQL PREPARE s1 FROM 'select * from liefert;'
     ```

  3. Anfrage ausführen:

     ```
     EXEC SQL EXECUTE s1 [USING :var1,...];
     ```

- Anfragestring kann Stringvariable sein (z.B. `:string1`)
- Statement kann Cursor zugeordnet und damit Ergebnisrelation durchlaufen werden (s. Bsp1)
- Anfragestring selbst kann Variablen enthalten, diese werden durch `USING ...` beim Aufruf mit akt. Variablenwerten aufgefüllt (s. Bsp2)

Embedded SQL

Beispiel1 (dynamisch)

```
#include<stdio.h>
#include<string.h>

EXEC SQL BEGIN DECLARE SECTION;
VARCHAR userid[9];
VARCHAR passwd[9];
VARCHAR bez[40];
int nummer;
char string1[256];

EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE s1 STATEMENT;

EXEC SQL DECLARE c CURSOR FOR
SELECT artnr, artname FROM artikel
WHERE artname like '%milch%';

EXEC SQL DECLARE c1 CURSOR FOR s1;

EXEC SQL INCLUDE SQLCA;

int main(void)
{
char user[9];
char passwort[9];
char u_bez[50];

printf("Ihr Login-Name?");
scanf("%s",user);
printf("\nIhr Passwort?");
scanf("%s",passwort);

EXEC SQL WHENEVER SQLERROR GOTO fehler;

strcpy( (char *) userid.arr, user);
userid.len = strlen( (char *) userid.arr);
strcpy( (char *) passwd.arr, passwort);
```

```
passwd.len = strlen( (char *) passwd.arr);
EXEC SQL CONNECT :userid IDENTIFIED BY :passwd;
printf("Welche Artikel (Wort in Art.bezeichnung) interessieren Sie?
");
scanf("%s",u_bez);

strcat(string1,"select artnr, artname from artikel where artname
like '%");
strcat(string1,u_bez);
strcat(string1,"%'");

printf("\n\nIhre Anfrage lautet:\n%s\n",string1);

EXEC SQL PREPARE s1 FROM :string1;
printf("\n\nAlle %s-Artikel: \n",u_bez);
EXEC SQL EXECUTE s1;
EXEC SQL OPEN c1;

EXEC SQL WHENEVER NOT FOUND GOTO next;

if(sqlca.sqlcode == 0) {
printf("Artikelnr:\t\tArtikelname\n");
while(1){
EXEC SQL FETCH c1 INTO :nummer, :bez;
bez.arr[bez.len]='\0';
printf("%-10d\t\t %s\n",nummer,bez.arr);
}
}

next: EXEC SQL CLOSE c1;
printf("\n\n");

EXEC SQL WHENEVER NOT FOUND GOTO ende;

printf("Alle Milch-Artikel: \n");
EXEC SQL OPEN c;
if(sqlca.sqlcode == 0) {
printf("Artikelnr:\t\tArtikelname\n");
while(1){
EXEC SQL FETCH c INTO :nummer, :bez;
bez.arr[bez.len]='\0';
printf("%-10d\t\t %s\n",nummer,bez.arr);
```

```
}
}
EXEC SQL CLOSE c;
ende: EXEC SQL COMMIT RELEASE;
return 0;

fehler: printf("\nFehler! %70s\n",sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK RELEASE;
exit(1);
}
```

Embedded SQL

Beispiel2 (dynamisch)

```
#include<stdio.h>
#include<string.h>

EXEC SQL BEGIN DECLARE SECTION;
VARCHAR userid[9];
VARCHAR passwd[9];
VARCHAR bez[40];
int nummer;
char string1[256];
int u_bez;
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE s1 STATEMENT;
EXEC SQL DECLARE c1 CURSOR FOR s1;

EXEC SQL INCLUDE SQLCA;

int main(void)
{
char user[9];
char passwort[9];

.........

printf("Welcher Artikel (Artikelnr) interessiert Sie? ");
scanf("%d",&u_bez);

EXEC SQL PREPARE s1 FROM 'select artnr, artname from artikel where
artnr = :u_bez';

printf("\n\nArtikel: \n");
EXEC SQL EXECUTE s1 USING :u_bez;
```

```
EXEC SQL OPEN c1;

EXEC SQL WHENEVER NOT FOUND GOTO ende;

if(sqlca.sqlcode == 0) {
printf("Artikelnr:\t\tArtikelname\n");
while(1){
EXEC SQL FETCH c1 INTO :nummer, :bez;
bez.arr[bez.len]='\0';
printf("%-10d\t\t %s\n",nummer,bez.arr);
}
}

EXEC SQL CLOSE c1;
printf("\n\n");

ende: EXEC SQL COMMIT RELEASE;
return 0;

fehler: printf("\nFehler! %70s\n",sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK RELEASE;
exit(1);
}
```

DB-Anwendungsprogrammierung: Trigger

Allgemein:

- Trigger sind ECA-Regeln: Event-Condition-Action
- wenn auslösendes Event auftritt und Condition erfüllt ist, wird Action ausgeführt (ähnlich Prozeduren, die nicht explizit aufgerufen werden müssen
- Trigger nicht in SQL92 enthalten, für SQL3 vorgesehen
- Trigger in Oracle enthalten, sehr unterschiedlich zum SQL3-Vorschlag
- Trigger heißen auch 'aktive Regeln', DB mit Triggern 'aktive DB'
- Anwendungen von Triggern: Integritätsüberwachungen, Aktualisierung abgeleiteter Daten, Protokollierung von Änderungen, Anstoßen von Geschäftsprozessen

Trigger

Auslösende Ereignisse (Event):

- Einfügen in eine bestimmte Tabelle (nicht in Views)
- Löschen aus einer bestimmten Tabelle (nicht in Views)
- Änderung in einer bestimmten Spalte
- weitere Auslöser denkbar (Datumsabhängigkeit, bei jedem `commit`, vor/nach Ausführung einer Prozedur,...), in Oracle aber nicht vorgesehen

Bedingungen (Condition): in Oracle nur sehr einfache Bedingungen, beziehen sich auf das gerade betroffene Tupel; Aktionsteil kann aber bel. komplex sein

Aktionen (Action): bel. PL/SQL-Code, aber ohne `commit/rollback`;
`raise_application_error` aufrufbar -> SQL-Anweisung, die zu Fehler geführt hat, kann abgebrochen werden, alle (auch vom Trigger) vorgenommenen Änderungen werden rückgängig gemacht

Trigger

Beispiel:

- zum Warenhaus Relation Bestand mit Attributen Artnr, Anzahl, mindAnzahl; Relation Bestellung mit Attributen Datum und Artnr;
  Bestellungen sollen automatisch erzeugt werden, wenn akt. Bestand unter Mindestbestand sinkt

- ```
  CREATE TRIGGER Nachbestellung
  AFTER UPDATE OF Anzahl ON Bestand
  FOR EACH ROW
  WHEN (new.Anzahl < new.mindAnzahl)
  BEGIN    INSERT INTO Bestellung
  VALUES (SYSDATE, :new.Artnr);
  END;
  ```

- Alternative: Bedingung im Aktiosteil `CREATE TRIGGER`
  ```
  Nachbestellung
  AFTER UPDATE OF Anzahl ON Bestand
  FOR EACH ROW
  BEGIN
  IF :new.Anzahl < :new.mindAnzahl THEN
  INSERT INTO Bestellung
  VALUES (SYSDATE, :new.Artnr);
  END IF;
  END;
  ```
  ```

Trigger

Zeitpunkt des Triggeraufrufs:

- `BEFORE`: Trigger wird vor Ausführung der auslösenden Operation aufgerufen
- `AFTER`: Trigger wird nach Ausführung der auslösenden Operation aufgerufen, kann schon veränderten Zustand sehen

Triggerarten in Oracle:

- Befehlstrigger: wird nur einmal vor/nach Operation aufgerufen; haben keine Möglichkeit, auf die von Änderung betroffenen Tupel zuzugreifen
- Zeilentrigger (`FOR EACH ROW`): wird einmal für jedes geänderte Tupel aufgerufen; in Trigger keine Anfragen erlaubt, die sich auf andere Tupel beziehen; Zugriff auf alten Datensatz: `old`, auf neuen Datensatz: `new`, jeweils implizit definiert vom Typ relation%ROWTYPE (im PL/SQL-Block mit : zugreifen)

Trigger

Sonstiges:

- Action kann selbst DB-Änderungen enthalten -> kann weitere Trigger auslösen (Problem: Triggerausführung könnte nicht terminieren)

- in Oracle Trigger formulierbar, die bei verschiedenen Events 'feuern':

  ```
  CREATE TRIGGER Nachbestellung
  AFTER INSERT OR UPDATE ON Bestand
  ...
  ```

  Falls später Unterscheidung nötig: in PL/SQL mit `IF INSERTING THEN...` auf jeweilige Operation beziehen

- in sqlplus: Triggerdefinition mit einem '/' abschließen

- Syntax:

  ```
  CREATE TRIGGER Triggername BEFORE |AFTER Operation ON Relation
  [FOR EACH ROW [WHEN ( Bedingung )] ]
  [DECLARE deklaration;....]
  BEGIN anweisung;.....END;
  ```

  Syntax von `Operation`: `DELETE | INSERT | UPDATE [OF spalte1,...]`
  `[ OR DELETE | INSERT | UPDATE [OF spalte1,...]`
  `[ OR ...]]`

- Löschen von Triggern: `DROP TRIGGER Triggername;`

9. Transaktionen, Datenschutz, Datensicherheit

Transaktionen:

- Folge von DB-Operationen, die bzgl. Integritätsüberwachung als eine Einheit angesehen werden

- DB muß nur vor und nach Transaktionsausführung in konsistentem Zustand sein

- müssen ACID-Eigenschaften erfüllen:
  - A: Atomarität; Transaktion wird ganz oder gar nicht ausgeführt
  - C: Konsistenz; DB muß nach Transaktion in zulässigem Zustand sein
  - I: Isolation; evtl. parallel ablaufende Transaktionen sind isoliert, können sich nicht gegenseitig beeinflussen
  - D: Dauerhaftigkeit; Wirkung einer erfolgreich beendeten Transaktion ist dauerhaft

Transaktionen

- Beispiel:

  Kontobuchung eines Betrags s von Kontonr. x nach Kontonr. y:
  IB: Summe der Kontostände vor und nach Umbuchung muß unverändert sein;
  während der Ausführung: nach Abbuchung von s von x ist IB nicht erfüllt, erst wieder nach Zubuchung auf y -> während laufender Transaktion kann IB nicht erfüllt sein

- in der Praxis:
  - Mehrbenutzerbetrieb
  - nebenläufige, konkurrierende Prozesse
  - -> atomarer, völlig isolierter Ablauf von Transaktionen nicht möglich
  - Hardwarefehler, Laufzeitfehler,...

- Forderung: Ergebnis einer Transaktion soll so aussehen, als sei sie nach dem ACID-Prinzip abgelaufen

Transaktionen

Abhilfe: Sperren (Locks)

- von Datenelementen:
  - ganze Relation
  - einzelne Tupel
  - einzelne Attributwerte
  - einzelne Seiten

- je größer Datenelement, umso geringer Verwaltungsaufwand, aber auch umso geringer Parallelität

- lock: Transaktion sperrt Datenelement vor Bearbeitung; ist Datenelement schon (von anderer Transaktion) gesperrt: warten, bis Sperre aufgehoben, dann sperren

- unlock: Entsperren des Datenelements nach Bearbeitung

Transaktionen

Probleme: Verklemmungen

- Behinderungen: eine Transaktion versucht, ein Datenelement zu sperren; dieses ist ständig durch andere Transaktionen abwechselnd gesperrt

- Deadlock:

  T1: lock A; ...; lock B;...;unlock A;...unlock B;
  T2: lock B; ...; lock A;...;unlock B;...unlock A;

- DBMS muß Verklemmungen vorbeugen, erkennen, behandeln
  - z.B. Behinderungen durch Warteschlangen, in die sich Transaktionen einreihen müssen
  - z.B. durch Forderung, daß alle Locks einer Transaktions gleichzeitig geschehen müssen
  - z.B. durch lineare Ordnung auf Datenelementen (lock A immer vor lock B)
  - z.B. bei Deadlocks durch Stoppen von Transaktionen, später erneutes Anlaufen

Transaktionen

Serialisierbarkeit:

- Ausführung konkurrierender Transaktionen ist **korrekt**, wenn Ergebnis dem einer seriellen Abarbeitung der Transaktionen (in bel. Reihenfolge) entspricht

- Schedule: Plan, der eine geordnete Folge von lock-, read-, write- und unlock-Aktionen für eine Menge von Transaktionen enthält

- serieller Schedule: alle Aktionen jeder Transaktion folgen unmittelbar aufeinander

- Schedule serialisierbar: Ergebnis eines Schedules ist äquivalent zu dem eines seriellen Schedules

- Beispiel:

S1:		S2:		S3	
T1	T2	T1	T2	T1	T2
read A		read A		read A	
A=A-10			read B	A=A-10	
write A		A=A-10			read B
read B			B=B-20	write A	
B=B+10		write A			B=B-20
write B			write B	read B	
	read B	read B			write B
	B=B-20		read C	B=B+10	
	write B	B=B+10			read C
	read C		C=C+20	write B	
	C=C+20	write B			C=C+20
	write C		write C		write C

S1: seriell, S2: nicht seriell aber serialisierbar, S3: nicht serialisierbar

Transaktionen

verschiedene Transaktionsmodelle: welche garantieren Serialisierbarkeit?

- einfaches lock-unlock-Modell:
  - Sperren setzen:
    lock A - mit implizitem read A
    unlock A - mit implizitem write A
    dazwischen Funktion(en), die den Wert von A verändern
  - Test auf Serialisierbarkeit: mittels Vorgängergraph
    Algorithmus: gegeben T1,..., Tn mit Schedule; bilde gerichteten Graphen, dessen Knoten den Ti entsprechen; suche Paare im Schedule der Art: Ti: unlock A.....Tj: lock A (nächstes nach unlock); jedes solche Paar bildet Kante von Ti nach Tj;
    hat Graph schließlich keinen Kreis: Schedule ist serialisierbar
  - Beispiel:

T1:	T2:	T3
lock A		
	lock B	
	lock C	
	unlock B	
lock B		
unlock A		
	lock A	
	unlock C	
	unlock A	
		lock A
		lock C
unlock B		
		unlock C
		unlock A

Transaktionen

Transaktionsmodelle

- 2-Phasen-Sperr-Protokoll:
  - alle Sperren (1. Phase) kommen vor allen Freigaben (2. Phase)
  - erfüllen alle Transaktionen jeweils obige Bedingung: Schedule immer serialiserbar
  - Deadlock-Problem aber damit nicht gelöst

- read-write-Modell: Unterscheidung zwischen read-only- und read-write-Zugriff

- read-only-write-only-Modell: berücksichtigt auch write-only-Zugriff

- optimistische Methoden: sperren keine Datenelemente; lassen bei jeder Lese-/Schreiboperation prüfen, ob serielle Ordnung verletzt ist

Transaktionen

Transaktionsabbruch:

- bisher: Annahme alle Transaktionen beenden Arbeit
- Problem: Transaktion kann vorher sterben (Deadlock, ^C, Division durch 0...)
- Unterscheidung:
  - laufende Transaktion (vor COMMIT): kann noch abgebrochen werden, Änderungen nicht dauerhaft gespeichert
  - abgeschlossene Transaktion (nach COMMIT): darf wegen Eigenschaft D nicht zurückgesetzt werden
  - -> kritischer Befehl ist COMMIT
- Beispiel:

	T1:	T2:
1	lock A	
2	read A	
3	A=A-1	
4	write A	
5	lock B	
6	unlock A	
7		lock A
8		read A
9		A=A*2
10	read B	
11		write A
12		COMMIT
13		unlock A
14	B=B/A (A sei 0)	

Transaktionen

Transaktionsabbruch:

- Was müßte passieren?
  - lock B (T1) lösen
  - neuen Wert von A (Zeile 3,4; T1) durch alten Wert ersetzen
  - T2 hat dann aber falschen Wert gelesen -> T2 insgesamt zurücksetzten
  - falls zwischen Zeile 13,14 andere Transaktionen A gelesen hätten: diese auch zurücksetzen (cascading rollback)

- aber: cascading rollback unerwünscht (widerspricht Transaktionsdefinition)

- -> strenges 2-Phasen-Sperr-Protokoll:
  - Transaktion darf nicht in DB schreiben, bevor sie COMMIT erreicht hat
  - vor COMMIT werden keine unlocks ausgelöst
  - -> (neuer) Wert in DB kann nur von COMMITed Transaktion geschrieben werden, Transaktion ist sicher dauerhaft

Transaktionen

Transaktionen in Oracle (in sqlplus, PL/SQL):

i.A.: Oracle sperrt alle benötigten Daten automatisch; explizite Angaben:

- `COMMIT [WORK]`: dauerhafte Speicherung der bisherigen Änderungen, Lösen aller Sperren, Löschen alles Savepoints der Transaktion, Start einer neuen Transaktion;
  vor u. nach jedem DDL-Befehl: automatisches `COMMIT`

- `ROLLBACK [WORK]`: Rücksetzen aller Änderungen seit dem letzten Transaktionsbeginn, Beenden der Transaktion, Löschen aller Savepoints der Transaktion, Lösen aller Sperren;

- `ROLLBACK [WORK] TO [SAVEPOINT] spname`: Rollback der aktuellen Transaktion bis zum Savepoint spname, Löschen aller Savepoints nach spname, Lösen aller Sperren seit dem Savepoint spname

- `SAVEPOINT spname`: Setzen eines Savepoints namens spname

- `SET TRANSACTION READ ONLY | READ WRITE | ISOLATION LEVEL SERIALIZABLE | ISOLATION LEVEL READ COMMITTED | USE ROLLBACK SEGMENT rb_seg_name`:
  setzt den Modus der aktuellen Transaktion (gilt nur bis zum nächsten `COMMIT`, `ROLLBACK`, `DDL`-Kommando), Voreinstellung: `READ WRITE`

- `LOCK TABLE relname1 [,relname2,..] IN modus MODE [NOWAIT]`:
  Sperren von Relation(en) oder View(s); überschreibt

automatische Sperren; erlaubte Modi:
`ROW SHARE`, `SHARE UPDATE`: erlaubt gleichzeitigen lesenden Zugriff, verbietet Updates; kein anderer Benutzer kann `EXCLUSIVE`-Modus auf Relation setzen
`ROW EXCLUSIVE`: erlaubt gleichzeitigen lesenden Zugriff, verhindert `SHARE`-Sperren durch andere; automatisch gewählter Modus bei `UPDATE`, `INSERT`, `DELETE`
`SHARE`: erlaubt gleichzeitigen lesenden Zugriff, verbietet Updates; kann von verschiedenen Benutzern auf gleicher Relation gegeben werden
`EXCLUSIVE`:erlaubt lesenden Zugriff, verbietet jede andere Aktivität anderer
-> keine Relation kann gegen Lesen anderer gesperrt werden!

- `SELECT ...FOR UPDATE` (in PL/SQL): expl. Sperren einer Relation/von Tupeln, Sperre gilt bis `COMMT/ROLLBACK`

Transaktionen im Embedded SQL: wie oben mit Präfix EXEC SQL

Datenschutz, Datensicherheit

Datenschutz: Schutz der Daten gegen unbefugtes Lesen, Modifizieren, Löschen

div. Techniken:

- Benutzer-Identifikation (Paßwort,..)

- Physikal. Schutzmaßnahmen, Kryptographie

- Benutzerrechte:

  - Rechte, Anfragen an bestimmte Relationen anderer zu stellen (`GRANT SELECT ON relname TO jutta;`)

  - Rechte, bestimmte 'Änderungsoperationen' durchzuführen (`GRANT UPDATE ON relname TO public; GRANT CREATE TRIGGER TO gremeyer;`)

  - speziell: Möglichkeit, erhaltenes Recht an andere weiterzugeben (`GRANT SELECT ON relname TO gremeyer WITH GRANT OPTION;`)

  - Rückname von vergebenen Rechten (`REVOKE recht ON relname FROM benutzer [RESTRICT|CASCADE]`)

- Rechte nicht auf Basisrelationen, sondern nur auf Ausschnitten (Views) -> Sichten im Rahmen von Datenschutz wichtig

Datenschutz, Datensicherheit

Datensicherheit in Datenbanken:

- Wiederherstellung der Daten bei Systemfehlern: Recovery

- 2 Speicherarten:
  instabiler: Hauptspeicher, Cache; Inhalt geht bei Systemfehler verloren
  stabiler: Platte, Diskette, CD,..; Inhalt geht bei Fehler der Speichermediums verloren

- Maßnahmen gegen Datenverlust

  - vom instabilen Speicher: Log-Buch, Journal, Wiederherstellungsprotokolle, Einsatz von Schatten-Speichern

  - vom stabilen Speicher: Backups, Archive, komplette Spiegelungen

Datenschutz, Datensicherheit

Log-Buch:
enthält vollständige Historie mit allen Änderungen auf der Datenbank; z.B. Eintrag (T,begin) bei Start einer Transaktion T, (T,A,a) für T schreibt für Attribut A Wert a,... (T,A,a',a) für T schreibt für A bei altem Wert a' neuen Wert a, (T,commit), (T,savepoint s),...

Problem: Log-Buch in instabilem Speicher: kann verloren gehen -> Log-Seiten müssen sofort auf stabilen Speicher geschrieben werden

Journal:
neben Log-Daten auch Daten über Benutzer, Zugriffsstatistiken, etc

Recovery: Algorithmus, wird nach Systemfehler gestartet, untersucht Log-Buch, setzt DB in letztmöglichen konsistenten Zustand zurück;
Algorithmus: Log-Buch rückwärts durchlaufen bis zu letzten savepoint- oder commit-Einträgen; ab dort vorwärts alle (T,A,a)-Einträge finden, a für A in DB schreiben; Transaktionen ohne (T,commit) oder mit (T,abort) ignorieren, dafür nur Warnung ausgeben ('T not committed')

10. Ausblick

- objektorientierte Datenbanksysteme
  - adäquatere Modellierung von Non-Standard-Anwendungen
  - kein einheitliches Datenmodell, fehlende Standards
  - fehlende 'saubere' Konzepte für grundlegende Funktionalitäten
- objekt-relationale Datenbanksysteme
  - (erweiterbares) relationales DB-System
  - (benutzerdefinierte) Klassenbibliotheken mit neuen komplexen Datentypen plus Operationen darauf
- Föderierte Datenbanksysteme
  - zur Integration von heterogenen Datenbanken
  - für systemübergreifende Anwendungen
  - unter Beibehaltung der Autonomie der bestehenden heterogenen Systeme